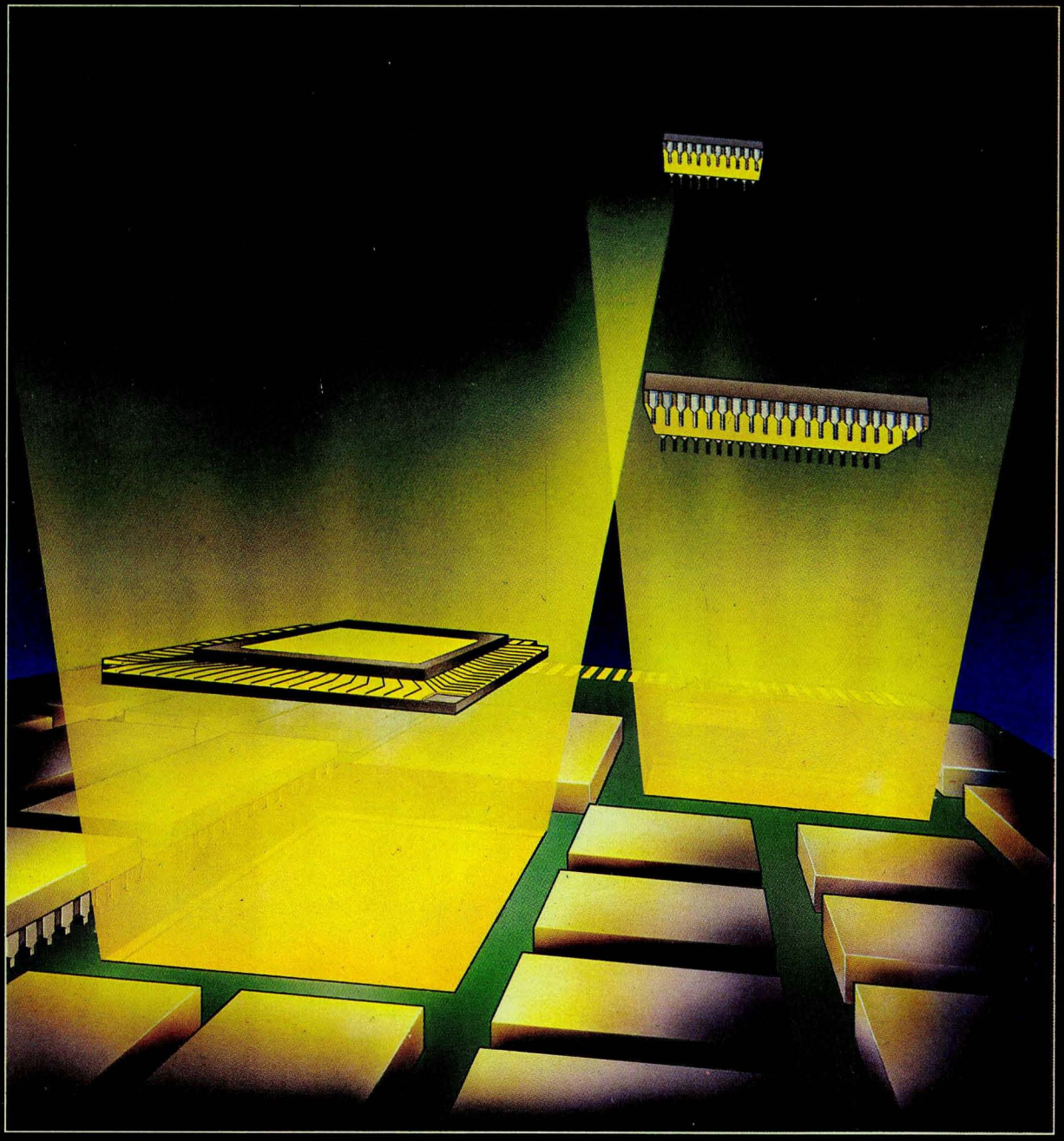
intel

Microsystem Components Handbook Volume II



Order Number: 230843-001

LITERATURE

In addition to the product line Handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846) provides an overview of Intel's complete product line and customer services.

Consult the INTEL LITERATURE GUIDE for a complete listing of Intel literature. TO ORDER literature in the United States, write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

1984 HANDBOOKS	U.S. PRICE*
Memory Components Handbook (Order No. 210830) Contains all application notes, article reprints, data sheets, and other design information on RAMs, DRAMs, EPROMs, E^2PROMs, Bubble Memories.	**$15.00**
Telecommunication Products Handbook (Order No. 230730) Contains all application notes, article reprints, and data sheets for telecommunication products.	**7.50**
Microcontroller Handbook (Order No. 210918) Contains all application notes, article reprints, data sheets, and design information for the MCS-48, MCS-51 and MCS-96 families.	**15.00**
Microsystem Components Handbook (Order No. 230843) Contains application notes, article reprints, data sheets, technical papers for microprocessors and peripherals. (2 Volumes) (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc. Consult the Literature Guide for prices and order numbers.)	**20.00**
Military Handbook (Order No. 210461) Contains complete data sheets for all military products. Information on Leadless Chip Carriers and on Quality Assurance is also included.	**10.00**
Development Systems Handbook (Order No. 210940) Contains data sheets on development systems and software, support options, and design kits.	**10.00**
OEM Systems Handbook (Order No. 210941) Contains all data sheets, application notes, and article reprints for OEM boards and systems.	**15.00**
Software Handbook (Order No. 230786) Contains all data sheets, applications notes, and article reprints available directly from Intel, as well as 3rd Party software.	**10.00**

*** Prices are for the U.S. only.**

intel®

MICROSYSTEM COMPONENTS HANDBOOK

VOLUME 2

1984

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, $\hat{\imath}$, ICE, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMMX, Insite, Intel, int$_e$l, int$_e$lBOS, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

*CP/M-86 is a Trademark of Digital Research, Inc.

CHAPTER 8

CHAPTER 9

Peripherals (continued)

6

Peripherals Section

intel

APPLICATION NOTE

AP-153

June 1983

Designing with the 8256

Charles T. Yager
Applications Engineer

ORDER NUMBER: 210907-001

INTRODUCTION

The INTEL 8256 MUART is a Multifunction Universal Asynchronous Receiver Transmitter designed to be used for serial asynchronous communication while also providing hardware support for parallel I/O, timing, counting and interrupt control. Its versatile design allows it to be directly connected to the MCS®-85, iAPX-86, iAPX-88, iAPX-186, and iAPX-188 microcomputer systems plus the MCS-48 and MCS-51 family of single-chip microcomputers.

The four commonly used peripheral functions contained in the MUART are:

1) Full-duplex, double-buffered serial asynchronous Receiver/Transmitter with an on-chip Baud Rate Generator
2) Two - 8-bit parallel I/O ports
3) Five - 8-bit counters/timers
4) 8-level priority interrupt controller

This manual can be divided into two parts. The first part describes the MUART in detail, including its functions, registers and pins. This section also describes the interface between the MUART and Intel CPUs plus a discussion on programming considerations. The second section provides an application example: a MUART-based line printer multiplexer. The Appendix contains software listings for the line printer multiplexer and some useful reference information.

DESCRIPTION OF THE MUART

The MUART can be logically partitioned into seven sections: the microprocessor bus interface, the command and status registers, clocking circuitry, asynchronous serial communication, parallel I/O, timer/event counters, and the interrupt controller. This can be seen from the block diagram of the 8256 MUART as shown in Figure 1. The MUART's pin configuration can be seen in Figure 2.

Microprocessor Bus Interface

The microprocessor bus interface is the hardware section of the MUART which allows a μP to communicate with the MUART. It consists of tristate bi-directional data-bus buffers, an address latch, a chip select ($\overline{CS}$) latch and bus control logic. In order to provide all of the MUART's functions in a 40-pin DIP while retaining direct register addressing, a multiplexed address/data bus is used.

Address/Data Bus

The MUART contains 16 internal directly addressable read/write registers. Four of the eight address/data lines are used to generate the address. When using 8-bit microprocessors such as MCS-85, MCS-48 and MCS-51, AD0 - AD3 are used to address the 16 internal registers while Address/Data line 4 (AD4) is not used for addressing. For 16-bit systems, AD1 - AD4 are used to generate the address for the internal data registers and AD0 is used as a second active low chip select.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The 8256 bus interface uses the standard bus control signals which are compatible with all Intel peripherals and microprocessors. The chip select signal ($\overline{CS}$), typically derived from an address decoder, is latched along with the address on the falling edge of ALE. As a result, chip select does not have to remain low for the entire bus cycle. However, the data bus buffers will remain tristated unless an $\overline{RD}$ or a $\overline{WR}$ signal becomes active while chip select has been latched in low.

INT, $\overline{INTA}$

The INT and $\overline{INTA}$ signals are used to interrupt the CPU and receive the CPU's acknowledgment to the interrupt request. The MUART can vector the CPU to the appropriate service routine depending on the source of the interrupt.

RESET

When a high level occurs on the RESET pin, the MUART is placed in a known initial state. This initial state is described under "Hardware Reset."

Command and Status Register

There are three command registers and one status register as shown in Figure 1. The three command registers are read/write registers while the status register is a read only. The command registers configure the MUART for its operating environment (i.e., 8 or 16 bits CPU, system clock frequency). In addition, they direct its higher level functions such as controlling the UART, selecting modes of operation for the interrupt controller, and choosing the fundamental frequency for the timers. Command Register 3 is the only register in the MUART which is a bit set/reset register, allowing the programmer to simply perform one write to set or reset any of the bits.

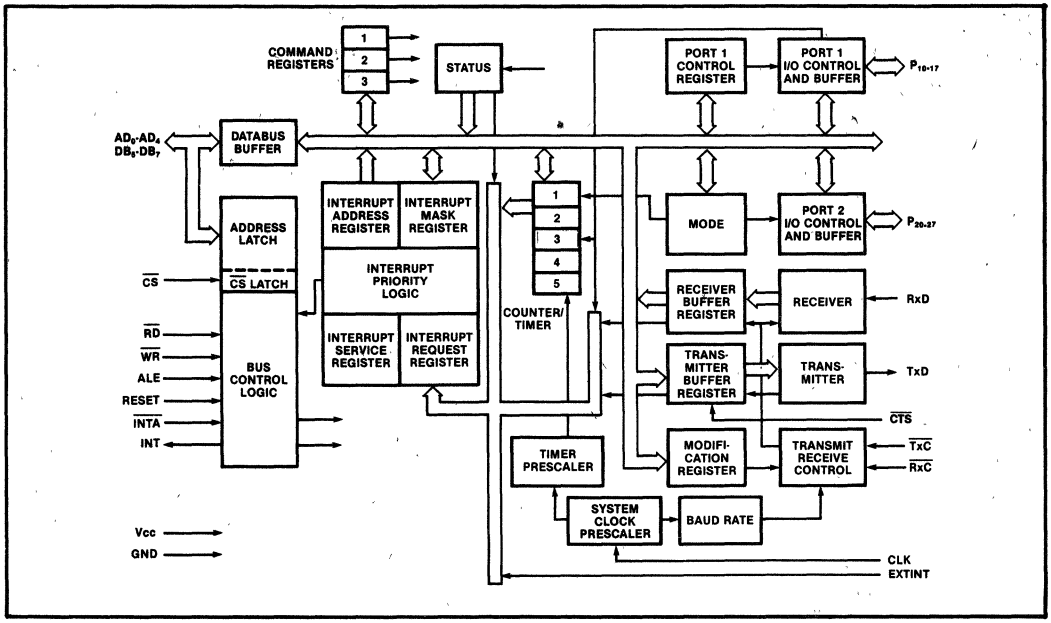


Figure 1. Block Diagram of the 8256 MUART

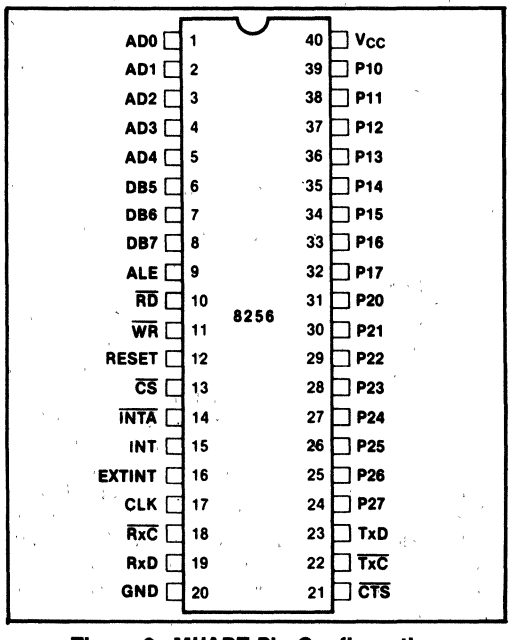


Figure 2. MUART Pin Configuration

The status register provides all of the information about the status of the UART's transmitter and receiver as well as the status of the interrupt pin. The status register is the only read only register in the MUART.

CLOCK CIRCUITRY

The clock for the five timers and baud rate generator is derived from the system clock. The system clock, pin 17 (CLK), is fed into a system clock prescaler which in turn feeds the five timers and the baud rate generator. The MUART's system clock can be asynchronous to the microprocessor's clock.

System Clock Prescaler

The system clock prescaler is a programmable divider which normalizes the internal clocking frequency for the timers and baud rate generator to 1.024MHz. It divides the system clock (CLK) by 1, 2, 3, or 5, allowing clock frequencies of 1.024MHz, 2.048MHz, 3.072MHz or 5.12MHz. (The commonly used 6.144MHz crystal frequency for the 8085 results in a 3.072MHz frequency from the 8085's CLK pin.) If the system clock is not one of the four frequencies mentioned above, then the frequency of the baud rate generator and the timers will be nonstandard;

however, the MUART will still run as long as the system clock meets the data sheet tcy spec.

Timer Prescaler

The timer prescaler permits the user to select one of two fundamental timing frequencies for all of the MUART's timers, either 1KHz or 16KHz. The frequency selection is made via Command Register 0.

Asynchronous Serial Interface

The asynchronous serial interface of the MUART is a full-duplex double-buffered transmitter and receiver with separate control registers. The standard asynchronous format is used as shown in Figure 3. The operation of the UART section of the MUART is very similar to the operation of the 8251A USART.

Receiver Section of the UART

The serial asynchronous receiver section contains a serial shift register, a receiver buffer register and receiver control logic. The serial input data is clocked into the receive shift register from the RxD pin at the specified baud rate. The sampling actually takes place at the rising edge of RxC, assuming an external clock,

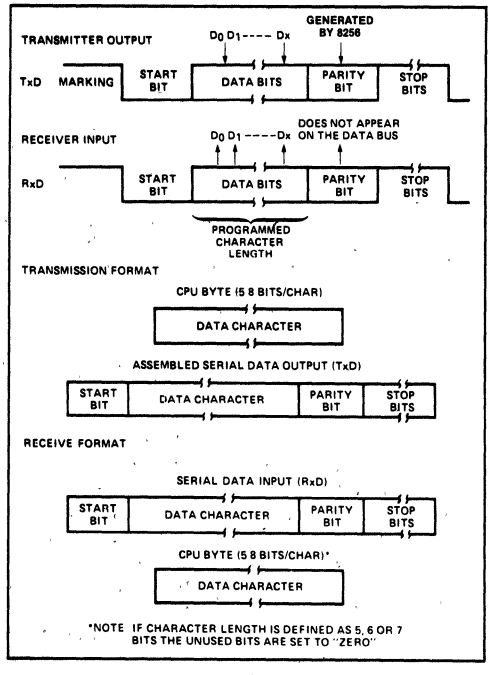


Figure 3. Asynchronous Format

or at the rising edge of the internal baud clock. When the receiver is enabled but inactive, the receive logic is sampling RxD at either 32 or 64 times the bit rate, looking for a change from the Mark (high) to the Space (low) state. This is commonly referred to as the start bit search mode. When this state change occurs, the receive logic waits one half of a bit time and then samples RxD again. If RxD is still in the Space state, the receive logic begins to clock in the receive data beginning one bit period later. If RxD has returned to the Mark state (i.e., false start bit), the receive logic will return to the start bit search mode.

Normally the received data is sampled in the center of each bit, however it is possible to adjust the location where the bit is sampled. This feature is controlled by the modification register.

The bit rate of the serial receive data is derived from either the internal baud rate generator or an external clock. When using an external clock, the programmer has a choice of three sampling rates: 1x, 32x, or 64x, Using the internal baud rate generator, the sampling rates are all 64x except for 19.2 Kbps which is 32x.

When the serial shift register clocks in the stop bit, an internal load pulse is generated which transfers the contents of the shift register into the receive buffer. This transfer takes place during the first half of the first stop bit. The load pulse also triggers several other signals relevant to the receive section including Receive Buffer Full (RBF), Parity Error (PE), Overrun Error (OE), and Framing Error (FE). These four status bits are updated after the middle of the first stop bit when the receive buffer has already been latched. Each one of these four status bits are latched. They are reset on the rising edge of the first read pulse (RD) addressed to the status register. A complete description of the status register is given in the section "Description of the Registers."

When the serial receiver is disabled (via bit 6 of Command Register 3) the load pulse is suppressed. The result is that the receive buffer is not loaded with the contents of the shift register, and the RBF, PE, OE, and FE bits in the status register are not updated. Even though the receiver is disabled, the serial shift register will still be clocking in the data from RxD, if any. This means that the receiver will still be synchronized with the start and stop bits. For example, if the receiver is enabled via Command Register 3 in the middle of receiving a serial character, the character will still be assembled correctly. When the receiver is disabled the last character received will remain in the receive buffer. On power-up the value in the receive buffer is undefined.

Whenever a character length of fewer than 8 bits is programmed, the most significant bits of a received character will read as zero. Also, the receiver will only check the first stop bit of any character, regardless of how many stop bits are programmed into the device.

Receive Break Detect

A Receive Break occurs when RxD remains in the space state for one character time, including the parity bit (if any) and the first stop bit. The MUART will set the Break Detect status bit (BD) when it receives a break. The Break Detect status bit is set after the middle of the first stop bit. If the MUART detects a break it will inhibit the receive buffer load pulse, thus the receive buffer will not be loaded with the null character, and none of the four status bits (PE, OE, FE, and RBF) will be updated. The last character received will remain in the receive buffer. A break detect state has the same effect as disabling the receiver—they both inhibit the load pulse—therefore one can think of the break status as disabling the receiver.

The Break Detect status bit is latched. It is cleared by the rising edge of the read pulse addressed to the status register. If a break occurs, and then the RxD data line returns to the Mark state before the status register is read, the BD status bit will remain set until it is read. If RxD returns to the Mark state after the BD status bit has been read true, the BD status bit will be reset automatically without reading the status register.

The receive break detect logic of the MUART is independent of whether the receiver is enabled or disabled; therefore even if the receiver is disabled the MUART will recognize a break. When the RxD line returns to the Mark state after a break, the 8256 will be in the start bit search mode.

If the receiver interrupt level is enabled, break will generate an interrupt request regardless of whether the receiver is enabled. Another receive interrupt will not be generated until the RxD pin returns to the Mark state.

Transmitter Section of the UART

The serial asynchronous transmitter section of the MUART consists of a transmit buffer, a transmit (shift) register, and the associated control logic. There are two bits in the status register which indicate the status of the transmit buffer and transmit register: TBE (transmit buffer empty) and TRE (transmit register empty).

To transmit a character, a byte is written to the transmit buffer. The transmit buffer should only be written to when TBE = 1. When the transmit register is empty and $\overline{CTS} = 0$, the character will be automatically transferred from the transmit buffer into the transmit register. The data transfer from the transmit buffer to the transmit register takes place during the transmission of the start bit. After this transfer takes place, sometime at the beginning of the transmission of the first data bit, TBE is set to 1.

When the transmitter is idle, both TBE and TRE will be set to 1. After a character is written to the transmit buffer, TBE = 0 and TRE = 1. This state will remain for a short period of time, then the character will be transferred into the transmit register and the status bits will read TBE = 1 and TRE = 0. At this point a second character may be written to the transmit buffer after which TBE = 0 and TRE = 0. TBE will not be set to 1 again until the transmit register becomes empty and is reloaded with the byte in the transmit buffer.

The transmitter can be disabled only one way—using the $\overline{CTS}$ pin. When $\overline{CTS} = 0$ the transmitter is enabled, and when $\overline{CTS} = 1$ the transmitter is disabled. If the transmitter is idle and $\overline{CTS}$ goes from 0 to 1, disabling the transmitter, TBE and TRE will remain set to 1. Since TBE = 1 a character can be written into the transmit buffer. The character will be stored in the transmit buffer but it will not be transferred to the transmit register until $\overline{CTS}$ goes low.

If $\overline{CTS}$ goes from low to high during transmission of a character, the character in transmission will be completed and TxD will return to the Mark state. If the transmitter is full (i.e., TBE and TRE = 0), the transmit shift register will be emptied but the transmit buffer will not; therefore TBE = 0 and TRE = 1.

Transmitter Break Features

The MUART has three transmit break features: Break-In Detect, Transmit Break (TBRK), and Single Character Break (SBRK).

Break-In Detect - A Break-In condition occurs when the MUART is sending a serial message and the transmission line is forced to the space state by the receiving station. Break-In is usually used with half-duplex transmission so that the receiver can signal a break to the transmitter. Port 16 must be connected externally to the transmission line in order to detect a Break-In. If transmission voltage levels other than TTL are used, then proper buffering must be provided so that Port 16 on the MUART will receive the correct polarity and voltage levels.

When Break-In Detect is enabled, Port 16 is polled internally during the transmission of the last or only stop bit of a character. If this pin is low during transmission of the stop bit, the Break Detect status bit (BD) will be set. Break-In Detect and receive Break Detect are OR-ed to set the BD status bit. (Either one can set this bit.) The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on level 5, the transmit interrupt, while Break will generate an interrupt on level 4, the receive interrupt. If RxC and TxC are used for the serial bit rates, Break-In cannot be detected.

Transmit Break - This causes the TxD pin to be forced low for as long as the TBRK bit in Command Register 3 is set. While Transmit Break is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full, and a Transmit Break command is issued (command register 3, TBRK = 1), the entire character in the Transmit register is sent including the stop bits. TxD is then driven low and the character in the Transmit Buffer remains there until Transmit Break is disabled (command register 3, TBRK = 0). At this time TxD will go high for one bit time and then send the character in the Transmit Buffer.

Single Character Break - This causes TxD to be set low for one character including start bit, data bits, parity bit, and stop bits. The user can send a specific number of Break characters using this feature.

If both the Transmit Buffer and the Transmit Register are full and a Send Break command is issued (command register 3, SBRK = 1) the entire character in the Transmit Register is sent including the stop bits. TxD is driven low for one complete character time followed by a high for two bit times after which the character in the Transmit Buffer is sent.

Modification Register

The modification register is used to alter two standard functions of the receiver (start bit check, and sampling time) and to enable a special indicator flag for half-duplex operation (transmitter status). Disabling start bit check means that the receiver will not return to the start bit search mode if RxD has returned to the Mark state in the center of the start bit. It will simply proceed to assemble a character from the RxD pin regardless of whether it received a false start bit or not. The modification register also allows the user to define where within the receive data bits the MUART will sample.

Parallel I/O

The MUART contains 16 parallel I/O pins which are divided into two 8-bit ports. These two parallel I/O ports (Port 1 and Port 2) can be used for basic digital I/O such as setting a bit high or low, or for byte transfers using a two-wire handshake. Port 1 is bit programmable for input or output, so any combination of the eight bits in Port 1 can be selected as either an input or an output. Port 2 is nibble programmable, which means that all four bits in the upper or lower nibble have to be selected as either inputs or outputs. For byte transfers using the two- wire handshake, Port 2 can either input or output the byte while two bits in Port 1 are used for the handshaking signals.

All of the bits in Port 1 have alternate functions other than I/O ports. As mentioned above, when using the byte handshake mode, two bits on Port 1 are used for the handshaking signals. As a result, these two bits cannot be used for general purpose I/O. The other six bits in Port 1 also have alternate functions if they are not used as I/O ports. Table 1 lists each bit from Port 1 and its corresponding alternate function.

The bits in the Port 1 Control Register select whether the pins on Port 1 are inputs or outputs. The pins on Port 1 are selected as control pins through the other programming registers which are relevant to the control signal. Configuring a bit in Port 1 as a control function overrides its definition in the Port 1 Control Register. If the pins on Port 1 are redefined as control signals, the definition of whether the pin is an input or an output in the Port 1 Control Register remains unchanged. If the pins on Port 1 are converted back to I/O pins, they assume the state which was defined in the Port 1 Control Register.

Each parallel I/O port has a latch and drivers. When the port is in the output mode, the data written to the port is latched and driven on the pins. The data which is latched in the I/O ports remains unchanged unless the port is written to again. Reading the ports, whether the port is an input or output, gates the state at the pins onto the data bus. Writing to an input port has no effect on the pin, but the data is stored in the latch and will be output if the direction on the pin is changed later. Writing to a control pin on Port 1 has the same effect as writing to an input pin. If pins 2, 3, 5, and 6 in Port 1 are used for control signals, the contents of the respective output latches will be read, not the state of the control signals. If pins 0, 1, and 7 on

Table 1. Port 1 Control Signals

Pin Symbol	Pin Number	Control Function	Condition
P10 P11	39 38	$\overline{ACK}$ Control signals for Port 2 $\overline{OBF}$ 8-bit handshake output	Mode register P2C2 - P2C0 = 101
P10 P11	39 38	$\overline{STB}$ Control signals for Port 2 $\overline{IBF}$ 8-bit handshake input	Mode register P2C2 - P2C0 = 100
P12	37	Event counter 2 clock input	Mode register CT2 = 1
P13	36	Event counter 3 clock input	Mode register CT3 = 1
P14	35	Internal baud rate generator clock output	Mode word P2C0 - P2C2 = 111 Port 1 control word P14 = 1 Command Register 2 B3 - B0 $\geq$ 3H
P15	34	Timer 5 trigger input	Mode register T5C = 1
P16	33	Break-In detection input	Command Register 1 BRKI = 1
P17	32	External edge sensitive interrupt input	Command Register 1 BITI = 1

Port 1 are used for control signals, the state of the control signals will be read. If pin 4 on Port 1 is used as a test output for the internal baud rate, this clock signal will be output through the output latch, thus the information in the output latch will be lost.

The Two-Wire Byte Handshake

The 8256 can be programmed, via the Mode Register, to implement an input or output two-wire byte handshake. When the Mode Register is programmed for the byte handshake, Port 2 is used to transmit or receive the byte, and pins P10 and P11 are used for the two handshake control signals. Figures 4 and 5 on pages 7 through 10 show a block diagram and timing signals for the two-wire handshake input and output.

To set up the two-wire handshake output using interrupts one must first program the Mode Register, and then enable the interrupt via the interrupt mask register. An interrupt will not occur immediately after the two-wire handshake interrupt is enabled. The interrupt is triggered by the rising edge of $\overline{ACK}$. There are two ways to generate the first interrupt. Either the first data byte must be written to Port 2 and completely transferred before an interrupt will occur, or the two-wire handshake interrupt is enabled while $\overline{ACK}$ is low, and then $\overline{ACK}$ goes high.

Event Counters/Timers

The MUART's five 8-bit programmable counters/timers are binary presettable down counters. The distinction between timer and counter is determined by the clock source. A timer measures an absolute time interval, and its input clock frequency is derived from the MUART's system clock. A counter's input clock frequency is derived from a pulse applied to an external pin. The counter is decremented on the rising edge of this pulse.

When the counters/timers are configured as timers their clock source passes through two dividers: the system clock prescaler, and the timer prescaler. As mentioned before, the system clock prescaler normalizes the internal system clock to 1.024 MHz. The timer prescaler receives this normalized system clock and devides it down to either 1 kHz or 16 kHz, depending

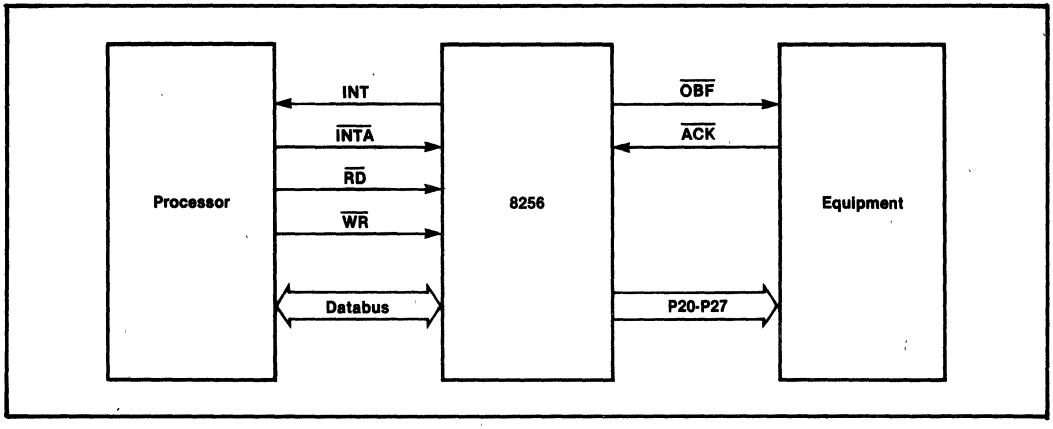


Figure 4. Block Diagram of Handshake Output

on how Command Register 1 is programmed. If more timing resolution is needed the clock frequency can be input externally through the I/O ports.

By programming the Mode Register, four of the 8-bit counters/timers can be cascaded to form two 16-bit counters. Counters/timers 3 and 5 can be cascaded together, and counters/timers 2 and 4 can be cascaded together. Counters/timers 2 and 3 are the lower bytes, while counters/timers 4 and 5 are the upper bytes in the cascaded mode.

Each counter can be loaded with an arbitrary initial value. Timer 5 is the only timer which has a special save register which holds its initial value. Whenever Timer 5 is loaded with an initial value the special save register is also loaded with this value. Timer 5 can be reloaded to its initial value from the detection of a high-to-low transition on Port P15.

The counters are decremented on the first rising edge of the clock after the initial value has been loaded. The setup time for loading the counter when using an external clock is specified in the data sheet. When using internal clocks, the user has no way of knowing the phase relationship of the clock to the write pulse; therefore the timing accuracy is one clock period.

The timers are counting continuously, and an interrupt request is issued any time a single counter or pair of cascaded counters reaches zero. If the timers are going to be used with interrupts, then the programmer should first load the timer with the initial value, then enable the interrupt. If the programmer enables the interrupt first, it is possible that the interrupt will occur before the initial value is loaded. When an interrupt from any one of the timers occurs, the corresponding bit in the interrupt mask register is automatically reset, preventing further interrupt requests from occuring.

The event counters/timers can be used in the following modes of operation:

Timer 1
— Serves as an 8-bit timer.

Event Counter/Timer 2
— Serves as an 8-bit timer or event counter, or cascaded with Timer 4 as a 16-bit timer or event counter.

Event Counter/Timer 3
— Serves as an 8-bit timer or event counter, or cascaded with Timer 5 as a 16-bit timer or event counter, with the additional modes of operation selectable for Timer 5.

Timer 4
— Serves as an 8-bit timer, or cascaded with Event Counter/Timer 2 as a 16-bit timer or event counter.

Timer 5
1) Non-retriggerable 8-bit timer
2) Retriggerable 8-bit timer whose initial value is loaded from a save register which starts following the negative transition of an external signal. Subsequent transitions of this signal after the counting has started, reloads the initial value and restarts the counting.
3) Cascaded with Event Counter/Timer 3, non-retriggerable 16-bit timer, which can be loaded with an initial value by two write operations.

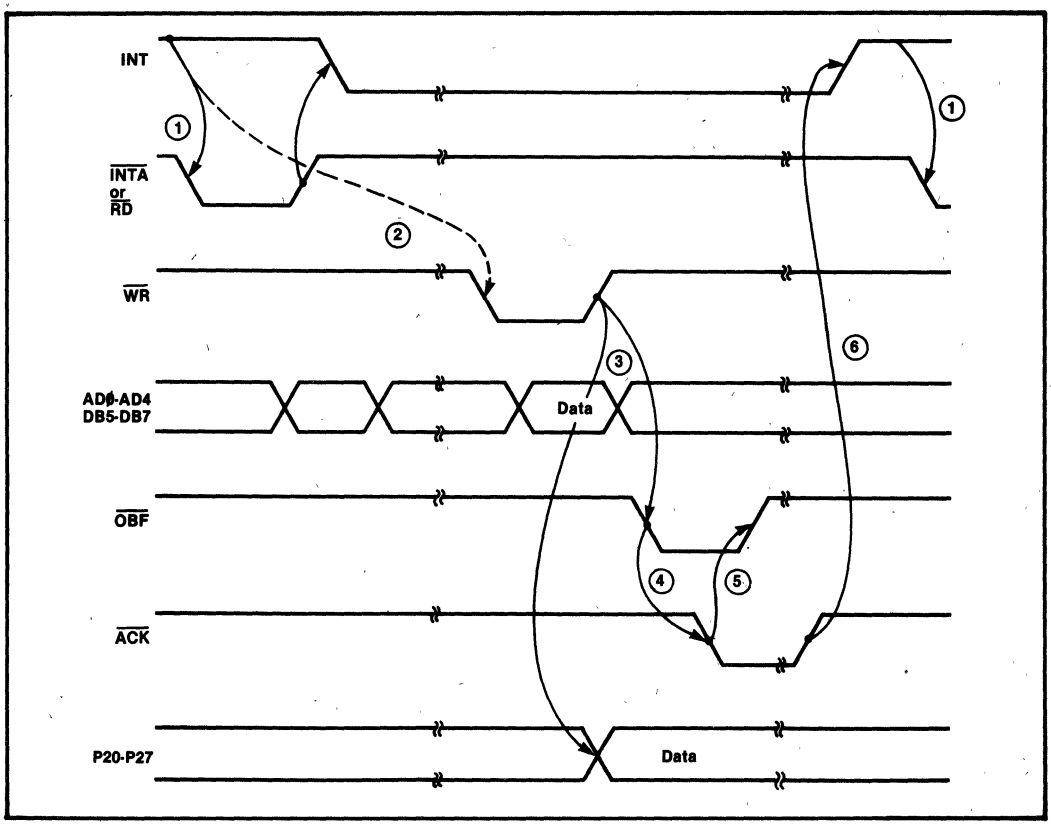


Figure 4a. Timing of Handshake Output

(1) The 8256 signals with INT that the equipment has accepted the last character and that the output latches are empty again.

(2) Thereupon, the microprocessor transfers the next data to the 8256.

(3) The rising edge of WR latches the data into port 2 (P20...P27) and "Output Buffer Full" (OBF) is set which indicates that a new byte is available.

(4) The equipment acknowledges with the falling edge of $\overline{ACK}$ that it recognized $\overline{OBF}$.

(5) Thereupon, the 8256 releases $\overline{OBF}$.

(6) The equipment acknowledges the data transfer with a rising edge of $\overline{ACK}$ which causes the 8256 to set INT.

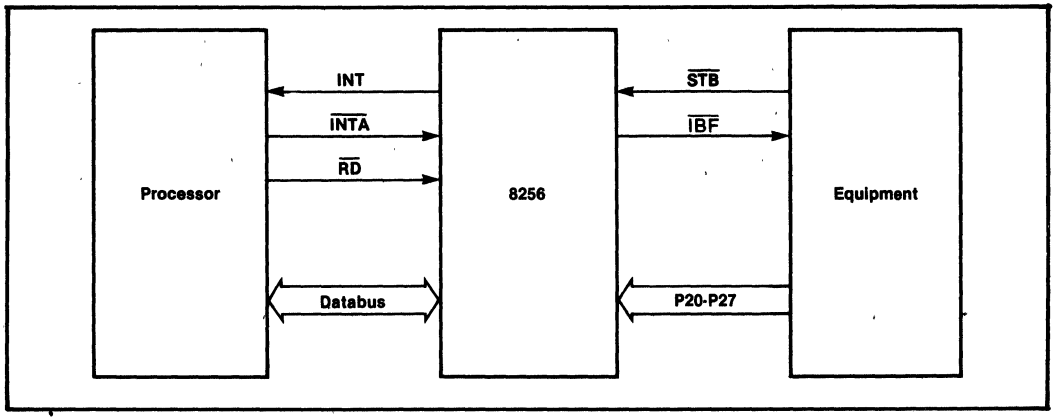


Figure 5. Block Diagram of Handshake Input

4) Cascaded with event counter/timer 3, non-retriggerable 16-bit event counter, which can be loaded with an initial value by two write operations.
5) Cascaded with Event Counter/Timer 3, retriggerable 16-bit timer. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically, Loading, starting, and retriggering operations follow the same pattern as in 2).
6) Cascaded with Event Counter/Timer 3, retriggerable 16-bit event counter. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2).

Interrupt Controller

In a microcomputer system there are several ways for the CPU to recognize that a peripheral device needs service. Two of the most common ways are the polling method and the interrupt service method.

In the polling method the CPU reads the status of each peripheral to determine whether it needs service. If the peripheral does not need service, the time the CPU spends polling is wasted; therefore this overhead results in increasing the execution time. Some systems must meet a specific request to response time such as a real time signal. In this case the programmer must guarantee that the peripheral is polled at a certain frequency. This polling frequency cannot always easily be met when the CPU must execute a main program as well as subroutines. Usually each peripheral has its own request to response time requirements; therefore the user must establish a priority scheme.

The interrupt method provides certain advantages over the polling method. When a peripheral device needs service it signals the CPU through hardware asynchronously, thus reducing the overhead of polling a device which does not need service. The CPU would typically finish the instruction it is executing, save the important registers, and acknowledge the peripheral's interrupt request. During the acknowledgment, the CPU reads a vector which directs the CPU to the starting location of the appropriate interrupt service routine. If several interrupt requests occur at the same time, special logic can prioritize the requests so that when the CPU acknowledges the interrupt, the highest priority request is vectored to the CPU.

An interrupt driven system requires additional hardware to control the interrupt request signal, priority, and vectoring. The 8256 integrates this additional hardware onto the chip. The interrupt controller on the MUART is directly compatible with the MCS-85, iAPX-86, iAPX-88, iAPX-186, iAPX-188 family of microcomputer systems, and it can also be used with other microprocessors as well. It contains eight priority levels, however, there are a total of 12 interruptable sources: 10 internal and 2 external. Since there are eight priority levels, only eight interrupts can be used at one time. The assignment of the interrupts used is selected by Command Register 1 and by the mode register. The MUART's interrupt sources have a fixed priority. Table 2 displays how the 12 interrupt sources are mapped into the 8 priority levels.

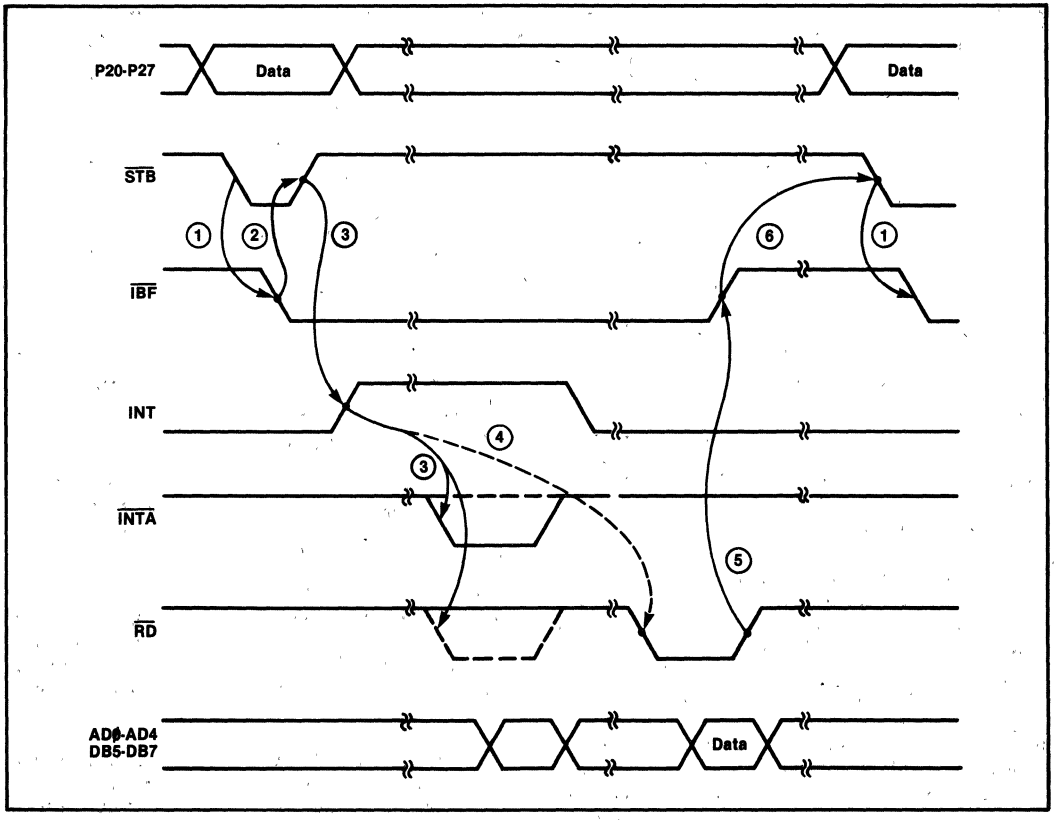


Figure 5a. Timing for Handshake Input

(1.) The equipment indicates with the falling edge of $\overline{STB}$ (Strobe) that a new character is available at port 2. The 8256 acknowledges the indication by activating $\overline{IBF}$ (Input Buffer Full).

(2.) Thereupon, the equipment releases $\overline{STB}$ and the 8256 latches the character.

(3.) The 8256 informs the microprocessor through INT that a new character is ready for transfer.

(4.) The microprocessor reads the character.

(5.) The rising edge of signal $\overline{RD}$ resets signal $\overline{IBF}$.

(6.) This action signals to the equipment that the input latches of the 8256 are empty and the next character can be transferred.

Table 2. Mapping of Interrupt Sources to Priority Levels

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

MCS®-85/8256 Interrupt Operation

The 8256 is compatible with the 8085 interrupt vectoring method when the 8086 bit in Command Register 1 of the MUART is set to 0. This is the default condition after a hardware reset. The 8085 has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. When the MUART's interrupt acknowledge feature is enabled (IAE bit 5 Command Register 3 = 1) the MUART's INT Pin 15 should be tied to the 8085's INTR, and both the 8085 and the MUART's INTA pins should be tied together. All of the interrupt pins on the 8085 except INTR automatically vector the program counter to a specified location in memory. When the INTR pin becomes active (HIGH), assuming the 8085 has interrupts enabled, the 8085 fetches the next instruction from the data bus where it has been placed by the 8256 or some other interrupt controller. This instruction is usually a Call or an RST0 through RST7. Figure 6 shows the memory locations where the 8085 will vector to based on which type of interrupt occurred.

The 8085 can receive an interrupt request any time, since its INTR input is asynchronous. The 8085, however, doesn't always acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions.

At the end of each instruction cycle, the 8085 examines the state of its INTR pin. If an interrupt request is present and interrupts are enabled, the 8085 enters an interrupt machine cycle. During the interrupt machine cycle the 8085 automatically disables further interrupts until the EI instruction is executed. Unlike normal machine cycles, the interrupt machine

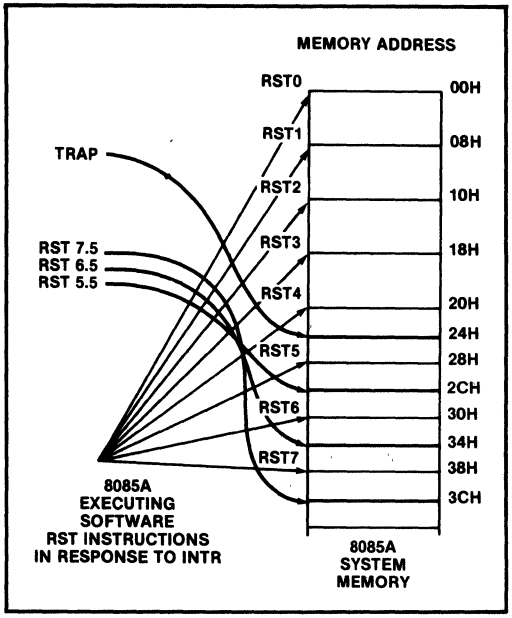


Figure 6. 8085A Hardware and Software RST Branch Locations

cycle doesn't increment the program counter. This ensures that the 8085 can return to the pre-interrupt program location after the interrupt service is completed. The 8085 issues an $\overline{\text{INTA}}$ pulse indicating that it is honoring the request and is ready to process the interrupt.

The 8256 can now vector program execution to the corresponding service routine. This is done during the first and only $\overline{\text{INTA}}$ pulse. Upon receiving the $\overline{\text{INTA}}$ pulse, the 8256 places the opcode RSTn on the data bus; where n equals 0 through 7 based on the level of the interrupt requested. The RSTn instruction causes the contents of the program counter to be pushed onto the stack, then transfers control to the instruction whose address is eight times n, as shown in Figure 6.

Note that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed in either the service routine or the main program before further interrupts can be processed.

For additional information on the 8085 interrupt operation and the RSTn instruction, refer to the *MCS-85 User's Manual.*

iAPX-86/88 - 8256 Interrupt Operation

The MUART is compatible with the 8086/8088 method of interrupt vectoring when the 8086 bit in Command Register 1 is set to 1. The MUART's INT pin is tied to the 8086/8088 INTR pin, and its $\overline{INTA}$ pin connected to the 8086/88's $\overline{INTA}$ pin. Like the 8085, the 8086/8088's INTR pin is also asynchronous so that an interrupt request can occur at any time. The 8086/8088 can accept or disregard requests on the INTR pin under software control instructions. These instructions set or clear the interrupt-enabled flag IF. When the 8086/8088 is powered-on or reset, the IF flag is cleared, disabling external interrupts on INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different from the 8085. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in PUSHF instruction). It then clears the IF flag, which disables interrupts. Finally, the contents of both the code segment register and the instruction pointer are pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and program location which are used to return from the service routine. The 8086/8088 then issues the first of two INTA pulses which signals the 8256 that the 8086/8088 has honored its interrupt request.

The 8256 is now ready to vector program execution to the appropriate service routine. Unlike the 8085 where the first INTA pulse is used to place an instruction on the data bus, the first INTA pulse from the 8086/8088 is used only to signal the 8256 of the honored request. The second INTA pulse causes the 8256 to place a single interrupt vector byte onto the data bus. The 8256 places the interrupt vector bytes 40H through 47H corresponding to the level of the interrupt to be serviced. Not used as a direct address, this interrupt vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt vector table. Each type in the interrupt vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 7 shows a block diagram of the interrupt vector table. When the 8086/8088 receives an interrupt vector byte, it multiplies its value by four to acquire the address of the interrupt type.

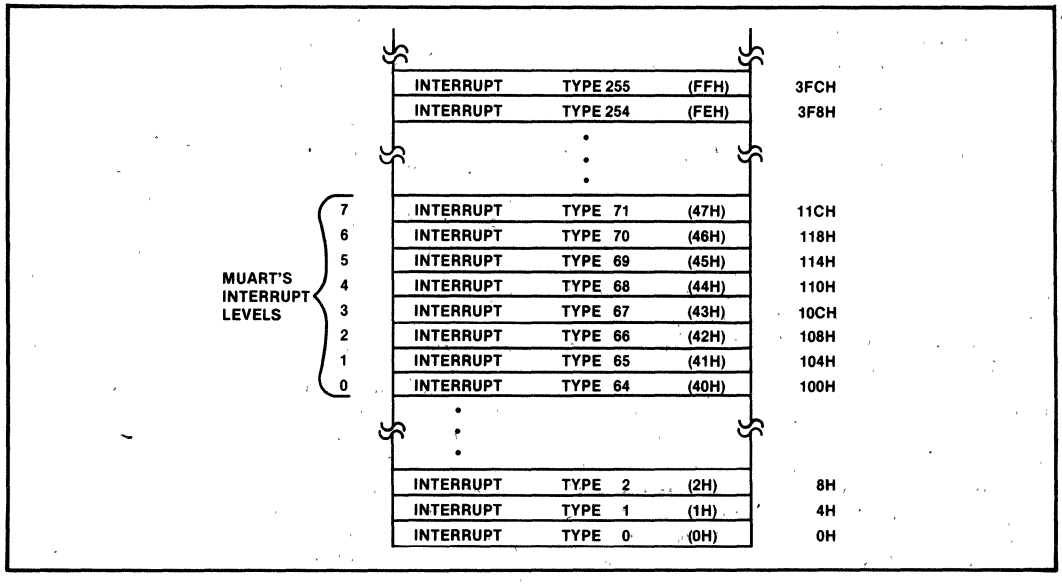


Figure 7. 8086/8088 Interrupt Vector Table

Once the service routine is completed the main program may be reentered by using an IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag; thus interrupts are re-enabled automatically when returning from the service routine. For further information refer to the *iAPX 86,88 User's Manual.*

Using the 8256's Interrupt Controller Without $\overline{\text{INTA}}$

There are several configurations where the 8256 will not have an $\overline{\text{INTA}}$ signal connected to it. Some examples are when using the 8256 with an 8051 or 8048, or when connecting the INT pin on the 8256 to the 8085's RST 7.5, RST 6.5, or RST 5.5 inputs. In these configurations the $\overline{\text{IAE}}$ bit in Command Register 3 is set to 0, and the $\overline{\text{INTA}}$ pin on the 8256 is tied high. When the interrupt occurs the CPU should branch to a service routine which reads the interrupt address register to determine which interrupt request level occured. The interrupt address register contains the level of the interrupt multiplied by four. Reading the interrupt address register is equivalent in effect to the $\overline{\text{INTA}}$ signal; it clears the INT pin and indicates to the MUART that the interrupt request has been acknowledged. After the CPU reads the value in the interrupt address register, it can add an offset to this value and branch to an interrupt vector table which contains jump instructions to the appropriate interrupt service routines. An 8085 program which demonstrates this routine is given is Figure 8.

Table 3 summarizes the priority levels and the interrupt vectors which the 8256 sends back to the CPU. Note that when using Timer 1 there is a conflict present between RST0 in the 8085 mode and a hardware reset, because both expect instructions starting at address 0H. However, there is a way to distinguish between the two. After a hardware reset, all control registers are reset to a value of 0H; therefore when using Timer 1, Reset and RST0 can be distinguished by reading one of the control registers of the 8256 which has not been programmed with a value of 0H. The control registers will contain the previously programmed values if RST0 occurs.

Interrupt Registers

The 8256's interrupt controller has several registers associated with it: an Interrupt Mask Register, an Interrupt Address Register, an Interrupt Request Register, an Interrupt Service Register, and a Priority Controller. Only the Interrupt Mask Registers and the Interrupt Address Register can be accessed by the user.

Interrupt Mask Registers

The Interrupt Mask Registers consist of two write registers — the Set Interrupts Register and Reset Interrupts Register, and one read register — the Interrupt Enable Register. Each one of the eight levels of interrupts may be individually enabled or disabled through these registers. Writing a one to any of the bits in the Set Interrupts Register enables the corresponding interrupt level, while writing a one to a bit in the Reset Interrupts Register disables the corresponding interrupt level. Reading the Interrupt Enable Register allows the user to determine which interrupt levels are enabled. The bits which are set to one in the Interrupt Enable Register correspond to the levels which are enabled. All of the interrupt levels will remain enabled until disabled by the Reset Interrupts Register except the counter/timer interrupts which automatically disable themselves when they reach zero.

```
INTA:   IN      INTADD      ;Read the Interrupt Address Register
        MOV     L, A        ;Put the interrupt address in HL
        XRA     A
        MOV     H, A

        LXI     B, TABLE    ;Load BE with the interrupt table offset
        DAD     B           ;Add the offset to the interrupt address
        PCHL                ;Jump to the interrupt vecor table
```

Figure 8. Software Interrupt Acknowledge Routine

Table 3. Assignment of Interrupt Levels to Interrupt Sources

Interrupt Level	Restart Command 8085 mode	Interrupt Vector 8086 mode	Interrupt Address	Trigger Mode	Sources (Only one source can be assigned at any time)	Selection by
Highest Priority 0	RST0	40H	0H	edge	Timer 1	-
1	RST1	41H	4H	edge	Event Counter/Timer 2 or external interrupt request on Port 1 P17	Command word 1 BITI (bit 2)
2	RST2	42H	8H	level	Input EXTINT	-
3	RST3	43H	CH	edge	Event Counter/Timer 3 or cascaded event counters/ timers 3 and 5	Mode word T35 (bit 7)
4	RST4	44H	10H	edge	Serial receiver	-
5	RST5	45H	14H	edge	Serial transmitter	-
6	RST6	46H	28H	edge	Timer 4 or cascaded event counters/timers 2 and 4	Mode word T24 (bit 6)
7 Lowest Priority	RST7	47H	1CH	edge	Timer 5 or Port 2 with handshaking interrupt request	Mode word P2C2 - P2C0 (bits 2...0)

Note:
If no interrupt requests are pending and $\overline{\text{INTA}}$ cycle occurs, interrupt level 2 will be the default value vectored to the CPU.

Interrupt requests occurring when the corresponding interrupt level is disabled are lost. An interrupt will only occur if the interrupt is enabled before the interrupt request occurs.

Interrupt Address Register

The Interrupt Address Register contains an identifier for the currently requested interrupt level. The numerical value in this register is equal to the interrupt level mutliplied by four. It can be used in lieu of an $\overline{\text{INTA}}$ signal to vector the CPU to the appropriate interrupt service routine. Reading this register has the same effect as the $\overline{\text{INTA}}$ pulse: it clears the INT pin and indicates an interrupt acknowledgement to the MUART. If the Interrupt Address Register is read while no interrupts are pending, the external interrupt EXTINT will be the default value, 08H.

Interrupt Request Register

The Interrupt Request Register latches all pending interrupt requests unless they are masked off. The request is set whenever the associated event occurs.

Interrupt Service Register

In the fully nested mode of operation, every interrupt request which is granted service is entered into this register. The appropriate bit will be set whenever the interrupt is acknowledged by $\overline{\text{INTA}}$ or by reading the Interrupt Address Register. At the same time, the corresponding bit in the Interrupt Request Register is reset. The Interrupt Service Register bit remains set until the microcomputer transfers the End Of Interrupt command (EOI) to the device by writing it into Command Register 3. In the normal mode the bits in the Interrupt Service Register are never set.

Priority Controller

The priority controller selects the highest priority request in the Interrupt Request Register from up to eight requests pending. If the INTA signal is enabled and becomes active, the priority controller will cause the highest priority level in the Interrupt Request Register to be vectored back to the CPU, regardless of whether the 8256 is in the normal mode or the nested mode. In the normal mode, if any bits are set in the Interrupt Request Register, the INT pin is activated. The highest priority level in the Interrupt Request register will be transferred to the Interrupt Address Register at the same time the interrupt request occurs. In the Fully Nested mode, the priorities of all pending requests are compared to the priorities in the Interrupt Service Register. If there is a higher priority in the Interrupt Request Register than in the Interrupt Service Register, the INT signal will be activated and the new interrupt level will be loaded into the Interrupt Address Register.

Interrupt Modes

There are two modes of operation for the interrupt controller: a normal mode and a fully nested mode. In the normal mode the CPU should only be a maximum of one interrupt level deep; therefore, the CPU can be interrupted only while in the main program and not while in an interrupt service routine. In the fully nested mode it is possible for the CPU to be nested up to eight interrupt levels deep. Using the fully nested mode, the MUART will activate the INT pin only when a higher priority than the one in service is requested. The fully nested mode is used to protect high priority interrupt service routines from being interrupted by equal or lower priority requests.

Normal Mode

In the normal mode of operation the 8256 will activate the INT pin whenever any of the bits in the Interrupt Request Register are set. The bits in the Interrupt Request Register can be set only if the corresponding interrupts are enabled. If more than one interrupt request bit is set, the MUART will always place the highest priority level in the Interrupt Address Register and vector this level to the CPU during an INTA cycle. When the CPU acknowledges the interrupt request, using either the INTA signal or by reading the Interrupt Address Register, the corresponding Interrupt Request Register bit is reset. Since the Interrupt Service Register bits are never set, there is no indication in the MUART that an interrupt service routine is in progress. Therefore, the priority controller will interrupt the CPU again if any of the interrupt request bits are set, regardless of whether the next request is a higher, lower, or equal priority.

The implied way to design a program using the normal mode is to have the CPU's interrupt flag enabled during portions of the main program, but to leave the interrupt flag disabled while the CPU is executing code in an interrupt service routine. This way, the CPU can never be interrupted in an interrupt service routine. Upon completion of an interrupt service routine the program can enable the CPU's interrupt flag, then return to the main program.

Figure 9 shows an example of how the normal mode of interrupts may operate. As the CPU begins executing code in the main program, certain I/O ports, variables, and arrays need to be initialized. During this time the CPU's interrupt flag is disabled. Once the program has completed the initialization routine and can accept an interrupt, the interrupt flag is enabled. In the 8085 this is done with the assembly language instruction EI, and on the 8086 with STI.

A short time later, an interrupt request comes in on Level 4. Since the CPU's interrupt flag is enabled, the interrupt acknowledge signal is activated and the CPU branches off to Interrupt Service Routine 4. While the CPU is executing code in Interrupt Service Routine 4, an interrupt request comes in on Level 6 and then a short time later on Level 2. The 8256 activates the INT signal; however, the CPU ignores this because its interrupt flag is disabled. Upon returning to the main program the interrupt flag is enabled. When the interrupt acknowledge signal is activated, the MUART places the highest priority interrupt request on the data bus regardless of the order in which the requests came in. Therefore, during the interrupt acknowledge the MUART vectors the indirect address for Interrupt Level 2. The INT signal is not cleared after the acknowledge because there is still a pending interrupt.

The normal mode of operation is advantageous in that it simplifies programming and lowers code requirements within interrupt routines; however, there are also several disadvantages. One disadvantage is that the interrupt response time for higher priority interrupts may be excessive. For example, if the CPU is executing code in an interrupt service routine during a higher priority request, the CPU will not branch off to the higher priority service routine until the current interrupt service routine is completed. This delay time may not be acceptable for interrupts such as the serial receiver or a real time signal. For these cases the MUART provides the nested mode.

Nested Mode

In the nested mode of operation, whenever a bit in the Interrupt Request Register is set, the Priority Con-

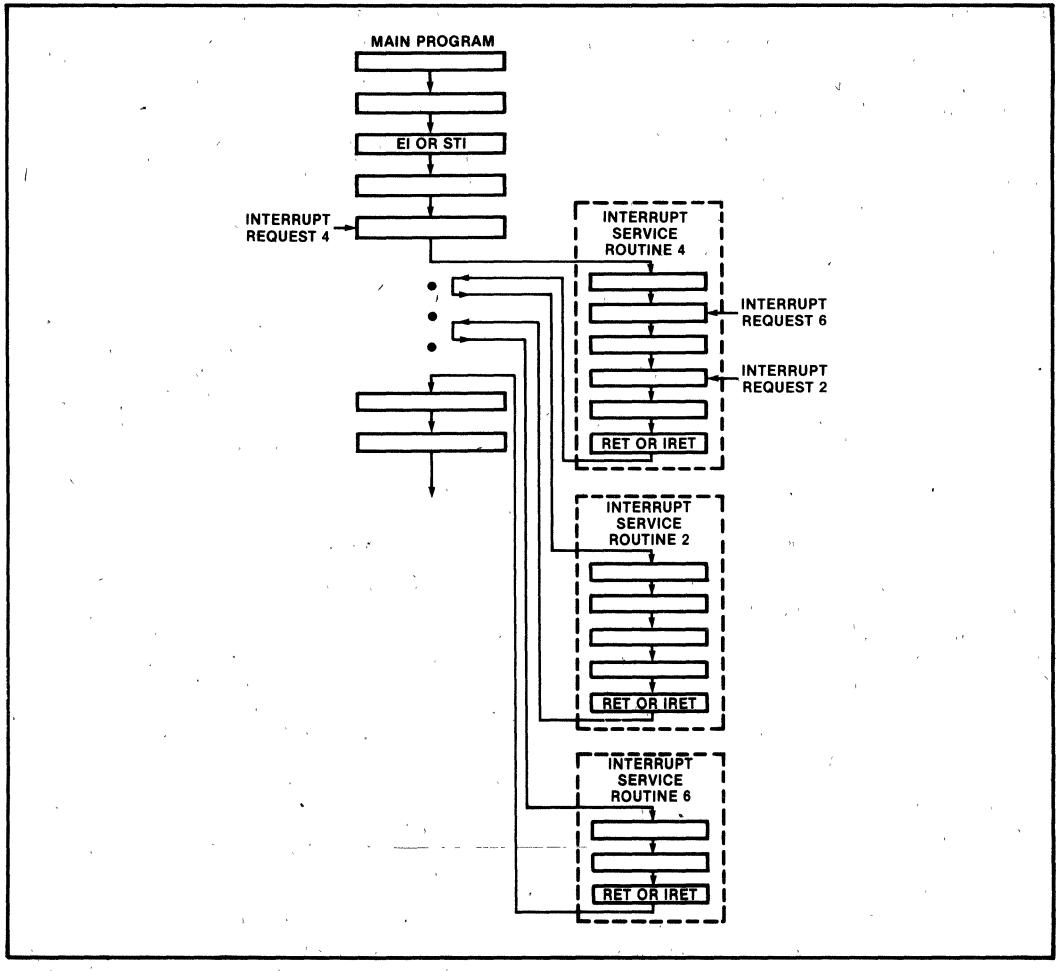


Figure 9. Normal Interrupt Mode Example

troller compares the Interrupt Request Register to the Interrupt Service Register. If the bit set in the Request Register is of a higher priority than the highest priority bit set in the Service Register, the MUART will activate the INT signal and update the Interrupt Address Register. If the bit in the Request Register is of equal or lower priority than the highest priority bit set in the Service Register, the INT signal will not be activated. When an $\overline{\text{INTA}}$ signal is activated or the Interrupt Address Register is read, the corresponding bit in the Request Register which caused the INT signal to be asserted is reset and set in the Service Register. When an EOI (End Of Interrupt) command is issued, the highest priority bit in the Service Register is reset.

Figure 10 shows an example of the program flow using the nested mode of interrupts. During the main program an interrupt request is generated from Level 4. Since the interrupt flag is enabled, the interrupt acknowledge signal is activated, and the microprocessor is vectored to Service Routine 4. During Service Routine 4, Level 2 requests an interrupt. Since Level 2 is a higher priority than Level 4, the 8256 activates its INT signal. An interrupt

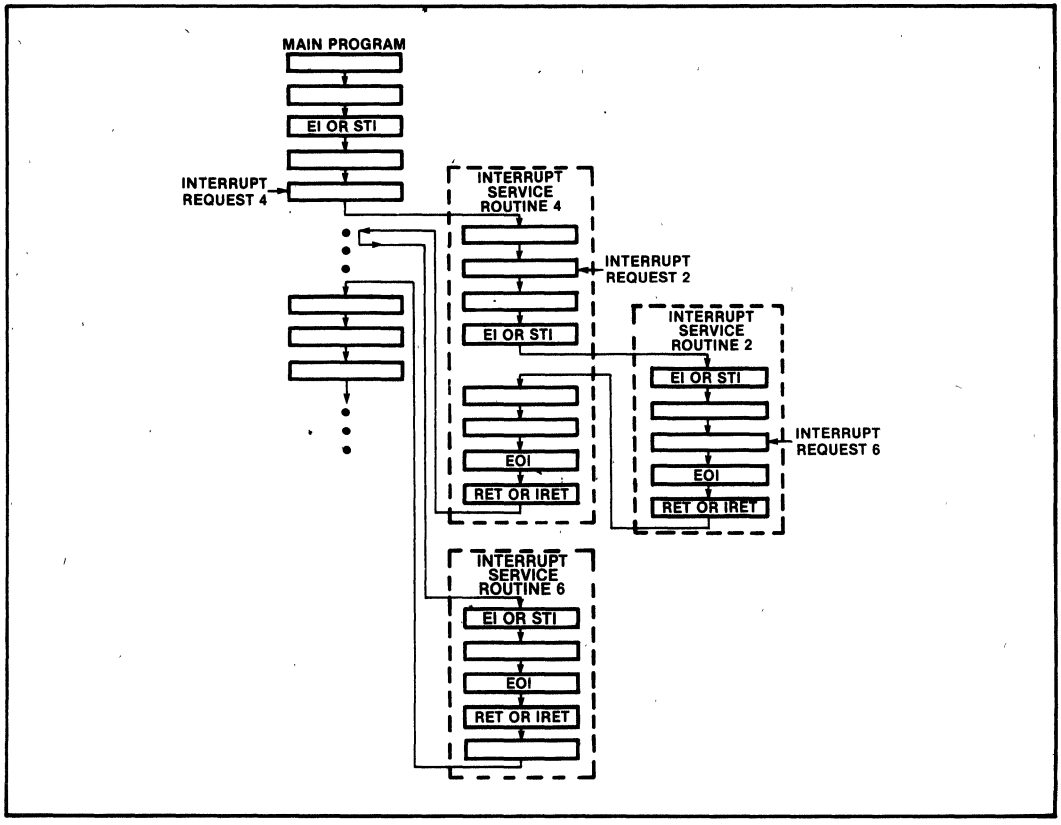


Figure 10. Fully Nested Interrupt Mode Example

acknowledge is not generated because the interrupt flag is disabled. This section of code in Service Routine 4 is protected and cannot be interrupted. A protected section of code may reinitialize a timer, take a sample, or update a global variable. When the interrupt flag is enabled the microprocessor acknowledges the interrupt and vectors into Service Routine 2. Service Routine 2 immediately enables the interrupt flag because it does not have a protected section of code. During Service Routine 2, Interrupt Request 6 is generated. However, the MUART will not interrupt the microprocessor until service routines 2 and 4 have issued the EOI command.

Edge Triggering

The MUART has a maximum of two external interrupts—EXTINT and P17. EXTINT is a dedicated interrupt pin which is level triggered, where P17 is either an I/O port or an edge triggered interrupt. If P17 is selected as an interrupt through Command Register 1 and its interrupt level is enabled, it will generate an interrupt when the level on this pin changes from low to high. The edge triggered mode incorporates an edge lockout feature. This means that after the rising edge of an interrupt request and the acknowledgment of the request, the positive level on

P17 won't generate further interrupts. Before another interrupt can be generated P17 must return low.

External devices which generate a pulse for an interrupt request can use the edge triggered mode as long as the minimum high time specified in the data sheet is met.

Level Triggering

The external interrupt (EXTINT pin 16) is the only level triggered interrupt on the MUART. The 8256 will recognize any active (high) level on the EXTINT as an interrupt request. The EXTINT pin must stay high until a short time after the rising edge of the first $\overline{\text{INTA}}$ pulse. If the voltage level on the EXTINT pin is high then goes low, the bit in the interrupt request register corresponding to EXTINT will be reset.

In the normal mode of operation if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT signal will remain active. If the microprocessor's interrupt flag is immediately reenabled, another interrupt will occur. Unless repeated interrupt generation is desired, the programmer should not reenable the CPU's interrupt flag until EXTINT has gone low.

In the nested mode of operation, if EXTINT is still high after the $\overline{\text{INTA}}$ pulse has been activated, the INT signal will not be reactivated. This is because in the nested mode only a higher priority interrupt than the one being serviced can activate the INT signal. The EXTINT pin should go inactive (low) before the EOI command is issued if an immediate interrupt is not desired.

Depending upon the particular design and application, the EXTINT pin has a number of uses. For example, it can provide repeated interrupt generation in the normal mode. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another use of the EXTINT pin is that a number of external interrupt requests can be wire-ORed. This can't be done using P17, for if a device makes an interrupt request while P17 is high (from another request), its transition will be shadowed. Note that when a wire-OR'ed scheme is used, the actual requesting device has to be determined by the software in the service routine.

Cascading the MUART's Interrupt Controller

Cascading the MUART's interrupt controller is necessary in an interrupt driven system which contains more than one interrupt controller, such as a system using more than one MUART, or using a MUART with another interrupt controller like the 8259A. For a system which uses several MUART's, one of them is tied directly to the microprocessor's INT and $\overline{\text{INTA}}$ pins, while the remaining MUARTs are daisy-chained using the EXTINT and INT pins. This is shown in Figure 11.

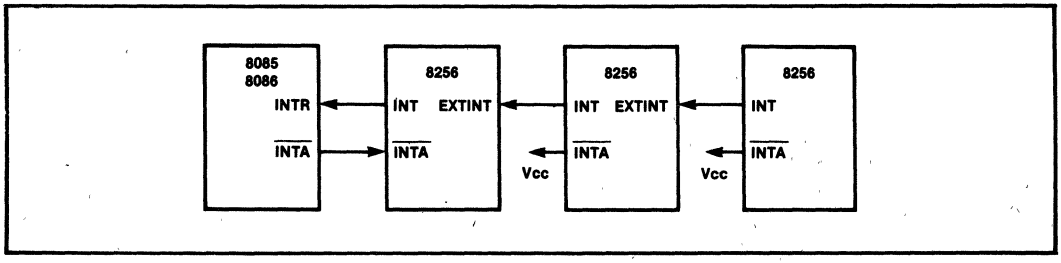


Figure 11. Cascading the MUART's Interrupt Controller

Using the configuration in Figure 11, when the microprocessor receives an interrupt, it generates an interrupt acknowledge and branches into an interrupt service routine. For the interrupt service routine of the external interrupt, EXTINT Level 2, the microprocessor will read the next MUART's interrupt address register and branch to the appropriate service routine. In effect, this would be a software interrupt acknowledge. An example of this type of interrupt acknowledge is given in Figure 8. If the last MUART in the chain indicated an external interrupt, the microprocessor would simply return to the main program; however, this would be an error condition caused by a spurious interrupt. A flow chart of the software to handle cascaded interrupts is given in Figure 12.

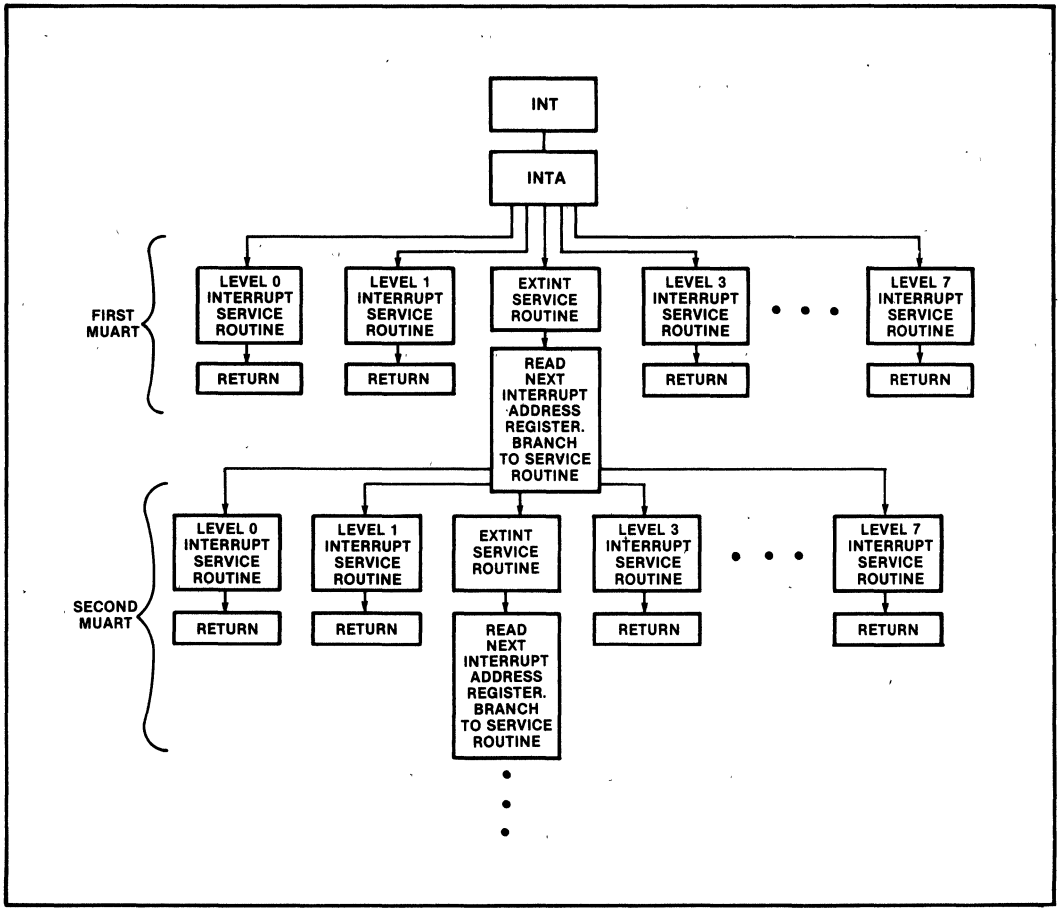


Figure 12. Flow Chart to Resolve Interrupt Request When Cascading MUART Interrupt Controllers

Some consideration should be given to the priority of the interrupts when cascading MUARTs. If all of the MUART's Level 0 and Level 1 interrupts are disabled, the highest priority interrupt is the EXTINT. In this case the last MUART in the chain would have the highest priority; however, it would take the longest time to propagate back to the CPU. If, however, Level 0 or Level 1 interrupts were enabled, the closer to the microprocessor the MUART is, the higher the priority these two levels would have.

When using the 8256 interrupt controller along with some other interrupt controller, such as the 8259A, the MUART's INT signal would simply be tied to one of the interrupt controller's request inputs. The service routine for the MUART's interrupt request would initially perform the software interrupt acknowledge before servicing the MUART's interrupt request. A block diagram of this configuration is given in Figure 13.

Polling the MUART

If interrupts are not used, the only other way to control the MUART is to poll it. It is still possible to use the priority structure of the MUART with polling. In this mode of operation the MUART's INT signal (Pin 15) is not used, and the $\overline{INTA}$ pin is tied high. Since the INT pin's level is duplicated in the MSB of the Status Register, a program can poll this bit. When it becomes set, the program could read the Interrupt Address Register to determine the cause. Either the normal or nested mode of operation can be used. Note that the functions used with this polled method must have their interrupts enabled.

It is also possible to poll the counters/timers, parallel I/O, and UART separately. To control the UART, one could poll the Status Register. Byte handshakes with the parallel I/O can be controlled by polling Port 1. Finally, each counter/timer has its own register which can be polled.

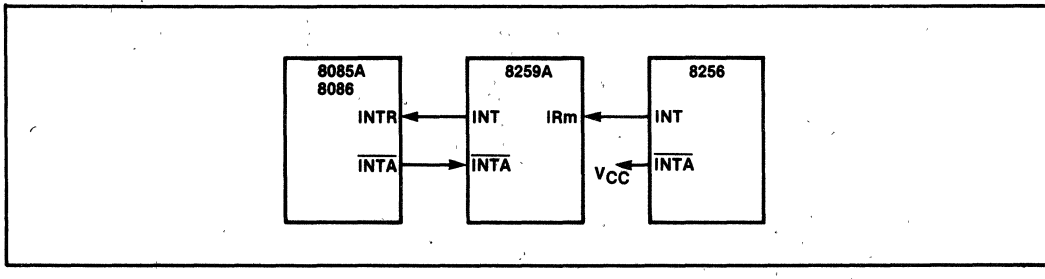


Figure 13. Connecting the 8256 to the 8259A Interrupt Controller

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
AD0-AD4 DB5-DB7	1-5 6-8	I/O	**Address/Data:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**Address Latch Enable:** Latches the 5 address lines on AD0-AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**Read Control:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**Write Control:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**Reset:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**Chip Select:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.
$\overline{INTA}$	14	I	**Interrupt Acknowledge:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	0	**Interrupt Request:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**External Interrupt:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**System Clock:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**Receive Clock:** If the baud rate bits in Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**Receive Data:** Serial data input.
$\overline{CTS}$	21	I	**Clear To Send:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-0FH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 of the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs.

Symbol	Pin No.	Type	Name and Function
			If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**Transmit Clock:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**Transmit Data:** Serial data output.
P27-P20	24-31	I/O	**Parallel I/O Port 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	I/O	**Parallel I/O Port 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
GND	20	PS	**Ground:** Power supply and logic ground reference.
Vcc	40	PS	**Power:** +5V power supply.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 4 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)					(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ=0, the timer input frequency is 16KHz (62.5μs). If FRQ=1, the timer input frequency is 1 KHz (1ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086=0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first $\overline{\text{INTA}}$. In 8086 mode (8086=1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to $\overline{\text{INTA}}$ is for 8086 interrupts where the first $\overline{\text{INTA}}$ is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second $\overline{\text{INTA}}$.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A

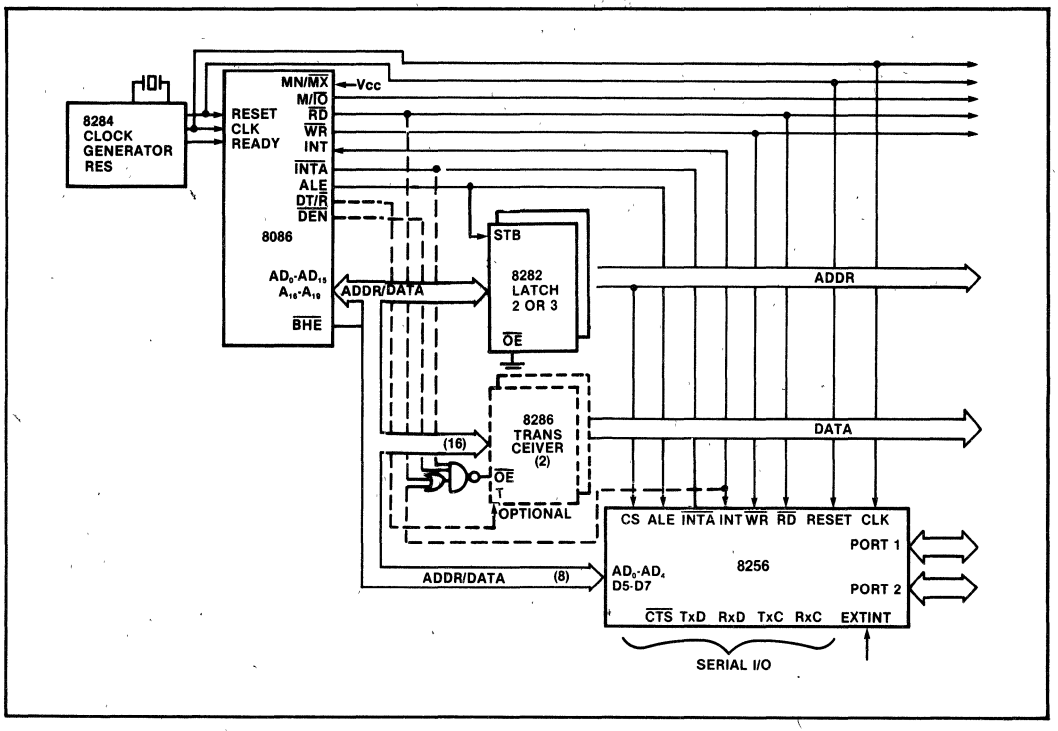


Figure 16. 8086 Min Mode/8256 Interface

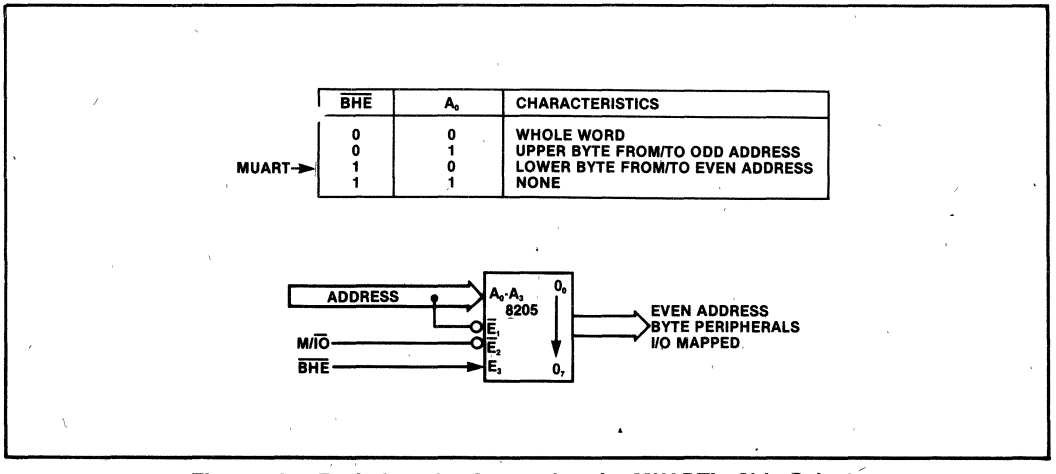


Figure 16a. Technique for Generating the MUART's Chip Select

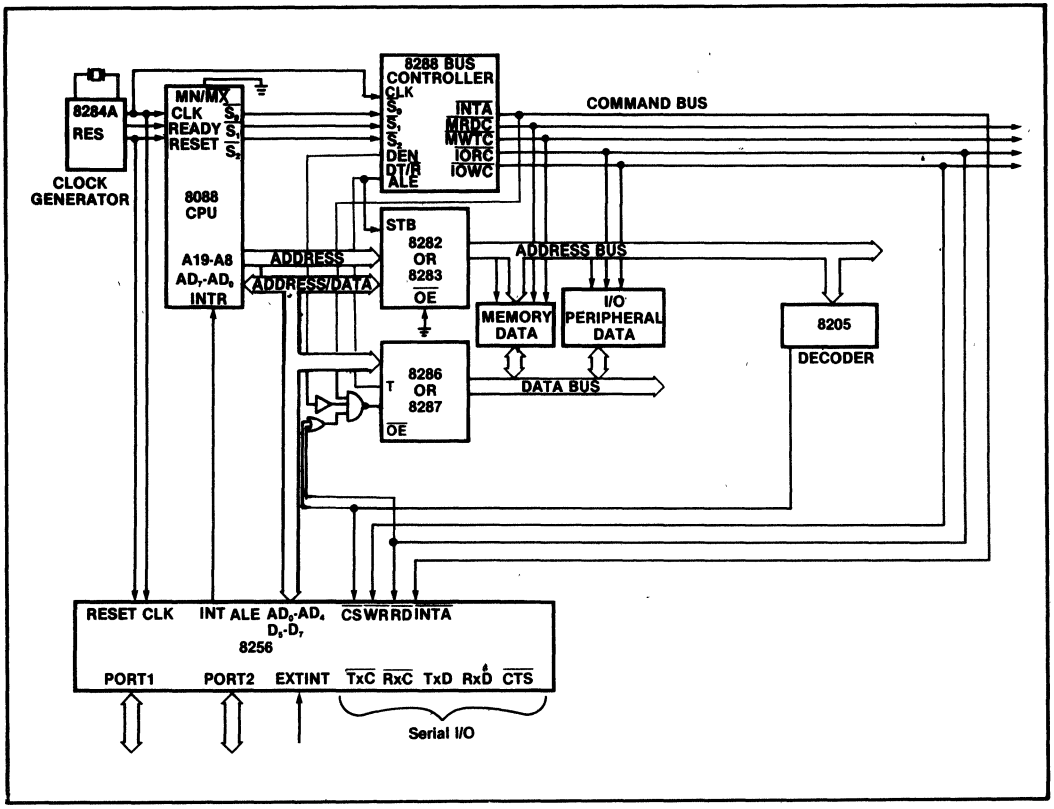


Figure 17. 8088 Max Mode/8256 Interface

READING PORT 1 AND PORT 2

Reading the ports gates the state at the pins onto the data bus if they are defined as I/O pins. A read operation transfers the contents of the associated output latches of pins P12, P13, P15, and P16, which are defined as control function pins. Reading control pins P10, P11, and P17 delivers the state of these pins.

Operating the Event Counters/Timers

The event counters/timers can be loaded with an initial value at any time. Reading event counters/timers is possible without interfering with the counting process.

LOADING EVENT COUNTERS/TIMERS

Loading event counters/timers 1-5 under their respective addresses transfers the data present on the data bus as an initial value into the addressed event counter/timer. The event counter/timer counts from the new initial value immediately following the data transfer (exception: retriggerable mode of Timer 5, or 3 and 5)

Cascaded counters/timers can be loaded with an initial value using one of two procedures:
1) Only the event counter/timer representing the most significant byte will be loaded. The event counter/timer representing the least significant byte is set to 0FFH automatically. Counting is started immediately after the data transfer.
2) The event counter/timer representing the most significant byte will be loaded, causing the least significant byte to be set to 0FFH automatically. Counting is started immediately following the data transfer. Next, the counter representing the least significant byte will be loaded and counting is started

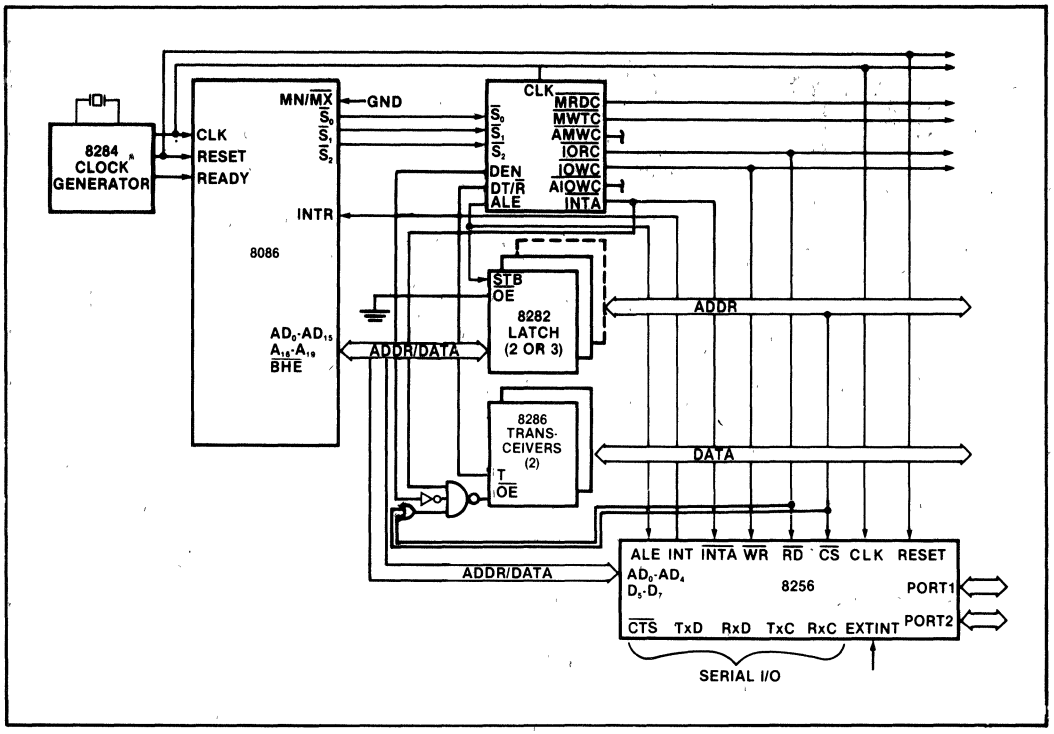


Figure 18. 8086 Max Mode/3256 Interface

again, but this time with a complete 16-bit initial value. The least significant byte of the initial value must be transferred before the counter representing the least significant byte exhibits its zero transition to prevent the most significant byte of the initial value from being decremented improperly.

In the case of an 8-bit initial value for Timer 5 or for cascaded Event Counter/Timer 3 and 5, the initial value for Timer 5 is loaded from a save register, if it is operated in retriggerable counting mode. Counting is started after an initial value has been transferred whenever a high-to-low transition occurs on Port P15.

Cascaded Event Counter/Timer 3 and 5 operating in retriggerable counting mode can be loaded directly with an initial value for Timer 5 representing the most significant byte; Event Counter/Timer 3 will be set to 0FFH automatically.

READING EVENT COUNTERS/TIMERS

Reading event counters/timers 1-5 from their respective addresses gates the counter contents onto the data bus. The counter contents gated onto the data bus remain stable during the read operation while the counter just being read continues to count. The minimum time between the two read operations from the same counter is 1 usec.

The procedure to be followed when reading cascaded event counters/timers is:

1) The event counter/timer representing the most significant byte will be read first. At this time, the least significant byte is latched into read latches.

2) When the event counter/timer representing the least significant byte is addressed, the byte stored in the read latches will be gated onto the data bus. The value stored in the read latches remains valid until it is read, the cascading condition is removed, or a write

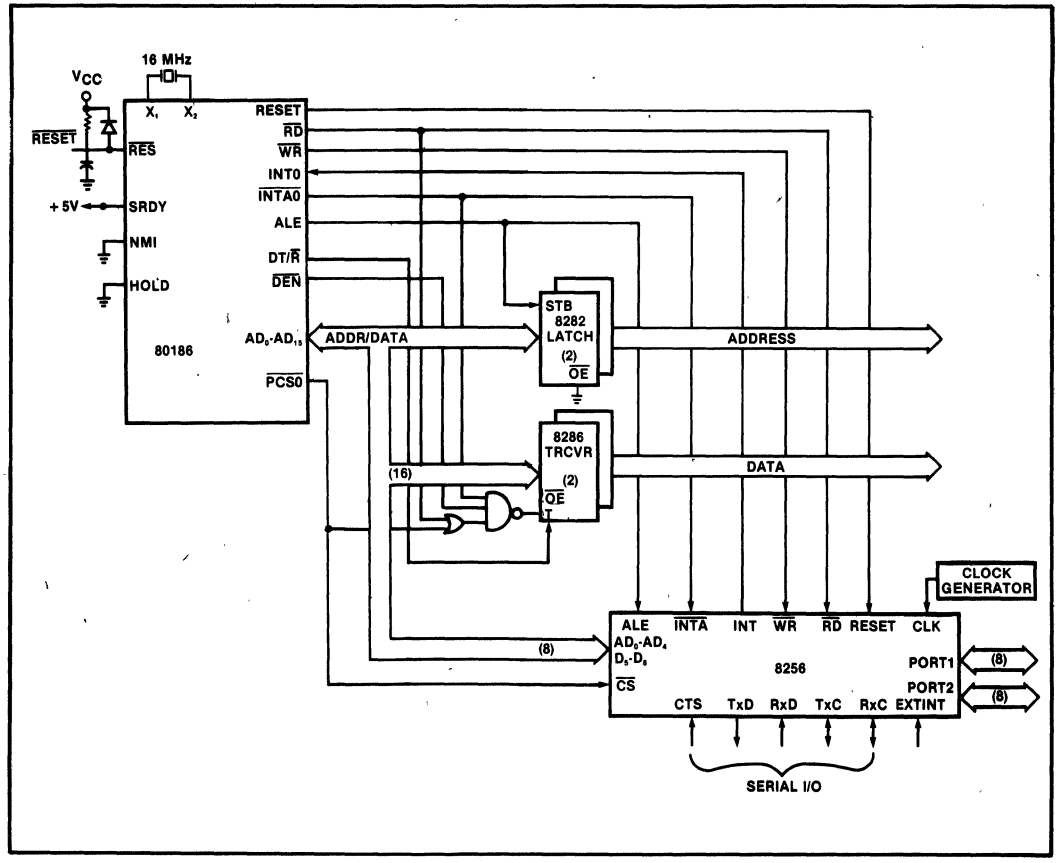


Figure 19. 80186/8256 Interface

operation affecting one of the two event counters/timers is executed.

The time between reading the most significant byte and the least significant byte must be at least 1 usec.

Note:
For cascaded event counters/timers the least significant counter/timer is latched after reading the most significant counter/timer. If the lower byte changes from 00H to 0FFH between the reading of the MSB and the latching of the LSB, the carry from the most significant event counter/timer to the least significant event counter/timer is lost.

Therefore, it is necessary to repeat the whole reading once if the value of the least significant event counter/timer is 0FFH. Doing this will avoid working with a wrong value (correct value + 255).

APPLICATION EXAMPLE

This section describes how the 8256 was designed into a Line Printer Multiplexer (LPM). This application example was chosen because it employs a majority of the MUART's features. The information in this section will be applicable to many other designs since it describes some common software and hardware aspects of using the MUART.

Description of the Line Printer Multiplexer (LPM)

The Line Printer Multiplexer allows up to eight workstations to share one printer. The workstations transmit serial asynchronous data to the LPM. The LPM receives the serial data, buffers it, then transmits

Table 4. MUART Registers

Read Registers								8085 Mode: AD3 / 8086 Mode: AD4	AD2 / AD3	AD1 / AD2	AD0 / AD1	Write Registers							
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	RBF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(1R)					(1W)		

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0...3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of evary data bit of a character. Thus, data can be loaded, e.g., into a shift register external-

ly. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
(2R)					(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1) All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.

2) The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.

3) The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.

4) If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle

(marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{\text{INTA}}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	nibble	input	input
0	0	1	nibble	input	output
0	1	0	nibble	output	input
0	1	1	nibble	output	output
1	0	0	byte handshake	input	
1	0	1	byte handshake	output	
1	1	0	*DO NOT USE*		
1	1	1	test		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value ≥ 3H.

Note:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with

an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse. When a high-to-low transition occurs, Timer 5 is preset to its saved value, But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 5.

Note:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Table 5. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	-	internal clock
2	8-bit timer	T24 = 0, CT2 = 0	internal clock
	8-bit event counter	T24 = 0, CT2 = 1	P12 pin 37
3	8-bit timer	T35 = 0, CT3 = 0	internal clock
	8-bit event counter	T35 = 0, CT3 = 1	P13 pin 36
4	8-bit timer	T24 = 0	internal clock
5	8-bit timer, normal mode	T35 = 0, T5C = 0	internal clock
	8-bit timer, retriggerable mode	T35 = 0, T5C = 1	internal clock
2 and 4 cascaded	16-bit timer	T24 = 1, CT2 = 0	internal clock
	16-bit event counter	T24 = 1, CT2 = 1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35 = 1, T5C = 0, CT3 = 0	internal clock
	16-bit event counter, normal mode	T35 = 1, T5C = 0, CT3 = 1	P13 pin 36
	16-bit timer, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 0	internal clock
	16-bit event counter, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 1	P13 pin 36

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4R)				(4W)			

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2, $\overline{OBF}$ indicates that a character has been loaded into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset high by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0
		(5R)		(5W = enable, 6W = disable)			

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register

0	0	0	D4	D3	D2	0	0
(6R)					Interrupt Level Indication		

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge $\overline{INTA}$; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0
	(7R)				(7W)		

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set. Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
	(8R)				(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
	(9R)				(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
	($0A_{16}$-$0E_{16}$R)				($0A_{16}$-$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded. writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X 256+255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
			($0F_{16}$R)				

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is requied to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section under "Receive Break Detect" and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}W$)

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

Note:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

- A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
  * The start bit check is enabled.
  * Status Register Bit 0 (FE) indicates framing error.
  * The sampling time of the serial receiver is the bit center.

  A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length	
0	1	1	1	1	1	(Start of Bit)
0	1	1	1	0	2	
0	1	1	0	1	3	
0	1	1	0	0	4	
0	1	0	1	1	5	
0	1	0	1	0	6	
0	1	0	0	1	7	
0	1	0	0	0	8	
0	0	1	1	1	9	
0	0	1	1	0	10	
0	0	1	0	1	11	
0	0	1	0	0	12	
0	0	0	1	1	13	
0	0	0	1	0	14	
0	0	0	0	1	15	
0	0	0	0	0	16	(Bit center)
1	1	1	1	1	17	
1	1	1	1	0	18	
1	1	1	0	1	19	
1	1	1	0	0	20	
1	1	0	1	1	21	
1	1	0	1	0	22	
1	1	0	0	1	23	
1	1	0	0	0	24	
1	0	1	1	1	25	
1	0	1	1	0	26	
1	0	1	0	1	27	
1	0	1	0	0	28	
1	0	0	1	1	29	
1	0	0	1	0	30	
1	0	0	0	1	31	
1	0	0	0	0	32	(End of Bit)

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT is inactive (LOW).

4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6) Status Register Bit 0 implies framing error.

7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 8085, 8086, 8088, and 80186 microprocesors. Figures 14 through 19 display the block diagrams for these interfaces. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines.

In Figure 15, the 8088 min mode, the 8205 chip select decoder is connected to the 8088's address bus lines A8-A15. These address lines are stable throughout the entire instruction cycle. However, the MUART's chip select signal could have been derived from A16/S3-A19/S6.

Figure 16 shows the 8256 interfaced with an 8086 in the min mode. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space. Figure 16A shows a table and a diagram of how the 8256 may be selected in an 8086 system where the MUART is I/O mapped and used on the lower byte of the address/data bus.

PROGRAMMING

Initialization

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Word 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. It is a good idea to reset the part before initialization. (Either a hardware or a software reset will do.)

Operating the Serial Interface

The microprocessor transfers data to the serial interface by writing bytes to the Transmit Buffer Register. Receive characters are transferred by reading the Receiver Buffer Register. The Status Register provides all of the necessary information to operate the serial I/O, including when to write to the Transmit Buffer, and when to read the Receive Buffer and error information.

Transmitting

The transmitter and the receiver may be operated by using either polling or interrupts. If polling is used then the software may poll the Status Register and write a byte to the Transmit Buffer whenever TBE = 1. Writing a byte to the Transmit Buffer clears the TBE

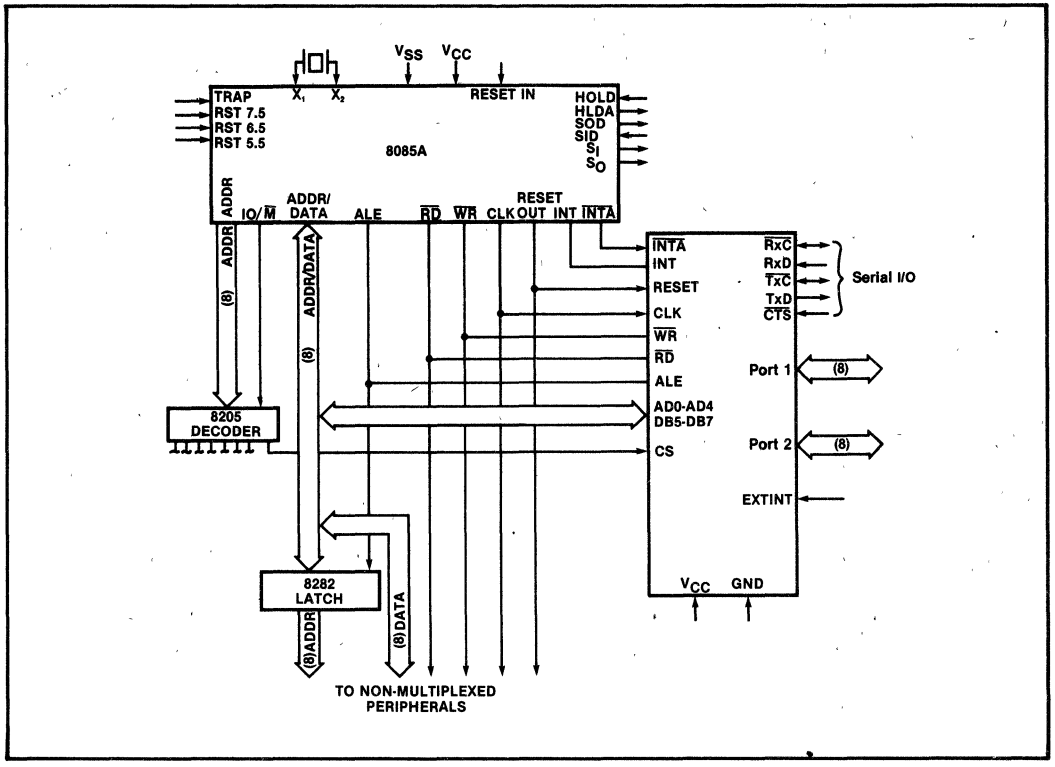


Figure 14. 8085/8256 Interface

status bit. If the $\overline{CTS}$ pin is low, then the Transmit Buffer will transfer the data to the Transmit Register when it becomes empty. When this transfer takes place the TRE bit is reset, and the TBE bit is set indicating the next byte may be written to the Transmit Buffer. If $\overline{CTS}$ is high, disabling the transmitter, the data byte will remain in the Transmit Buffer and TBE will remain low until $\overline{CTS}$ goes low. The transmitter can only buffer one byte if it is disabled.

There is no way of knowing that the transmitter is disabled unless the $\overline{CTS}$ signal is fed into one of the I/O ports. Using the transmitter interrupt will free up the CPU to perform other functions while the transmitter is disabled or while the Transmit Buffer is full.

To enable the transmit interrupt feature Bit L5 in the Set Interrupt Register must be set. An interrupt request will not occur immediately after this bit has been set. Before any transmit interrupt request will occur a byte must be written to the Transmit Buffer. After the first byte has been written to the Transmit Buffer, a transmit interrupt request will occur, providing the transmitter is enabled.

There are three sources of transmitter interrupt requests: TBE = 1, TRE = 1, and Break-In Detect. Assuming the Break-In Detect feature is disabled, after the transmit interrupt is enabled and the first byte is written, a transmit interrupt request will be generated by TBE going active. The microprocessor can immediately write a byte to the Transmit Buffer without reading any status. However if Break-In Detect is enabled, the Status Register must be read to determine whether the transmit interrupt request was generated by Break-In Detect or TBE.

The TRE interrupt request can be used to indicate when the transmitter has completely sent all of the data. For example, using half-duplex communica-

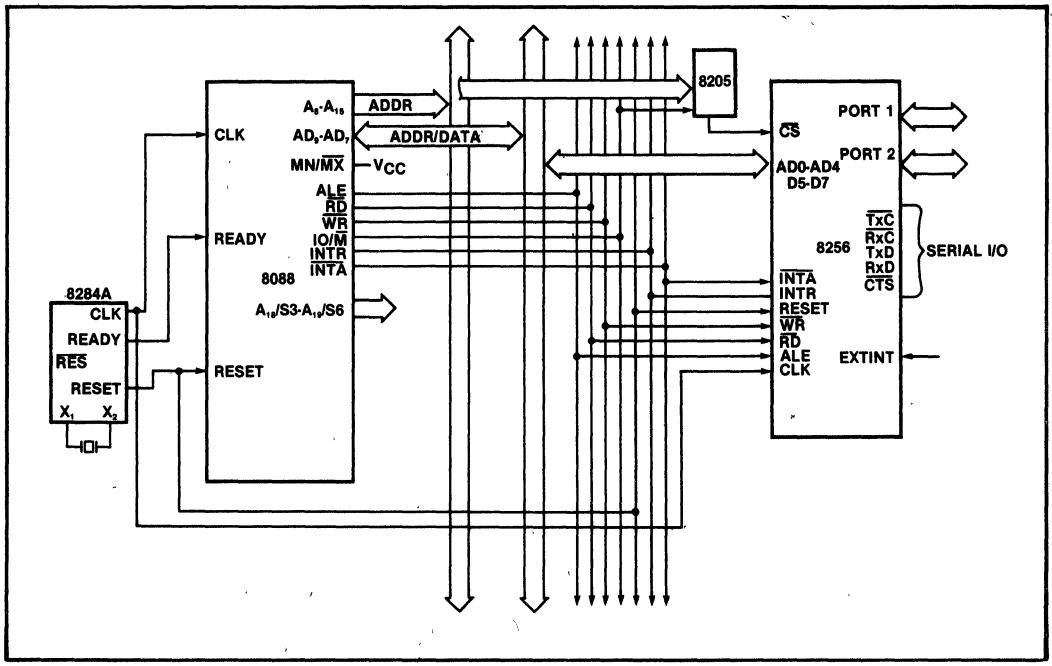


Figure 15. 8088 Min Mode/8256 Interface Multiplexed Bus

tions, all of the data written to the MUART must be transmitted before the line can be turned around. After the last byte is written, an interrupt request will be generated by TBE. If this interrupt is acknowledged without writing another byte, then the next transmitter interrupt request, TRE = 1, will indicate that the transmitter is empty and the line may be turned around.

RECEIVING

Valid data may be read from the Receive Buffer whenever the RBF bit in the Status Register is set. Reading the Receive Buffer resets the RBF status bit. The RBF bit in the Status Register can be used for polling. When the RBF bit is set, the three receive status bits, PE, OE, and FE are updated. These three status bits are reset when they are read. Therefore when the status register is read with RBF set, the three error status bit should be tested too.

If interrupts are used for serial receive data, the receiver must be enabled by setting the RxE bit in Command Register 3, and Bit L4 must be set in the Set Interrupt Register. When the receive interrupt request occurs the Receive Buffer may be read, but the status register should also be read since the receive interrupt could have been generated by the Break Detect. Also, reading the status register will indicate whether there were any errors in the received character.

Operating the Parallel Interface

Data can be transferred to or read from Port 1 and Port 2 by using the appropriate write and read operations.

LOADING PORT 1 and PORT 2

Writing to the ports transfers the data present on the data bus into the output latches. This operation is independent of the programmed I/O characteristics of the individual port pins. Writing to control or input ports has no effect on the state of the pins. Pins defined as outputs immediately assume the state which is associated with the transferred data. If inputs or control pins are reprogrammed into outputs, they assume the states stored in their output latches which were transferred by the most recent port write operation.

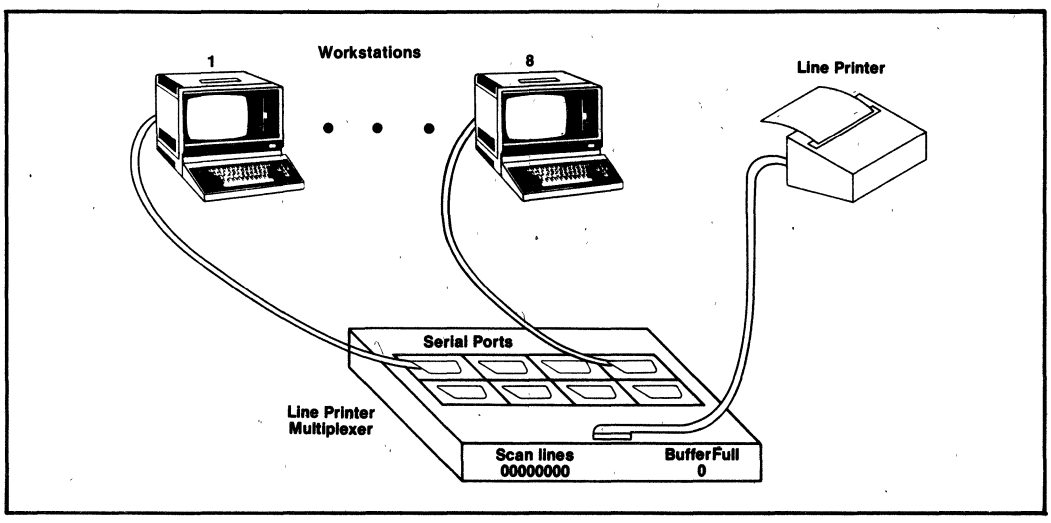


Figure 20. Using the Line Printer Multiplexer to Share a Line Printer

it to the line printer using a two-wire byte handshake Dataproducts interface. A conceptual diagram of this system is shown in Figure 20. Note that only one workstation can transmit at a time. This workstation will transmit its entire file before another workstation will be allowed to transmit.

The LPM sequentially polls each of the eight RS-232 ports for a Request To Send ($\overline{RTS}$). When it finds a serial port which has asserted $\overline{RTS}$, it configures itself for the appropriate data format and bit rate, establishes the connection and sends back to the serial port a Clear To Send ($\overline{CTS}$) which enables transmission. The LPM receives the serial asynchronous data, buffers it in a software FIFO, and transmits the data to the line printer. If the LPM detects an error in any of the serial characters it receives, it transmits an error message to the serial port and ignores the bad character. If the LPM does not receive a serial character after 18 seconds, it assumes that the transmission is complete. It transmits the final status to the serial port, and returns to scanning.

This LPM was designed to be used with single-user workstations and a 300 lines per minute line printer. These workstations are not multitasking; therefore in the middle of a file transfer when the CPU needs to reload its buffer from the disk, no serial data is transmitted. During this time the LPM is emptying its FIFO; thus, the line printer never stops printing.

The buffer size on the LPM was chosen to complement the disk access time on the workstations. Figure 21 illustrates the buffer size calculation. The line printer can print up to 300 lines per minute, or approximately 660 characters per second. This corresponds to a serial transmission rate of 6,600bps (assuming ASCII character codes and a parity bit) as shown in equation 1.

(1) Serial bit rate for the line printer $= \frac{(300\ \text{lines/min})*(132\ \text{char/line})*(10\ \text{bits/char})}{(60\ \text{sec/min})}$

The bottleneck in this data transfer is the line printer since the MUART and the workstations can both transmit and receive at 19.2Kbps. To realize the maximum data transfer rate of this system the LPM must guarantee that the average transfer rate to the line printer is 660 characters per second. The maximum amount of dead time that the serial port on the workstation is not transmitting, multiplied by 660 is the number of bytes which the LPM should buffer. It was determined through experimentation that it takes about 3 seconds to load 40K bytes of data from the disk into the workstation's RAM. During these 3 seconds no serial data is being sent; therefore the buffer size on the LPM should be 2K bytes. (Note: even though only a 2K byte FIFO is required, this design used an 8 Kbyte FIFO.)

To keep the LPM's buffer full the serial data rate must be greater than 6.6Kbps. The two bit rates which the

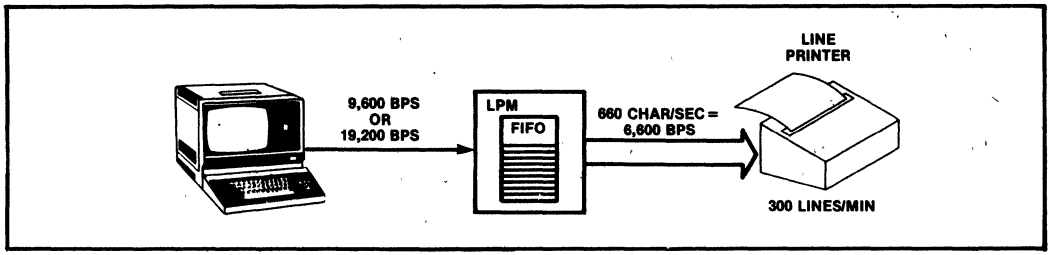


Figure 21. LPM Buffer Size Calculation

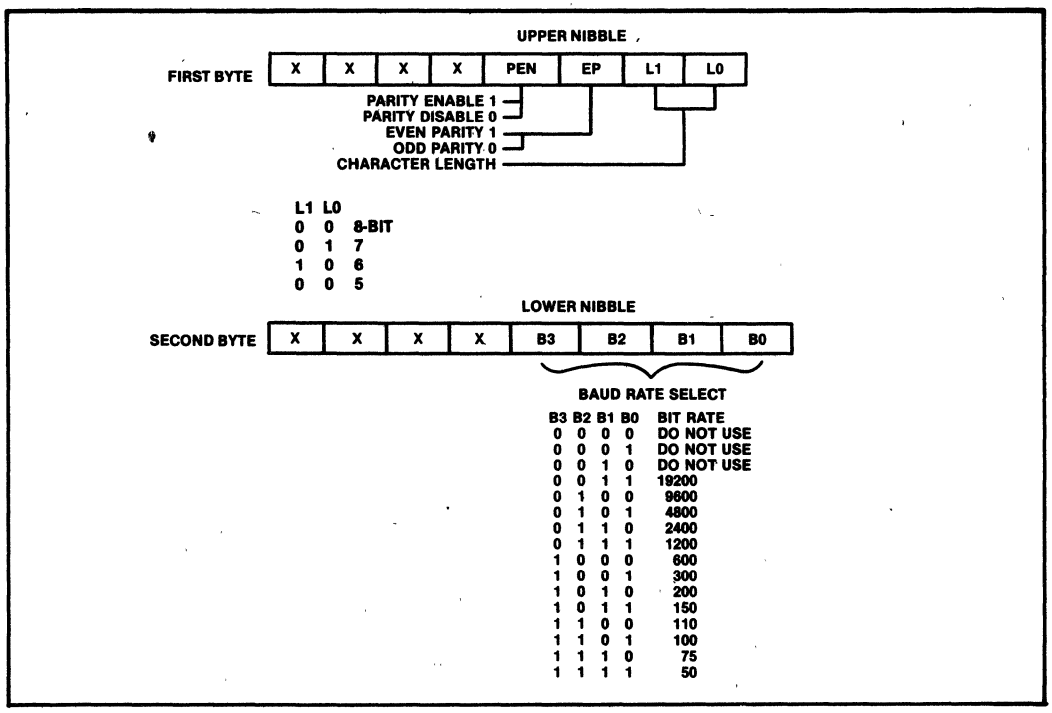


Figure 22. Programming Words Format for LPM

workstations use are 9.6Kbps and 19.2Kbps. The $\overline{CTS}$ signal is used to control the flow of the serial data so that the LPM buffer will not overflow.

Each serial port on the LPM can have a different bit rate, character length, and parity format. These parameters are programmable through the serial port. When the LPM powers up, or is reset, it expects a bit rate of 9600 bps, 7 bit characters, and odd parity. When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.

Description of the Hardware

Figure 23 shows a block diagram of the LPM. In addition to the standard components of most microprocessor systems such as CPU, RAM, and ROM this particular design requires a UART, timers, parallel I/O and an interrupt controller. The MUART is the ideal choice for this design since it integrates these four functions onto one device.

The eight serial I/O ports use four signals: Transmit Data (TxD), Receive Data (RxD), Request To Send ($\overline{RTS}$), and Clear To Send ($\overline{CTS}$). These four signals, controlled by the MUART, are connected to one port at a time using TTL multiplexers. The TTL multiplexers are interfaced to RS-232 transceivers to be electrically compatible with the RS-232 spec. The serial port select address is derived from three bits of the MUART's parallel I/O port (Port 1). Two more bits from Port 1 control $\overline{CTS}$ and $\overline{RTS}$, and another bit lights up an LED to indicate when the LPM's buffer is full. Parallel Port 2 and two bits from Port 1 are connected to the line printer implementing a two-wire byte handshake transfer. These signals are passed through a line driver so that they can reliably drive a long cable.

There are three timing functions needed for the LPM: a scan timer, a debounce timer, and a recieve timeout. The Scan timer determines the amount of time spent sampling $\overline{RTS}$ on each port before the next port is addressed. By using one of the MUART's timers to do this function, the CPU is free to perform other functions instead of implementing the timer in software. If $\overline{RTS}$ is recognized as true, the CPU branches into a debounce procedure. This procedure uses another one of the MUART's timers to wait 10 msec then sample $\overline{RTS}$ again, thus preventing any glitches from registering as a false $\overline{RTS}$. The receive timeout timer uses two 8-bit timers in the cascaded mode to measure an 18-second interval. After a valid $\overline{RTS}$ is recognized,

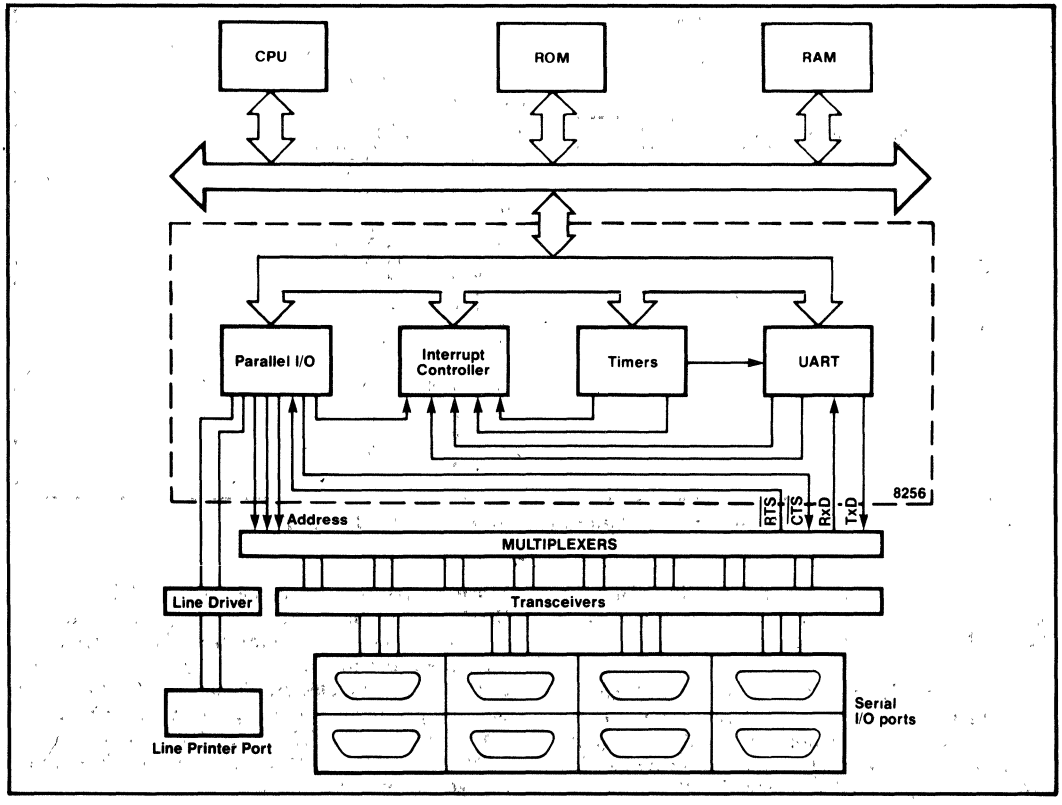


Figure 23. Functional Block Diagram of the Line Printer Multiplexer

the LPM sends back a $\overline{CTS}$ and initializes the receive timeout timer for 18 seconds. Each time a character is received by the LPM, this timer is reinitialized. If this timer times out, the LPM considers the transmission complete and returns to scanning.

The schematic diagram of the LPM is shown in Figure 24. The CPU is an 8088 used in the min mode. It is interfaced directly to the 8256. An 8282 latch is employed in the system so that nonmultiplexed bus memory can be used. A 2716 holds the entire program, and six 2016s (2K x 8 static RAMs) are used to store the buffer, temporary data, stack area, and interrupt vector table. The 2716 is located in the upper 2K of the 8088 address space (FF800-FFFFFH) so that the reset vectors can be stored starting at location FFFF0H. The RAM address space spans 0-2FFFH so that the interrupt vector table can be stored starting at location 0. The MUART is I/O mapped and its registers occupy even addresses from 0 to 1EH. Using an 8088 CPU the MUART must be placed in the 8086 mode since the $\overline{INTA}$ signal is used; hence the register addresses are all even numbers.

The line printer used provides a choice of two standard parallel interfaces: Centronics or Dataproducts. The Centronics interface uses a two-wire handshake pulsed strobe where the transmitter asserts a complete strobe pulse before an acknowledge is received. The Dataproducts interface is an interlocking two-wire handshake. The Dataproducts interface was chosen since it is directly compatible with the MUART's two-wire byte handshake. The MUART could also be connected to the Centronics interface; however, additional hardware would be necessary to generate the pulsed strobe for correct interrupt operation. Figure 25 shows the timing of the Dataproducts interface and Table 6 lists the connector pin configuration.

Table 6. Dataproducts Interface Line Functions

Signal	Description	Connector Pin
Data Request	Sent by printer to synchronize data transmission. When true, requests a character. Remains true until **Data strobe** is received, then goes false within 100 nsec.	E(return C)
Data Strobe	Sent by user system to cause printer to accept information on data lines. Should remain true until printer drops **Data Request** line. Data lines must stabilize for at least 50 nsec before **Data Strobe** is sent.	j(return m)
Data Bit 1	Bit 8 controls optional character set Refer to *Commands and Formats.*	B(return D)
Data Bit 2		F(return J)
Data Bit 3		L(return N)
Data Bit 4		R(return T)
Data Bit 5		V(return X)
Data Bit 6		Z(return b)
Data Bit 7		n(return k)
Data Bit 8		h(return e)
VFU Control (PI)	Optional control from user system. Used for VFU control. Data Request/Strobe timing is same as for data lines.	p(return s)
Ready	Sent to user system by printer. True when no Check condition exists.	CC(return EE)
On Line	Sent to user system by printer. True when **Ready** line is true and operator has activated ON LINE Pushbutton. Enables interface activity.	y(return AA)
Interface Verify	Jumper in printer connector. Continuity informs user system that connector is properly seated.	x to v
+5V	Supply voltage for Exerciser only.	HH

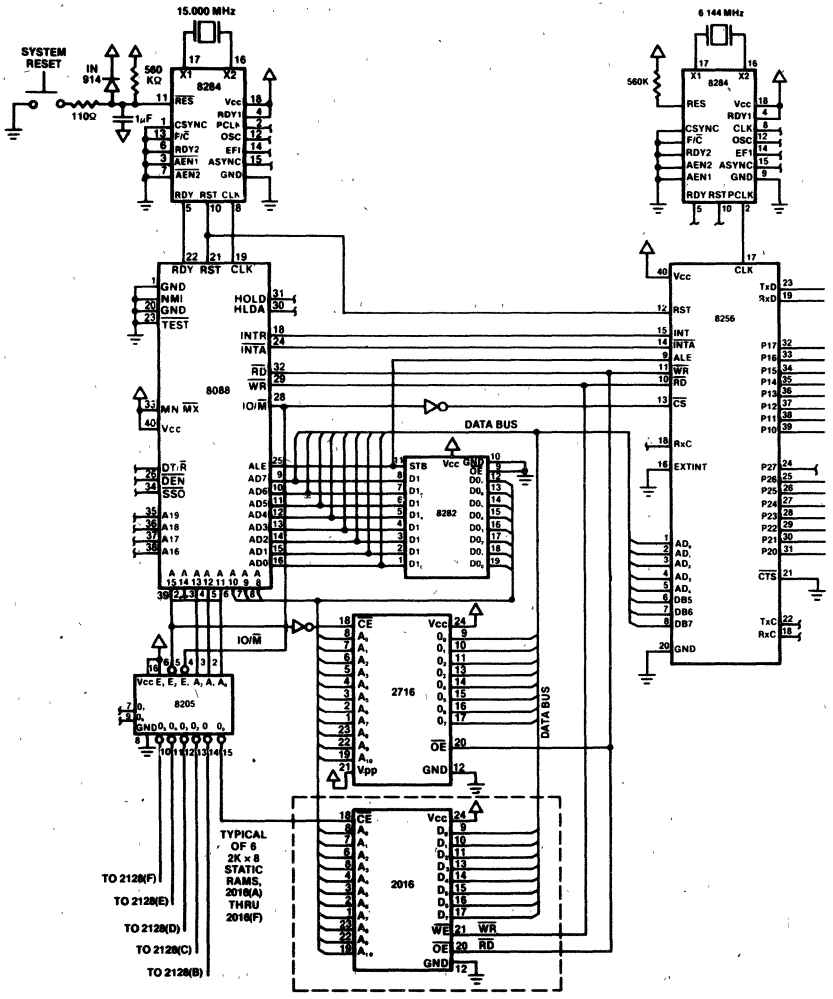


Figure 24. Schematic of LPM

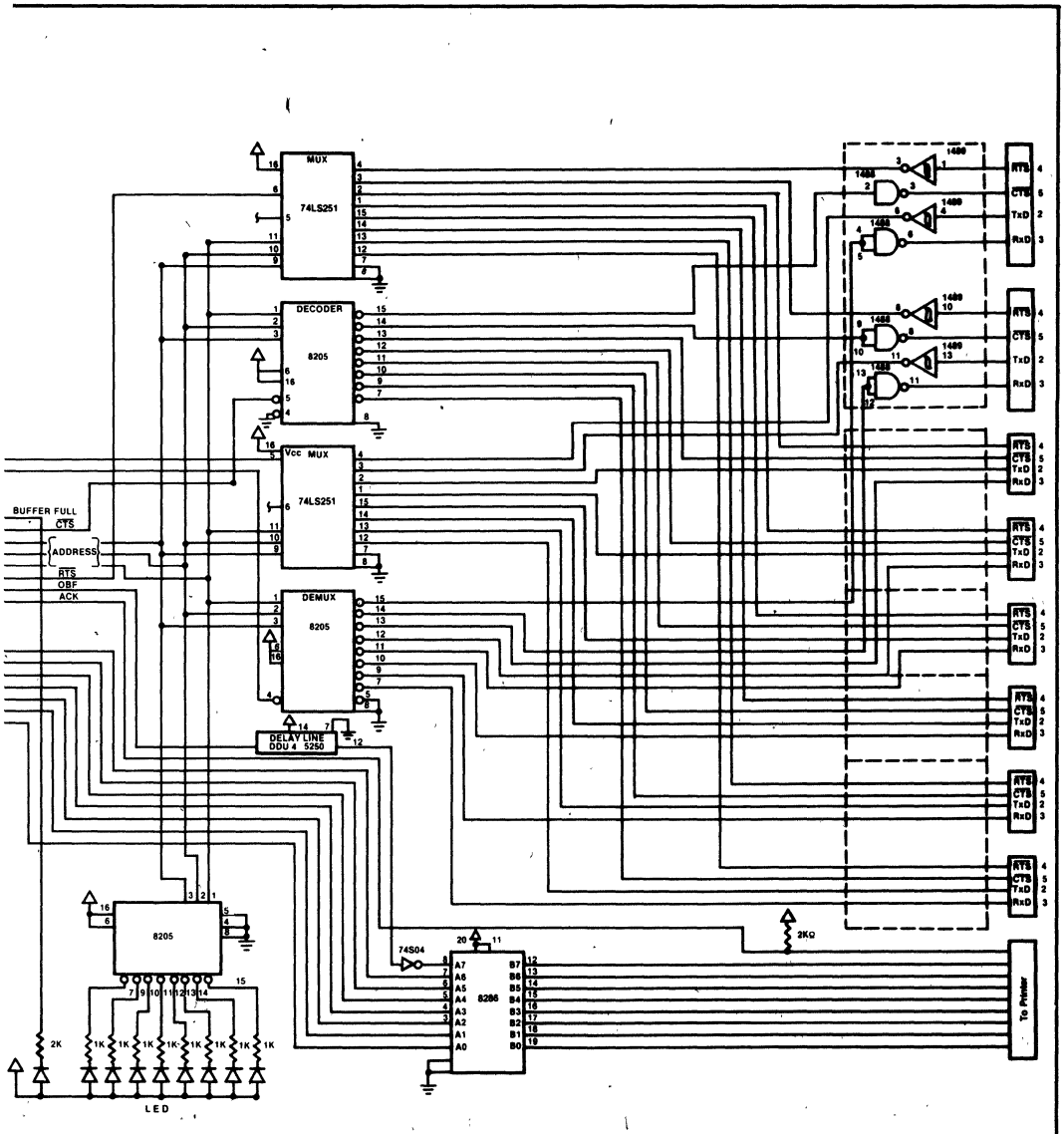


Figure 24. Schematic of LPM (Continued)

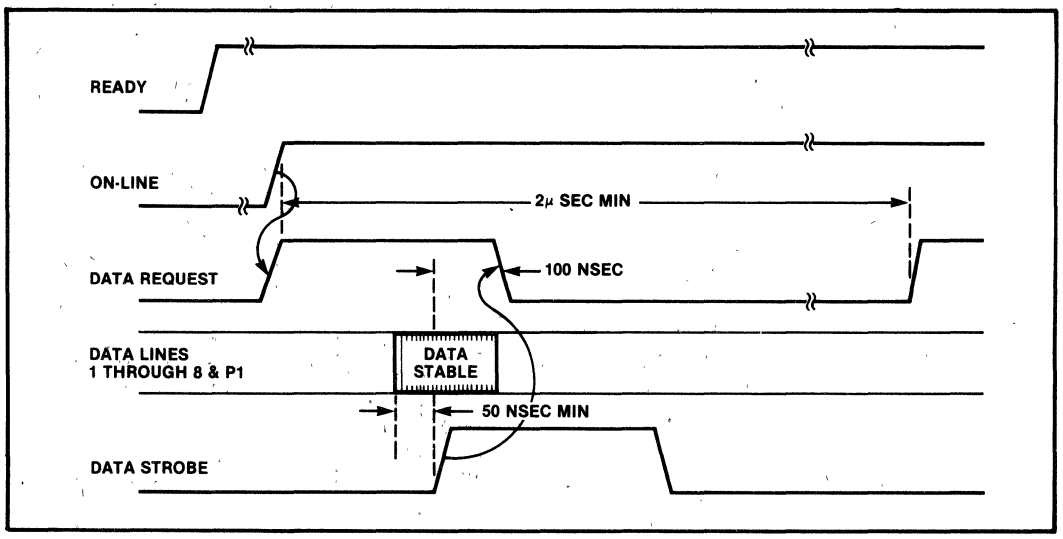


Figure 25. Timing of Dataproducts Interface

Only ten signals are used to interface the LPM to the line printer: Data Request, Data Strobe, and the eight data lines. The most significant data line is not used since the character code is 7-bit ASCII. Data Strobe connects to $\overline{OBF}$ on the MUART; however, for the Dataproducts interface this signal must be inverted. Data Request is connected to $\overline{ACK}$ on the MUART. When the line printer is ready to accept data, the Data Request signal goes high. The 8256 will not interrupt the CPU to transmit parallel data unless this signal is high.

The Dataproducts interface is slightly different from the MUART's two-wire handshake in that it latches the data on the leading edge of the strobe signal. When the MUART receives bytes it latches the data on the trailing edge. As a result the Dataproducts interface has a 50 nsec setup time for data stable to the leading edge of Data Strobe. In the LPM hardware a delay line was used to realize this setup time.

Description of the Software

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 26. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connected above are interrupt procedures. They are entered when the MUART interrupts the CPU and vectors an indirect address to it.

The LPM program uses nested interrupts; the priority of the interrupt procedures is given in Table 7.

Table 7. Line Printer Multiplexers' Interrupt Priority

	Priority	Source
Highest	0	Debounce timer
	1	Not Used
	2	Not Used
	3	Receive timer
	4	RxD Interrupt
	5	TxD Interrupt
	6	Scan timer
	7	LP Interrupt

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. In the steady state of the LPM's operation the UART will be receiving data, and the parallel port will be transmitting data. The serial receiver should be the highest priority since it can have overrun errors. This is the case because the debounce timer will be disabled, and the receive timeout interrupt will only occur when serial reception has ended. Therefore the RxD request can interrupt any other service routine, thus preventing any possibility of an overrun error.

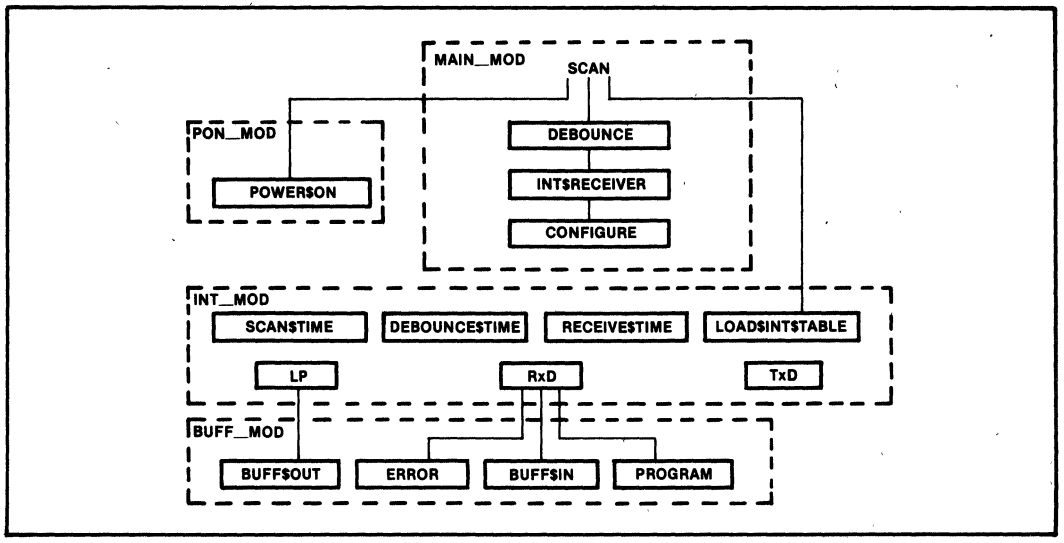


Figure 26. Block Diagram of LPM Software Structure

On power-up the CPU branches from 0FFFF0H to the INITCODE routine which is included in the machine code by the MDS locater utility. INITCODE initializes the 8088's segment registers, stack pointer, and instruction pointer, then it disabled interrupts and jumps into MAIN_MOD. The first executable instruction in MAIN_MOD calls POWER$ON, which initializes the MUART, flags, variables, and arrays. The MAIN_MOD calls LOADINTTABLE, which initializes the interrupt vector table. The CPU's interrupt is then enabled and the program enters into a DO FOREVER loop which scans the eight serial ports for an $\overline{\text{RTS}}$.

There are three software functions which employ the MUART's timers and interrupt controller to measure time intervals: SCAN, debounce, and INIT$RECEIVER. DEBOUNCE and INIT$RECEIVER procedures, employ the MUART's timers and interrupt controller to measure time intervals. The CPU remains in a loop for a specific amount of time before it proceeds with the next section of code. In this loop the CPU is waiting for a global status flag to change while servicing any interrupts which may occur. When the appropriate timer interrupt occurs, the interrupt service routine will set the global flag which causes the CPU to exit the loop and proceed to the next section of code. An example can be seen from the scan flow chart in Figure 27.

The first thing the program does before entering the loop is set the flag (in this case SCAN$DELAY) TRUE. The timer is initialized and the loop is entered. As long as SCAN$DELAY is TRUE the CPU will continue to sample $\overline{\text{RTS}}$. If $\overline{\text{RTS}}$ remains false for more than 100 msec, the timer interrupts the CPU and the interrupt service routine sets SCAN$DELAY FALSE. This causes the CPU to exit the loop and address the next port. The process is then repeated. If $\overline{\text{RTS}}$ becomes true while it is being sampled, the DEBOUNCE procedure is called.

DEBOUNCE does nothing more than wait 10 msec and sample $\overline{\text{RTS}}$ again using the same technique discussed above. If $\overline{\text{RTS}}$ is still valid INIT$RECEIVER is called, otherwise the CPU returns to scan.

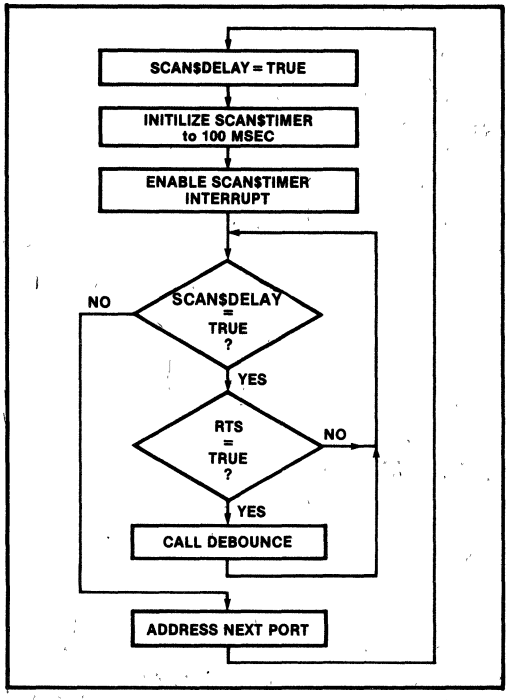


Figure 27. Scan Flow Chart

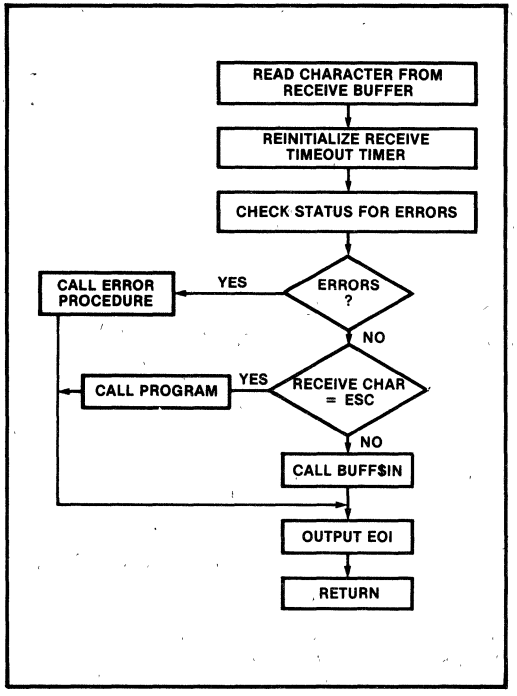


Figure 28. RxD Interrupt Procedure Flow Chart

INIT$RECEIVER calls CONFIGURE which programs the MUART for the bit rate, number of bits in a character, and parity format. This information is stored in an array called SERIAL$FORMAT, which contains a byte for each port. The bytes in the SERIAL$FORMAT array have the same bit definition as the two nibbles in the programming words in Figure 22. Upon returning to INIT$RECEIVER the receiver is enabled, the receive timeout timer is initialized, and the timer and receiver interrupts are enabled. CTS on the serial port is then set true, and the CPU enters a loop which does nothing except wait for 18 seconds. If no characters are received within 18 seconds, the receive timeout interrupt occurs and the loop flag is set false, which causes the CPU to exit the loop. If a character is received, a receive interrupt occurs, and the CPU vectors into the RxD interrupt service routine.

Figure 28 shows a flow chart of the RxD interrupt service routine. This routine begins by reading the receive buffer and reinitializing the receive timeout timer. There are two conditions to check for before the character can be inserted into the FIFO. First, if there are any errors in the received character, an ERROR procedure is called which reports back to the serial port what the error condition was. The character in error is discarded and the routine returns. The other condition is that if the received character is an ASCII ESC, the PROGRAM procedure is called. If neither one of these conditions occurs, the character is placed in the FIFO by the BUFF$IN procedure.

The LP interrupt routine is entered when the byte handshake interrupt request is acknowledged. This routine simply calls the BUFF$OUT procedure, which extracts a byte out of the FIFO. BUFF$OUT returns the byte to the LP interrupt procedure, which then writes it to Port 2. One small problem with getting the handshake interrupt going is that the first byte has to be written to Port 2 before the first handshake interrupt will occur. The problem is that the line printer may not be ready for the first byte. This would be indicated by DATA REQUEST being low. If the byte was written to the LP while DATA REQUEST is low, it would be lost. Note that if the handshake interrupt is enabled while DATA REQUEST is low, then DATA REQUEST goes high, the interrupt will occur without

writing the first byte. There are several ways to solve this problem. Port 1 can be read to find out what the state of the DATA REQUEST line is. If DATA REQUEST is low, the CPU can simply wait for the interrupt without writing the first byte. If DATA REQUEST is high, then the first data byte may be written. Another solution would be to write a NUL character as the first byte to Port 2. If DATA REQUEST is low, then a worthless character is lost. If DATA REQUEST is high, the NUL character would be sent to the line printer; however, it is not printed since NUL is a nonprintable character. The LPM program uses the NUL character solution.

BUFFER MANAGEMENT

The FIFO implementation uses an 8K byte array to store the characters. There are two pointers used as indexes in the array to address the characters: IN$POINTER and OUT$POINTER. IN$POINTER points to the location in the array which will store the next byte of data inserted. OUT$POINTER points to the next byte of data which will be removed from the array. Both IN$POINTER and OUT$POINTER are declared as words. Figure 29 illustrates the FIFO in a block diagram.

The BUFF$IN procedure receives a byte from the RxD interrupt routine and stores it in the array location pointed to by IN$POINTER, then IN$POINTER is incremented. Similarly, when BUFF$OUT is called by the LP interrupt routine, the byte in the array pointed to by OUT$POINTER is read. OUT$POINTER is incremented, and the byte which was read is passed back to the LP interrupt routine. Since IN$POINTER and OUT$POINTER are always incremented, they must be able to roll over when they hit the top of the 8K byte address space. This is done by clearing the upper three bits of each pointer after it is incremented.

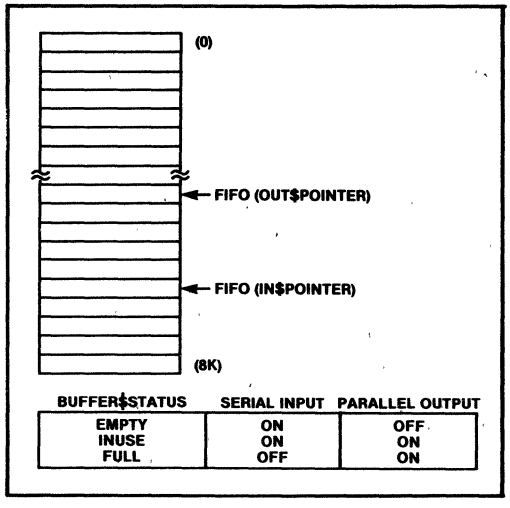


Figure 29. FIFO Structure and Status

IN$POINTER and OUT$PONTER not only point to the locations in the FIFO, they also indicate how many bytes are in the FIFO and whether the FIFO is full or empty. When a character is placed into the FIFO and IN$POINTER is incremented, the FIFO is full if IN$POINTER equals OUT$POINTER. When a character is read from the FIFO and OUT$POINTER is incremented, the FIFO is empty if OUT$POINTER equals IN$POINTER. If the buffer is neither full nor empty, then it is in use. A byte called BUFFER$STATUS is used to indicate one of these three conditions.

The software uses the buffer status information to control the flow into and out of the FIFO. When the FIFO is empty the handshake interrupt must be turned off. When the FIFO is full, $\overline{CTS}$ must be sent false so that no more data will be received. If the buffer status is in use, $\overline{CTS}$ is true and the handshake interrupt is enabled.

Figure 30 shows the flow chart of the BUFF$IN procedure. The BUFF$IN procedure begins by checking the BUFFER$STATUS. If it is empty and the character to be inserted into the FIFO is a CR or LF, the handshake interrupt is enabled, a NUL character is output, and the BUFFER$STATUS is set to IN-USE. The character passed to BUFF$IN from RxD is put into the FIFO. If the FIFO is now full, the BUFFER$STATUS is set to FULL, $\overline{CTS}$ is set false, and the buffer full LED is turned on.

Figure 31 shows the flow chart of the BUFF$OUT procedure. After the character is read from the FIFO, the FIFO is tested to determine if it is empty. If it is not empty, the BUFFER$STATUS is FULL and there are 200 bytes available in the FIFO, serial data reception is reenabled, and the FIFO fills again. While data is being received from the workstation, $\overline{CTS}$ toggles high and low, filling up and emptying the last 200 bytes in the FIFO. Referring to the top of the flow chart (FIFO empty test) if it's empty, the BUFFER$STATUS is set to EMPTY, and the handshake interrupt is disabled. During this time all interrupts

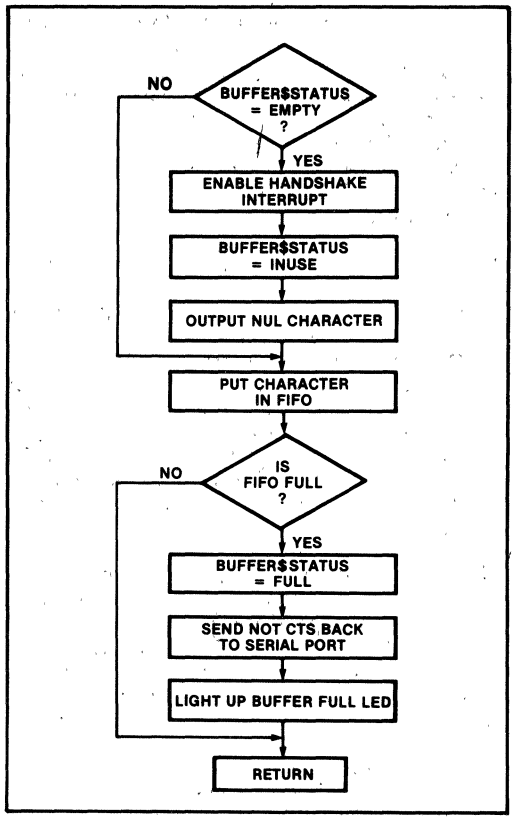


Figure 30. Flow Chart of the BUFF$IN Procedure

are disabled at the CPU. (Remember that the RxD interrupt routine can interrupt the LP and BUFF$OUT procedures since it has a higher priority, and the MUART is in the nested mode.)

If the CPU interrupt was not disabled during this time, the following events could occur which would cause the LPM to crash. Assume that the RxD interrupt occured where the asterisk is in the flow chart, after BUFFER$STATUS is set to EMPTY. The BUFF$IN procedure would set BUFFER$STATUS to INUSE and enable the handshake interrupt. When the RxD interrupt routine returned to BUFF$OUT, the handshake interrupt is disabled, but the BUFFER$STATUS is INUSE. The handshake interrupt could never be reenabled, and the FIFO would fill up. This is known as a critical section of code. Suspicion should arise for a critical section of code when two or more nested interrupt routines can affect the same status. One solution is to disable the interrupt flag at the CPU while the status and conditional operations are being modified.

The flow chart for the TxD interrupt procedure is given in Figure 32. For this program five different messages can be transmitted, and they are stored in ROM. It is possible to download the messages into a dedicated RAM buffer; however, the RAM buffer would have to be as large as the largest message. A more efficient way to transmit the messages is to read them from ROM. In this case the address of the first byte of the message would have to be accessible by the transmit interrupt procedure. Since parameters cannot be passed to interrupt procedures, this message pointer is declared PUBLIC in one module and EXTERNAL in the other modules.

To get the transmit interrupt started, the first byte of the message must be written to the transmit buffer. When a section of code decides to transmit a message serially, it loads the global message pointer with the address of the first byte of the message, enables the transmit interrupt, and calls the TxD interrupt procedure. Calling the TxD interrupt procedure writes the first byte to the transmit buffer to initiate transmit interrupts. This can be done by calling PL/M's built-in procedure CAUSE$INTERRUPT.

The transmit interrupt routine checks each byte before it writes it to the transmit buffer. The last character in each message is a 0, so if the character fetched is 0, the transmit interrupt is disabled and the character is ignored.

USING THE LPM WITH THE INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM, SERIES II OR SERIES III

A special driver program was written for the MDS to communicate to the LPM. This program, called WRITE, reads a specified file from the disk, expands any TAB characters, and transmits the data through Serial Channel 2 to the LPM. Serial Channel 2 was chosen because $\overline{CTS}$ and $\overline{RTS}$ are brought out to the RS-232 connector. The WRITE program is listed in appendix B. It was also necessary to modify the boot ROM of the development system so that Serial Channel 2 initializes with $\overline{RTS}$ false and a bit rate of 9600 bps.

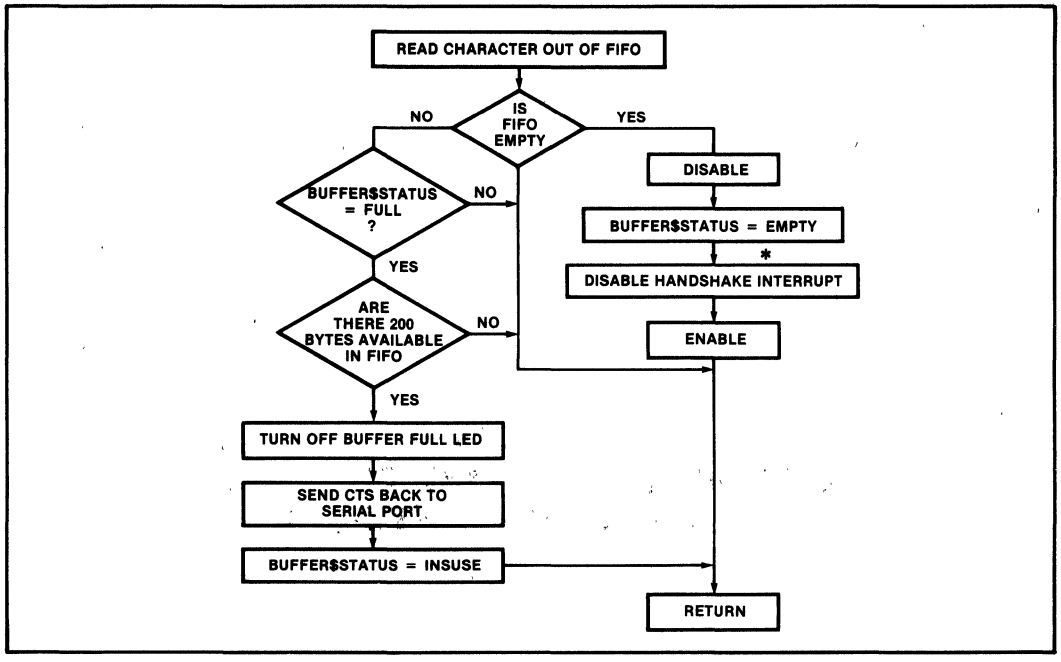


Figure 31. Flow Chart of the BUFF$OUT Procedure

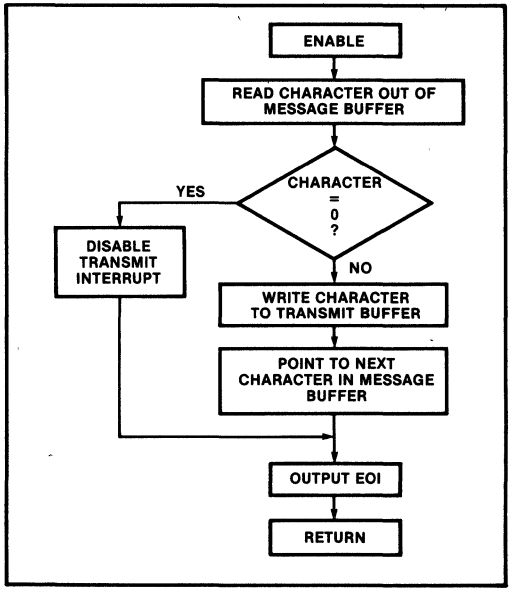


Figure 32. Flow Chart for TxD Interrupt Procedure

APPENDIX A
LISTING OF THE LINE PRINTER MULTIPLEXER SOFTWARE

```
PL/M-86 COMPILER     MAINMOD

SERIES-III PL/M-86 V1 0 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN  F1 MAIN OBJ
COMPILER INVOKED BY   PLM86 86  F1 MAIN SRC


                /*************************************************************************
                 *                                                                       *
                 *            MAIN MODULE FOR THE LINE PRINTER MULTIPLEXER               *
                 *                                                                       *
                 *************************************************************************/

                $DEBUG
  1             MAIN$MOD DO;

                /*************************************************************************
                * PORT 1 BIT CONFIGURATION                                               *
                *                                                                        *
                *  BUFFER FULL   CTS      ADDRESS    RTS   TWO WIRE HANDSHAKE           *
                *       B7        B6     B5 B4 B3    B2        B1 B0                     *
                *************************************************************************/

  2   1         DECLARE LITERALLY        'LITERALLY',
                        TRUE             LIT          'OFFH',
                        FALSE            LIT          '0',
                        FOREVER          LIT          'WHILE 1',

                        CMD$1            LIT          '0',     /*8256 REGISTERS*/
                        CMD$2            LIT          '2',
                        CMD$3            LIT          '4',
                        MODE             LIT          '6',
                        PORT$1$CTRL      LIT          '8',
                        SET$INT          LIT          '0AH',
                        INT$EN           LIT          '0AH',
                        RST$INT          LIT          '0CH',
                        INT$ADDR         LIT          '0CH',
                        TX$BUFF          LIT          '0EH',
                        RX$BUFF          LIT          '0EH',
                        PORT$1           LIT          '10H',
                        PORT$2           LIT          '12H',
                        DEBOUNCE$TIMER   LIT          '14H',
                        SCAN$TIMER       LIT          '1AH',
                        RECEIVE$TIMER    LIT          '1CH',
                        STATUS$REG       LIT          '1EH',

                        SCAN$INT         LIT          '40H',
                        DEBOUNCE$INT     LIT          '01H',
                        RECEIVER$INT     LIT          '10H',
                        TIME$OUT$INT     LIT          '08H',
                        TRANSMIT$INT     LIT          '20H',

                        EMPTY            LIT          '0',
                        INUSE            LIT          '1',
                        FULL             LIT          '2',

                        RTS              LIT          '(INPUT(PORT$1) AND 04H)',
```

```
PL/M-86 COMPILER    MAINMOD

                    BEGIN           LABEL           PUBLIC,

                    TEMP            BYTE
                    SCAN$DELAY      BYTE            PUBLIC,
                    DEBOUNCE$DELAY  BYTE            PUBLIC,
                    RECEIVE$DELAY   BYTE            PUBLIC,
                    PORT$PTR        BYTE            PUBLIC,
                    SERIAL$FORMAT(8)BYTE            PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */

                    MESSAGE$PTR     POINTER         EXTERNAL,
                    J               BYTE            EXTERNAL,
                    OK(1)           BYTE            EXTERNAL,
                    BUFFER$STATUS   BYTE            EXTERNAL,

            /************************************************************************
             *                  EXTERNAL PROCEDURE DECLARATIONS                     *
             ************************************************************************/

  3   1     POWER$ON PROCEDURE EXTERNAL;
  4   2     END POWER$ON,

  5   1     LOAD$INT$TABLE. PROCEDURE EXTERNAL,
  6   2     END LOAD$INT$TABLE;

            /************************************************************************
             *      SET THE BIT RATE AND DATA FORMAT FOR THE SERIAL PORT            *
             ************************************************************************/

  7   1     CONFIGURE: PROCEDURE ; /*Initialize bit rate and data format*/
  8   2     TEMP=SERIAL$FORMAT(SHR(PORT$PTR,3));
  9   2     OUTPUT(CMD$1)=((SHL(TEMP,2) AND 0C0H) OR 03H),
 10   2     OUTPUT(CMD$2)=(TEMP OR 30H);
 11   2     END CONFIGURE,

            /************************************************************************
             *                INITIALIZE SERIAL RECEIVER                            *
             ************************************************************************/

 12   1     INIT$RECEIVER   PROCEDURE,
 13   2     CALL CONFIGURE,                 /*Initialize 8256 serial port*/
 14   2     RECEIVE$DELAY=TRUE;
 15   2     OUTPUT(CMD$3)=0C0H,             /*Enable serial receiver*/
 16   2     OUTPUT(RECEIVE$TIMER)=70,       /*18 second TIME$OUT*/
 17   2     OUTPUT(SET$INT)=18H,            /*Enable RECEIVER and TIME$OUT interrupts*/
 18   2     IF (BUFFER$STATUS<>FULL)
                  THEN
 19   2         OUTPUT(PORT$1)=(INPUT(PORT$1) AND 0BFH), /*Send CTS TRUE*/

 20   2       DO WHILE RECEIVE$DELAY=TRUE, /* Wait here while receiving serial data */
 21   3       END,

                   /* After 18 seconds of not receiving a character, proceed */

 22   2     OUTPUT(SET$INT)=TRANSMIT$INT,     /* Send the terminating message */
 23   2     J=0,
 24   2     MESSAGE$PTR= @OK(0);
 25   2     CAUSE$INTERRUPT (45H),
```

```
PL/M-86 COMPILER    MAINMOD


  26   2      OUTPUT(PORT$1)=(INPUT(PORT$1) OR 40H), /*Send CTS FALSE*/
  27   2      OUTPUT(RST$INT)=18H,              /*Clear RECEIVER and TIMER Interrupts*/
  28   2      OUTPUT(CMD$3)=40H,                /*Disable serial receiver*/
  29   2      END INIT$RECEIVER,

              /*************************************************************
               *                        DEBOUNCE RTS                       *
               *************************************************************/

  30   1      DEBOUNCE PROCEDURE,
  31   2      DEBOUNCE$DELAY=TRUE,
  32   2      OUTPUT(DEBOUNCE$TIMER)=10, /* 10 msec debounce time delay */
  33   2      OUTPUT(SET$INT)=DEBOUNCE$INT,
  34   2          DO WHILE DEBOUNCE$DELAY=TRUE,
  35   3          END,
  36   2      IF RTS=0 THEN CALL INIT$RECEIVER,
  38   2      END DEBOUNCE,

              /*************************************************************
               *                      BEGIN MAIN PROGRAM                   *
               *************************************************************/


  39   1      BEGIN CALL POWER$ON,

  40   1      CALL LOAD$INT$TABLE,

  41   1      ENABLE,

  42   1      DO FOREVER,

  43   2      SCAN$DELAY=TRUE,
  44   2      OUTPUT(SCAN$TIMER)=100,     /*Spend 100 msec on each serial port sampling RTS*/
  45   2      OUTPUT(SET$INT)=SCAN$INT,

  46   2          DO WHILE SCAN$DELAY=TRUE,    /*Sample RTS*/
  47   3            IF RTS=0
                       THEN
  48   3              CALL DEBOUNCE,
  49   3          END,


  50   2      TEMP=INPUT(PORT$1),               /*Increment PORT$PTR*/
  51   2      PORT$PTR=TEMP AND 38H,
  52   2      TEMP=TEMP AND (NOT 38H),
  53   2      PORT$PTR=(PORT$PTR+8) AND 38H,

  54   2      OUTPUT(PORT$1)=TEMP OR PORT$PTR, /*Look at next serial port*/
  55   2      END, /*DO FOREVER*/
  56   1      END MAIN$MOD,


MODULE INFORMATION

     CODE AREA SIZE     = 011CH    284D


PL/M-86 COMPILER    MAINMOD

     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 000DH     13D
     MAXIMUM STACK SIZE = 000CH     12D
     159 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     INTMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE INTMOD
OBJECT MODULE PLACED IN  F1 INT OBJ
COMPILER INVOKED BY   PLM86 86  F1 INT SRC


                /*****************************************************************
                 *                                                               *
                 *        INTERRUPT MODULE  CONTAINS ALL INTERRUPT ROUTINES      *
                 *                          PLUS LOAD INTERRUPT TABLE PROCEDURE  *
                 *                                                               *
                 *****************************************************************/

                $DEBUG
  1             INT$MOD DO;
                $NOLIST

  3   1         DECLARE
                        ESC             LIT             '1BH',
                        SCAN$DELAY      BYTE            EXTERNAL,
                        DEBOUNCE$DELAY  BYTE            EXTERNAL,
                        RECEIVE$DELAY   BYTE            EXTERNAL,
                        MESSAGE$PTR     POINTER         EXTERNAL,
                        J               BYTE            EXTERNAL;

                /*****************************************************************
                 *                MESSAGES SENT TO SERIAL PORTS                  *
                 *****************************************************************/

                        OK (*) BYTE PUBLIC DATA ('TRANSMISSION COMPLETE',0AH,0DH,00),
                        BREAK (*) BYTE PUBLIC DATA ('BREAK DETECT ERROR',0AH,0DH,00),
                        PARITY (*)BYTE PUBLIC DATA ('PARITY ERROR DETECTED',0AH,0DH,00),
                        FRAME (*) BYTE PUBLIC DATA ('FRAMING ERROR DETECTED',0AH,0DH,00),
                        OVER$RUN(*)BYTE PUBLIC DATA('OVER RUN ERROR DETECTED',0AH,0DH,00);

                /*****************************************************************
                 *       EXTERNAL PROCEDURES CALLED BY THE INTERRUPT ROUTINES    *
                 *****************************************************************/
  4   1         ERROR PROCEDURE (STATUS) EXTERNAL;
  5   2         DECLARE STATUS  BYTE;
  6   2         END ERROR;

  7   1         PROGRAM.PROCEDURE EXTERNAL;
  8   2         END PROGRAM;

  9   1         BUFF$IN PROCEDURE (CHAR) EXTERNAL;
 10   2         DECLARE CHAR    BYTE;
 11   2         END BUFF$IN;

 12   1         BUFF$OUT PROCEDURE BYTE EXTERNAL;
 13   2         END BUFF$OUT;

                /*****************************************************************
                 *                  LOAD THE INTERRUPT TABLE                     *
                 *****************************************************************/

 14   1         LOAD$INT$TABLE PROCEDURE PUBLIC;
```

PL/M-86 COMPILER INTMOD 12/09/82

```
 15   2      CALL SET$INTERRUPT (40H, DEBOUNCE$TIME),
 16   2      CALL SET$INTERRUPT (43H, RECEIVE$TIME),
 17   2      CALL SET$INTERRUPT (44H, RXD),
 18   2      CALL SET$INTERRUPT (45H, TXD),
 19   2      CALL SET$INTERRUPT (46H, SCAN$TIME),
 20   2      CALL SET$INTERRUPT (47H, LP),

 21   2      END LOAD$INT$TABLE,

             /*************************************************************
              *                    INTERRUPT ROUTINES                     *
              *************************************************************/


             /*************************************************************
              *               SET SCAN DELAY FLAG FALSE                   *
              *************************************************************/

 22   1      SCAN$TIME PROCEDURE INTERRUPT 46H,

 23   2      ENABLE,
 24   2      SCAN$DELAY=FALSE,
 25   2      OUTPUT(CMD$3)=88H;              /*Output end for nested mode*/
 26   2      END SCAN$TIME,

             /*************************************************************
              *               SET DEBOUNCE DELAY FLAG FALSE               *
              *************************************************************/

 27   1      DEBOUNCE$TIME. PROCEDURE INTERRUPT 40H,
 28   2      DEBOUNCE$DELAY=FALSE,
 29   2      OUTPUT(CMD$3)=88H,
 30   2      END DEBOUNCE$TIME,

             /*************************************************************
              *               SET RECEIVE DELAY FLAG FALSE                *
              *************************************************************/

 31   1      RECEIVE$TIME PROCEDURE INTERRUPT 43H,
 32   2      ENABLE;
 33   2      RECEIVE$DELAY=FALSE,
 34   2      OUTPUT(CMD$3)=88H,
 35   2      END RECEIVE$TIME,

             /*************************************************************
              *                READ SERIAL RECEIVE BUFFER                 *
              *************************************************************/

 36   1      RXD PROCEDURE INTERRUPT 44H,

 37   2      DECLARE
                 STATUS          BYTE,
                 CHAR            BYTE,
 38   2      CHAR=INPUT(RX$BUFF),
 39   2      OUTPUT(RECEIVE$TIMER)=70,          /* REINITIALIZE RECEIVE TIME OUT */
 40   2      STATUS=INPUT(STATUS$REG) AND 0FH,
```

```
PL/M-86 COMPILER     INTMOD


  41    2           IF STATUS <> 0
                        THEN
  42    2             CALL ERROR (STATUS),

  43    2           ELSE IF CHAR=ESC
                           THEN
  44    2             CALL PROGRAM,
                    ELSE
  45    2              CALL BUFF$IN (CHAR),
  46    2       OUTPUT(CMD$3)=88H,
  47    2       END RXD,

               /*************************************************************************
                *                    SEND A BYTE TO THE LINE PRINTER                    *
                *************************************************************************/

  48    1       LP: PROCEDURE INTERRUPT 47H,
  49    2       ENABLE,
  50    2       OUTPUT(PORT$2)=BUFF$OUT,
  51    2       OUTPUT(CMD$3)=88H;
  52    2       END LP;

               /*************************************************************************
                *                    SEND A BYTE TO THE SERIAL PORTS                    *
                *************************************************************************/

  53    1       TXD PROCEDURE INTERRUPT 45H,
  54    2       DECLARE
                       MESSAGE BASED MESSAGE$PTR (1) BYTE,
                       I       BYTE,
  55    2       ENABLE,
  56    2       I=MESSAGE(J),
  57    2           IF I<>0
                       THEN OUTPUT(TX$BUFF)=I,
  59    2           ELSE OUTPUT(RST$INT)=TRANSMIT$INT,
  60    2           J=J+1,
  61    2           OUTPUT(CMD$3)=88H,
  62    2       END TXD,

  63    1       END INT$MOD,


MODULE INFORMATION

     CODE AREA SIZE     = 01BDH    445D
     CONSTANT AREA SIZE = 0078H    120D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0022H     34D
     181 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER    BUFFMOD

SERIES-III PL/M-86 V1 0 COMPILATION OF MODULE BUFFMOD
OBJECT MODULE PLACED IN  F1 BUFF OBJ
COMPILER INVOKED BY   PLM86 86 F1 BUFF SRC


                /*****************************************************************
                 *                                                               *
                 *    BUFFER MODULE  INSERTS AND REMOVES CHARACTERS FROM FIFO     *
                 *                   REPORTS SERIAL RECEIVE ERRORS AND           *
                 *                   RE-PROGRAMS SERIAL PORTS                    *
                 *                                                               *
                 *****************************************************************/

                $DEBUG
  1             BUFF$MOD DO,
                $NOLIST

  3   1         DECLARE
                        MESSAGE$PTR     POINTER         PUBLIC,
                        J               BYTE            PUBLIC,
                        OK(1)           BYTE            EXTERNAL,
                        BREAK(1)        BYTE            EXTERNAL,
                        PARITY(1)       BYTE            EXTERNAL,
                        FRAME(1)        BYTE            EXTERNAL,
                        OVER$RUN(1)     BYTE            EXTERNAL,
                        SERIAL$FORMAT(1)BYTE            EXTERNAL,
                        PORT$PTR        BYTE            EXTERNAL,

                        FIFO(8192)      BYTE,
                        IN$POINTER      WORD            PUBLIC,
                        OUT$POINTER     WORD            PUBLIC,
                        BUFFER$STATUS   BYTE            PUBLIC,

                /*****************************************************************
                 *                    INSERT CHARACTER INTO FIFO                 *
                 *****************************************************************/

  4   1         BUFF$IN PROCEDURE (CHAR) PUBLIC,
  5   2         DECLARE
                        CHAR    BYTE,

  6   2         IF ((BUFFER$STATUS=EMPTY) AND ((CHAR=LF) OR (CHAR=CR)))
                    THEN
  7   2                 DO;
  8   3                   OUTPUT(SET$INT)=HANDSHAKE$INT, /* Enable two-wire handshake interrupt */
  9   3                   BUFFER$STATUS=INUSE;
 10   3                   OUTPUT(PORT$2)=0,              /* Output NULL character to get
                                                               the interrupt started */
 11   3                 END,

 12   2         FIFO(IN$POINTER)=CHAR,   /* Put CHAR into FIFO and increment pointer */
 13   2         IN$POINTER=((INPOINTER+1) AND 1FFFH),

 14   2         IF (((IN$POINTER+4) AND 1FFFH)=OUT$POINTER) /* If the buffer is full, stop reception *
                    THEN
 15   2                 DO,  /* Send CTS FALSE, and light up buffer full LED */
```

```
PL/M-86 COMPILER     BUFFMOD

 16   3                OUTPUT(PORT$1)=((INPUT(PORT$1) OR 40H) AND 7FH),
 17   3                BUFFER$STATUS=FULL,
 18   3              END,
 19   2       END BUFF$IN;

              /************************************************************************
               *                      REMOVE CHARACTER FROM FIFO                      *
               ************************************************************************/

 20   1       BUFF$OUT PROCEDURE BYTE PUBLIC,
 21   2       DECLARE  CHAR   BYTE;
 22   2       CHAR=FIFO(OUT$POINTER),
 23   2       OUT$POINTER=((OUT$POINTER+1) AND 1FFFH),
 24   2       IF OUT$POINTER=IN$POINTER  /* If the buffer is EMPTY disable the output to LP */
                  THEN
 25   2                DO,
 26   3                  DISABLE,
 27   3                  BUFFER$STATUS=EMPTY,
 28   3                  OUTPUT(RST$INT)=HANDSHAKE$INT,
 29   3                  ENABLE,
 30   3                END,

              /*  If the buffer is ready to fill up again then send CTS TRUE */

 31   2       ELSE  IF ((BUFFER$STATUS=FULL) AND (((OUT$POINTER-200) AND 1FFFH)=IN$POINTER))
                       THEN
 32   2                   DO, /* Turn off buffer-full LED and turn on CTS */
 33   3                     OUTPUT(PORT$1)=((INPUT(PORT$1) AND 0BFH) OR 80H),
 34   3                     BUFFER$STATUS=INUSE,
 35   3                   END,
              RETURN CHAR,

 37   2       END BUFF$OUT,

              /************************************************************************
               *                    SEND ERROR MESSAGE TO SERIAL PORT                 *
               ************************************************************************/

 38   1       ERROR PROCEDURE (STATUS) PUBLIC,
 39   2               DECLARE STATUS   BYTE,
                         MESSAGE  BASED  MESSAGE$PTR(1) BYTE,

 40   2               IF (STATUS AND 02H)<>0
                        THEN
 41   2                     STATUS=2,
 42   2       ELSE    IF (STATUS AND 04H)<>0
                        THEN
 43   2                     STATUS=3,
 44   2       ELSE    IF (STATUS AND 08H)<>0
                        THEN
 45   2                     STATUS=4,
 46   2       ELSE    IF (STATUS AND 01H)<>0
                        THEN
 47   2                     STATUS=1,
                     DO CASE STATUS,
 49   3
 50   3                      MESSAGE$PTR=@FRAME(0),
```

```
PL/M-86 COMPILER    BUFFMOD

  51   3                  MESSAGE$PTR=@OVER$RUN(0),
  52   3                  MESSAGE$PTR=@PARITY(0),
  53   3                  MESSAGE$PTR=@BREAK(0),
  54   3              END,

  55   2              J=1,          /* Point to second character in string */
  56   2              OUTPUT(SET$INT)=TRANSMIT$INT,
  57   2              OUTPUT(TX$BUFF)=MESSAGE(0),
  58   2      END ERROR,

            /*************************************************************
             *               RELOAD SERIAL PORT CONFIGURE BYTE           *
             *************************************************************/

  59   1      PROGRAM PROCEDURE PUBLIC,
  60   2              DECLARE TEMP    BYTE,
                               CHAR    BYTE,

  61   2              DO WHILE (INPUT(STATUS$REG) AND 40H)=0, /* Wait for next byte */
  62   3              END,

  63   2              CHAR=INPUT(RX$BUFF);

  64   2              IF CHAR=0       /* If second byte is 0, exit program mode */
                         THEN
  65   2                     DO,
  66   3                         OUTPUT(RECEIVE$TIMER)=70;
  67   3                         CALL BUFF$IN (CHAR),
  68   3                         RETURN,
  69   3                     END,

  70   2              TEMP=(CHAR AND 0FH),

  71   2              DO WHILE (INPUT(STATUS$REG) AND 40H)=0,
  72   3              END,

  73   2              TEMP=(INPUT(RXBUFF) AND 0FH) OR SHL(TEMP,4),

  74   2              SERIAL$FORMAT (SHR(PORT$PTR,3))=TEMP,
  75   2      END PROGRAM,


  76   1      END BUFF$MOD,


MODULE INFORMATION

     CODE AREA SIZE     = 01E4H    484D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 200BH   8203D
     MAXIMUM STACK SIZE = 000AH     10D
     189 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     PON_MOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE PON_MOD
OBJECT MODULE PLACED IN  F1 PON OBJ
COMPILER INVOKED BY   PLM86 86  F1 PON SRC


                $DEBUG

                /*****************************************************************************
                 *                                                                           *
                 *       POWER ON INITIALIZATION OF THE LINE PRINTER MULTIPLEXER             *
                 *                                                                           *
                 *****************************************************************************/

   1            PON_MOD  DO;

                $NOLIST


   3    1       DECLARE BUFFER$STATUS    BYTE     EXTERNAL,
                        IN$POINTER       WORD     EXTERNAL,
                        OUT$POINTER      WORD     EXTERNAL,
                        PORT$PTR         BYTE     EXTERNAL,
                        SERIAL$FORMAT(8)BYTE      EXTERNAL;


   4    1       POWER$ON  PROCEDURE PUBLIC;

   5    2       DECLARE I                BYTE;


   6    2       DISABLE;

                                /*  INITIALIZE THE MUART */

   7    2       OUTPUT(CMD$1)=01000011B;          /*8086 MODE, FREQ=1KHz, 1 STOP BIT, &
                                                     7 BITS/CHARACTER*/
   8    2       OUTPUT(CMD$2)=10110100B;          /*ODD PARITY, SYSTEM CLOCK=1.024 MHz, &
                                                     9600 bps*/
   9    2       OUTPUT(CMD$3)=01111111B;          /*CLEAR CMD$3 REGISTER*/
  10    2       OUTPUT(CMD$3)=10110001B;          /*RESET, INTERRUPT ACKNOWLEDGE ENABLE, &
                                                     NESTED INTERRUPT MODE*/
  11    2       OUTPUT(MODE)=10000101B;           /*CASCADE TIMERS 3&5 FOR THE
                                                     RECEIVE$TIME$OUT TIMER, BYTE OUTPUT MODE*/

  12    2       OUTPUT(PORT$1$CTRL)=11111000B;    /*PORT 1  RTS=INPUT, THE REST ARE OUTPUTS*/

  13    2       OUTPUT(PORT$1)=11000000B;         /*POINT TO THE FIRST PORT, CTS IS FALSE,
                                                     AND BUFFER IS NOT FULL*/

                     /* INITIALIZE FLAGS, VARIABLES, AND ARRAYS */

  14    2       BUFFER$STATUS=EMPTY;
  15    2       IN$POINTER=0;   OUT$POINTER=0;
  17    2       PORT$PTR=0;

  18    2               DO I=0 TO 7;
```

```
PL/M-86 COMPILER     PON_MOD


  19   3               SERIAL$FORMAT(I)=10010100B,      /* ON POWER-UP ALL EIGHT SERIAL PORTS
                                                         DEFAULT TO 9600 bps, ODD PARITY, AND
                                                         7 BITS/CHARACTER*/
  20   3               END,
  21   2         END POWER$ON,
  22   1         END PON_MOD,


MODULE INFORMATION.

     CODE AREA SIZE      = 0058H      88D
     CONSTANT AREA SIZE  = 0000H       0D
     VARIABLE AREA SIZE  = 0001H       1D
     MAXIMUM STACK SIZE  = 0002H       2D
     98 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

APPENDIX B
LISTING OF THE WRITE PROGRAM

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE WRITEMOD
OBJECT MODULE PLACED IN  F1 WRITE OBJ
COMPILER INVOKED BY    F2 PLM80  F1 WRITE SRC


                $DEBUG
   1            WRITE$MOD DO;

                /*********************************************************************
                 *                                                                   *
                 *    WRITE PROGRAM  READS A FILE FROM A DISK AND COPIES             *
                 *                   IT TO SERIAL CHANNEL 2 ON THE MDS               *
                 *                                                                   *
                 *    SYNTAX OF WRITE  WRITE .DEVICE.NAME EXTENSION                  *
                 *                                                                   *
                 *********************************************************************/

   2    1       DECLARE  LIT             LITERALLY       'LITERALLY',
                         USART$DATA       LIT             '0F6H',
                         USART$STATUS     LIT             '0F7H',
                         RTS              LIT             '20H',
                         TXEN             LIT             '01H',
                         RXE              LIT             '04H',
                         CR               LIT             '0DH',
                         LF               LIT             '0AH',
                         TAB              LIT             '09H',
                         SP               LIT             '20H',
                         ESC              LIT             '1BH',
                         FORM$FEED        LIT             '0CH';

   3    1       DECLARE  AFT$IN           ADDRESS,
                         FILENAME(15)     BYTE,
                         STATUS           ADDRESS,
                         BUFFER(32000)    BYTE,
                         ACTUAL           ADDRESS,
                         CHAR$COUNT       ADDRESS,
                         BYE(42)          BYTE    INITIAL
                ('WROTE ',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,' TO THE LINE PRINTER',0AH,0DH),
                         I                ADDRESS,
                         J                BYTE;

                /*********************************************************************
                 *                    EXTERNAL SYSTEM LIB PROCEDURES                 *
                 *********************************************************************/

   4    1       OPEN
                        PROCEDURE (AFTNPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL,
   5    2                DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS,
   6    2       END OPEN;

   7    1       READ
                        PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL,
   8    2                DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
   9    2       END READ;

  10    1       WRITE.
```

PL/M-80 COMPILER

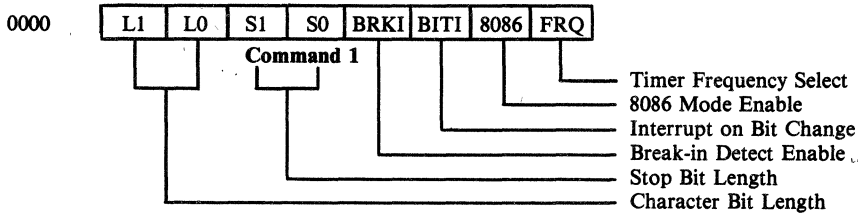
```
                        PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL,
 11    2                DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS,
 12    2        END WRITE;

 13    1        CLOSE:
                        PROCEDURE (AFTN, STATUS) EXTERNAL;
 14    2                DECLARE (AFTN, STATUS) ADDRESS,
 15    2        END CLOSE,

 16    1        ERROR
                        PROCEDURE (ERRNUM) EXTERNAL,
 17    2                DECLARE (ERRNUM) ADDRESS,
 18    2        END ERROR;

 19    1        EXIT:
                        PROCEDURE EXTERNAL;
 20    2        END EXIT,

                /************************************************************************
                 *               WAIT UNTIL USART TRANSMITTER IS READY                  *
                 ************************************************************************/

 21    1        TXRDY:
                        PROCEDURE;
 22    2                DO WHILE ( (INPUT(USART$STATUS) AND 01H) = 0 ),
 23    3                END;
 24    2        END TXRDY;

                /************************************************************************
                 *                          BEGIN MAIN PROGRAM                          *
                 ************************************************************************/
 25    1        BEGIN.
                        STATUS=0;

 26    1                CALL READ(1, FILENAME, 15, ACTUAL, STATUS),   /* Read in file and path name */

 27    1                IF STATUS <> 0
                             THEN
 28    1                           GO TO DONE;

 29    1                CALL OPEN( AFT$IN, .FILENAME, 1, 0, STATUS), /* Open up the file */

 30    1                IF STATUS <> 0
                             THEN
 31    1                           GO TO DONE;

 32    1        REPEAT:
                        CALL READ(AFT$IN, .BUFFER, 32000, .ACTUAL, STATUS),

 33    1                IF STATUS <> 0
                             THEN
 34    1                           GO TO DONE,

 35    1                CHAR$COUNT=0,   /* CHAR$COUNT keeps track of the tab columns in each line */

 36    1                OUTPUT(USART$STATUS)= RTS OR TXEN,
```

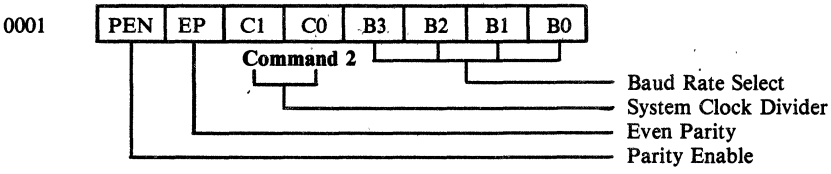
```
PL/M-80 COMPILER

  37   1          IF BUFFER(0)=FORM$FEED   /* If the first character is a form feed
                                              remove it  Form feeds are inserted at the
                                              end of a file */
                   THEN
  38   1                DO;
  39   2                   BUFFER(0)=00H;
  40   2                   CHAR$COUNT=-1;
  41   2                END;

  42   1          DO I = 0 TO (ACTUAL - 1);

  43   2                IF  (BUFFER(I)=TAB)  /* Replace TAB characters with the
                                                appropriate number of spaces */
                            THEN
  44   2                   DO;
  45   3                         CALL TXRDY;
  46   3                         OUTPUT(USART$DATA)=SP;
  47   3                         CHAR$COUNT=CHAR$COUNT+1;

  48   3                         DO WHILE ((CHAR$COUNT AND 0007H)<>0);
  49   4                              CALL TXRDY;
  50   4                              OUTPUT(USART$DATA)=SP;
  51   4                              CHAR$COUNT=CHAR$COUNT+1;
  52   4                         END;
  53   3                    END;
                       ELSE
  54   2                      IF BUFFER(I)=ESC /* If outputting ESC, then output a
                                                  0 next so the LPM does not get
                                                      re-programmed */
                               THEN
  55   2                          DO J=0 TO 1;
  56   3                             CALL TXRDY;
  57   3                             OUTPUT(USART$DATA)=0;
  58   3                          END;
                       ELSE /* If the character is not an ESC or TAB then output it */
  59   2                   DO;
  60   3                      CALL TXRDY;
  61   3                      OUTPUT(USART$DATA)=BUFFER(I);
  62   3                      IF (BUFFER(I)>1FH AND BUFFER(I)<7FH)

                                 THEN                    /* Only increment CHAR$COUNT
                                                            for printable characters */
  63   3                         CHAR$COUNT=CHAR$COUNT+1;

  64   3                      IF ( (BUFFER(I)=CR) OR (BUFFER(I)=LF) )
                                 THEN                 /* Reset CHAR$COUNT for CR or LF */
  65   3                         CHAR$COUNT=0;
  66   3                   END;
  67   2          END;

  68   1          IF ACTUAL = 32000 /*If the file is more than 32K, get some more data */
                     THEN
  69   1                GO TO REPEAT;

  70   1          CALL TXRDY;              /* Terminate file with CR, LF, and FF */
  71   1          OUTPUT(USART$DATA)=CR;
  72   1          CALL TXRDY;
```

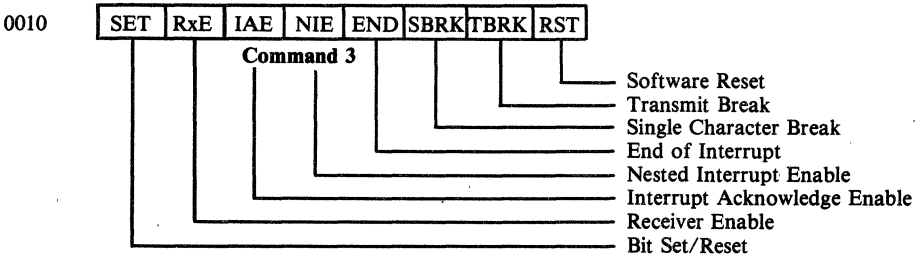
```
PL/M-80 COMPILER

  73   1                OUTPUT(USART$DATA)=LF;
  74   1                CALL TXRDY;
  75   1                OUTPUT(USART$DATA)=FORM$FEED;

  76   1                OUTPUT(USART$STATUS)=RXE OR TXEN; /* Shut off RTS */

  77   1                CALL CLOSE (AFT$IN, STATUS);

  78   1                DO I=0 TO 14;              /* Output sign off message */
  79   2                     IF FILENAME(I)=CR
                                THEN
  80   2                          GO TO SKIP;
  81   2                     BYE(I+5)=FILENAME(I);
  82   2                END;

  83   1           SKIP:  CALL WRITE(0,.BYE,42, STATUS);

  84   1                GO TO NEXT;

  85   1           DONE:
                        CALL ERROR(STATUS);

  86   1           NEXT:
                        CALL EXIT;

  87   1           END WRITE$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0209H    521D
     VARIABLE AREA SIZE = 7D44H  32068D
     MAXIMUM STACK SIZE = 0008H      8D
     191 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

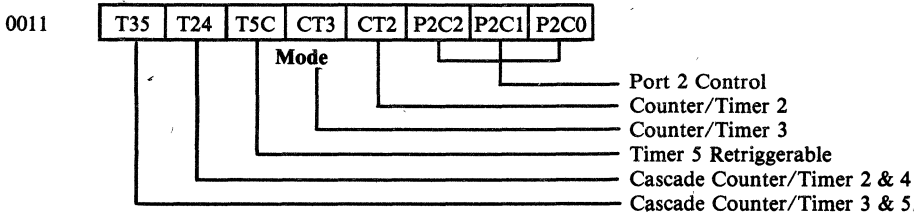
APPENDIX C
MUART REGISTERS

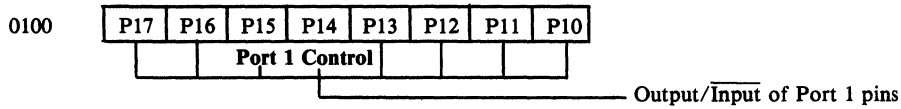
8085 Mode: AD3 AD2 AD1 AD0
8086 Mode: AD4 AD3 AD2 AD1

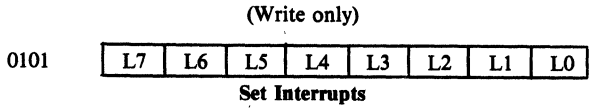


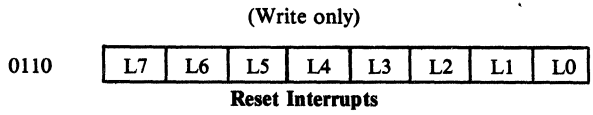


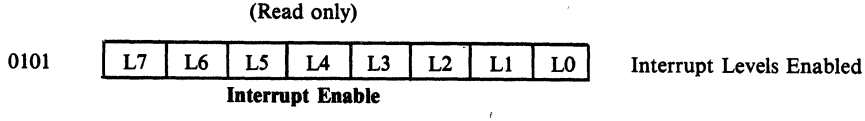


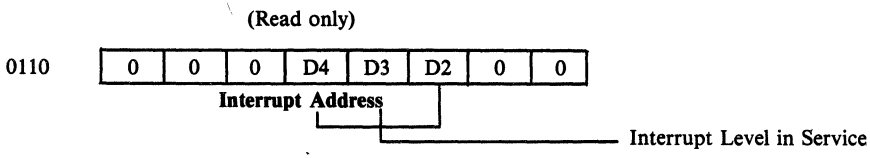


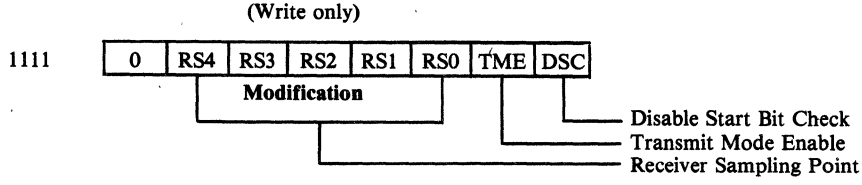

0100
P17 P16 P15 P14 P13 P12 P11 P10
Port 1 Control
Output/Input of Port 1 pins


(Write only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Set Interrupts
Enable

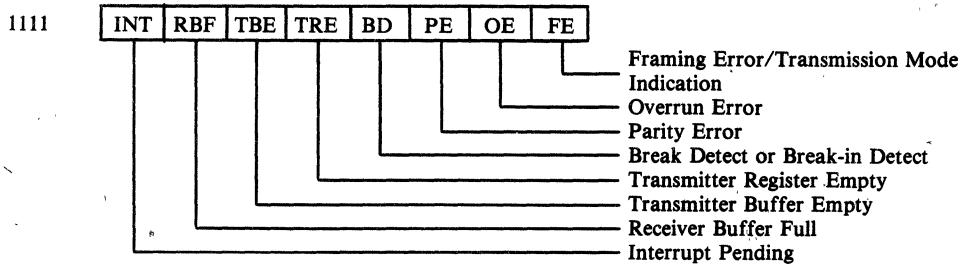

(Write only)
0110
L7 L6 L5 L4 L3 L2 L1 L0
Reset Interrupts
Disable


(Read only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Interrupt Enable
Interrupt Levels Enabled


(Read only)
0110
0 0 0 D4 D3 D2 0 0
Interrupt Address
Interrupt Level in Service

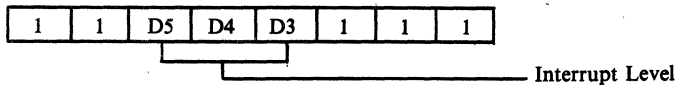

(Write only)
1111
0 RS4 RS3 RS2 RS1 RS0 TME DSC
Modification
Disable Start Bit Check
Transmit Mode Enable
Receiver Sampling Point

Status Register (Read only)

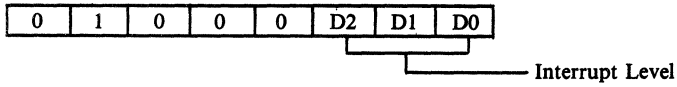


Response to INTA

8085-Mode (RST-instruction in response to $\overline{\text{INTA}}$)



8086-Mode (Interrupt Vector in response to second $\overline{\text{INTA}}$)



8231A
ARITHMETIC PROCESSING UNIT

- **Fixed Point Single and Double Precision (16/32 Bit)**
- **Floating Point Single Precision (32 Bit)**
- **Binary Data Formats**
- **Add, Subtract, Multiply and Divide**
- **Trigonometric and Inverse Trigonometric Functions**
- **Square Roots, Logarithms, Exponentiation**
- **Float to Fixed and Fixed to Float Conversions**
- **Stack Oriented Operand Storage**
- **Compatible with all Intel and most other Microprocessor Families**
- **Direct Memory Access or Programmed I/O Data Transfers**
- **End of Execution Signal**
- **General Purpose 8-Bit Data Bus Interface**
- **Standard 24 Pin Package**
- **+12 Volt and +5 Volt Power Supplies**
- **Advanced N-Channel Silicon Gate HMOS Technology**

The Intel® 8231A Arithmetic Processing Unit (APU) is a monolithic HMOS LSI device that provides high performance fixed and floating point arithmetic and floating point trigonometric operations. It may be used to enhance the mathematical capability of a wide variety of processor-oriented systems. Chebyshev polynomials are used in the implementation of the APU algorithms.

All transfers, including operand, result, status and command information, take place over an 8-bit bidirectional data bus. Operands are pushed onto an internal stack and commands are issued to perform operations on the data in the stack. Results are then available to be retrieved from the stack.

Transfers to and from the APU may be handled by the associated processor using conventional programmed I/O, or may be handled by a direct memory access controller for improved performance. Upon completion of each command, the APU issues an end of execution signal that may be used as an interrupt by the CPU to help coordinate program execution.

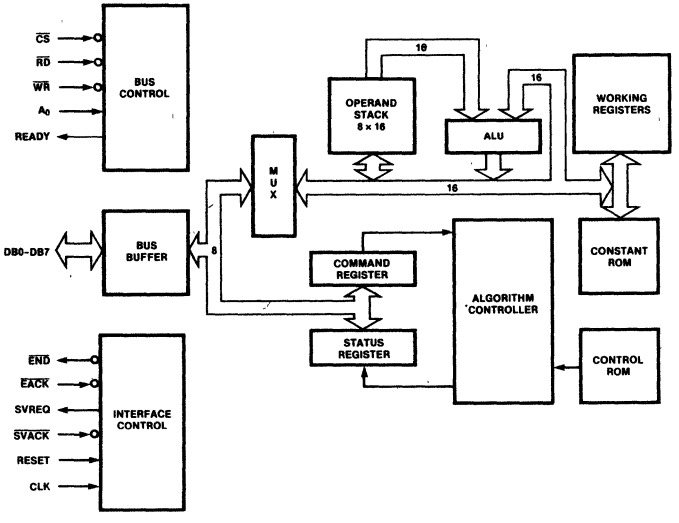


Figure 1. Block Diagram

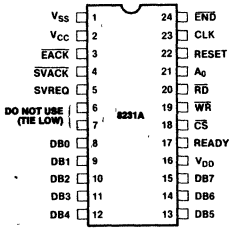


Figure 2. Pin Configuration

Table 1. Pin Description

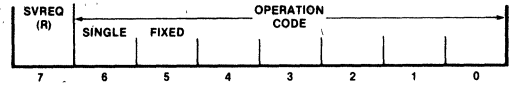
Symbol	Pin No.	Type	Name and Function
V_{CC}	2		**Power:** +5 Volt power supply.
V_{DD}	16		**Power:** +12 Volt power supply.
V_{SS}	1		**Ground.**
CLK	23	I	**Clock:** An external, TTL compatible, timing source is applied to the CLK pin.
RESET	22	I	**Reset:** The active high reset signal provides initialization for the chip. RESET also terminates any operation in progress. RESET clears the status register and places the 8231A into the idle state. Stack contents and command registers are not affected (5 clock cycles).
$\overline{CS}$	18	I	**Chip Select:** $\overline{CS}$ is an active low input signal which selects the 8231A and enables communication with the data bus.
A_0	21	I	**Address:** In conjunction with the $\overline{RD}$ and WR signals, the A_0 control line establishes the type of communication that is to be performed with the 8231A as shown below:

A_0	$\overline{RD}$	$\overline{WR}$	Function
0	1	0	Enter data byte into stack
0	0	1	Read data byte from stack
1	1	0	Enter command
1	0	1	Read status

$\overline{RD}$	20	I	**Read:** This active low input indicates that data or status is to be read from the 8231A if CS is low.
$\overline{WR}$	19	I	**Write:** This active low input indicates that data or a command is to be written into the 8231A if CS is low.
$\overline{EACK}$	3	I	**End of Execution:** This active low input clears the end of execution output signal ($\overline{END}$). If $\overline{EACK}$ is tied low, the $\overline{END}$ output will be a pulse that is one clock period wide.
$\overline{SVACK}$	4	I	**Service Request:** This active low input clears the service request output (SVREQ).
$\overline{END}$	24	O	**End:** This active low, open-drain output indicates that execution of the previously entered command is complete. It can be used as an interrupt request and is cleared by $\overline{EACK}$, RESET or any read or write access to the 8231.
SVREQ	5	O	**Service Request:** This active high output signal indicates that command execution is complete and that post execution service was requested in the previous command byte. It is cleared by $\overline{SVACK}$, the next command output to the device, or by RESET.
READY	17	O	**Ready:** This active high output indicates that the 8231A is able to accept communication with the data bus. When an attempt is made to read data, write data or to enter a new command while the 8231A is executing a command, READY goes low until execution of the current command is complete (See READY Operation, p. 5).
DB0-DB7	8-15	I/O	**Data Bus:** These eight bidirectional lines provide for transfer of commands, status and data between the 8231A and the CPU. The 8231A can drive the data bus only when CS and RD are low.

COMMAND STRUCTURE

Each command entered into the 8231A consists of a single 8-bit byte having the format illustrated below:



Bits 0-4 select the operation to be performed as shown in the table. Bits 5-6 select the data format appropriate to the selected operation. If bit 5 is a 1, a fixed point data format is specified. If bit 5 is a 0, floating point format is specified. Bit 6 selects the precision of the data to be operated upon by fixed point commands only (if bit 5 = 0, bit 6 must be 0). If bit 6 is a 1, single-precision (16-bit) operands are assumed. If bit 6 is a 0, double-precision (32-bit) operands are indicated. Results are undefined for all illegal combinations of bits in the command byte. Bit 7 indicates whether a service request is to be issued after the command is executed. If bit 7 is a 1, the service request output (SVREQ) will go high at the conclusion of the command and will remain high until reset by a low level on the service acknowledge pin ($\overline{SVACK}$) or until completion of execution of the succeeding command where service request (bit 7) is 0. Each command issued to the 8231A requests post execution service based upon the state of bit 7 in the command byte. When bit 7 is a 0, SVREQ remains low.

Table 2. 32-Bit Floating Point Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags[4] Affected
ACOS	Inverse Cosine of A	0 6	R U U U	S, Z, E
ASIN	Inverse Sine of A	0 5	R U U U	S, Z, E
ATAN	Inverse Tangent of A	0 7	R B U U	S, Z
CHSF	Sign Change of A	1 5	R B C D	S, Z
COS	Cosine of A (radians)	0 3	R B U U	S, Z
EXP	e^A Function	0 A	R B U U	S, Z, E
FADD	Add A and B	1 0	R C D U	S, Z, E
FDIV	Divide B by A	1 3	R C D U	S, Z, E
FLTD	32-Bit Integer to Floating Point Conversion	1 C	R B C U	S, Z
FLTS	16-Bit Integer to Floating Point Conversion	1 D	R B C U	S, Z
FMUL	Multiply A and B	1 2	R C D U	S, Z, E
FSUB	Subtract A from B	1 1	R C D U	S, Z, E
LOG	Common Logarithm (base 10) of A	0 8	R B U U	S, Z, E
LN	Natural Logarithm of A	0 9	R B U U	S, Z, E
POPF	Stack Pop	1 8	B C D A	S, Z
PTOF	Stack Push	1 7	A A B C	S, Z
PUPI	Push π onto Stack	1 A	R A B C	S, Z
PWR	B^A Power Function	0 B	R C U U	S, Z, E
SIN	Sine of A (radians)	0 2	R B U U	S, Z
SQRT	Square Root of A	0 1	R B C U	S, Z, E
TAN	Tangent of A (radians)	0 4	R B U U	S, Z, E
XCHF	Exchange A and B	1 9	B A C D	S, Z

Table 3. 32-Bit Integer Instructions

Instruction	Description	Hex[1] Code	Stack Contents[2] After Execution A B C D	Status Flags[4] Affected
CHSD	Sign Change of A	3 4	R B C D	S, Z, O
DADD	Add A and B	2 C	R C D A	S, Z, C, E
DDIV	Divide B by A	2 F	R C D U	S, Z, E
DMUL	Multiply A and B (R = lower 32-bits)	2 E	R C D U	S, Z, O
DMUU	Multiply A and B (R = upper 32-bits)	3 6	R C D U	S, Z, O
DSUB	Subtract A from B	2 D	R C D A	S, Z, C, O
FIXD	Floating Point to Integer Conversion	1 E	R B C U	S, Z, O
POPD	Stack Pop	3 8	B C D A	S, Z
PTOD	Stack Push	3 7	A A B C	S, Z
XCHD	Exchange A and B	3 9	B A C D	S, Z

Table 4. 16-Bit Integer Instructions

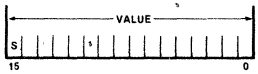
Instruction	Description	Hex[1] Code	Stack Contents[3] After Execution A_U A_L B_U B_L C_U C_L D_U D_L	Status Flags[4] Affected
CHSS	Change Sign of A_U	7 4	R A_L B_U B_L C_U C_L D_U D_L	S, Z, O
FIXS	Floating Point to Integer Conversion	1 F	R B_U B_L C_U C_L U U U	S, Z, O
POPS	Stack Pop	7 8	A_L B_U B_L C_U C_L D_U D_L A_U	S, Z
PTOS	Stack Push	7 7	A_U A_U A_L B_U B_L C_U C_L D_U	S, Z
SADD	Add A_U and A_L	6 C	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
SDIV	Divide A_L by A_U	6 F	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUL	Multiply A_L by A_U (R = lower 16-bits)	6 E	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SMUU	Multiply A_L by A_U (R = upper 16-bits)	7 6	R B_U B_L C_U C_L D_U D_L U	S, Z, E
SSUB	Subtract A_U from A_L	6 D	R B_U B_L C_U C_L D_U D_L A_U	S, Z, C, E
XCHS	Exchange A_U and A_L	7 9	A_L A_U B_U B_L C_U C_L D_U D_L	S, Z
NOP	No Operation	0 0	A_U A_L B_U B_L C_U C_L D_U D_L	

Notes: 1. In the hex code column, SVREQ is a 0.
2. The stack initially is composed of four 32-bit numbers (A, B, C, D). A is equivalent to Top Of Stack (TOS) and B is Next On Stack (NOS). Upon completion of a command the stack is composed of: the result (R); undefined (U), or the initial contents (A, B, C, or D).
3. The stack initially is composed of eight 16-bit numbers (A_U, A_L, B_U, B_L, C_U, C_L, D_U, D_L). A_U is the TOS and A_L is NOS. Upon completion of a command the stack is composed of: the result (R); undefined (U); or the initial contents (A_U, A_L, B_U, B_L, . . .).
4. Nomenclature: Sign (S); Zero (Z); Overflow (O); Carry (C); Error Code Field (E).

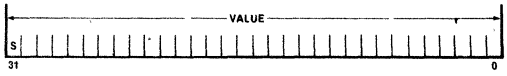
DATA FORMATS

The 8231A arithmetic processing unit handles operands in both fixed point and floating point formats. Fixed point operands may be represented in either single (16-bit operands) or double precision (32-bit operands), and are always represented as binary, two's complement values.

SINGLE PRECISION FIXED POINT FORMAT



DOUBLE PRECISION FIXED POINT FORMAT



The sign (positive or negative) of the operand is located in the most significant bit (MSB). Positive values are represented by a sign bit of zero (S = 0). Negative values are represented by the two's complement of the corresponding positive value with a sign bit equal to 1 (S = 1). The range of values that may be accommodated by each of these formats is $-32{,}768$ to $+32{,}767$ for single precision and $-2{,}147{,}483{,}648$ to $+2{,}147{,}483{,}647$ for double precision.

Floating point binary values are represented in a format that permits arithmetic to be performed in a fashion analogous to operations with decimal values expressed in scientific notation.

$$(5.83 \times 10^{2})\,(8.16 \times 10^{1}) = (4.75728 \times 10^{4})$$

In the decimal system, data may be expressed as values between 0 and 10 times 10 raised to a power that effectively shifts the implied decimal point right or left the number of places necessary to express the result in conventional form (e.g., 47,572.8). The value-portion of the data is called the mantissa. The exponent may be either negative or positive.

The concept of floating point notation has both a gain and a loss associated with it. The gain is the ability to represent the significant digits of data with values spanning a large dynamic range limited only by the capacity of the exponent field. For example, in decimal notation if the exponent field is two digits wide, and the mantissa is five digits, a range of values (positive or negative) from 1.0000×10^{-99} to $9.9999 \times 10^{+99}$ can be accommodated. The loss is that only the significant digits of the value can be represented. Thus there is no distinction in this representation between the values 123451 and 123452, for example, since each would be expressed as: 1.2345×10^{5}. The sixth digit has been discarded. In most applications where the dynamic range of values to be represented is large, the loss of significance, and hence accuracy of results, is a minor consideration. For greater precision a fixed point format could be chosen, although with a loss of potential dynamic range.

The 8231A is a binary arithmetic processor and requires that floating point data be represented by a fractional mantissa value between .5 and 1 multiplied by 2 raised to an appropriate power. This is expressed as follows:

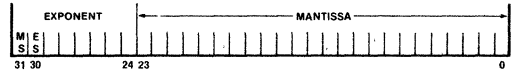
$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

For example, the value 100.5 expressed in this form is $0.1100\ 1001 \times 2^{7}$. The decimal equivalent of this value may be computed by summing the components (powers of two) of the mantissa and then multiplying by the exponent as shown below:

$$\begin{aligned}\text{value} &= (2^{-1} + 2^{-2} + 2^{-5} + 2^{-8}) \times 2^{7} \\ &= 0.5 + 0.25 + 0.03125 + 0.00290625) \times 128 \\ &= 0.78515625 \times 128 \\ &= 100.5\end{aligned}$$

FLOATING POINT FORMAT

The format for floating point values in the 8231A is given below. The mantissa is expressed as a 24-bit (fractional) value; the exponent is expressed as a two's complement 7-bit value having a range of -64 to $+63$. The most significant bit is the sign of the mantissa (0 = positive, 1 = negative), for a total of 32 bits. The binary point is assumed to be to the left of the most significant mantissa bit (bit 23). All floating point data values must be normalized. Bit 23 must be equal to 1, except for the value zero, which is represented by all zeros.

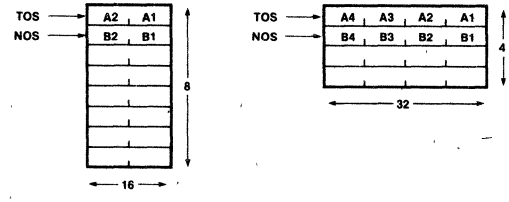


The range of values that can be represented in this format is $\pm(2.7 \times 10^{-20}$ to $9.2 \times 10^{18})$ and zero.

FUNCTIONAL DESCRIPTION

STACK CONTROL

The user interface to the 8231A includes access to an 8 level 16-bit wide data stack. Since single precision fixed point operands are 16-bits in length, eight such values may be maintained in the stack. When using double precision fixed point or floating point formats four values may be stored. The stack in these two configurations can be visualized as shown below:



Data are written onto the stack, eight bits at a time, in the order shown (A1, A2, A3, . . .). Data are removed from the stack in reverse byte order (A4, A3, A2 . . .). Data should be entered onto the stack in multiples of the number of bytes appropriate to the chosen data format.

DATA ENTRY

Data entry is accomplished by bringing the chip select ($\overline{CS}$), the command/data line (A_0), and $\overline{WR}$ low, as shown in the timing diagram. The entry of each new data word "pushes down" the previously entered data and places the new byte on the top of stack (TOS). Data on the bottom of the stack prior to a stack entry are lost.

DATA REMOVAL

Data are removed from the stack in the 8231A by bringing chip select ($\overline{CS}$), command/data (A_0), and $\overline{RD}$ low as shown in the timing diagram. The removal of each data word redefines TOS so that the next successive byte to be removed becomes TOS. Data removed from the stack rotates to the bottom of the stack.

COMMAND ENTRY

After the appropriate number of bytes of data have been entered onto the stack, a command may be issued to perform an operation on that data. Commands which require two operands for execution (e.g., add) operate on the TOS and NOS values. Single operand commands operate only on the TOS.

Commands are issued to the 8231A by bringing the chip select ($\overline{CS}$) line low, command data (A_0) line high, and $\overline{WR}$ line low as indicated by the timing diagram. After a command is issued, the CPU can continue execution of its program concurrently with the 8231A command execution.

COMMAND COMPLETION

The 8231A signals the completion of each command execution by lowering the End Execution line ($\overline{END}$). Simultaneously, the busy bit in the status register is cleared and the Service Request bit of the command register is checked. If it is a "1" the service request output level (SVREQ) is raised. $\overline{END}$ is cleared on receipt of an active low End Acknowledge ($\overline{EACK}$) pulse. Similarly, the service request line is cleared by recognition of an active low Service Acknowledge ($\overline{SVACK}$) pulse.

READY OPERATION

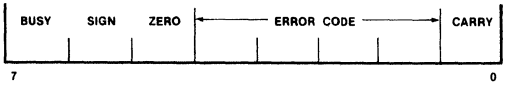
An active high ready (READY) is provided. This line is high in its quiescent state and is pulled low by the 8231A under the following conditions:

1. A previously initiated operation is in progress (device busy) and Command Entry has been attempted. In this case, the READY line will be pulled low and remain low until completion of the current command execution. It will then go high, permitting entry of the new command.
2. A previously initiated operation is in progress and stack access has been attempted. In this case, the READY line will be pulled low, will remain in that state until execution is complete, and will then be raised to permit completion of the stack access.
3. The 8231A is not busy, and data removal has been requested. READY will be pulled low for the length of time necessary to transfer the byte from the top of stack to the interface latch, and will then go high, indicating availability of the data.
4. The 8231A is not busy, and a data entry has been requested. READY will be pulled low for the length of time required to ascertain if the preceding data byte, if any, has been written to the stack. If so READY will immediately go high. If not, READY will remain low until the interface latch is free and will then go high.
5. When a status read has been requested, READY will be pulled low for the length of time necessary to transfer the status to the interface latch, and will then be raised to permit completion of the status read. Status may be read whether or not the 8231A is busy.

When READY goes low, the APU expects the bus control signals present at the time to remain stable until READY goes high.

DEVICE STATUS

Device status is provided by means of an internal status register whose format is shown below:



BUSY: Indicates that 8231A is currently executing a command (1=Busy)

SIGN: Indicates that the value on the top of stack is negative (1 = Negative)

ZERO: Indicates that the value on the top of stack is zero (1 = Value is zero)

ERROR CODE: This field contains an indication of the validity of the result of the last operation. The error codes are:

- 0000 — No error
- 1000 — Divide by zero
- 0100 — Square root or log of negative number
- 1100 — Argument of inverse sine, cosine, or e^x too large
- XX10 — Underflow
- XX01 — Overflow

CARRY: Previous operation resulted in carry or borrow from most significant bit. (1 = Carry/Borrow, 0 = No Carry/No Borrow.)

If the BUSY bit in the status register is a one, the other status bits are not defined; if zero, indicating not busy, the operation is complete and the other status bits are defined as given above.

READ STATUS

The 8231A status register can be read by the CPU at any time (whether an operation is in progress or not) by bringing the chip select ($\overline{CS}$) low, the command/data line (A_0) high, and lowering $\overline{RD}$. The status register is then gated onto the data bus and may be input by the CPU.

EXECUTION TIMES

Timing for execution of the 8231A command set is contained below. All times are given in terms of clock cycles. Where substantial variation of execution times

is possible, the minimum and maximum values are quoted; otherwise, typical values are given. Variations are data dependent.

Total execution times may require allowances for operand transfer into the APU, command execution, and result retrieval from the APU. Except for command execution, these times will be heavily influenced by the nature of the data, the control interface used, the speed of memory, the CPU used, the priority allotted to DMA and Interrupt operations, the size and number of operands to be transferred, and the use of chained calculations, etc.

Table 5. Command Execution Times

Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles	Command Mnemonic	Clock Cycles
SADD	17	FADD	54-368	LN	4298-6956	POPF	12
SSUB	30	FSUB	70-370	EXP	3794-4878	XCHS	18
SMUL	84-94	FMUL	146-168	PWR	8290-12032	XCHD	26
SMUU	80-98						
SDIV	84-94	FDIV	154-184	NOP	4	XCHF	26
DADD	21	SORT	800	CHSS	23	PUPI	16
DSUB	38	SIN	4464	CHSD	27		
DMUL	194-210	COS	4118	CHSF	18		
DMUU	182-218						
DDIV	208	TAN	5754	PTOS	16		
FIXS	92-216	ASIN	7668	PTOD	20		
FIXD	100-346	ACOS	7734	PTOF	20		
FLTS	98-186	ATAN	6006	POPS	10		
FLTD	98-378	LOG	4474-7132	POPD	12		

DERIVED FUNCTION DISCUSSION

Computer approximations of transcendental functions are often based on some form of polynomial equation, such as:

$$F(X) = A_0 + A_1X + A_2X^2 + A_3X^3 + A_4X^4 \ldots \quad (1\text{-}1)$$

The primary shortcoming of an approximation in this form is that it typically exhibits very large errors when the magnitude of $|X|$ is large, although the errors are small when $|X|$ is small. With polynomials in this form, the error distribution is markedly uneven over any arbitrary interval.

A set of approximating functions exists that not only minimizes the maximum error but also provides an even distribution of errors within the selected data representation interval. These are known as Chebyshev Polynomials and are are based upon cosine functions. These functions are defined as follows:

$$T_n(X) = \text{Cos } n\theta; \text{ where } n = 0,1,2 \ldots \quad (1\text{-}2)$$
$$\theta = \text{Cos}^{-1}X$$

The various terms of the Chebyshev series can be computed as shown below:

$$T_0(X) = \text{Cos}(0 \cdot \theta) = \text{Cos}(0) = 1 \quad (1\text{-}4)$$
$$T_1(X) = \text{Cos}(\text{Cos}^{-1}X) = X \quad (1\text{-}5)$$
$$T_2(X) = \text{Cos } 2\theta = 2\text{Cos}^2\theta - 1 = 2\text{Cos}^2(\text{Cos}^{-1}X) - 1 \quad (1\text{-}6)$$
$$= 2X^2 - 1$$

In general, the next term in the Chebyshev series can be recursively derived from the previous term as follows:

$$T_n(X) = 2X[T_n - 1(X)] - T_n - 2(X); n \geq 2 \quad (1\text{-}7)$$

Common logarithms are computed by multiplication of the natural logarithm by the conversion factor 0.43429448 and the error function is therefore the same as that for natural logarithm. The power function is realized by combination of natural log and exponential functions according to the equation:

$$X^Y = e^{Y\text{Ln}x}.$$

The error for the power function is a combination of that for the logarithm and exponential functions.

Each of the derived functions is an approximation of the true function. Thus the result of a derived function will have an error. The absolute error is the difference between the function's result and the true result. A more useful measure of the function's error is relative error (absolute error/true result). This gives a measurement of the significant digits of algorithm accuracy. For the derived functions except LN, LOG, and PWR the relative error is typically 4×10^{-7}. For PWR the relative error is the summation of the EXP and LN errors, 7×10^{-7}. For LN and LOG, the absolute error is 2×10^{-7}.

APPLICATION INFORMATION

The diagram in Figure 4 shows the interface connections for the APU with operand transfers handled by an 8237 DMA controller, and CPU coordination handled by an Interrupt Controller. The APU interrupts the CPU to indicate that a command has been completed. When the performance enhancements provided by the DMA and Interrupt operations are not required, the APU interface can be simplified as shown in Figure 3. The 8231A APU is designed with a general purpose 8-bit data bus and interface control so that it can be conveniently used with any general 8-bit processor.

In many systems it will be convenient to use the microcomputer system clock to drive the APU clock input. In the case of 8080A systems it would be the ϕ2TTL signal. Its cycle time will usually fall in the range of 250 ns to 1000 ns, depending on the system speed.

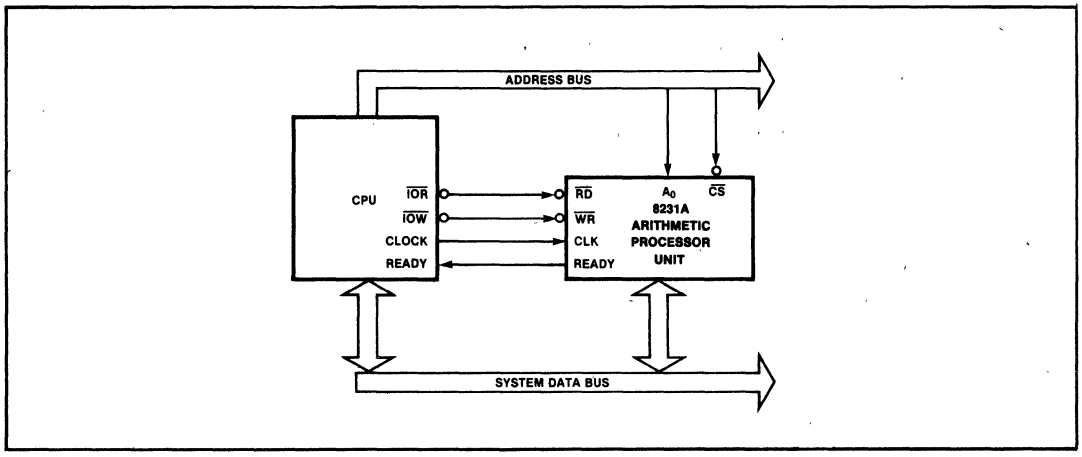


Figure 3. Minimum Configuration Example

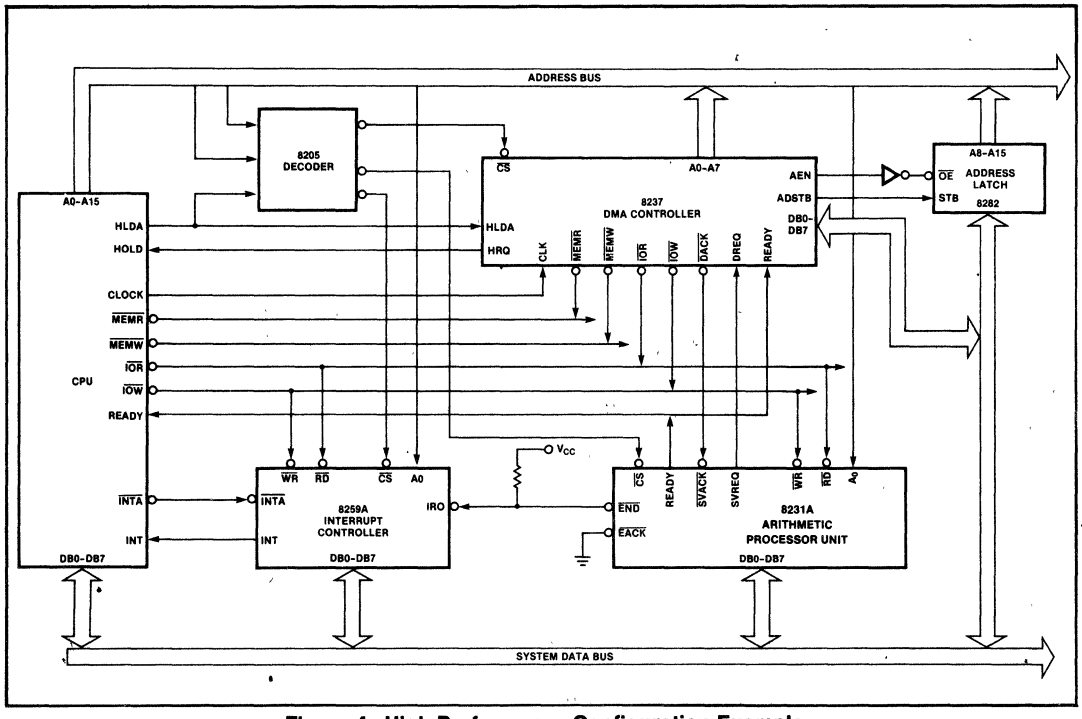


Figure 4. High Performance Configuration Example

ABSOLUTE MAXIMUM RATINGS*

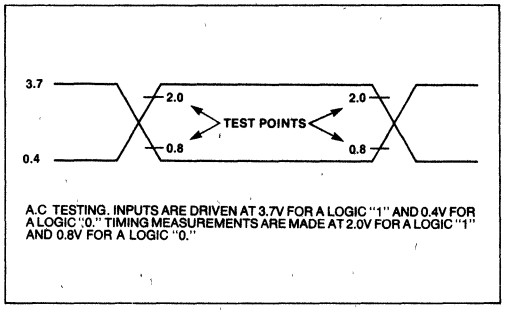
Storage Temperature ..... −65°C to +150°C
Ambient Temperature Under Bias ..... 0°C to 70°C
V_{DD} with Respect to V_{SS} ..... −0.5V to +15.0V
V_{CC} with Respect to V_{SS} ..... −0.5V to +7.0V
All Signal Voltages with Respect to V_{SS} ..... −0.5V to +7.0V
Power Dissipation 2.0W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may effect device reliability.*

D.C. AND OPERATING CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = +5V ± 10%, V_{DD} = +12V ± 10%)

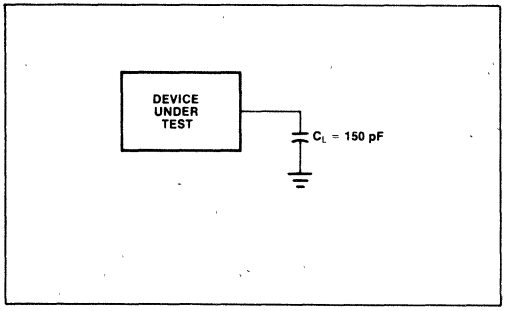
Parameters	Description	Min.	Typ.	Max.	Units	Test Conditions
V_{OH}	Output HIGH Voltage	3.7			Volts	I_{OH} = −200 μA
V_{OL}	Output LOW Voltage			0.4	Volts	I_{OL} = 3.2 mA
V_{IH}	Input HIGH Voltage	2.0		V_{CC}	Volts	
V_{IL}	Input LOW Voltage	−0.5		0.8	Volts	
I_{IL}	Input Load Current			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Data Bus Leakage			±10	μA	V_{SS} +0.45 ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current		50	95	mA	
I_{DD}	V_{DD} Supply Current		50	95	mA	
C_O	Output Capacitance		8		pF	fc = 1.0 MHz, Inputs = 0V
C_I	Input Capacitance		5		pF	
C_{IO}	I/O Capacitance		10		pF	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 3.7V FOR A LOGIC "1" AND 0.4V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

A.C. TESTING LOAD CIRCUIT



A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V, V_{CC} = +5V ± 10%, V_{DD} = +12V ± 10%)

READ OPERATION

Symbol	Parameter		8231A-8		8231A		Units
			Min.	Max.	Min.	Max.	
t_{AR}	A_0, $\overline{CS}$ Setup to $\overline{RD}$		0		0		ns
t_{RA}	A_0, $\overline{CS}$ Hold from $\overline{RD}$		0		0		ns
t_{RY}	READY ↓ from $\overline{RD}$ ↓ Delay (Note 2)			150		100	ns
t_{YR}	READY ↑ to $\overline{RD}$ ↑		0		0		ns
t_{RRR}	READY Pulse Width (Note 3)	Data	3.5 t_{CY} + 50		3.5 t_{CY} + 50		ns
		Status	1.5 t_{CY} + 50		1.5 t_{CY} + 50		ns
t_{RDE}	Data Bus Enable from $\overline{RD}$ ↓		50		50		ns
t_{DRY}	Data Valid to READY ↑		0		0		ns
t_{DF}	Data Float after $\overline{RD}$ ↑		50	200	50	100	ns

WRITE OPERATION

Symbol	Parameter		8231A-8		8231A		Units
			Min.	Max.	Min.	Max.	
t_{AW}	A_0, $\overline{CS}$ Setup to $\overline{WR}$		0		0		ns
t_{WA}	A_0, $\overline{CS}$ Hold after $\overline{WR}$		60		25		ns
t_{WY}	READY ↓ from $\overline{WR}$ ↓ Delay (Note 2)			150		100	ns
t_{YW}	READY ↑ to $\overline{WR}$ ↑		0		0		ns
t_{RRW}	READY Pulse Width (Note 4)			50		50	ns
t_{WI}	Write Inactive Time (Note 4)	Command	4 t_{CY}		4 t_{CY}		ns
		Data	5 t_{CY}		5 t_{CY}		ns
t_{DW}	Data Setup to $\overline{WR}$		150		100		ns
t_{WD}	Data Hold after $\overline{WR}$		20		20		ns

OTHER TIMINGS

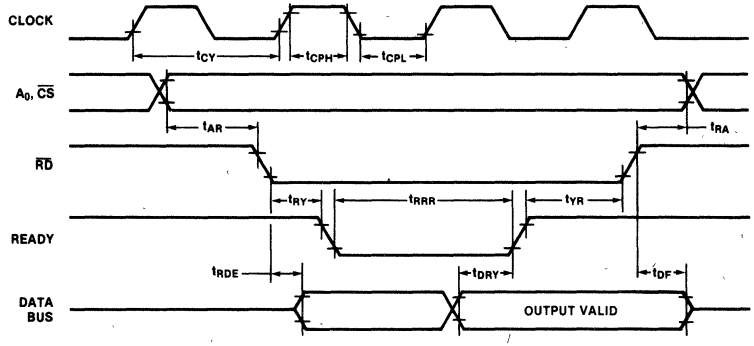
Symbol	Parameter	8231A-8		8231A		Units
		Min.	Max.	Min.	Max.	
t_{CY}	Clock Period	480	5000	250	2500	ns
t_{CPH}	Clock Pulse High Width	200		100		ns
t_{CPL}	Clock Pulse Low Width	240		120		ns
t_{EE}	$\overline{END}$ Pulse Width (Note 5)	400		200		ns
t_{EAE}	$\overline{EACK}$ ↓ to $\overline{END}$ ↑ Delay		200		150	ns
t_{AA}	$\overline{EACK}$ Pulse Width	100		50		ns
t_{SA}	$\overline{SVACK}$ ↓ to SVREQ ↓ Delay		300		150	ns
t_{SS}	$\overline{SVACK}$ Pulse Width	100		50		ns

NOTES:

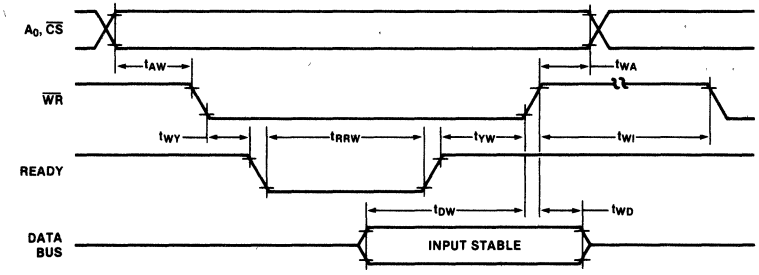
1. Typical values are for T_A=25°C, nominal supply voltages and nominal processing parameters.
2. READY is pulled low for both command and data operations.
3. Minimum values shown assume no previously entered command is being executed for the data access. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. Status may be read at any time without exceeding the time shown.
4. READY low pulse width is less than 50 ns when writing into the data port or the control port as long as the duty cycle requirement (t_{WI}) is observed and no previous command is being executed. t_{WI} may be safely violated as long as the extended t_{RRW} that results is observed. If a previously entered command is being executed, READY low pulse width is the time to complete execution plus the time shown. These timings refer specifically to the 8231A.
5. $\overline{END}$ low pulse width is specified for $\overline{EACK}$ tied to VSS. Otherwise t_{EAE} applies.

WAVEFORMS

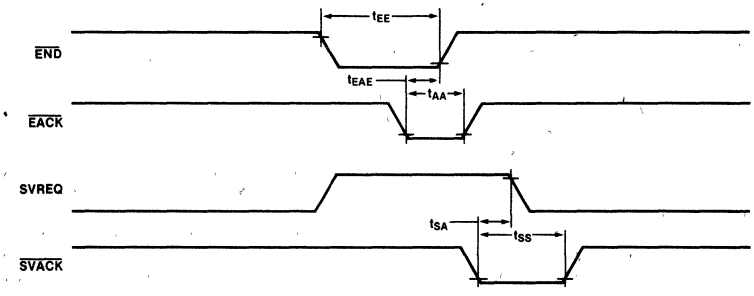
READ OPERATION



WRITE OPERATION



INTERRUPT OPERATION



8253/8253-5
PROGRAMMABLE INTERVAL TIMER

- **MCS-85™ Compatible 8253-5**
- **3 Independent 16-Bit Counters**
- **DC to 2.6 MHz**
- **Programmable Counter Modes**
- **Count Binary or BCD**
- **Single +5V Supply**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**

The Intel® 8253 is a programmable counter/timer device designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. All modes of operation are software programmable.

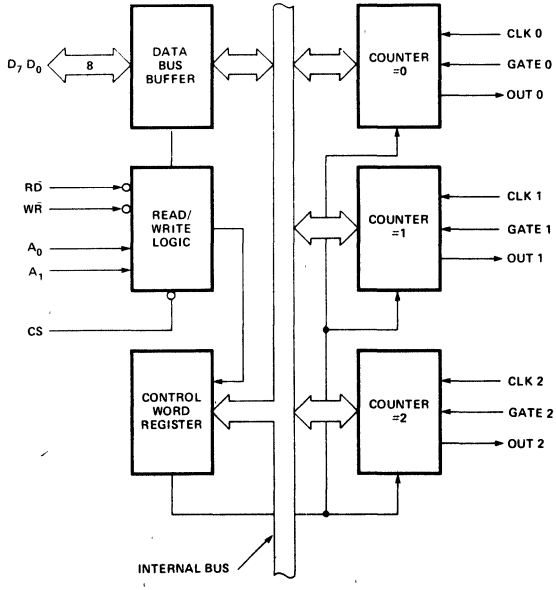


Figure 1. Block Diagram

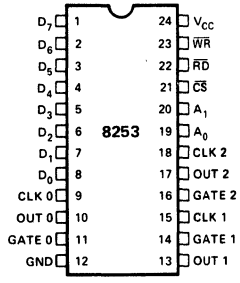


Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

General

The 8253 is a programmable interval timer/counter specifically designed for use with the Intel™ Microcomputer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

- Programmable Rate Generator
- Event Counter
- Binary Rate Multiplier
- Real Time Clock
- Digital One-Shot
- Complex Motor Controller

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

$\overline{RD}$ (Read)

A "low" on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

$\overline{WR}$ (Write)

A "low" on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8253. No reading or writing will occur unless the device is selected. The $\overline{CS}$ input has no effect upon the actual operation of the counters.

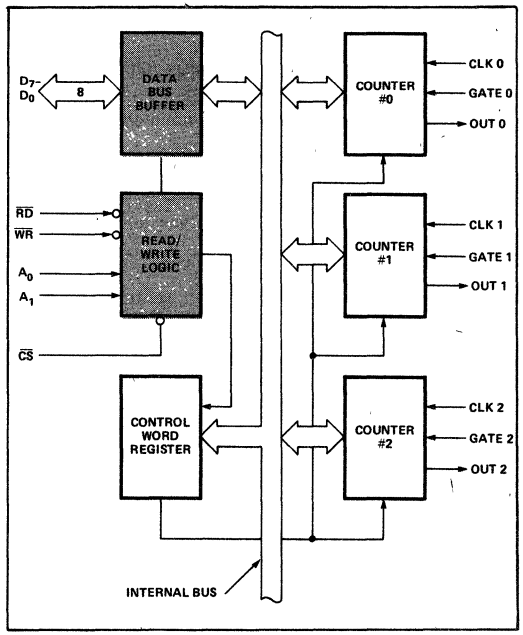


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operational MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

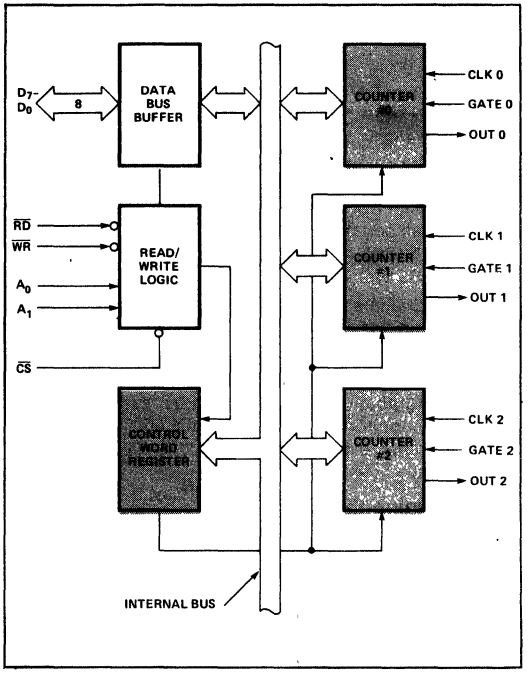


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The $\overline{CS}$ can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel® 8205 for larger systems.

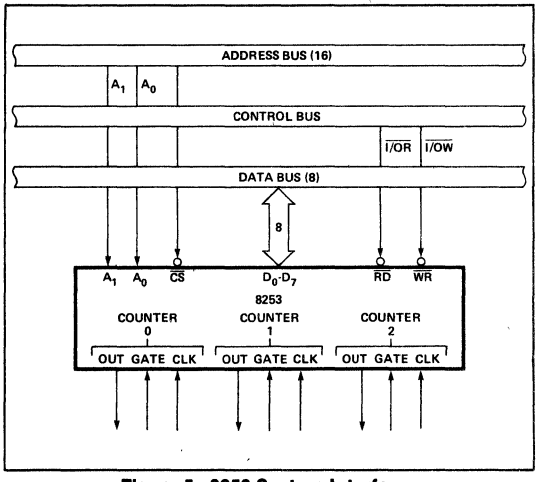


Figure 5. 8253 System Interface

OPERATIONAL DESCRIPTION

General

The complete functional definition of the 8253 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253

All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register. (A0, A1 = 11)

Control Word Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL — Read/Load:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section)
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Loading

The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE Definition

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

(1) Write 1st byte stops the current counting.
(2) Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for $(N+1)/2$ counts and low for $(N-1)/2$ counts.

In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

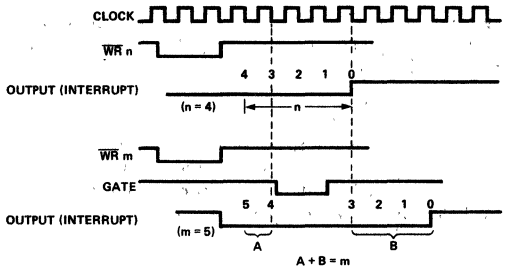
If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

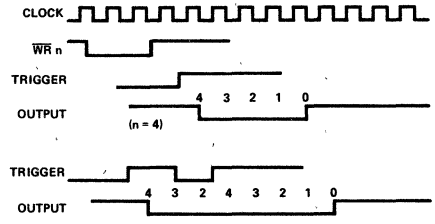
Modes \ Signal Status	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 6. Gate Pin Operations Summary

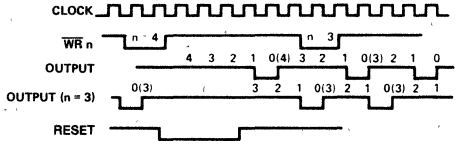
MODE 0: Interrupt on Terminal Count



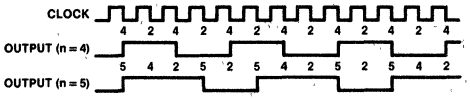
MODE 1: Programmable One-Shot



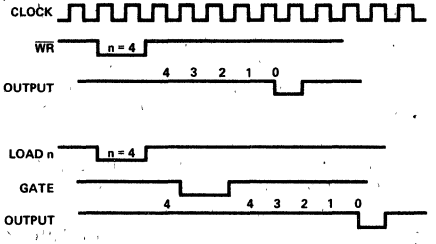
MODE 2: Rate Generator



MODE 3: Square Wave Generator



MODE 4: Software Triggered Strobe



MODE 5: Hardware Triggered Strobe

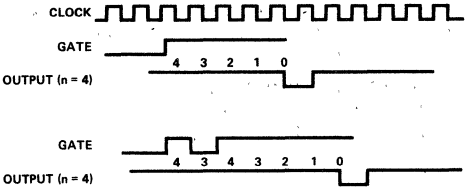


Figure 7. 8253 Timing Diagrams

8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1)

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it must be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeroes into a count register will result in the maximum count (2^{16} for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

	MODE Control Word Counter n
LSB	Count Register byte Counter n
MSB	Count Register byte Counter n

Note: Format shown is a simple example of loading the 8253 and does not imply that it is the only format that can be used.

Figure 8. Programming Format

			A1	A0
No. 1		MODE Control Word Counter 0	1	1
No. 2		MODE Control Word Counter 1	1	1
No. 3		MODE Control Word Counter 2	1	1
No. 4	LSB	Count Register Byte Counter 1	0	1
No. 5	MSB	Count Register Byte Counter 1	0	1
No. 6	LSB	Count Register Byte Counter 2	1	0
No. 7	MSB	Count Register Byte Counter 2	1	0
No. 8	LSB	Count Register Byte Counter 0	0	0
No. 9	MSB	Count Register Byte Counter 0	0	0

Note: The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully utilized.

Figure 9. Alternate Programming Formats

Read Operations

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253 the programmer can select the counter to be read (remember that no read operation of the mode register is allowed A0, A1-11) The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter must be inhibited either by controlling the Gate input or by external logic that inhibits the clock input The contents of the counter selected will be available as follows

first I/O Read contains the least significant byte (LSB).

second I/O Read contains the most significant byte (MSB)

Due to the internal logic of the 8253 it is absolutely necessary to complete the entire reading procedure If two bytes are programmed to be read then two bytes must be read before any loading WR command can be sent to the same counter

Read Operation Chart

A1	A0	RD	
0	0	0	Read Counter No. 0
0	1	0	Read Counter No. 1
1	0	0	Read Counter No. 2
1	1	0	Illegal

Reading While Counting

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available

MODE Register for Latching Count

A0, A1 = 11

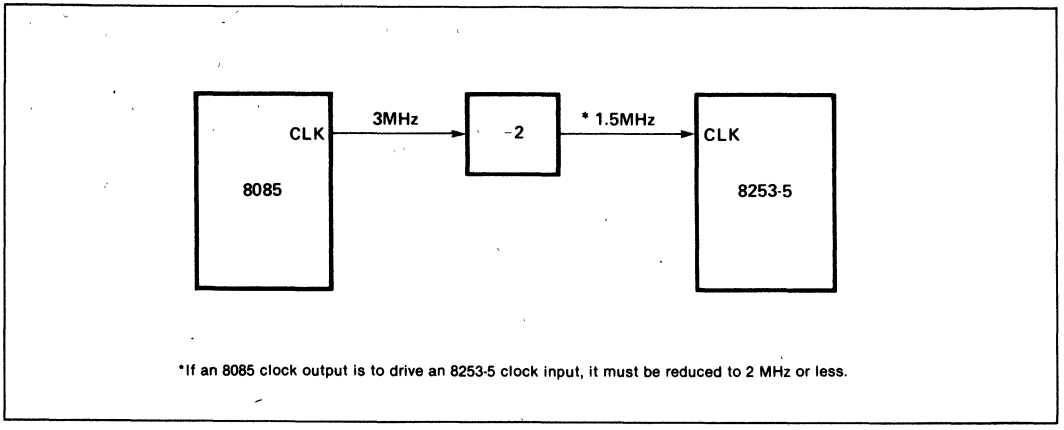
D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 — specify counter to be latched

D5,D4 — 00 designates counter latching operation.

X — don't care

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.



*If an 8085 clock output is to drive an 8253-5 clock input, it must be reduced to 2 MHz or less.

Figure 10. MCS-85™ Clock Interface*

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground ..... −0.5 V to +7 V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%) *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC}+.5V	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 2
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to .45V
I_{CC}	V_{CC} Supply Current		140	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, GND = 0V) *

Bus Parameters (Note 3)

READ CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{READ}$	50		30		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		5		ns
t_{RR}	$\overline{READ}$ Pulse Width	400		300		ns
t_{RD}	Data Delay From $\overline{READ}$[4]		300		200	ns
t_{DF}	$\overline{READ}$ to Data Floating	25	125	25	100	ns
t_{RV}	Recovery Time Between READ and Any Other Control Signal	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before $\overline{WRITE}$	50		30		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	30		30		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		300		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		250		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		30		ns
t_{RV}	Recovery Time Between $\overline{WRITE}$ and Any Other Control Signal	1		1		μs

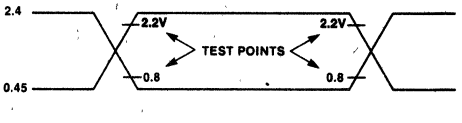
CLOCK AND GATE TIMING

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{CLK}	Clock Period	380	dc	380	dc	ns
t_{PWH}	High Pulse Width	230		230		ns
t_{PWL}	Low Pulse Width	150		150		ns
t_{GW}	Gate Width High	150		150		ns
t_{GL}	Gate Width Low	100		100		ns
t_{GS}	Gate Set Up Time to CLK↑	100		100		ns
t_{GH}	Gate Hold Time After CLK↑	50		50		ns
t_{OD}	Output Delay From CLK↓[4]		400		400	ns
t_{ODG}	Output Delay From Gate↓[4]		300		300	ns

NOTES:
1. I_{OL} = 2.2 mA.
2. I_{OH} = −400 μA.
3. AC timings measured at V_{OH} 2.2, V_{OL} = 0.8.
4. C_L = 150pF.

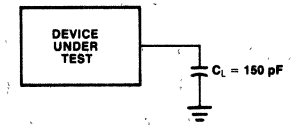
* For Extended Temperature EXPRESS, use M8253 electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.2V FOR A LOGIC "1" AND 0.8V FOR A LOGIC 0.

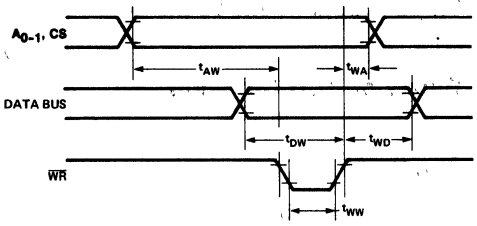
A.C. TESTING LOAD CIRCUIT



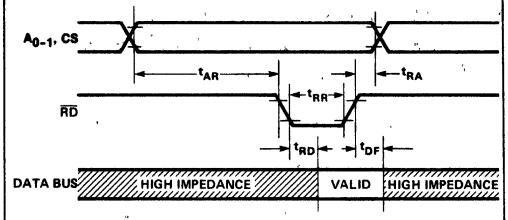
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

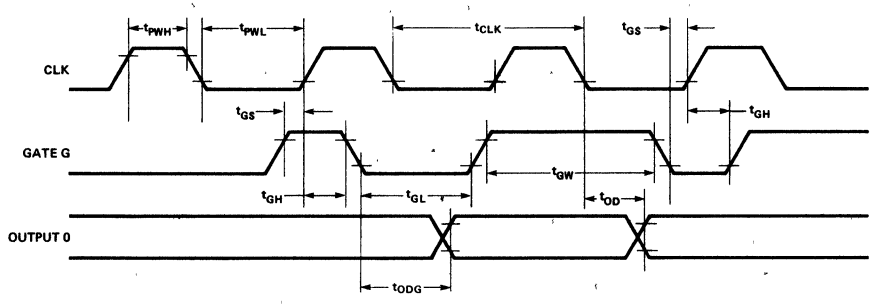
WRITE TIMING



READ TIMING



CLOCK AND GATE TIMING



PRELIMINARY

8254
PROGRAMMABLE INTERVAL TIMER

- **Compatible with Most Microprocessors Including 8080A, 8085A, iAPX 88 and iAPX 86**
- **Handles Inputs from DC to 8 MHz (10 MHz for 8254-2)**
- **Six Programmable Counter Modes**
- **Status Read-Back Command**
- **Three Independent 16-bit Counters**
- **Binary or BCD Counting**
- **Single +5V Supply**
- **Available in EXPRESS —Standard Temperature Range**

The Intel® 8254 is a counter/timer device designed to solve the common timing control problems in microcomputer system design. It provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz. All modes are software programmable. The 8254 is a superset of the 8253.

The 8254 uses HMOS technology and comes in a 24-pin plastic or CERDIP package.

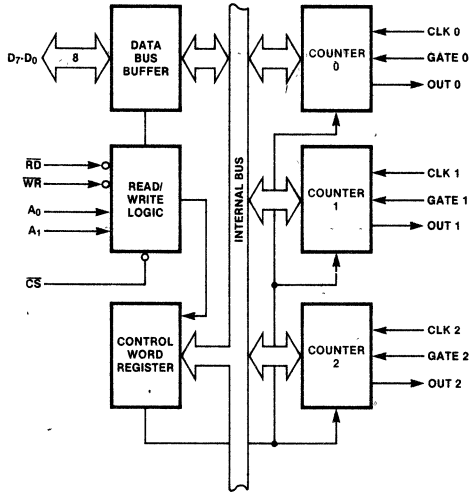


Figure 1. 8254 Block Diagram

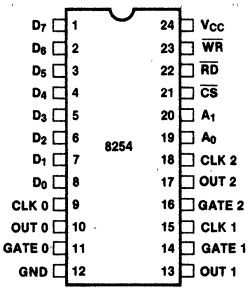


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_7-D_0	1-8	I/O	**Data:** Bi-directional three state data bus lines, connected to system data bus.
CLK 0	9	I	**Clock 0:** Clock input of Counter 0.
OUT 0	10	O	**Output 0:** Output of Counter 0.
GATE 0	11	I	**Gate 0:** Gate input of Counter 0.
GND	12		**Ground:** Power supply connection.
V_{CC}	24		**Power:** +5V power supply connection.
$\overline{WR}$	23	I	**Write Control:** This input is low during CPU write operations.
$\overline{RD}$	22	I	**Read Control:** This input is low during CPU read operations.
$\overline{CS}$	21	I	**Chip Select:** A low on this input enables the 8254 to respond to $\overline{RD}$ and $\overline{WR}$ signals. $\overline{RD}$ and $\overline{WR}$ are ignored otherwise.
A_1, A_0	20-19	I	**Address:** Used to select one of the three Counters or the Control Word Register for read or write operations. Normally connected to the system address bus.
			A_1 / A_0 / Selects: 0 0 Counter 0; 0 1 Counter 1; 1 0 Counter 2; 1 1 Control Word Register
CLK 2	18	I	**Clock 2:** Clock input of Counter 2.
OUT 2	17	O	**Out 2:** Output of Counter 2.
GATE 2	16	I	**Gate 2:** Gate input of Counter 2.
CLK 1	15	I	**Clock 1:** Clock input of Counter 1.
GATE 1	14	I	**Gate 1:** Gate input of Counter 1.
OUT 1	13	O	**Out 1:** Output of Counter 1.

FUNCTIONAL DESCRIPTION

General

The 8254 is a programmable interval timer/counter designed for use with Intel microcomputer systems. It is a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8254 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in software, the programmer configures the 8254 to match his requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU. Software overhead is minimal and variable length delays can easily be accommodated.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:

- Real time clock
- Event counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

Block Diagram

DATA BUS BUFFER

This 3-state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus (see Figure 3).

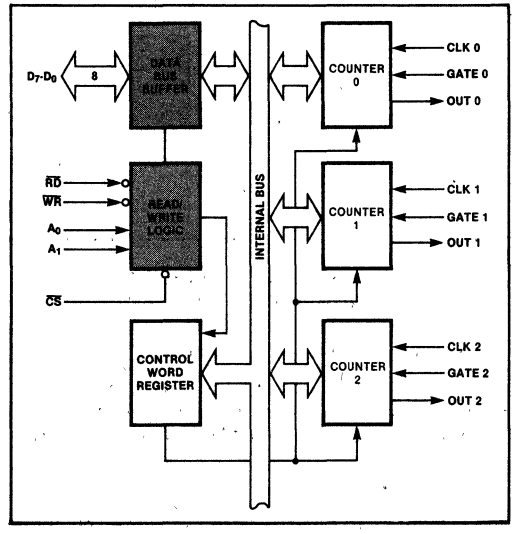


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

READ/WRITE LOGIC

The Read/Write Logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254. A_1 and A_0 select one of the three counters or the Control Word Register to be read from/written into. A "low" on the $\overline{RD}$ input tells the 8254 that the CPU is reading one of the counters. A "low" on the $\overline{WR}$ input tells the 8254 that the CPU is writing either a Control Word or an initial count. Both $\overline{RD}$ and $\overline{WR}$ are qualified by $\overline{CS}$; $\overline{RD}$ and $\overline{WR}$ are ignored unless the 8254 has been selected by holding $\overline{CS}$ low.

CONTROL WORD REGISTER

The Control Word Register (see Figure 4) is selected by the Read/Write Logic when $A_1,A_0 = 11$. If the CPU then does a write operation to the 8254, the data is stored in the Control Word Register and is interpreted as a Control Word used to define the operation of the Counters.

The Control Word Register can only be written to; status information is available with the Read-Back Command.

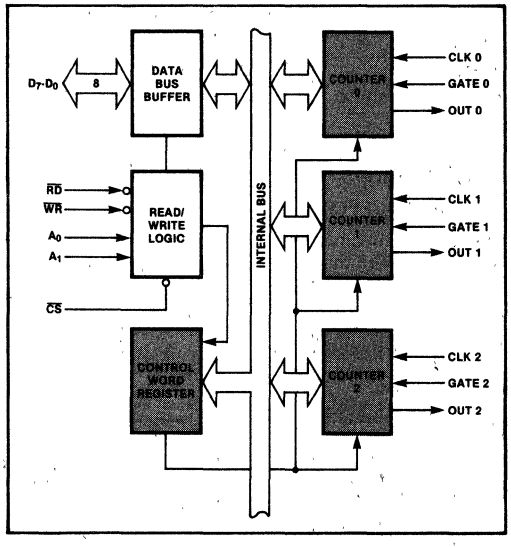


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

COUNTER 0, COUNTER 1, COUNTER 2

These three functional blocks are identical in operation, so only a single Counter will be described. The internal block diagram of a single counter is shown in Figure 5.

The Counters are fully independent. Each Counter may operate in a different Mode.

The Control Word Register is shown in the figure; it is not part of the Counter itself, but its contents determine how the Counter operates.

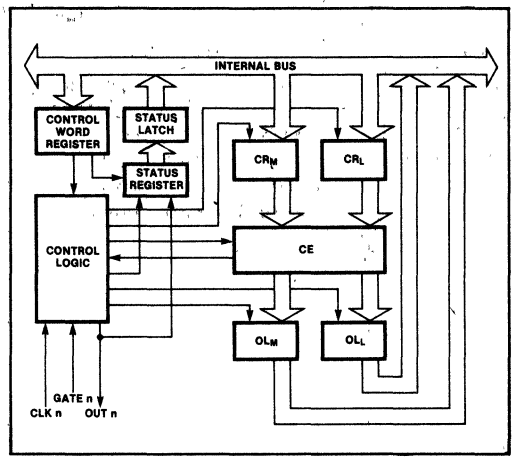


Figure 5. Internal Block Diagram of a Counter

The status register, shown in the Figure, when latched, contains the current contents of the Control Word Register and status of the output and null count flag. (See detailed explanation of the Read-Back command.)

The actual counter is labelled CE (for "Counting Element"). It is a 16-bit presettable synchronous down counter.

OL_M and OL_L are two 8-bit latches. OL stands for "Output Latch"; the subscripts M and L stand for "Most significant byte" and "Least significant byte" respectively. Both are normally referred to as one unit and called just OL. These latches normally "follow" the CE, but if a suitable Counter Latch Command is sent to the 8254, the latches "latch" the present count until read by the CPU and then return to "following" the CE. One latch at a time is enabled by the counter's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that the CE itself cannot be read; whenever you read the count, it is the OL that is being read.

Similarly, there are two 8-bit registers called CR_M and CR_L (for "Count Register"). Both are normally referred to as one unit and called just CR. When a new count is written to the Counter, the count is stored in the CR and later transferred to the CE. The Control Logic allows one register at a time to be loaded from the internal bus. Both bytes are transferred to the CE simultaneously. CR_M and CR_L are cleared when the Counter is programmed. In this way, if the Counter has been programmed for one byte counts (either most significant byte only or least significant byte only) the other byte will be zero. Note that the CE cannot be written into; whenever a count is written, it is written into the CR.

The Control Logic is also shown in the diagram. CLK n, GATE n, and OUT n are all connected to the outside world through the Control Logic.

8254 SYSTEM INTERFACE

The 8254 is a component of the Intel Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A_0, A_1 connect to the A_0, A_1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel 8205 for larger systems.

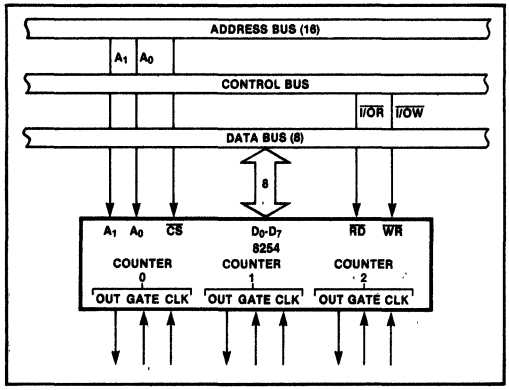


Figure 6. 8254 System Interface

OPERATIONAL DESCRIPTION

General

After power-up, the state of the 8254 is undefined. The Mode, count value, and output of all Counters are undefined.

How each Counter operates is determined when it is programmed. Each Counter must be programmed before it can be used. Unused counters need not be programmed.

Programming the 8254

Counters are programmed by writing a Control Word and then an initial count.

All Control Words are written into the Control Word Register, which is selected when $A_1,A_0 = 11$. The Control Word itself specifies which Counter is being programmed.

By contrast, initial counts are written into the Counters, not the Control Word Register. The A_1,A_0 inputs are used to select the Counter to be written into. The format of the initial count is determined by the Control Word used.

Control Word Format

$A_1,A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.

Figure 7. Control Word Format

Write Operations

The programming procedure for the 8254 is very flexible. Only two conventions need to be remembered:

1) For each Counter, the Control Word must be written before the initial count is written.
2) The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1,A_0 inputs), and each Control Word specifies the Counter it applies to (SC0,SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

	A_1	A_0
Control Word — Counter 0	1	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0
Control Word — Counter 1	1	1
LSB of count — Counter 1	0	1
MSB of count — Counter 1	0	1
Control Word — Counter 2	1	1
LSB of count — Counter 2	1	0
MSB of count — Counter 2	1	0

	A_1	A_0
Control Word — Counter 2	1	1
Control Word — Counter 1	1	1
Control Word — Counter 0	1	1
LSB of count — Counter 2	1	0
MSB of count — Counter 2	1	0
LSB of count — Counter 1	0	1
MSB of count — Counter 1	0	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0

	A_1	A_0
Control Word — Counter 0	1	1
Control Word — Counter 1	1	1
Control Word — Counter 2	1	1
LSB of count — Counter 2	1	0
LSB of count — Counter 1	0	1
LSB of count — Counter 0	0	0
MSB of count — Counter 0	0	0
MSB of count — Counter 1	0	1
MSB of count — Counter 2	1	0

	A_1	A_0
Control Word — Counter 1	1	1
Control Word — Counter 0	1	1
LSB of count — Counter 1	0	1
Control Word — Counter 2	1	1
LSB of count — Counter 0	0	0
MSB of count — Counter 1	0	1
LSB of count — Counter 2	1	0
MSB of count — Counter 0	0	0
MSB of count — Counter 2	1	0

NOTE: IN ALL FOUR EXAMPLES, ALL COUNTERS ARE PROGRAMMED TO READ/WRITE TWO-BYTE COUNTS.
THESE ARE ONLY FOUR OF MANY POSSIBLE PROGRAMMING SEQUENCES.

Figure 8. A Few Possible Programming Sequences

Read Operations

It is often desirable to read the value of a Counter without disturbing the count in progress. This is easily done in the 8254.

There are three possible methods for reading the Counters. The first is through the Read-Back command. The second is a simple read operation of the Counter, which is selected with the A_1,A_0 inputs. The only requirement is that 1) the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic; or 2) the count must first be latched. Otherwise, the count may be in process of changing when it is read, giving an undefined result.

COUNTER LATCH COMMAND

The other method involves a special software command called the "Counter Latch Command". Like a Control Word, this command is written to the Control Word Register, which is selected when $A_1,A_0 = 11$. Also like a Control Word, the SC0,SC1 bits select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

A_1,A_0=11; CS=0; RD=1; WR=0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0 — specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4 — 00 designates Counter Latch Command

X — don't care

NOTE: DON'T CARE BITS (X) SHOULD BE 0 TO INSURE COMPATIBILITY WITH FUTURE INTEL PRODUCTS.

Figure 9. Counter Latching Command Format

The selected Counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the Counter is reprogrammed). The count is then unlatched automatically and the OL returns to "following" the counting element (CE). This allows reading the contents of the Counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one Counter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the Counter in any way.

If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the Counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other Counters may be inserted between them.

Another feature of the 8254 is that reads and writes of the same Counter may be interleaved; for example, if the Counter is programmed for two byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a Counter is programmed to read/write two-byte counts, the following precaution applies: A program must not transfer control between reading the first and second byte to another routine which also reads from that same Counter. Otherwise, an incorrect count will be read.

READ-BACK COMMAND

The read-back command allows the user to check the count value, programmed Mode, and current state of the OUT pin and Null Count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D3,D2,D1=1.

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{COUNT}$	$\overline{STATUS}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = LATCH COUNT OF SELECTED COUNTER(S)
D_4: 0 = LATCH STATUS OF SELECTED COUNTER(S)
D_3: 1 = SELECT COUNTER 2
D_2: 1 = SELECT COUNTER 1
D_1: 1 = SELECT COUNTER 0
D_0: RESERVED FOR FUTURE EXPANSION; MUST BE 0

Figure 10. Read-Back Command Format

The read-back command may be used to latch multiple counter output latches (OL) by setting the $\overline{COUNT}$ bit D5=0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple count read-back commands are issued to the same counter without reading the count, all but the first are ignored; i.e., the count which will be read is the count at the time the first read-back command was issued.

The read-back command may also be used to latch status information of selected counter(s) by setting $\overline{STATUS}$ bit D4 = 0. Status must be latched to be read; status of a counter is accessed by a read from that counter.

The counter status format is shown in Figure 11. Bits D5 through D0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D7 contains the current state of the OUT pin. This allows the user to monitor the counter's output via software, possibly eliminating some hardware from a system.

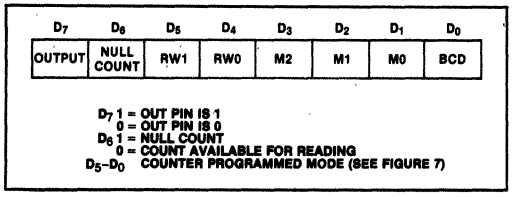


Figure 11. Status Byte

NULL COUNT bit D6 indicates when the last count written to the counter register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

THIS ACTION:	CAUSES:
A. WRITE TO THE CONTROL WORD REGISTER:[1]	NULL COUNT=1
B. WRITE TO THE COUNT REGISTER (CR);[2]	NULL COUNT=1
C. NEW COUNT IS LOADED INTO CE (CR→CE);	NULL COUNT=0

[1] ONLY THE COUNTER SPECIFIED BY THE CONTROL WORD WILL HAVE ITS NULL COUNT SET TO 1. NULL COUNT BITS OF OTHER COUNTERS ARE UNAFFECTED.

[2] IF THE COUNTER IS PROGRAMMED FOR TWO-BYTE COUNTS (LEAST SIGNIFICANT BYTE THEN MOST SIGNIFICANT BYTE) NULL COUNT GOES TO 1 WHEN THE SECOND BYTE IS WRITTEN.

Figure 12. Null Count Operation

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored; i.e., the status that will be read is the status of the counter at the time the first status read-back command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both $\overline{\text{COUNT}}$ and $\overline{\text{STATUS}}$ bits D5,D4=0. This is functionally the same as issuing two separate read-back commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status read-back commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Figure 13.

Command								Description	Result
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
1	1	0	0	0	0	1	0	Read back count and status of Counter 0	Count and status latched for Counter 0
1	1	1	0	0	1	0	0	Read back status of Counter 1	Status latched for Counter 1
1	1	1	0	1	1	0	0	Read back status of Counters 2, 1	Status latched for Counter 2, but not Counter 1
1	1	0	1	1	0	0	0	Read back count of Counter 2	Count latched for Counter 2
1	1	0	0	0	1	0	0	Read back count and status of Counter 1	Count latched for Counter 1, but not status
1	1	1	0	0	0	1	0	Read back status of Counter 1	Command ignored, status already latched for Counter 1

Figure 13. Read-Back Command Example

If both count and status of a counter are latched, the first read operation of that counter will return latched status, regardless of which was latched first. The next one or two reads (depending on whether the counter is programmed for one or two type counts) return latched count. Subsequent reads return unlatched count.

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_1	A_0	
0	1	0	0	0	Write into Counter 0
0	1	0	0	1	Write into Counter 1
0	1	0	1	0	Write into Counter 2
0	1	0	1	1	Write Control Word
0	0	1	0	0	Read from Counter 0
0	0	1	0	1	Read from Counter 1
0	0	1	1	0	Read from Counter 2
0	0	1	1	1	No-Operation (3-State)
1	X	X	X	X	No-Operation (3-State)
0	1	1	X	X	No-Operation (3-State)

Figure 14. Read/Write Operations Summary

Mode Definitions

The following are defined for use in describing the operation of the 8254.

CLK pulse: a rising edge, then a falling edge, in that order, of a Counter's CLK input.

trigger: a rising edge of a Counter's GATE input.

Counter loading: the transfer of a count from the CR to the CE (refer to the "Functional Description")

MODE 0: INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

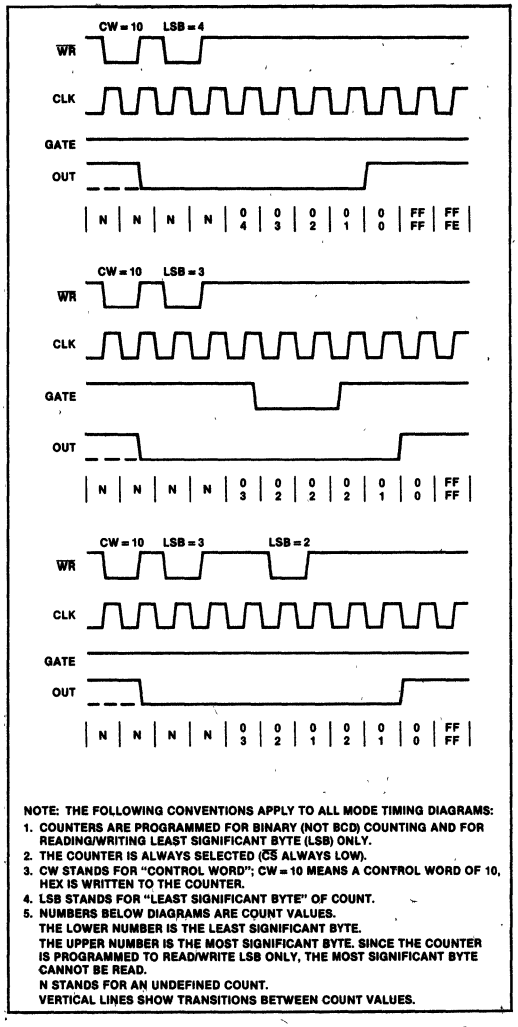
After the Control Word and initial count are written to a Counter, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go high until N + 1 CLK pulses after the initial count is written.

If a new count is written to the Counter, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte disables counting. OUT is set low immediately (no clock pulse required).
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later; no CLK pulse is needed to load the Counter as this has already been done.



NOTE: THE FOLLOWING CONVENTIONS APPLY TO ALL MODE TIMING DIAGRAMS:
1. COUNTERS ARE PROGRAMMED FOR BINARY (NOT BCD) COUNTING AND FOR READING/WRITING LEAST SIGNIFICANT BYTE (LSB) ONLY.
2. THE COUNTER IS ALWAYS SELECTED ($\overline{CS}$ ALWAYS LOW).
3. CW STANDS FOR "CONTROL WORD"; CW = 10 MEANS A CONTROL WORD OF 10, HEX IS WRITTEN TO THE COUNTER.
4. LSB STANDS FOR "LEAST SIGNIFICANT BYTE" OF COUNT.
5. NUMBERS BELOW DIAGRAMS ARE COUNT VALUES.
   THE LOWER NUMBER IS THE LEAST SIGNIFICANT BYTE.
   THE UPPER NUMBER IS THE MOST SIGNIFICANT BYTE. SINCE THE COUNTER IS PROGRAMMED TO READ/WRITE LSB ONLY, THE MOST SIGNIFICANT BYTE CANNOT BE READ.
   N STANDS FOR AN UNDEFINED COUNT.
   VERTICAL LINES SHOW TRANSITIONS BETWEEN COUNT VALUES.

Figure 15. Mode 0

MODE 1: HARDWARE RETRIGGERABLE ONE-SHOT

OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero. OUT will then go high and remain high until the CLK pulse after the next trigger.

After writing the Control Word and initial count, the Counter is armed. A trigger results in loading the Counter and setting OUT low on the next CLK pulse, thus starting the one-shot pulse. An initial count of N will result in a one-shot pulse N CLK cycles in duration. The one-shot is retriggerable, hence OUT will remain low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected unless the Counter is retriggered. In that case, the Counter is loaded with the new count and the one-shot pulse continues until the new count expires.

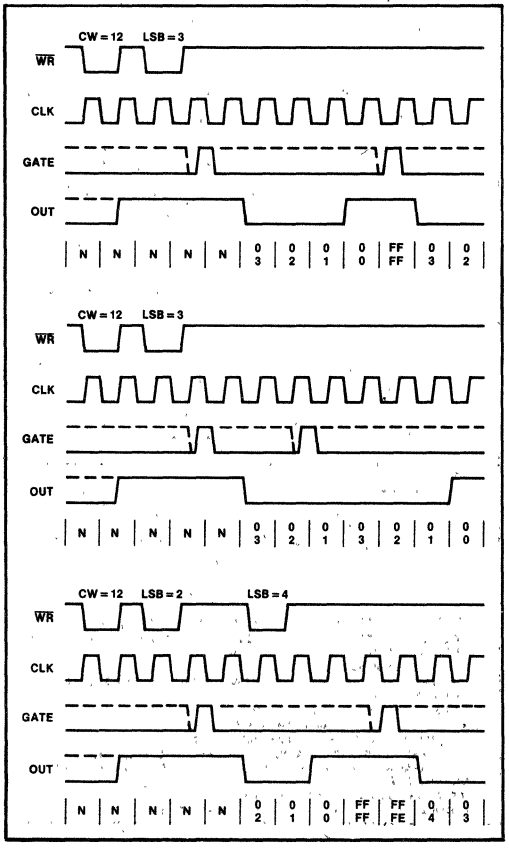


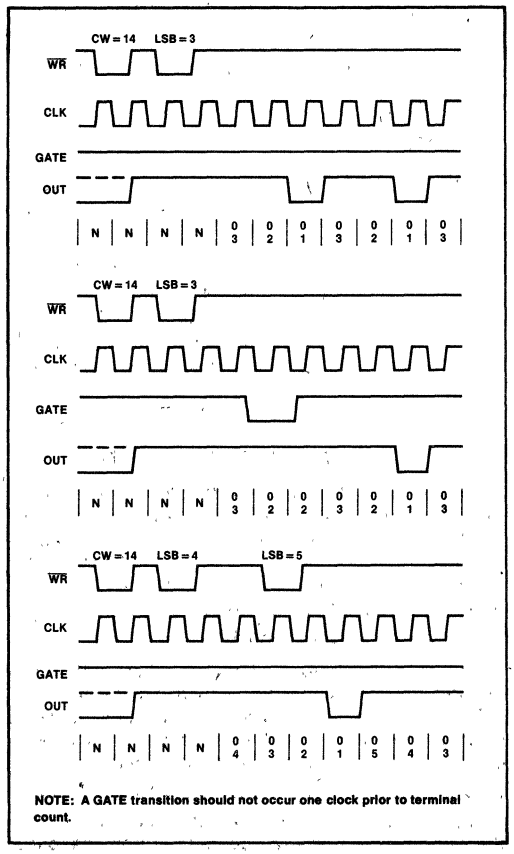
Figure 16. Mode 1

MODE 2: RATE GENERATOR

This Mode functions like a divide-by-N counter. It is typiclaly used to generate a Real Time Clock interrupt. OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated. Mode 2 is periodic; the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately. A trigger reloads the Counter with the initial count on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. OUT goes low N CLK Pulses after the initial count is written. This allows the Counter to be synchronized by software also.



NOTE: A GATE transition should not occur one clock prior to terminal count.

Figure 17. Mode 2

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current period, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current counting cycle. In mode 2, a COUNT of 1 is illegal.

MODE 3: SQUARE WAVE MODE

Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low while OUT is low, OUT is set high immediately; no CLK pulse is required. A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

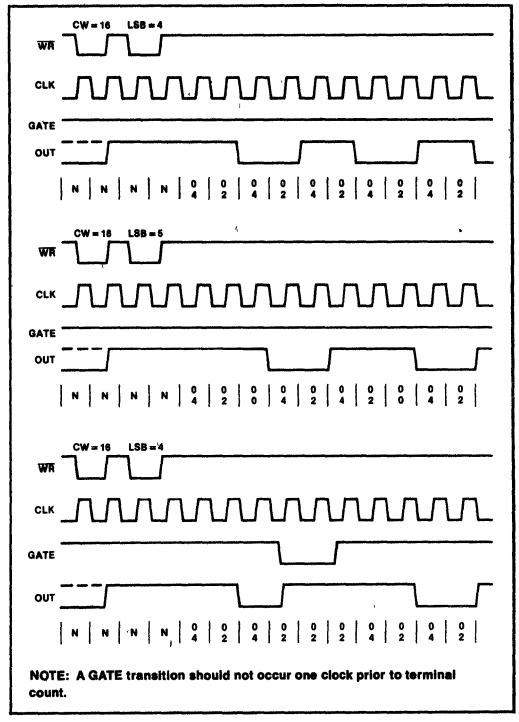
After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This allows the Counter to be synchronized by software also.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

Mode 3 is implemented as follows:

Even counts: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially high. The initial count minus one (an even number) is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes low and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes high again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT will be high for (N + 1)/2 counts and low for (N – 1)/2 counts.



NOTE: A GATE transition should not occur one clock prior to terminal count.

Figure 18. Mode 3

MODE 4: SOFTWARE TRIGGERED STROBE

OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is "triggered" by writing the initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

After writing a Control Word and initial count, the Counter will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1) Writing the first byte has no effect on counting.
2) Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be "retriggered" by software. OUT strobes low N + 1 CLK pulses after the new count of N is written.

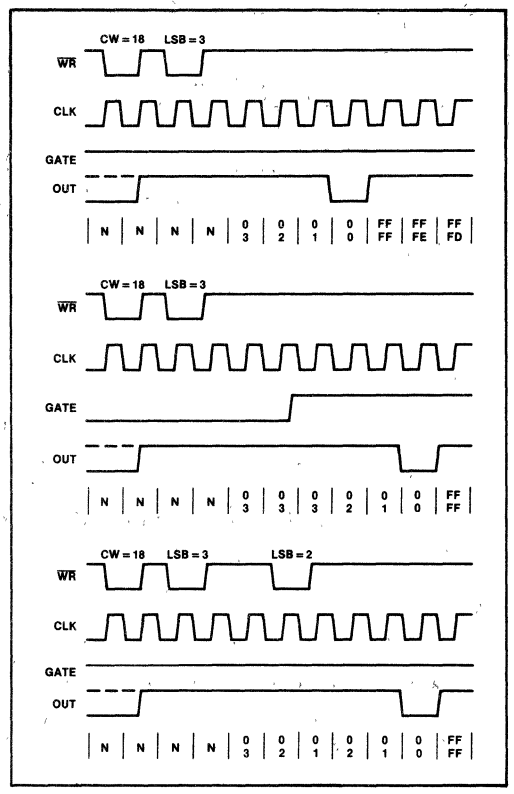


Figure 19. Mode 4

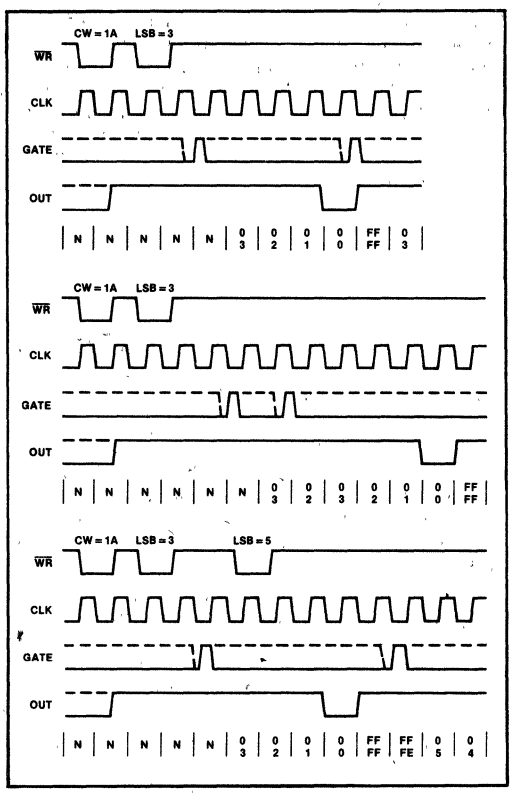


Figure 20. Mode 5

MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE)

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again.

After writing the Control Word and initial count, the counter will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after a trigger.

A trigger results in the Counter being loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable. OUT will not strobe low for N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the curent counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

Signal Status Modes	Low Or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	1) Initiates counting 2) Resets output after next clock	—
2	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
3	1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

Figure 21. Gate Pin Operations Summary

Mode	Min Count	Max Count
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

NOTE: 0 IS EQUIVALENT TO 2^{16} FOR BINARY COUNTING AND 10^4 FOR BCD COUNTING.

Figure 22. Minimum and Maximum Initial Counts

Operation Common to All Modes

PROGRAMMING

When a Control Word is written to a Counter, all Control Logic is immediately reset and OUT goes to a known initial state; no CLK pulses are required for this.

GATE

The GATE input is always sampled on the rising edge of CLK. In Modes 0, 2, 3, and 4 the GATE input is level sensitive, and the logic level is sampled on the rising edge of CLK. In Modes 1, 2, 3, and 5 the GATE input is rising-edge sensitive. In these Modes, a rising edge of GATE (trigger) sets an edge-sensitive flip-flop in the Counter. This flip-flop is then sampled on the next rising edge of CLK; the flip-flop is reset immediately after it is sampled. In this way, a trigger will be detected no matter when it occurs—a high logic level does not have to be maintained until the next rising edge of CLK. Note that in Modes 2 and 3, the GATE input is both edge- and level-sensitive. In Modes 2 and 3, if a CLK source other than the system clock is used, GATE should be pulsed immediately following $\overline{WR}$ of a new count value.

COUNTER

New counts are loaded and Counters are decremented on the falling edge of CLK.

The largest possible initial count is 0; this is equivalent to 2^{16} for binary counting and 10^4 for BCD counting.

The Counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5 the Counter "wraps around" to the highest count, either FFFF hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic; the Counter reloads itself with the initial count and continues counting from there.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to Absolute Maximum Rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
I_{IL}	Input Load Current		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{CC}	V_{CC} Supply Current		170	mA	

CAPACITANCE (T_A=25°C, V_{CC}=GND=0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$, GND = 0V)

Bus Parameters (Note 1)

READ CYCLE

Symbol	Parameter	8254		8254-2		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{RD}\downarrow$	45		30		ns
t_{SR}	$\overline{CS}$ Stable Before $\overline{RD}\downarrow$	0		0		ns
t_{RA}	Address Hold Time After $\overline{RD}\uparrow$	0		0		ns
t_{RR}	$\overline{RD}$ Pulse Width	150		95		ns
t_{RD}	Data Delay from $\overline{RD}\downarrow$		120		85	ns
t_{AD}	Data Delay from Address		220		185	ns
t_{DF}	$\overline{RD}\uparrow$ to Data Floating	5	90	5	65	ns
t_{RV}	Command Recovery Time	200		165		ns

Note 1: AC timings measured at V_{OH} = 2.0V, V_{OL} = 0.8V.

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8254		8254-2		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{SW}	$\overline{CS}$ Stable Before $\overline{WR}\downarrow$	0		0		ns
t_{WA}	Address Hold Time $\overline{WR}\uparrow$	0		0		ns
t_{WW}	$\overline{WR}$ Pulse Width	150		95		ns
t_{DW}	Data Setup Time Before $\overline{WR}\uparrow$	120		95		ns
t_{WD}	Data Hold Time After $\overline{WR}\uparrow$	0		0		ns
t_{RV}	Command Recovery Time	200		165		ns

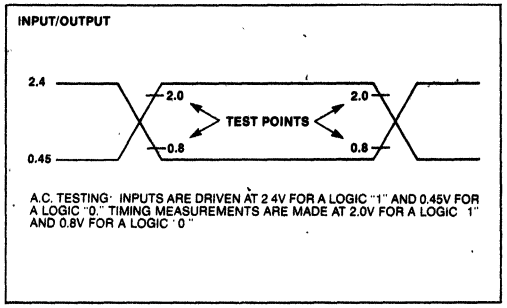
CLOCK AND GATE ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$, GND = 0V)

Symbol	Parameter	8254		8254-2		Unit
		Min.	Max.	Min.	Max.	
t_{CLK}	Clock Period	125	DC	100	DC	ns
t_{PWH}	High Pulse Width	60[3]		30[3]		ns
t_{PWL}	Low Pulse Width	60[3]		50[3]		ns
t_R	Clock Rise Time		25		25	ns
t_F	Clock Fall Time		25		25	ns
t_{GW}	Gate Width High	50		50		ns
t_{GL}	Gate Width Low	50		50		ns
t_{GS}	Gate Setup Time to CLK$\uparrow$	50		40		ns
t_{GH}	Gate Hold Time After CLK$\uparrow$	50[2]		50[2]		ns
t_{OD}	Output Delay from CLK$\downarrow$		150		100	ns
t_{ODG}	Output Delay from Gate$\downarrow$		120		100	ns
t_{WC}	CLK Delay for Loading	0	55	0	55	ns
t_{WG}	Gate Delay for Sampling	−5	50	−5	40	ns
t_{WO}	OUT Delay from Mode Write		260		240	ns
t_{CL}	CLK Set Up for Count Latch	−40	45	−40	40	ns

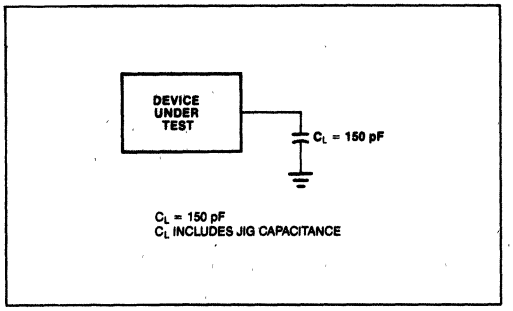
Note 2: In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns (70 ns for the 8254-2) of the rising clock edge may not be detected.

Note 3: Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.

A.C. TESTING INPUT, OUTPUT WAVEFORM

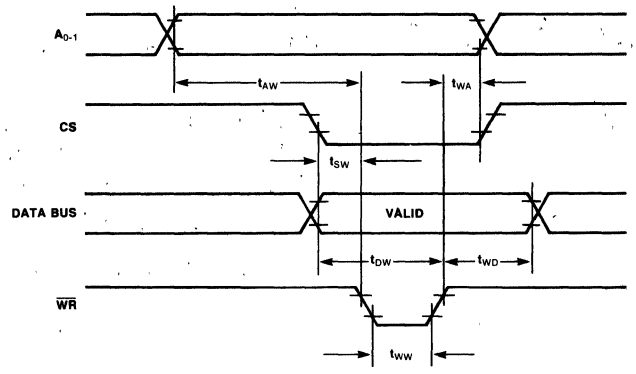


A.C. TESTING LOAD CIRCUIT

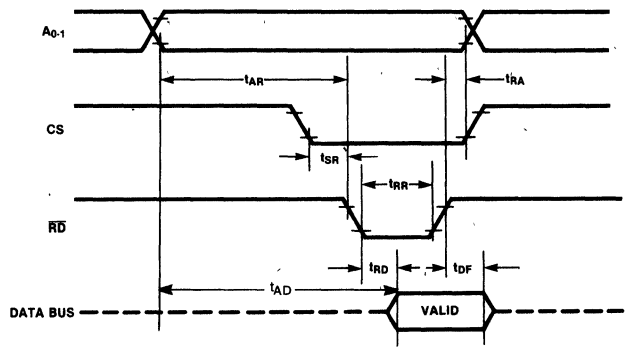


WAVEFORMS

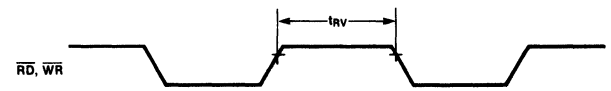
WRITE



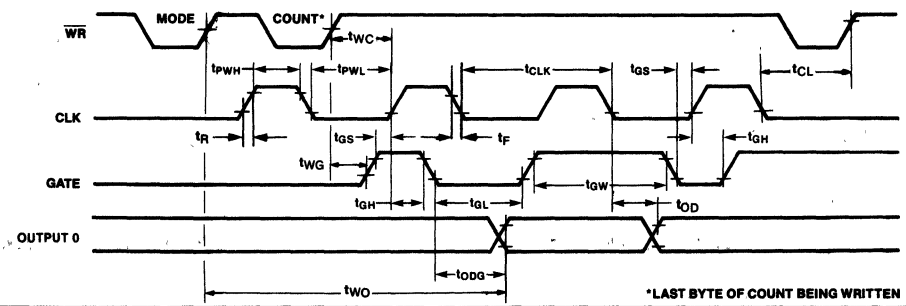
READ



RECOVERY



CLOCK AND GATE



8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- Reduces System Package Count
- Improved DC Driving Capability
- Available in EXPRESS
  - —Standard Temperature Range
  - —Extended Temperature Range

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

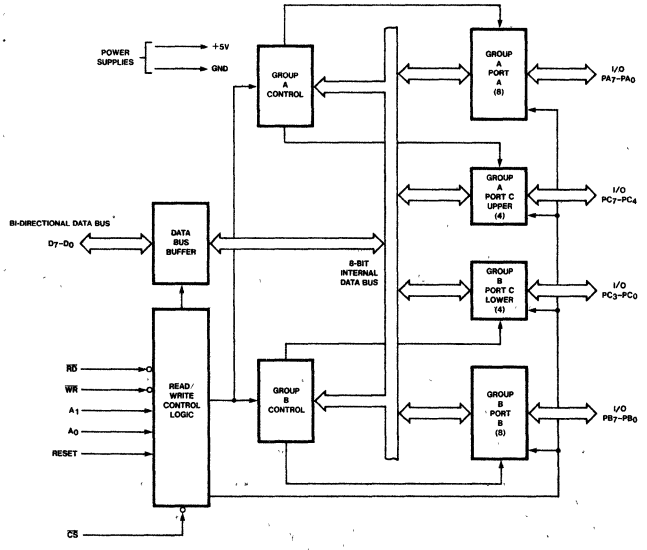


Figure 1. 8255A Block Diagram

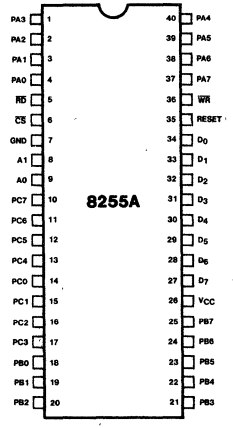


Figure 2. Pin Configuration

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communiction between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A ⇒ DATA BUS
0	1	0	1	0	PORT B ⇒ DATA BUS
1	0	0	1	0	PORT C ⇒ DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS ⇒ PORT A
0	1	1	0	0	DATA BUS ⇒ PORT B
1	0	1	0	0	DATA BUS ⇒ PORT C
1	1	1	0	0	DATA BUS ⇒ CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS ⇒ 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS ⇒ 3-STATE

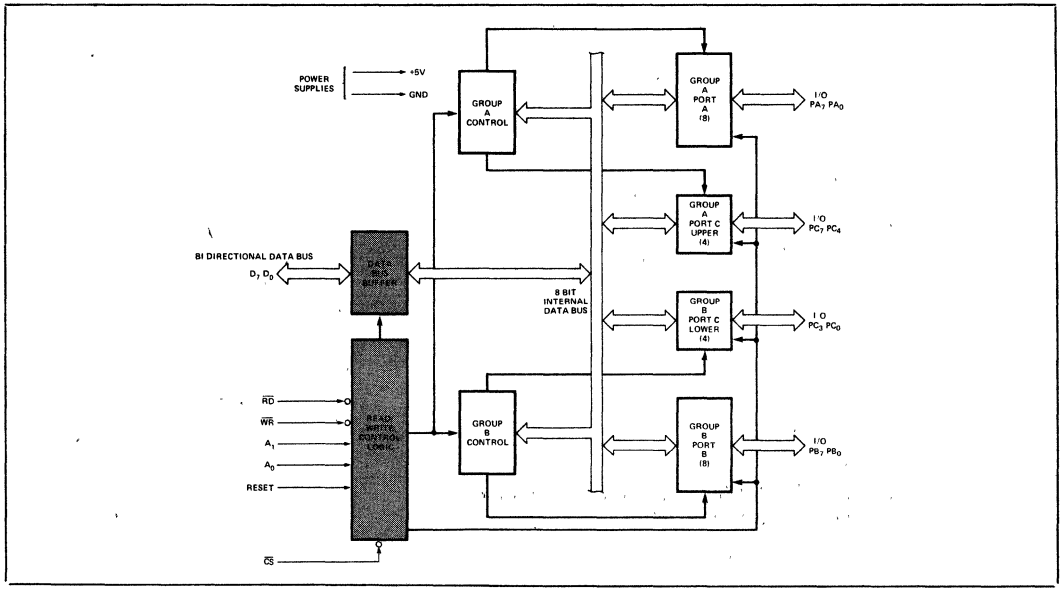


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can **Only** be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

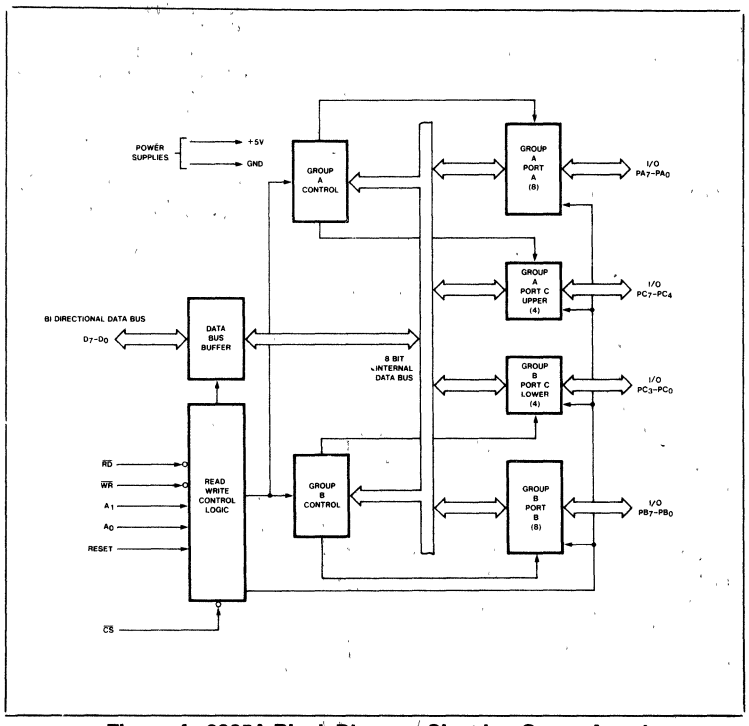
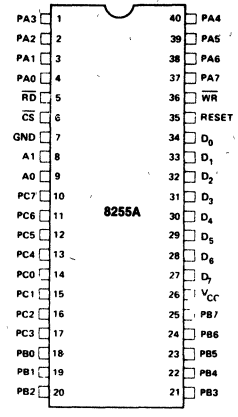


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions

PIN CONFIGURATION



PIN NAMES

D_7 D_0	DATA BUS (BI DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7 PB0	PORT B (BIT)
PC7 PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 – Basic Input/Output
Mode 1 – Strobed Input/Output
Mode 2 – Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

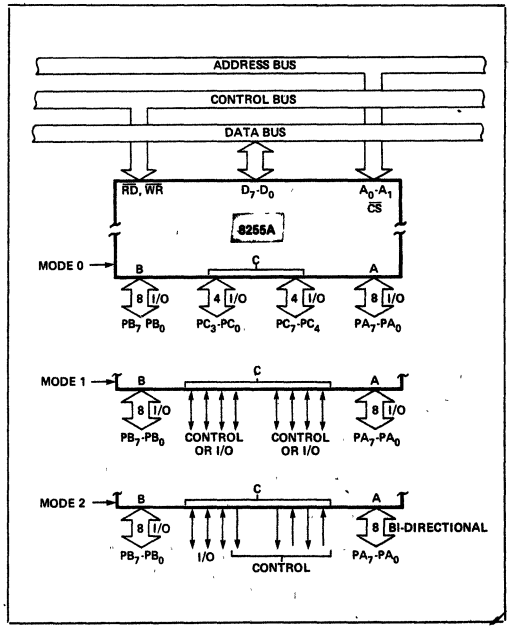


Figure 5. Basic Mode Definitions and Bus Interface

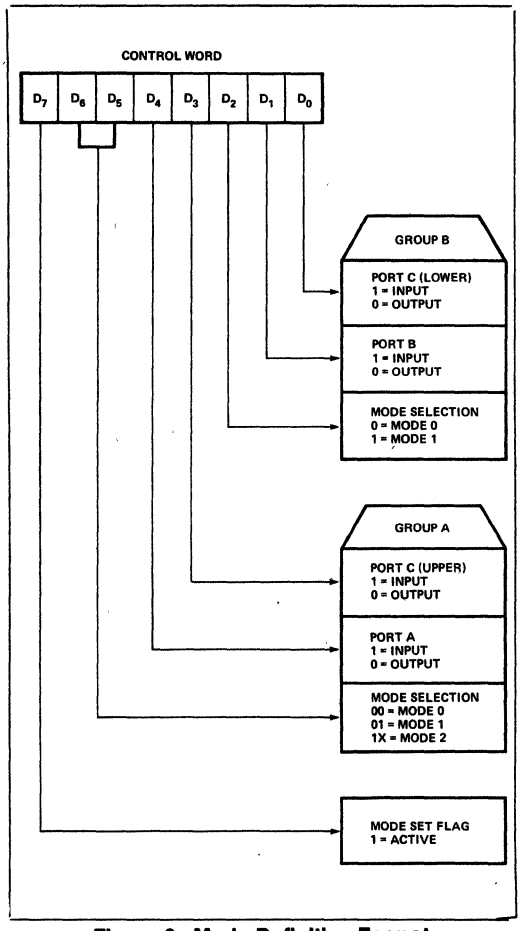


Figure 6. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

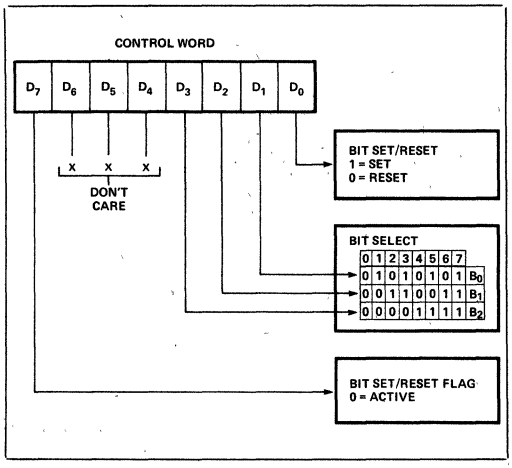


Figure 7. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) — INTE is SET — Interrupt enable
(BIT-RESET) — INTE is RESET — Interrupt disable

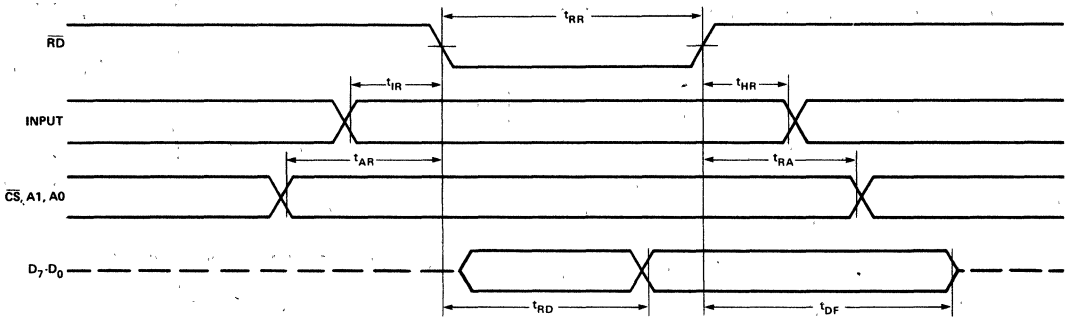
Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

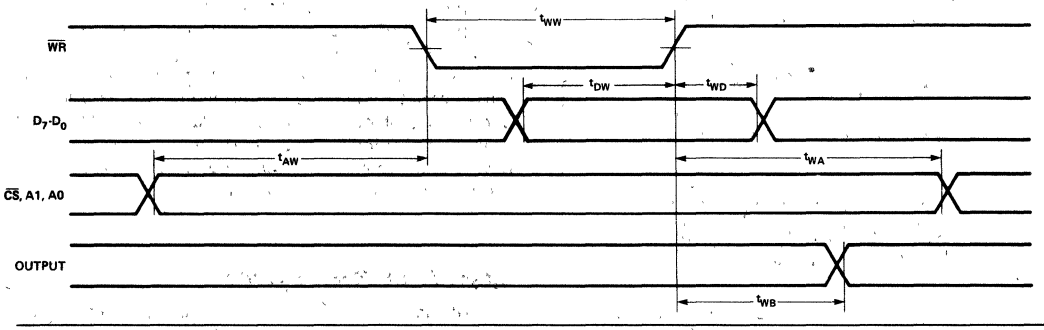
MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



MODE 0 (Basic Input)



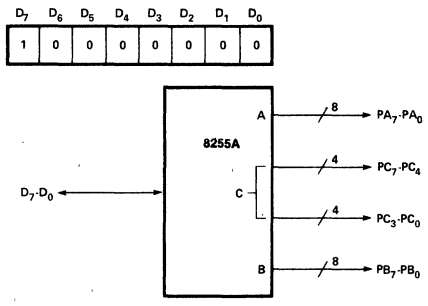
MODE 0 (Basic Output)

MODE 0 Port Definition

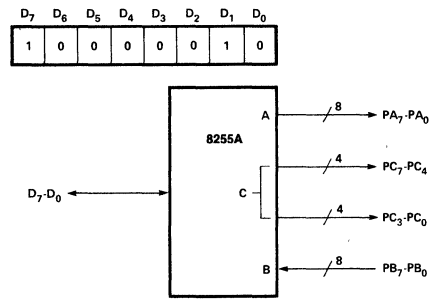
A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE 0 Configurations

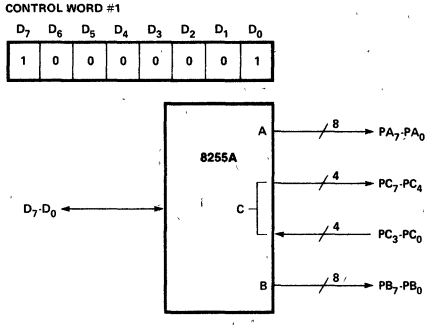
CONTROL WORD =0



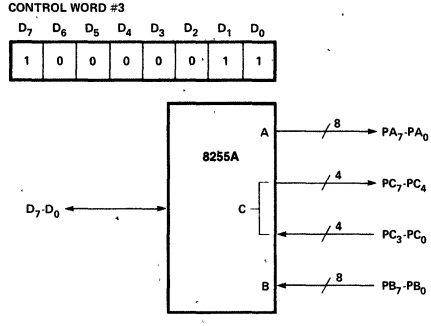
CONTROL WORD #2



CONTROL WORD #1

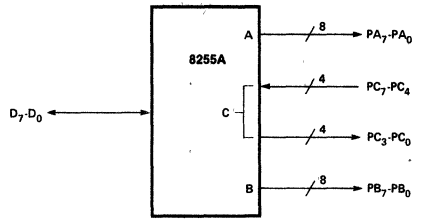


CONTROL WORD #3



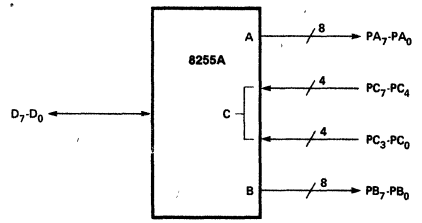
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



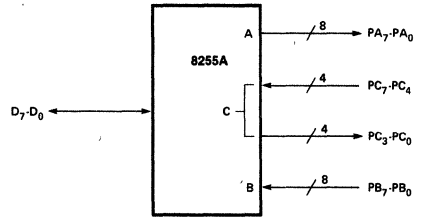
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



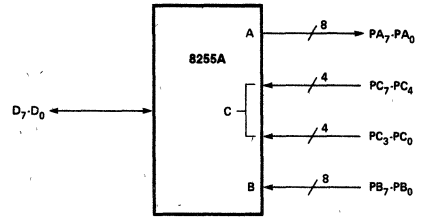
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



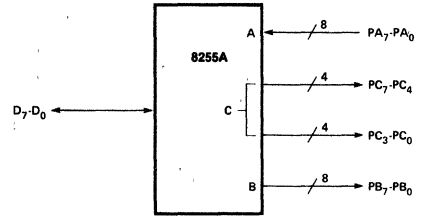
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



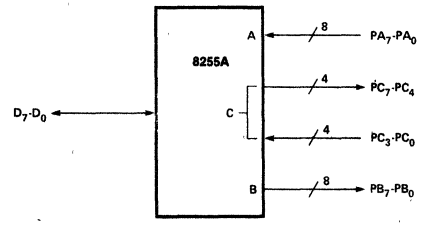
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



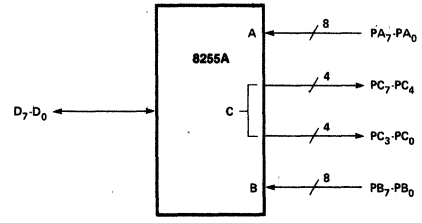
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



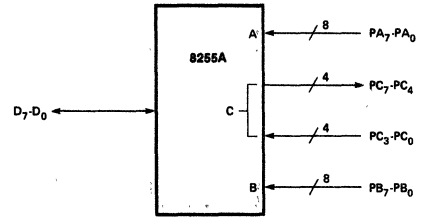
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



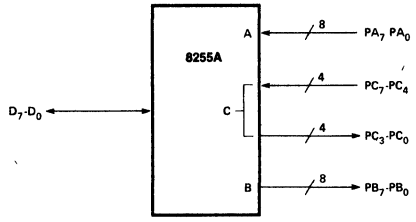
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



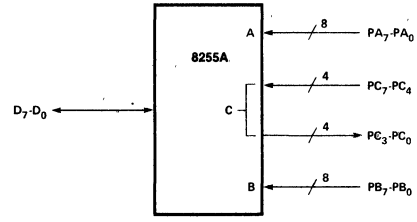
CONTROL WORD #12

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0



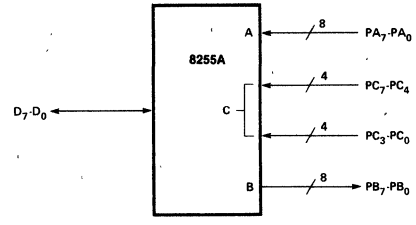
CONTROL WORD #14

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	0



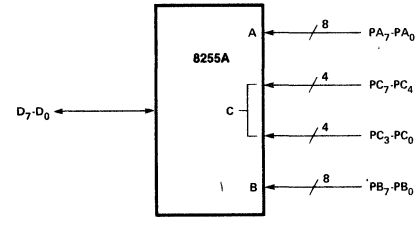
CONTROL WORD #13

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	1



CONTROL WORD #15

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	1



Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and Port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

STB (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

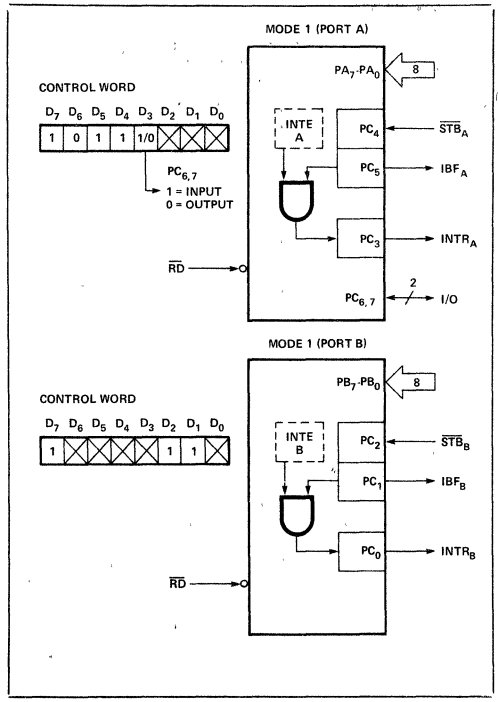


Figure 8. MODE 1 Input

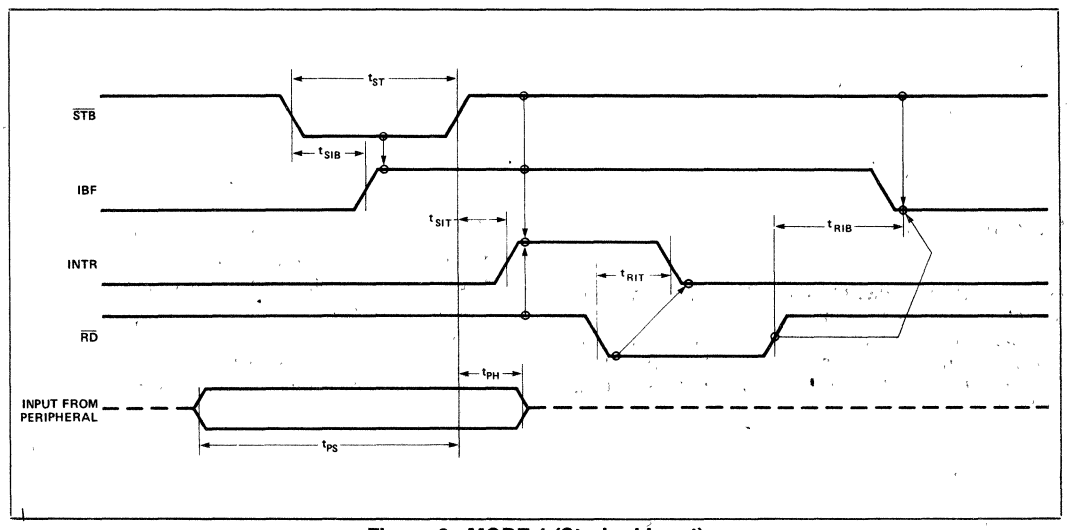


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The OBF output will go "low" to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by ACK Input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a "one", OBF is a "one" and INTE is a "one". It is reset by the falling edge of WR.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

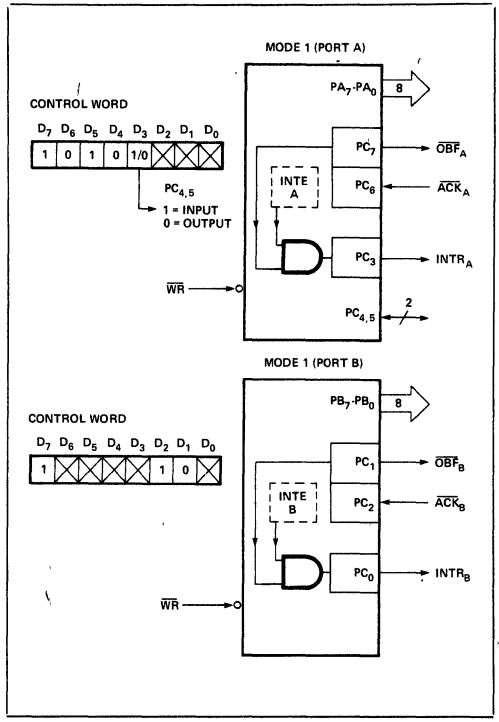


Figure 10. MODE 1 Output

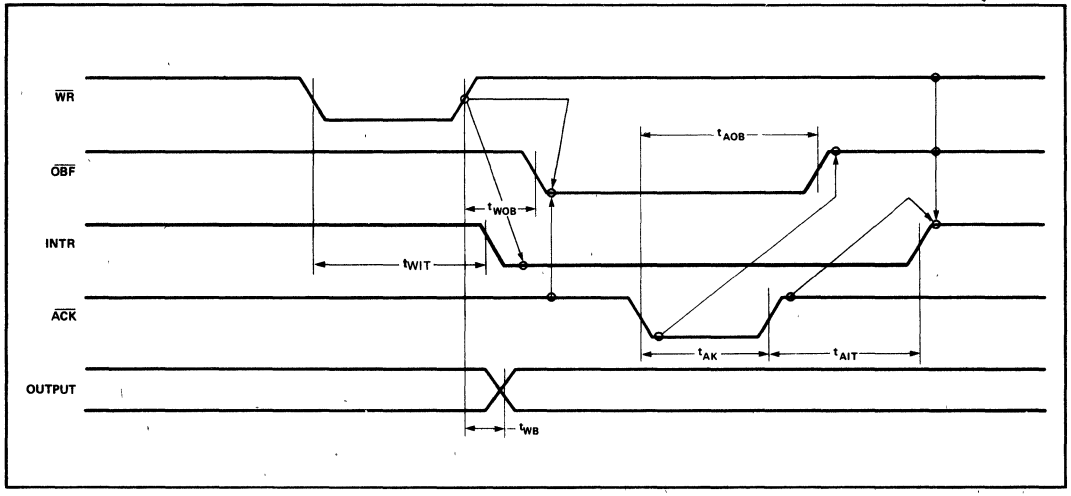


Figure 11. Mode 1 (Strobed Output)

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

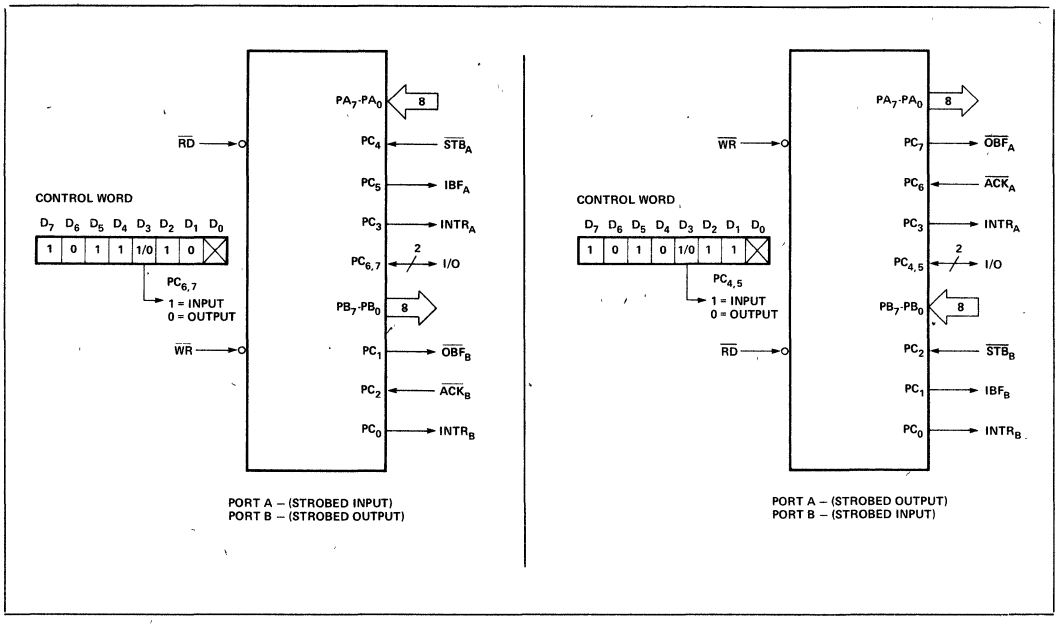


Figure 12. Combinations of MODE 1

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Ful). The OBF output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input)

STB (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

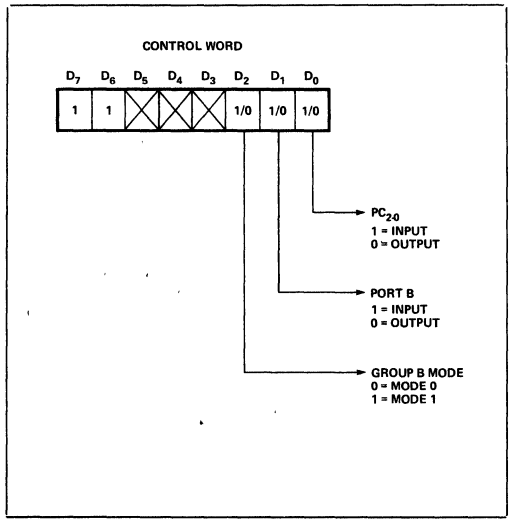


Figure 13. MODE Control Word

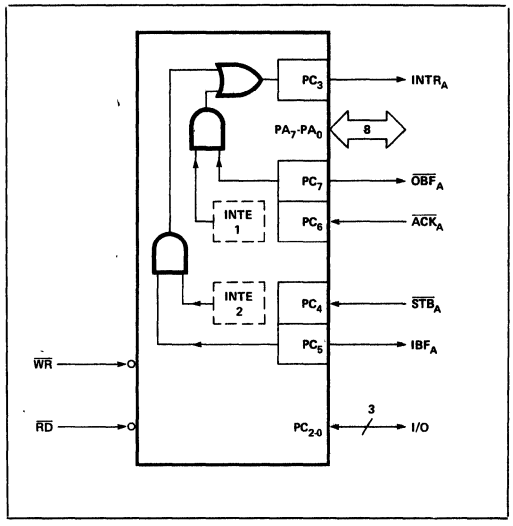


Figure 14. MODE 2

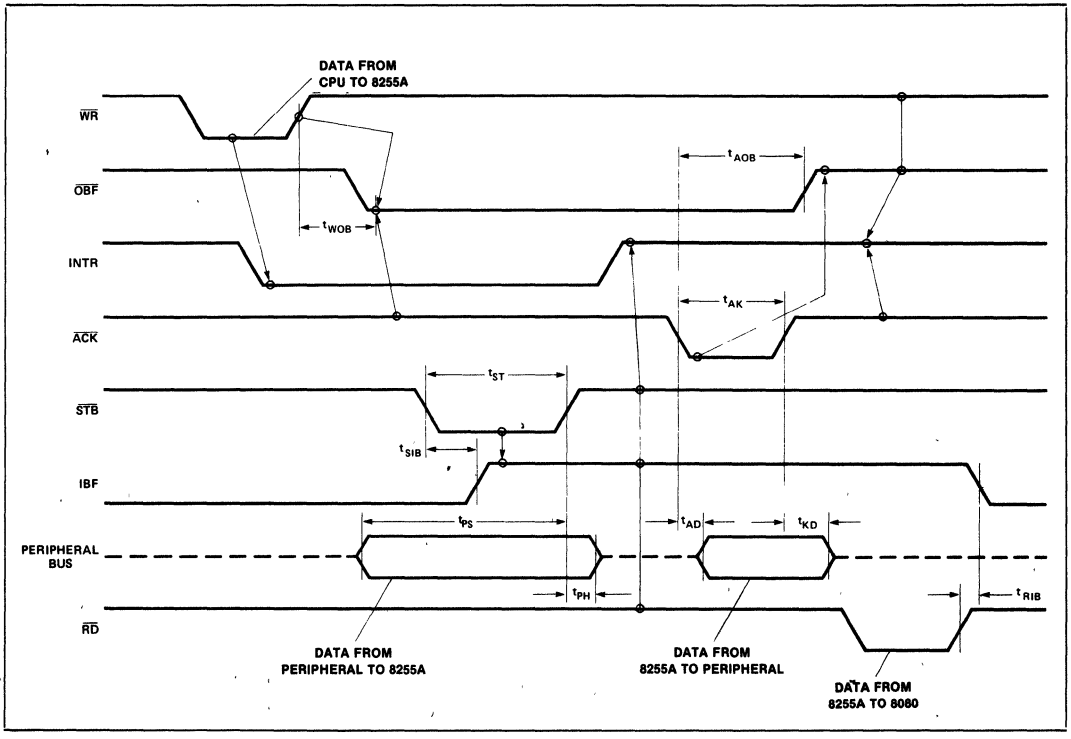


Figure 15. MODE 2 (Bidirectional)

NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
($INTR = IBF \cdot \overline{MASK} \cdot \overline{STB} \cdot \overline{RD} + \overline{OBF} \cdot \overline{MASK} \cdot \overline{ACK} \cdot \overline{WR}$)

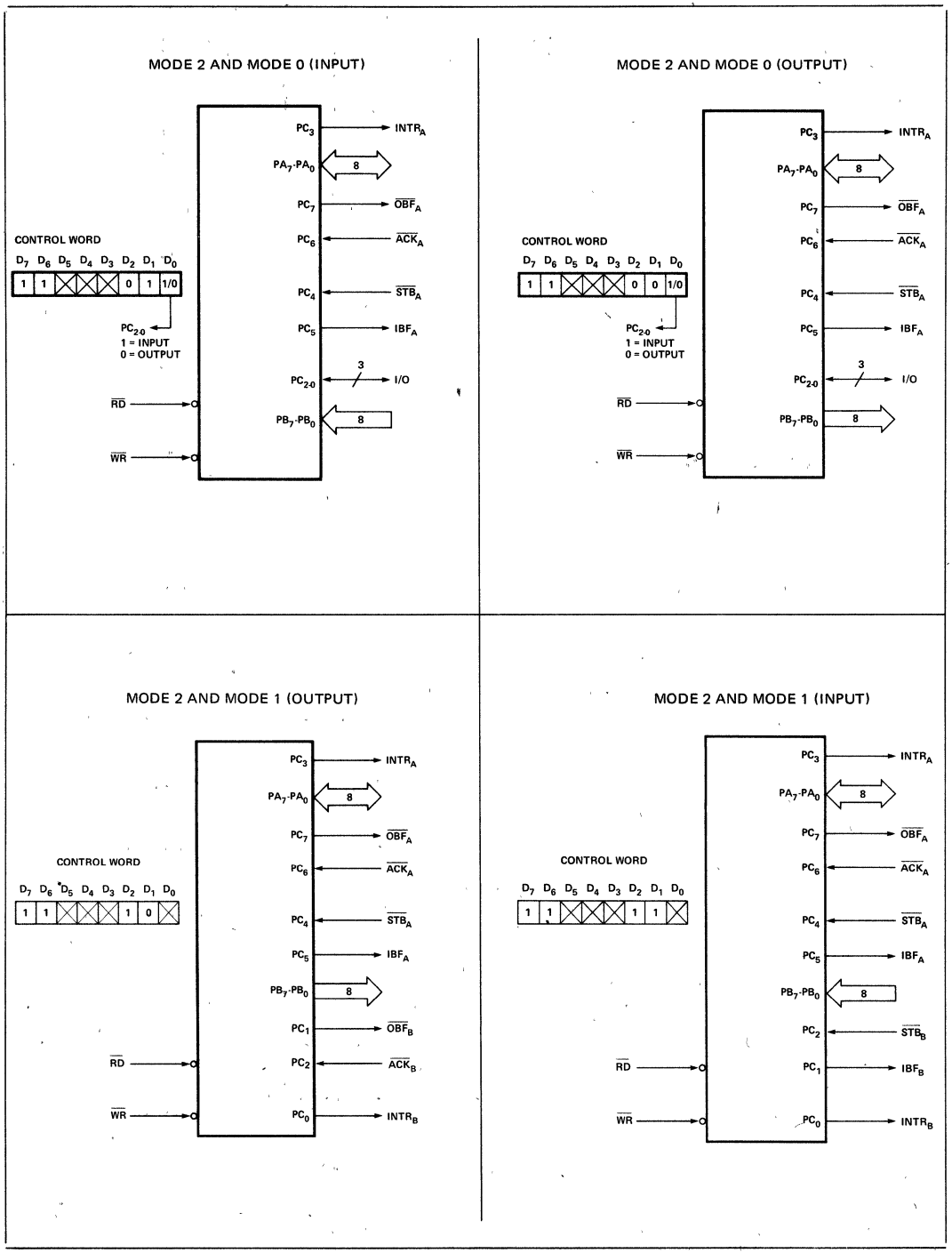


Figure 16. MODE ¼ Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	⟷	
PA_1	IN	OUT	IN	OUT	⟷	
PA_2	IN	OUT	IN	OUT	⟷	
PA_3	IN	OUT	IN	OUT	⟷	
PA_4	IN	OUT	IN	OUT	⟷	
PA_5	IN	OUT	IN	OUT	⟷	
PA_6	IN	OUT	IN	OUT	⟷	
PA_7	IN	OUT	IN	OUT	⟷	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs –
All input lines can be accessed during a normal Port C read.

If Programmed as Outputs –
Bits in C upper (PC_7-PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3-PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

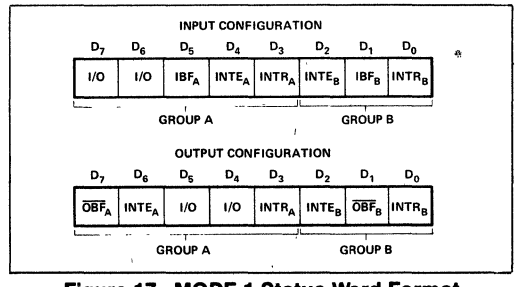


Figure 17. MODE 1 Status Word Format

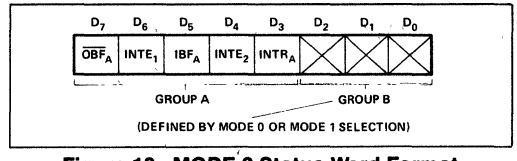


Figure 18. MODE 2 Status Word Format

APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 present a few examples of typical applications of the 8255A.

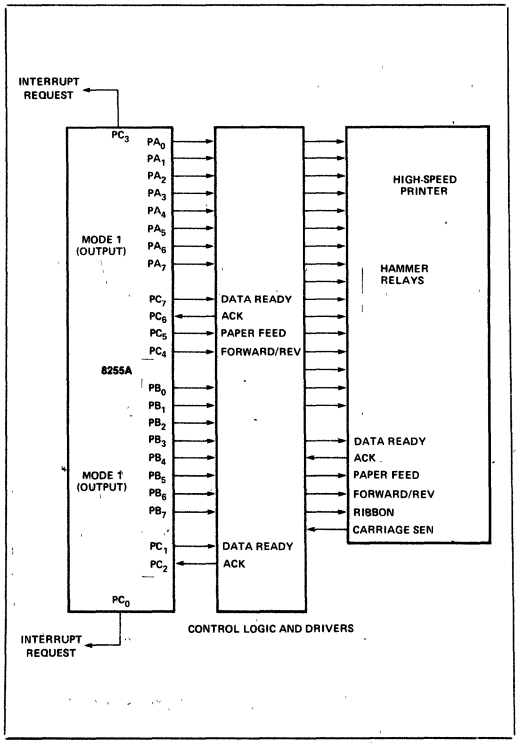


Figure 19. Printer Interface

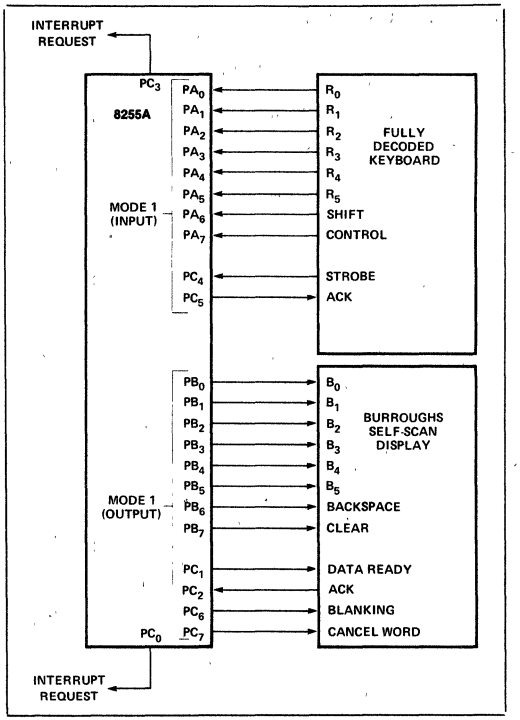


Figure 20. Keyboard and Display Interface

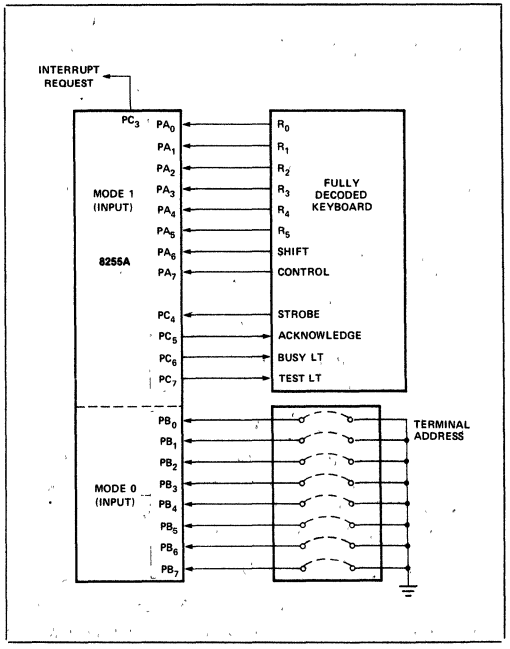


Figure 21. Keyboard and Terminal Address Interface

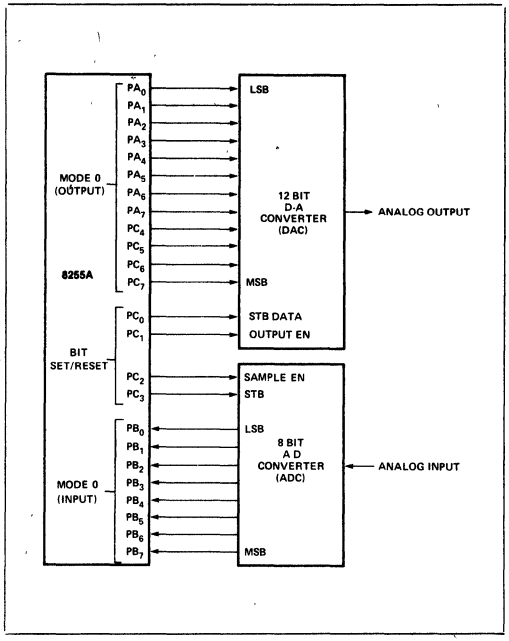


Figure 22. Digital to Analog, Analog to Digital

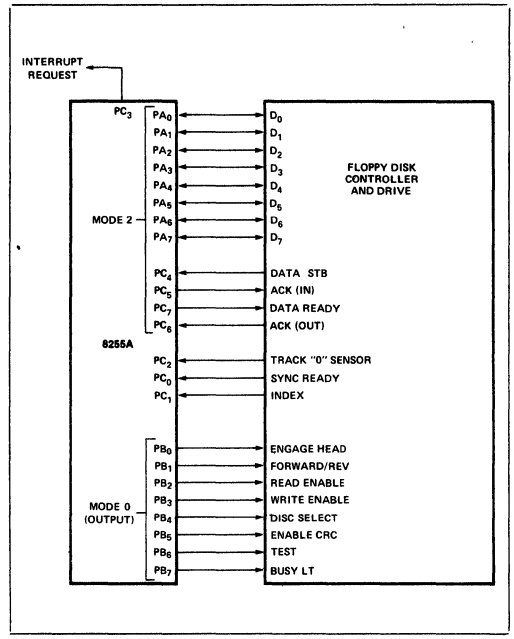


Figure 23. Basic Floppy Disk Interface

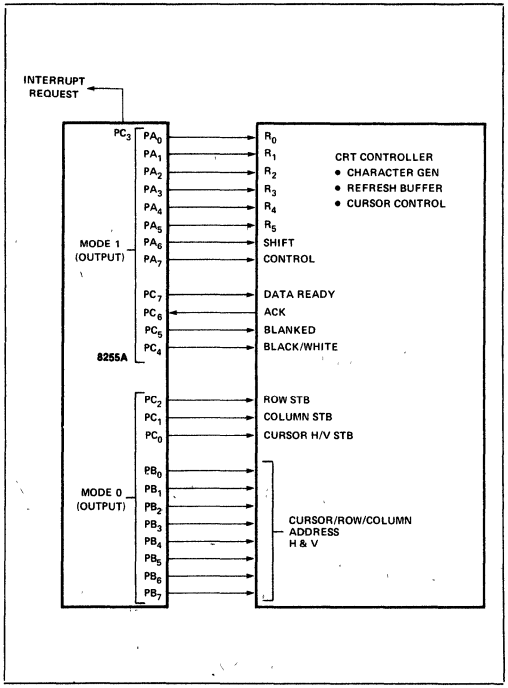


Figure 24. Basic CRT Controller Interface

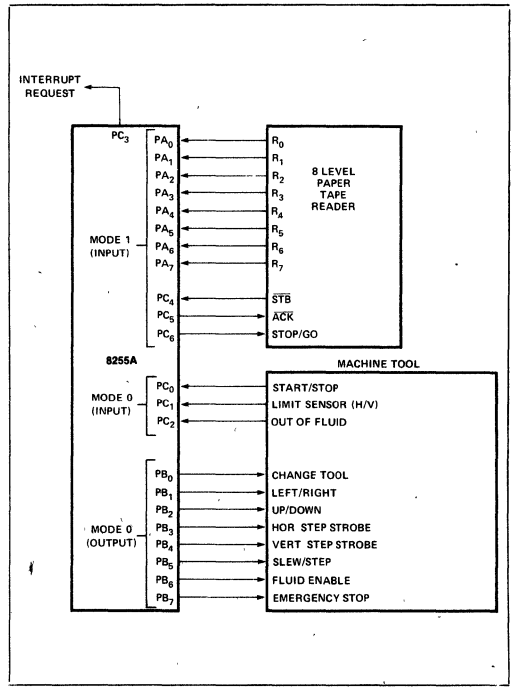


Figure 25. Machine Tool Controller Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . –65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . . –0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V) *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL} (DB)	Output Low Voltage (Data Bus)		0.45*	V	I_{OL} = 2.5mA
V_{OL}(PER)	Output Low Voltage (Peripheral Port)		0.45*	V	I_{OL} = 1.7mA
V_{OH}(DB)	Output High Voltage (Data Bus)	2.4		V	I_{OH} = -400μA
V_{OH}(PER)	Output High Voltage (Peripheral Port)	2.4		V	I_{OH} = -200μA
I_{DAR}[1]	Darlington Drive Current	-1.0	-4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to .45V

NOTE:
1. Available on any 8 pins from Port B and C.

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%, GND = 0V) *

Bus Parameters

READ

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before READ	0		0		ns
t_{RA}	Address Stable After READ	0		0		ns
t_{RR}	READ Pulse Width	300		300		ns
t_{RD}	Data Valid From READ[1]		250		200	ns
t_{DF}	Data Float After READ	10	150	10	100	ns
t_{RV}	Time Between READs and/or WRITEs	850		850		ns

A.C. CHARACTERISTICS (Continued)

WRITE

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before WRITE	0		0		ns
t_{WA}	Address Stable After WRITE	20		20		ns
t_{WW}	WRITE Pulse Width	400		300		ns
t_{DW}	Data Valid to WRITE (T.E.)	100		100		ns
t_{WD}	Data Valid After WRITE	30		30		ns

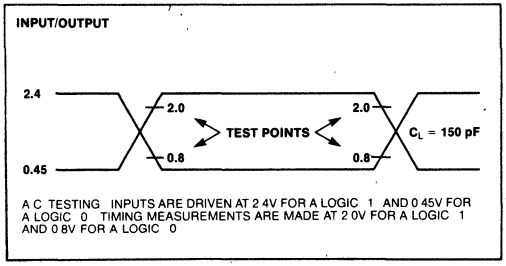
OTHER TIMINGS

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{WB}	WR = 1 to Output[1]		350		350	ns
t_{IR}	Peripheral Data Before RD	0		0		ns
t_{HR}	Peripheral Data After RD	0		0		ns
t_{AK}	ACK Pulse Width	300		300		ns
t_{ST}	STB Pulse Width	500		500		ns
t_{PS}	Per. Data Before T.E. of STB	0		0		ns
t_{PH}	Per. Data After T.E. of STB	180		180		ns
t_{AD}	ACK = 0 to Output[1]		300		300	ns
t_{KD}	ACK = 1 to Output Float	20	250	20	250	ns
t_{WOB}	WR = 1 to OBF = 0[1]		650		650	ns
t_{AOB}	ACK = 0 to OBF = 1[1]		350		350	ns
t_{SIB}	STB = 0 to IBF = 1[1]		300		300	ns
t_{RIB}	RD = 1 to IBF = 0[1]		300		300	ns
t_{RIT}	RD = 0 to INTR = 0[1]		400		400	ns
t_{SIT}	STB = 1 to INTR = 1[1]		300		300	ns
t_{AIT}	ACK = 1 to INTR = 1[1]		350		350	ns
t_{WIT}	WR = 0 to INTR = 0[1,3]		450		450	ns

NOTES:
1. Test Conditions: C_L = 150 pF.
2. Period of Reset pulse must be at least 50μs during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR↑ may occur as early as $\overline{WR}$↓.

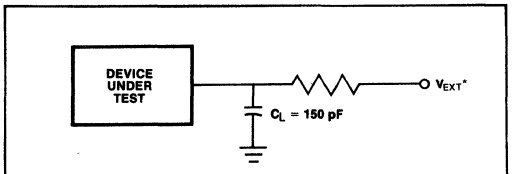
* For Extended Temperature EXPRESS, use M8255A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

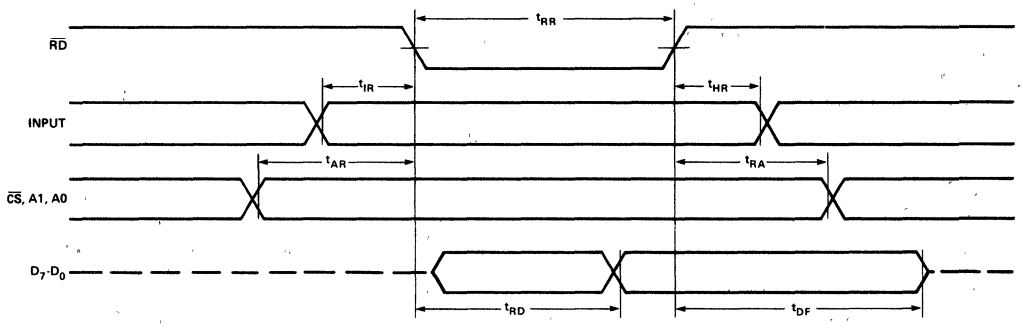
A.C. TESTING LOAD CIRCUIT



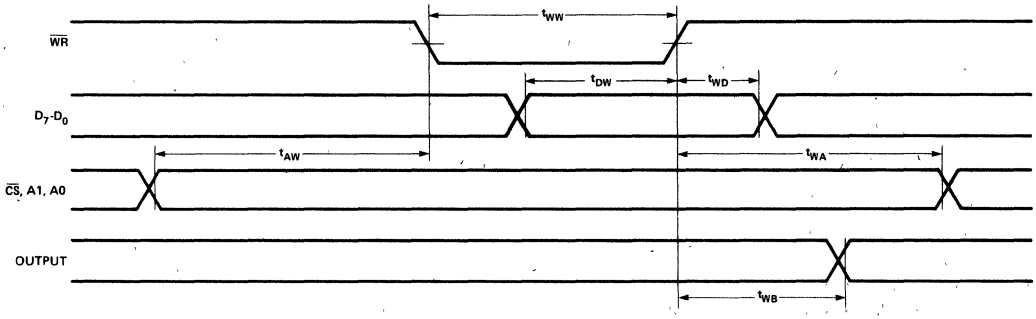
*V_{EXT} IS SET AT VARIOUS VOLTAGES DURING TESTING TO GUARANTEE THE SPECIFICATION C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

MODE 0 (BASIC INPUT)

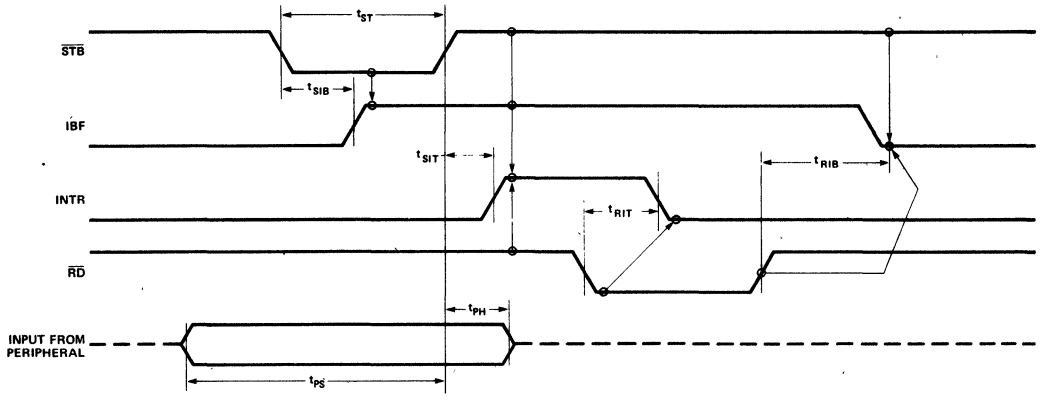


MODE 0 (BASIC OUTPUT)

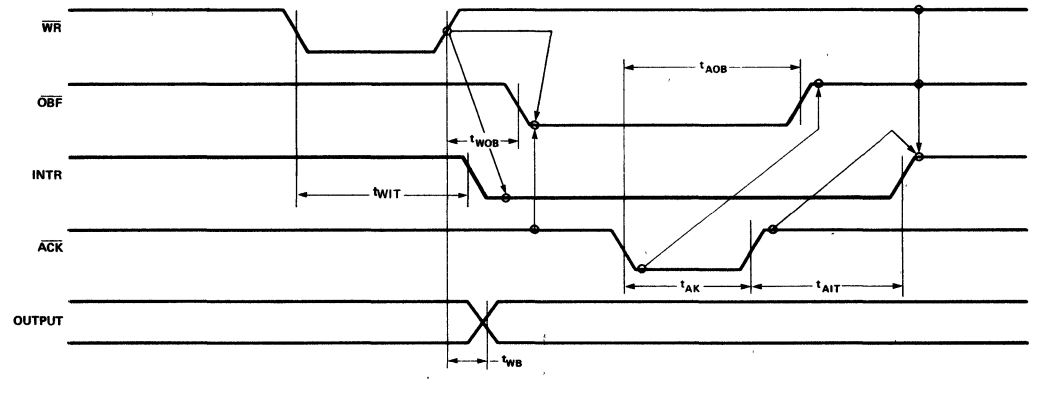


WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

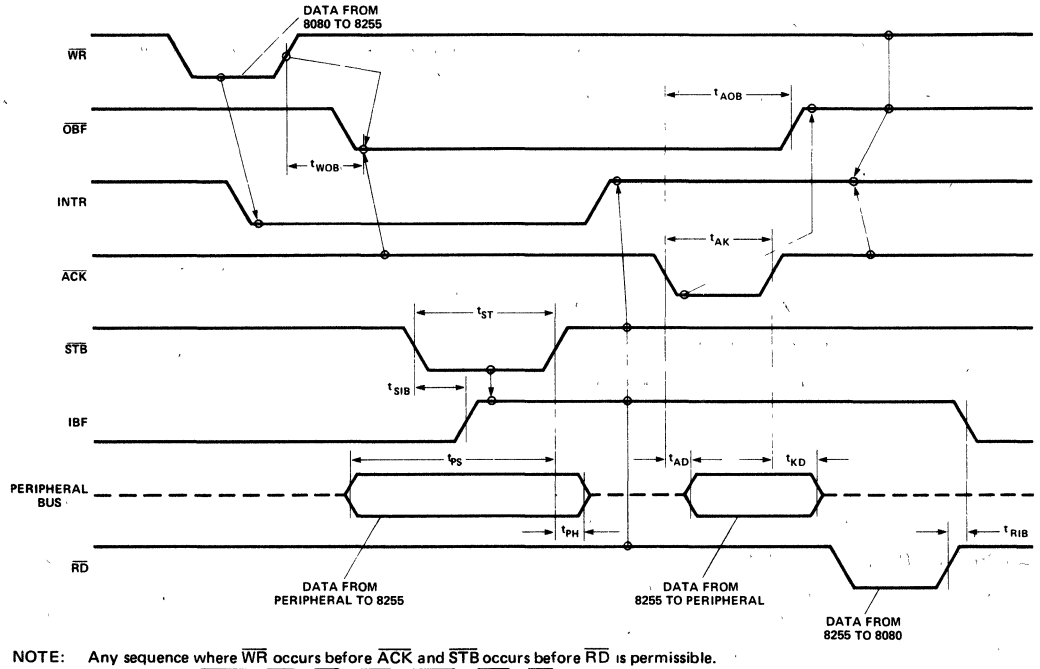


MODE 1 (STROBED OUTPUT)



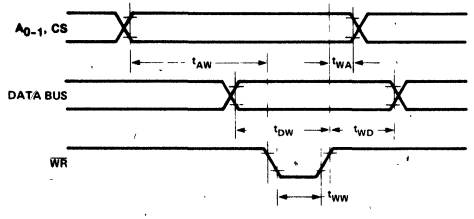
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

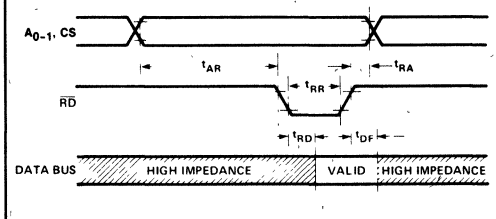


NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

WRITE TIMING



READ TIMING



ADVANCE INFORMATION

8256AH
MULTIFUNCTION UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER (MUART)

- **Programmable Serial Asynchronous Communications Interface for 5-, 6-, 7-, or 8-Bit Characters, 1, 1½, or 2 Stop Bits, and Parity Generation**
- **On-Board Baud Rate Generator Programmable for 13 Common Baud Rates up to 19.2K Bits/second, or an External Baud Clock Maximum of 1M Bit/second**
- **Five 8-Bit Programmable Timer/Counters; Four Can Be Cascaded to Two 16-Bit Timer/Counters**
- **Two 8-Bit Programmable Parallel I/O Ports; Port 1 Can Be Programmed for Port 2 Handshake Controls and Event Counter Inputs**
- **Eight-Level Priority Interrupt Controller Programmable for 8085 or iAPX 86, iAPX 88 Systems and for Fully Nested Interrupt Capability**
- **Programmable System Clock to 1 ×, 2 ×, 3 ×, or 5 × 1.024 MHz**

The Intel® 8256AH Multifunction Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. It is designed to interface to the 8086/88, iAPX 186/188, and 8051 to perform serial communications, parallel I/O, timing, event counting, and priority interrupt functions. All of these functions are fully programmable through nine internal registers. In addition, the five timer/counters and two parallel I/O ports can be accessed directly by the microprocessor.

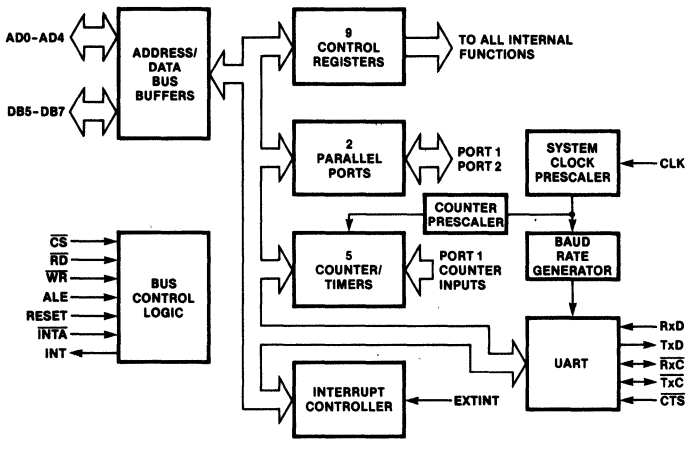


Figure 1. MUART Block Diagram

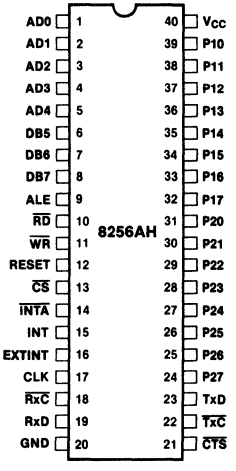


Figure 2. MUART Pin Configuration

AUGUST 1983
ORDER NUMBER: 230759-001

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AD0-AD4 DB5-DB7	1-5 6-8	I/O	**ADDRESS/DATA:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**ADDRESS LATCH ENABLE:** Latches the 5 address lines on AD0-AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**READ CONTROL:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**WRITE CONTROL:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**RESET:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**CHIP SELECT:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.
$\overline{INTA}$	14	I	**INTERRUPT ACKNOWLEDGE:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	O	**INTERRUPT REQUEST:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**EXTERNAL INTERRUPT:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an $\overline{INTA}$ occurs or the interrupt address register is read.
CLK	17	I	**SYSTEM CLOCK:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**RECEIVE CLOCK:** If the baud rate bits in the Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**RECEIVE DATA:** Serial data input.
GND	20	PS	**GROUND:** Power supply and logic ground reference.

Table 1. Pin Description (continued)

Symbol	Pin	Type	Name and Function
$\overline{CTS}$	21	I	**CLEAR TO SEND:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitter serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-OFH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where ½ the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs. If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**TRANSMIT CLOCK:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**TRANSMIT DATA:** Serial data output.
P27-P20	24-31	I/O	**PARALLEL I/O PORT 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	I/O	**PARALLEL I/O PORT 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
V_{CC}	40	PS	**POWER:** +5V power supply.

FUNCTIONAL DESCRIPTION

The 8256AH Multi-Function Universal Asynchronous Receiver-Transmitter (MUART) combines five commonly used functions into a single 40-pin device. The MUART performs asynchronous serial communications, parallel I/O, timing, event counting, and interrupt control. For detailed application information, see Intel Ap Note #153, Designing with the 8256.

Serial Communications

The serial communications portion of the MUART contains a full-duplex asynchronous receiver-transmitter (UART). A programmable baud rate generator is included on the MUART to permit a variety of operating speeds without external components. The UART can be programmed by the CPU for a variety of character sizes, parity generation and detection, error detection, and start/stop bit handling. The receiver checks the start and stop bits in the center of the bit, and a break halts the reception of data. The transmitter can send breaks and can be controlled by an external enable pin.

Parallel I/O

The MUART includes 16 bits of general purpose parallel I/O. Eight bits (Port 1) can be individually changed from input to output or used for special I/O functions. The other eight bits (Port 2) can be used as nibbles (4 bits) or as bytes. These eight bits also include a handshaking capability using two pins on Port 1.

Counter/Timers

There are five 8-bit counter/timers on the MUART. The timers can be programmed to use either a 1 kHz or 16 kHz clock generated from the system clock. Four of the 8-bit counter/timers can be cascaded to two 16-bit counter/timers, and one of the 8-bit counter/timers can be reset to its initial value by an external signal.

Interrupts

An eight-level priority interrupt controller can be configured for fully nested or normal interrupt priority. Seven of the eight interrupts service functions on the MUART (counter/timers, UART), and one external interrupt is provided which can be used for a particular function or for chaining interrupt controllers or more MUARTs. The MUART will support 8085 and 8086/88 systems with direct interrupt vectoring, or the MUART can be polled to determine the cause of the interrupt. If additional interrupt control capability is needed, the MUART's interrupt controller can be cascaded into another MUART, into an Intel 8259A Programmable Interrupt Controller, or into the interrupt controller of the iAPX 186/188 High-Integration Microprocessor.

INITIALIZATION

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Byte 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. The MUART should be reset before initialization. (Either a hardware or a software reset will do.)

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 80186 microprocessor. Figure 3 displays the block diagram for this interface. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte.

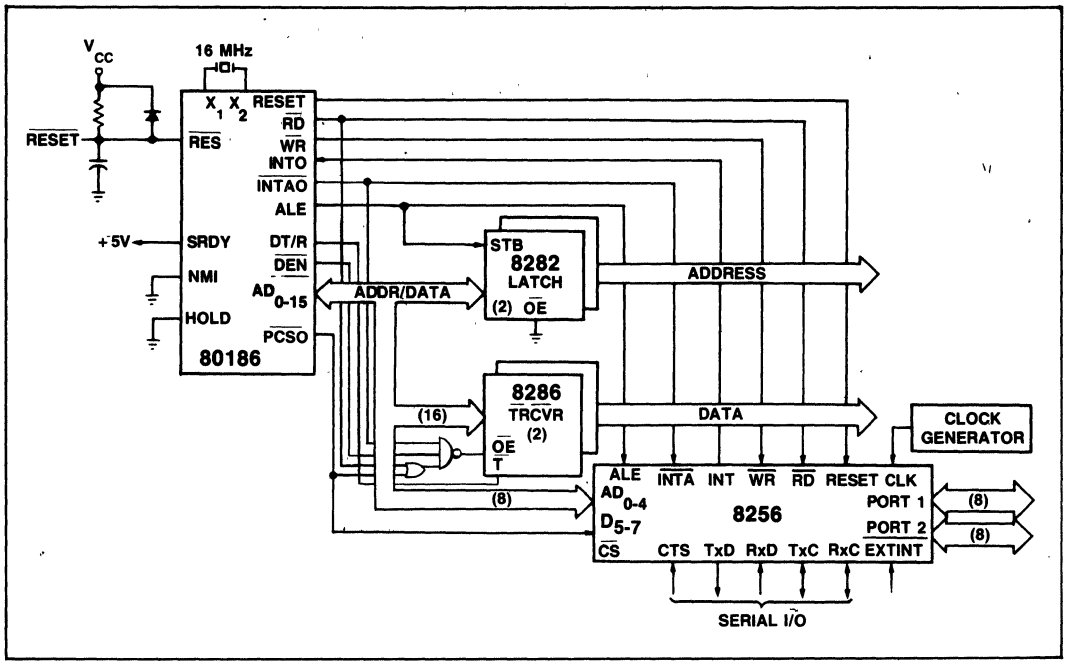


Figure 3. 80186/8256 Interface

the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 2 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)					(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16 kHz (62.5μs). If FRQ = 1, the timer input frequency is 1 KHz (1 ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first INTA. In In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to INTA is for 8086 interrupts where the first INTA is ignored, and an interrupt vector (40H to 47H is placed on the bus in response to the second INTA.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

Table 2. MUART Registers

Read Registers	8085 Mode: AD3 AD2 AD1 AD0 8086 Mode: AD4 AD3 AD2 AD1	Write Registers
L1 L0 S1 S0 BRKI BITI 8086 FRQ Command 1	0 0 0 0	L1 L0 S1 S0 BRKI BITI 8086 FRQ Command 1
PEN EP C1 C0 B3 B2 B1 B0 Command 2	0 0 0 1	PEN EP C1 C0 B3 B2 B1 B0 Command 2
0 RxE IAE NIE 0 SBRK TBRK 0 Command 3	0 0 1 0	SET RxE IAE NIE END SBRK TBRK RST Command 3
T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0 Mode	0 0 1 1	T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0 Mode
P17 P16 P15 P14 P13 P12 P11 P10 Port 1 Control	0 1 0 0	P17 P16 P15 P14 P13 P12 P11 P10 Port 1 Control
L7 L6 L5 L4 L3 L2 L1 L0 Interrupt Enable	0 1 0 1	L7 L6 L5 L4 L3 L2 L1 L0 Set Interrupts
D7 D6 D5 D4 D3 D2 D1 D0 Interrupt Address	0 1 1 0	L7 L6 L5 L4 L3 L2 L1 L0 Reset Interrupts
D7 D6 D5 D4 D3 D2 D1 D0 Receiver Buffer	0 1 1 1	D7 D6 D5 D4 D3 D2 D1 D0 Transmitter Buffer
D7 D6 D5 D4 D3 D2 D1 D0 Port 1	1 0 0 0	D7 D6 D5 D4 D3 D2 D1 D0 Port 1
D7 D6 D5 D4 D3 D2 D1 D0 Port 2	1 0 0 1	D7 D6 D5 D4 D3 D2 D1 D0 Port 2
D7 D6 D5 D4 D3 D2 D1 D0 Timer 1	1 0 1 0	D7 D6 D5 D4 D3 D2 D1 D0 Timer 1
D7 D6 D5 D4 D3 D2 D1 D0 Timer 2	1 0 1 1	D7 D6 D5 D4 D3 D2 D1 D0 Timer 2
D7 D6 D5 D4 D3 D2 D1 D0 Timer 3	1 1 0 0	D7 D6 D5 D4 D3 D2 D1 D0 Timer 3
D7 D6 D5 D4 D3 D2 D1 D0 Timer 4	1 1 0 1	D7 D6 D5 D4 D3 D2 D1 D0 Timer 4
D7 D6 D5 D4 D3 D2 D1 D0 Timer 5	1 1 1 0	D7 D6 D5 D4 D3 D2 D1 D0 Timer 5
INT RBF TBE TRE BD PE OE FE Status	1 1 1 1	0 RS4 RS3 RS2 RS1 RS0 TME DSC Modification

BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
(1R)				(1W)			

Programming bits 0. . .3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0. . .3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0. . .32 Kbaud.

If bits 0. . .3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, ampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input: 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of every data bit of a character. Thus, data can be loaded, e.g., into a shift register externally. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIW	END	SBRK	TBRK	RST
	(2R)				(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1. All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.
2. The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.
3. The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.
4. If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has not effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of CTS is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be re-enabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle (marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit reset the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section of AP-153 under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{INTA}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value ≥ 3H.

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake	Input	
1	0	1	Byte Handshake	Output	
1	1	0	*DO NOT USE*		
1	1	1	Test		

NOTE:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with an initial value. The first trigger pulse causes the initial value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse.

When a high-to-low transition occurs, Timer 5 is preset to its saved value, But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 3.

NOTE:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4W)					(4W)		

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2 $\overline{OBF}$ indicates that a character has been loaded

Table 3. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	—	Internal clock
2	8-bit timer	T24=0, CT2=0	Internal clock
	8-bit event counter	T24=0, CT2=1	P12 pin 37
2	8-bit timer	T35=0, CT3=0	Internal clock
	8-bit event counter	T35=0, CT3=1	P13 pin 36
4	8-bit timer	T24=0	Internal clock
5	8-bit timer, normal mode	T35=0, T5C=0	Internal clock
	8-bit timer, retriggerable mode	T35=0, T5C=1	Internal clock
2 and 4 cascaded	16-bit timer	T24=1, CT2=0	Internal clock
	16-bit event counter	T24=1, CT2=1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35=1, T5C=0, CT3=0	Internal clock
	16-bit event counter, normal mode	T35=1, T5C=0, CT3=1	P13 pin 36
	16-bit timer, retriggerable mode	T35=1, T5C=1, CT3=0	Internal clock
	16-bit event counter, retriggerable mode	T35=1, T5C=1, CT3=1	P13 pin 36

into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. IBF is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0
	(5R)				(5W=enable, (6W=disable)		

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register

0	0	0	D4	D3	D2	0	0
(6R)						Interrupt Level Indication	

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge INTA; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0
	(7R)				(7W)		

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set.

Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming CTS is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
(8R)					(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
(9R)					(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
($0A_{16}$-$0E_{16}$R)					($0A_{16}$-$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while RD is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded, writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X *256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
($0F_{16}$R)							

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is required to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section of AP-153 under "Receive Break Detect" and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If CTS is low, the Transmitter Register will be loaded during the transmission of the start bit. If CTS is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until CTS goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by INTA or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}W$)

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

NOTE:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
- The start bit check is enabled.
- Status Register Bit 0 (FE) indicates framing error.
- The sampling time of the serial receiver is the bit center.

A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

1. Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2. Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3. The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT IS INACTIVE (LOW).

4. The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5. The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6. Status Register Bit 0 implies framing error.

7. The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length
0	1	1	1	1	1 (Start of Bit)
0	1	1	1	0	2
0	1	1	0	1	3
0	1	1	0	0	4
0	1	0	1	1	5
0	1	0	1	0	6
0	1	0	0	1	7
0	1	0	0	0	8
0	0	1	1	1	9
0	0	1	1	0	10
0	0	1	0	1	11
0	0	1	0	0	12
0	0	0	1	1	13
0	0	0	1	0	14
0	0	0	0	1	15
0	0	0	0	0	16 (Bit center)
1	1	1	1	1	17
1	1	1	1	0	18
1	1	1	0	1	19
1	1	1	0	0	20
1	1	0	1	1	21
1	1	0	1	0	22
1	1	0	0	1	23
1	1	0	0	0	24
1	0	1	1	1	25
1	0	1	1	0	26
1	0	1	0	1	27
1	0	1	0	0	28
1	0	0	1	1	29
1	0	0	1	0	30
1	0	0	0	1	31
1	0	0	0	0	32 (End of Bit)

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to −150°C
Voltage On Any Pin	
With Respect to ground	−0.5V to −7V
Power Dissication	1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A= 0°C to 70°C, V_{CC}= +5.0V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+ 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL}= 2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH}= −400 μA
I_{IL}	Input Leakage		10 −10	μA μA	V_{IN}= V_{CC} V_{IN}= 0V
I_{LO}	Output Leakage		10 −10	μA μA	V_{OUT}= V_{CC} V_{OUT}= 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A= 25°C, V_{CC}= GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c= 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{cc} = +5.0V ± 10%, GND = 0V)

BUS PARAMETERS

Symbol	Parameter	8256AH		Units
		Min.	Max.	
tLL	ALE Pulse Width	70		ns
tCSL	$\overline{CS}$ to ALE Setup Time	0		ns
tAL	Address to ALE Setup Time	20		ns
tLA	Address Hold Time After ALE	30		ns
tLC	ALE to $\overline{RD}/\overline{WR}$	20		ns
tCC	$\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ Pulse Width	200		ns
tRD	Data Valid from $\overline{RD}$ (1)		150	ns
tDF	Data Float After $\overline{RD}$ (2)		70	ns
tDW	Data Valid to $\overline{WR}$	200		ns
tWD	Data Valid After $\overline{WR}$	50		ns
tCL	$\overline{RD}/\overline{WR}$ Control to Latch Enable	25		ns
tLDR	ALE to Data Valid		180	ns
tRST	Reset Pulse Width	500		ns
tRV	Recovery Time Between $\overline{RD}/\overline{WR}$	500		ns

TIMER/COUNTER PARAMETERS

tCPI	Counter Input Cycle Time (P12, P13)	2.2		μs
tCPWH	Counter Input Pulse Width High	1.1		μs
tCPWL	Counter Input Pulse Width Low	1.1		μs
tTPI	Counter Input↑ to INT↑ at Terminal Count		2.5	μs
tTIH	LOAD Pulse High Time Counter 5	1.1		μs
tTIL	LOAD Pulse Low Time Counter 5	1.1		μs
tPP	Counter 5 Load Before Next Clock Pulse on P13	1.1		μs
tCR	External Count Clock↑ to $\overline{RD}$↓ to Ensure Clock is Reflected in Count	2.2		μs
tRC	$\overline{RD}$↑ to External Count Clock↑ to Ensure Clock is not Reflected in Count	0		ns
tCW	External Count Clock↑ ro $\overline{WR}$↑ to Ensure Count Written is Not Decremented	2.2		μs
tWC	$\overline{WR}$↑ to External Count Clock to Ensure Count Written is Decremented	0		ns

INTERRUPT PARAMETERS

tDEX	EXTINT↑ to INT↑		200	ns
tDPI	Interrupt request on P17↑ to INT↑		2tCY +500	ns
tPI	Pulse Width of Interrupt Request on P17	tCY+ 100		ns
tHEA	$\overline{INTA}$↑ or $\overline{RD}$↑ to EXTINT↑	30		ns
tHIA	$\overline{INTA}$↑ or $\overline{RD}$↑ to INT↓		300	ns

A.C. CHARACTERISTICS (continued)

SERIAL INTERFACE AND CLOCK PARAMETERS

Symbol	Parameter	8256AH		Units
		Min.	Max.	
tCY	Clock Period	195	10,000	ns
tCLKH	Clock High Pulse Width	65		ns
tCLKL	Clock Low Pulse Width	65		ns
tR	Clock Rise Time		30	μs
tF	Clock Fall Time		30	ns
tSCY	Serial Clock Period (4)	975		ns
tSPD	Serial Clock High (4)	350		ns
tSPW	Serial Clock Low (4)	350		ns
tSTD	Internal Status Update Delay From Center of Stop Bit (5)		300	ns
tDTX	$\overline{TxC}$ to TxD Data Valid		300	ns
tIRBF	INT Delay From Center of First Stop Bit		2tCY +500	ns
tITBE	INT Delay From Falling Edge of Transmit Clock at end of Start Bit		2tCY +500	ns
tCTS	Pulse Width for Single Character Transmission	(6)		

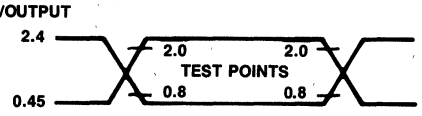
PARALLEL I/O PORT PARAMETERS

tWP	$\overline{WR}$ ↑ to P1/P2 Data Valid		0	ns
tPR	P1/P2 Data Stable Before $\overline{RD}$ ↓ (7)	300		ns
tRP	P1/P2 Data Hold Time	50		ns
tAK	$\overline{ACK}$ Pulse Width	150		ns
tST	Strobe Pulse Width	tSIB		ns
tPS	Data Setup to $\overline{STB}$ ↑	50		ns
tPH	Data Hold After $\overline{STB}$ ↑	50		ns
tWOB	$\overline{WR}$ ↑ to $\overline{OBF}$ ↑		250	ns
tAOB	$\overline{AKC}$ ↓ $\overline{OBF}$ ↓		250	ns
tSIB	$\overline{STB}$ ↓ to $\overline{IBF}$ ↓		250	ns
tRI	$\overline{RD}$ ↑ to $\overline{IBF}$ ↑		250	ns
tSIT	$\overline{STB}$ ↑ to $\overline{INT}$ ↑		2tCY +500	ns
tAIT	$\overline{ACK}$ ↑ to $\overline{INT}$ ↑		2tCY +500	ns
tAED	$\overline{OBF}$↓ to $\overline{ACK}$ ↓ Delay	0		ns

NOTES:
1. C_L = pF all outputs.
2. Measured from logic "one" or "zero" to 1.5V at C_L = 150 pF.
3. P12, P13 are external clock inputs.
4. Note that RxC may be used as an input only in 1X mode, otherwise it will be an output.
5. The center of the Stop Bit will be the receiver sample time, as programmed by the modification register.
6. 1/16th bit length for 32X, 64X; 100 ns for 1X.
7. To ensure t_{RD} spec is met.

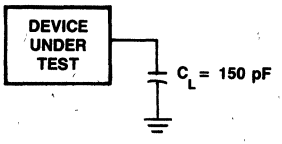
WAVEFORMS

A.C. TESTING INPUT, OUTPUT WAVEFORM



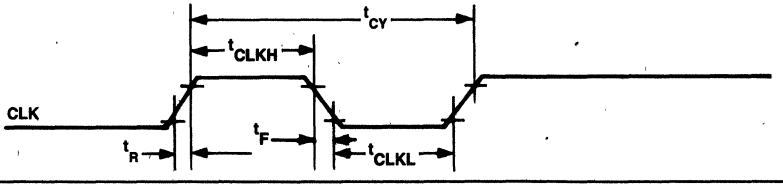
NOTES:
A.C. testing: inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

A.C. TESTING LOAD CIRCUIT

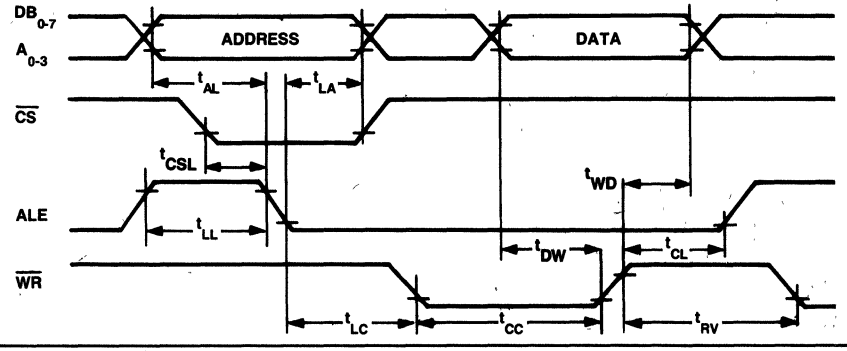


NOTES:
C_L = 150 pF
C_L includes jig capacitance

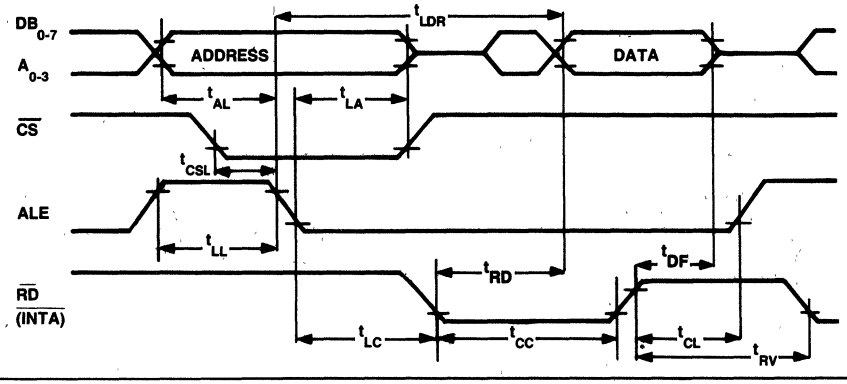
SYSTEM CLOCK



WRITE CYCLE

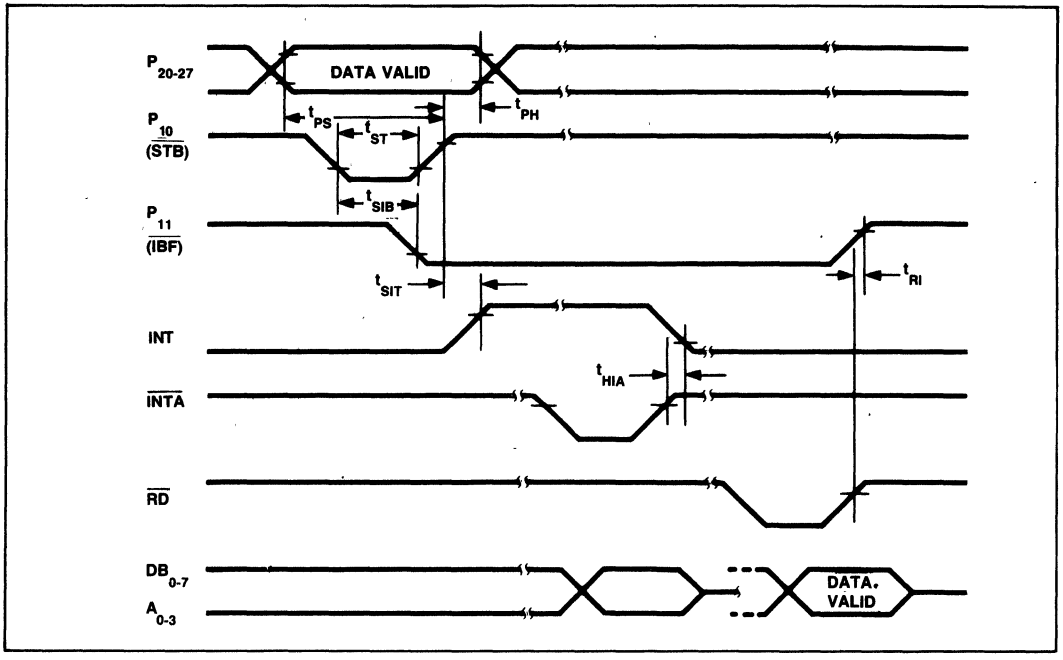


READ CYCLE

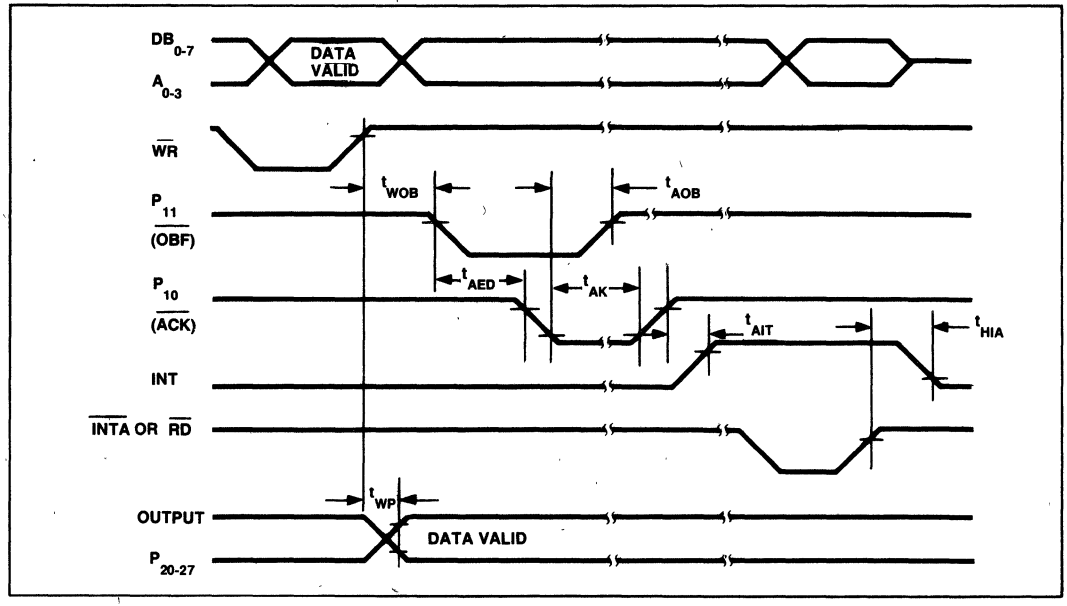


WAVEFORMS (Continued)

PARALLEL PORT HANDSHAKING - INPUT MODE

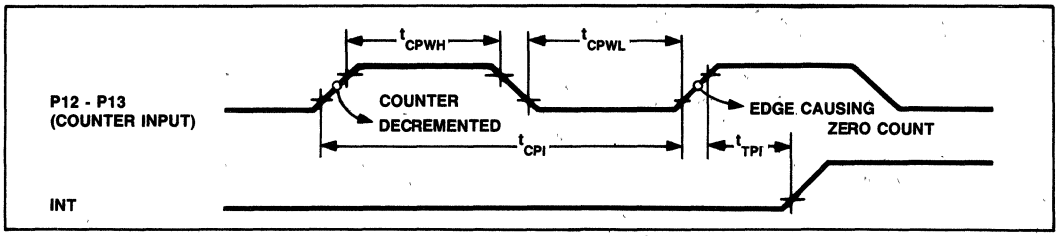


PARALLEL PORT HANDSHAKING - OUTPUT MODE

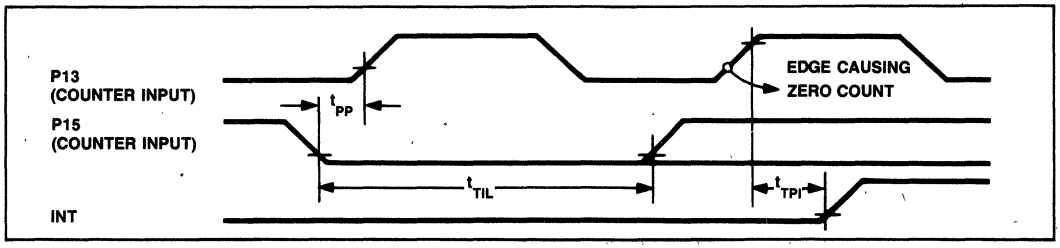


230759-001

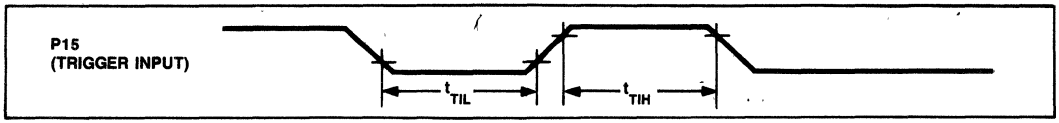
COUNT PULSE TIMINGS



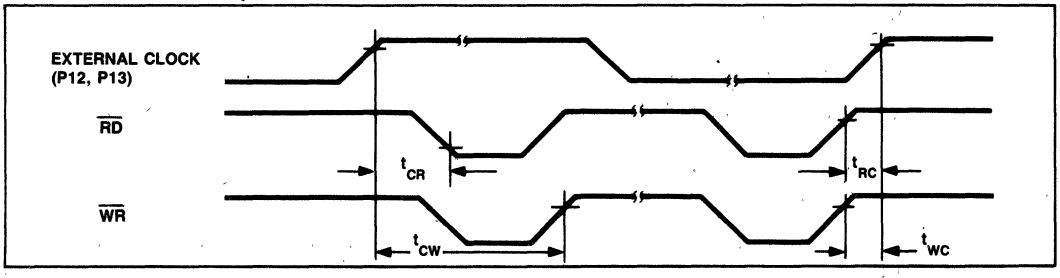
LOADING TIMER (OR CASCADED COUNTER/TIMER 3 AND 5)



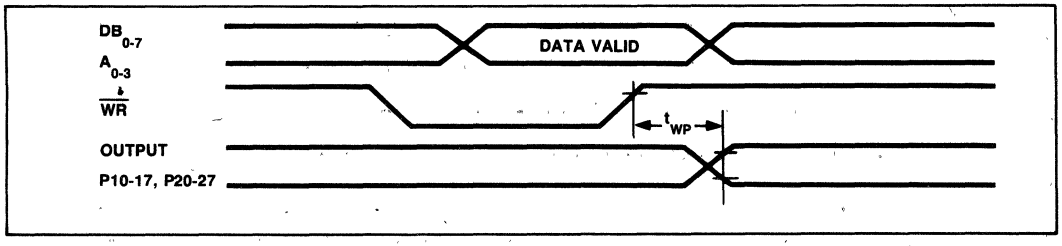
TRIGGER PULSE FOR TIMER 5 (CASCADED EVENT COUNTER/TIMER 3 AND 5)



COUNTER TIMER TIMING

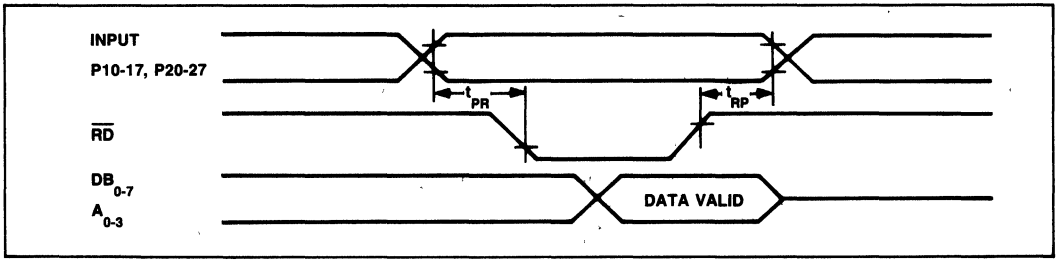


OUTPUT FROM PORT 1 AND PORT 2

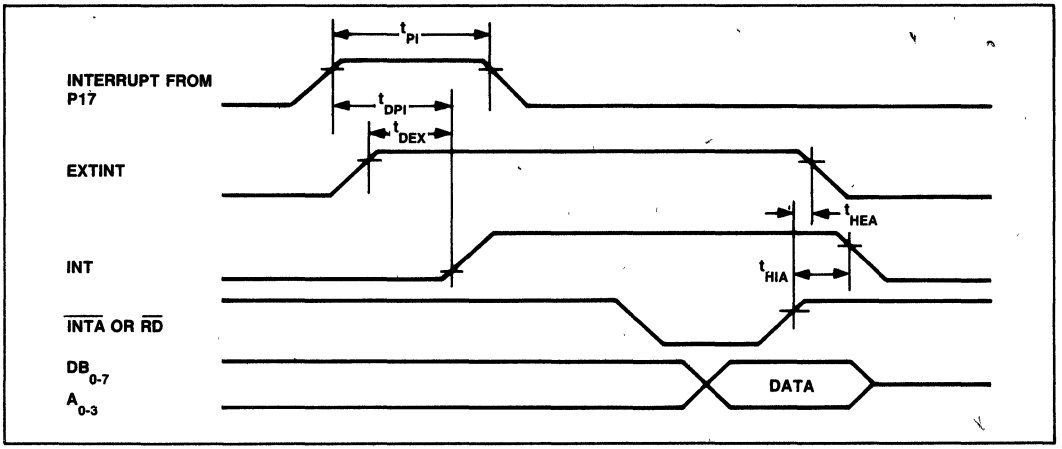


230759-001

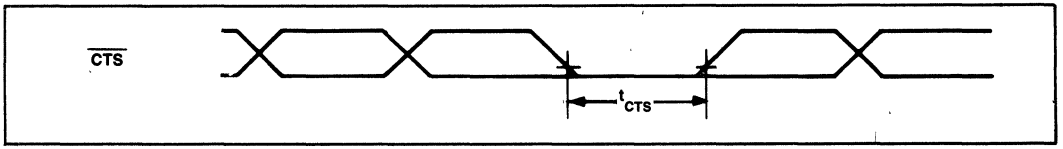
INPUT FROM PORT 1 AND PORT 2



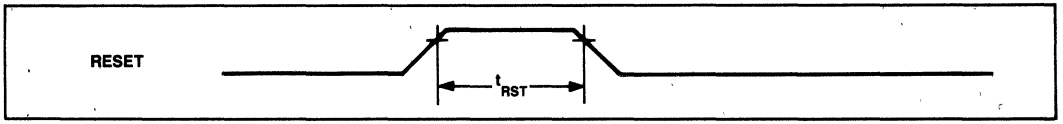
INTERRUPT TIMING



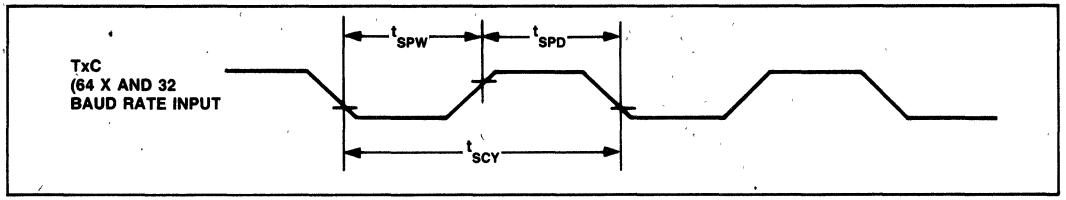
$\overline{CTS}$ FOR SINGLE CHARACTER TRANSMISSION



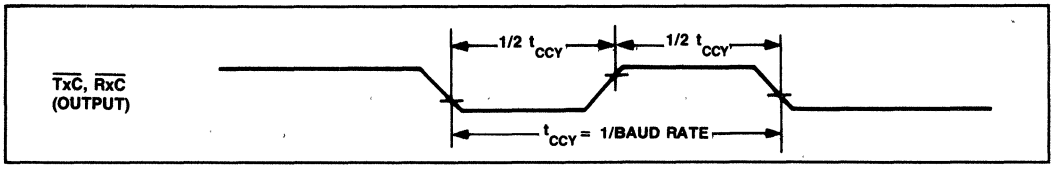
RESET TIMING



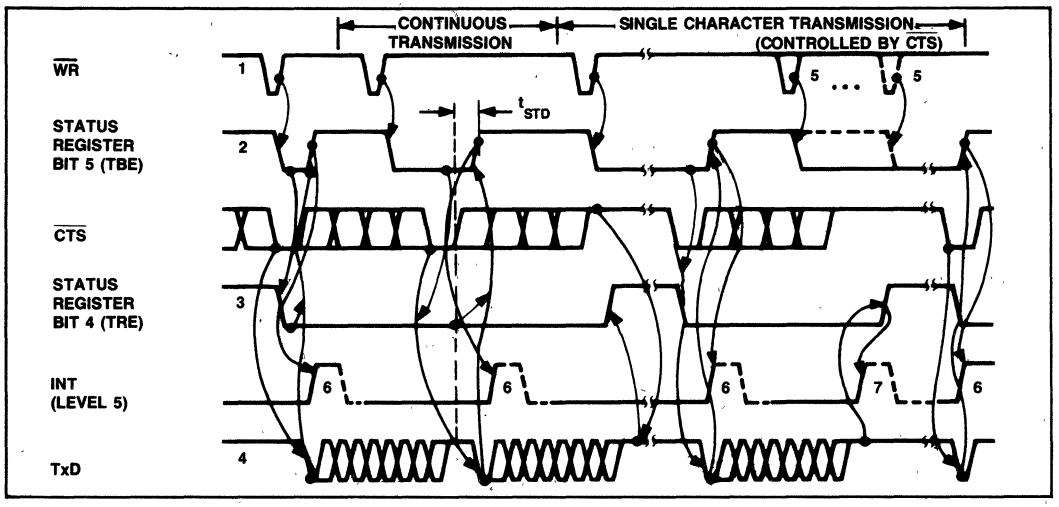
EXTERNAL BAUD RATE CLOCK FOR SERIAL INTERFACE



TRANSMITTER AND RECEIVER CLOCK FROM INTERNAL CLOCK SOURCE



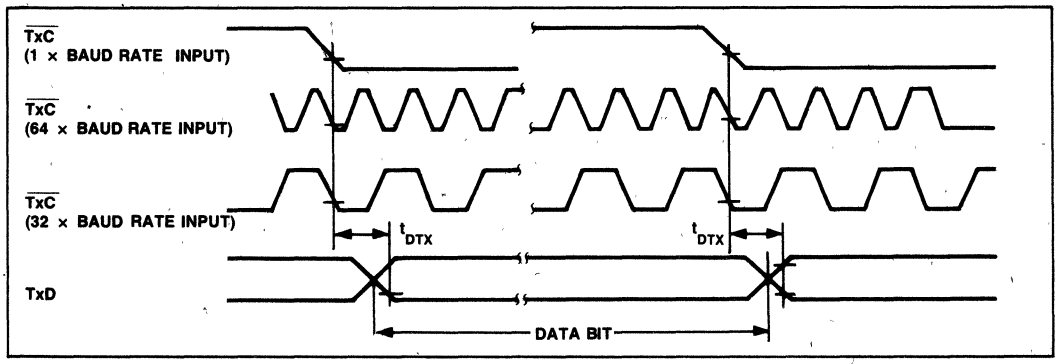
TRANSMISSION OF CHARACTERS ON SERIAL INTERFACE



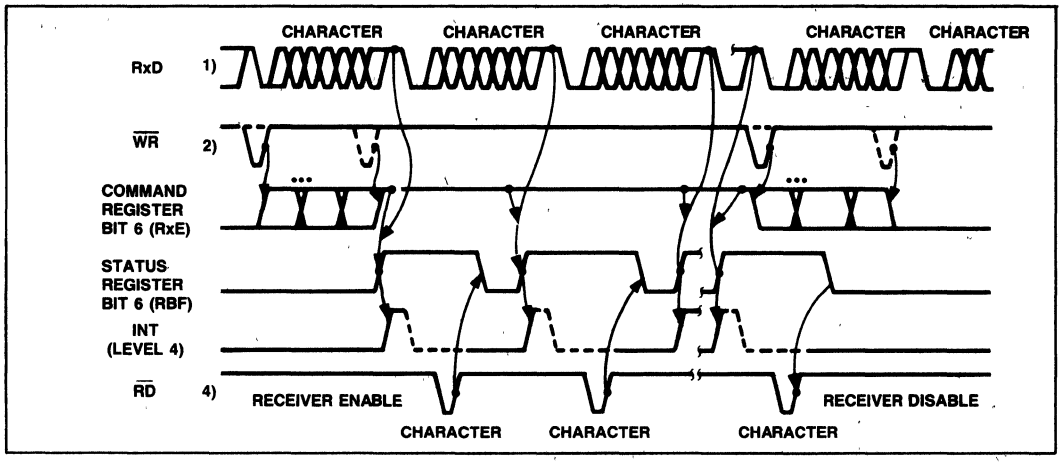
NOTES:
1. Load transmitter buffer register.
2. Transmitter buffer register is empty.
3. Transmitter register is empty.
4. Character format for this example: 7 Data Bits with Parity Bit and 2 Stop Bits.
5. Loading of transmitter buffer register must be complete before CTS goes low.
6. Interrupt due to transmitter buffer register empty.
7. Interrupt due to transmitter register empty.

No Status bits are altered when RD is active.

DATA BIT OUTPUT ON SERIAL INTERFACE

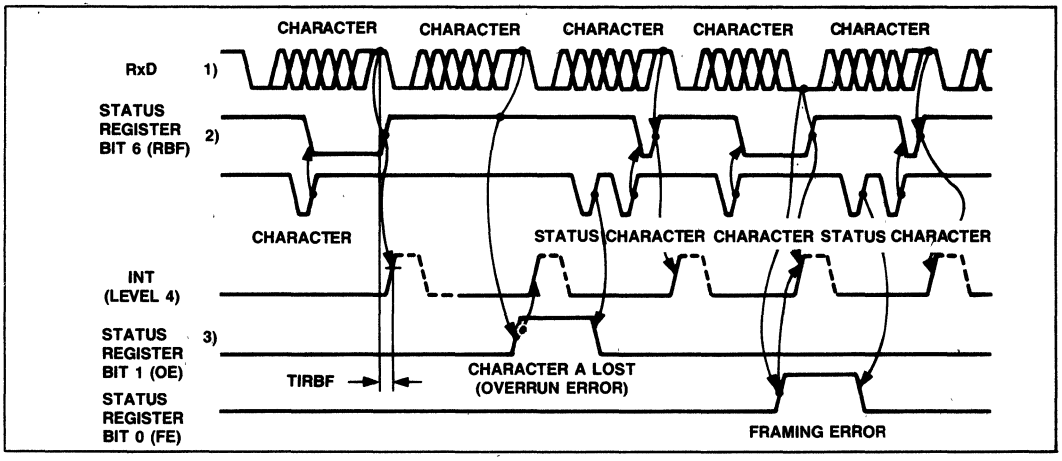


CONTINUOUS RECEPTION OF CHARACTERS ON SERIAL INTERFACE WITHOUT ERROR CONDITION



NOTES:
1. Character format for this example: 6 data bits with parity bit and one stop bit.
2. Set or reset bit 6 of command register 3 (enable receiver).
3. Receiver buffer located.
4. Read receiver buffer register.

ERROR CONDITIONS DURING RECEPTION OF CHARACTERS ON THE SERIAL INTERFACE



NOTES:
1. Character format for this example: 6 data bits without parity and one stop bit.
2. Receiver buffer register loaded.
3. Overrun error.
4. Framing error.
5. Interrupt from receiver buffer register loading.
6. Interrupt from overrun error.
7. Interrupt from framing error and loading receiver buffer register.

No status bits are altered when $\overline{RD}$ is active.

8279/8279-5
PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- **Simultaneous Keyboard Display Operations**
- **Scanned Keyboard Mode**
- **Scanned Sensor Mode**
- **Strobed Input Entry Mode**
- **8-Character Keyboard FIFO**
- **2-Key Lockout or N-Key Rollover with Contact Debounce**
- **Dual 8- or 16-Numerical Display**
- **Single 16-Character Display**
- **Right or Left Entry 16-Byte Display RAM**
- **Mode Programmable from CPU**
- **Programmable Scan Timing**
- **Interrupt Output on Key Entry**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16X8 display RAM which can be organized into dual 16X4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

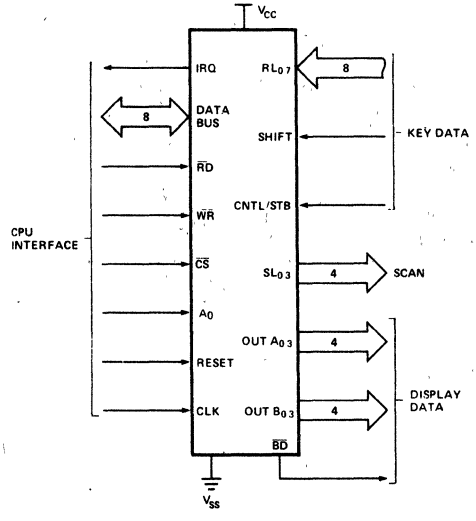


Figure 1. Logic Symbol

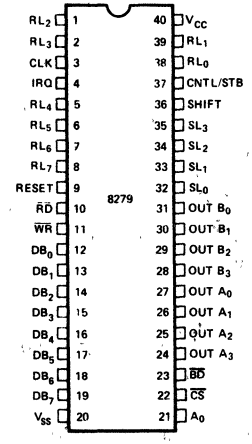


Figure 2. Pin Configuration

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Descriptions

Symbol	Pin No.	Name and Function
DB_0-DB_7	8	**Bi-directional data bus:** All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	1	**Clock:** Clock from system used to generate internal timing.
RESET	1	**Reset:** A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display —left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	1	**Chip Select:** A low on this pin enables the interface functions to receive or transmit.
A_0	1	**Buffer Address:** A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
$\overline{RD}$, $\overline{WR}$	2	**Input/Output Read and Write:** These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	1	**Interrupt Request:** In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V_{SS}, V_{CC}	2	**Ground and power supply pins.**
SL_0-SL_3	4	**Scan Lines:** Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL_0-RL_7	8	**Return Line:** Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	1	**Shift:** The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	1	**Control/Strobed Input Mode:** For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A_0-OUT A_3 OUT B_0-OUT B_3	4 4	**Outputs:** These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL_0-SL_3) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
$\overline{BD}$	1	**Blank Display:** This output is used to blank the display during digit switching or by a display blanking command.

FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

Input Modes

- Scanned Keyboard — with encoded (8 x 8 key keyboard) or decoded (4 x 8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8 x 8 matrix switches) or decoded (4 x 8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0$, $A_3 = D_7$).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$ and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by $\overline{CS}$. The character of the information, given or desired by the CPU, is identified by A_0. A logic one means the information is a command or status. A logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\overline{CS}$ = 1), the devices are in a high impedance state. The drivers input during $\overline{WR} \bullet \overline{CS}$ and output during $\overline{RD} \bullet \overline{CS}$.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with $A_0 = 1$ and then sending a $\overline{WR}$. The command is latched on the rising edge of $\overline{WR}$.

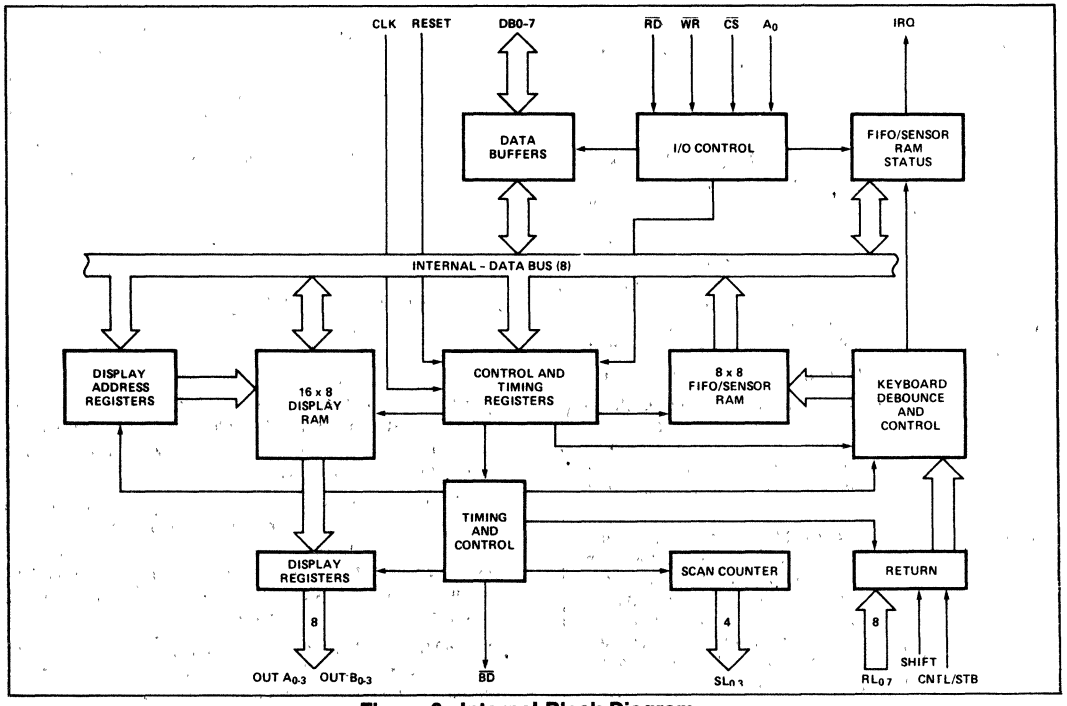


Figure 3. Internal Block Diagram

The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a ÷ N prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note than when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row If the debounce circuit detects a closed switch, it waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse

FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM In Keyboard or Strobed Input modes, it is a FIFO Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error The status can be read by an $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

SOFTWARE OPERATION

8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with $\overline{CS}$ low and A_0 high and are loaded to the 8279 on the rising edge of $\overline{WR}$.

Keyboard/Display Mode Set

	MSB							LSB
Code:	0	0	0	D	D	K	K	K

Where DD is the Display Mode and KKK is the Keyboard Mode.

DD

0	0	8 8-bit character display — Left entry
0	1	16 8-bit character display — Left entry*
1	0	8 8-bit character display — Right entry
1	1	16 8-bit character display — Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

KKK

0	0	0	Encoded Scan Keyboard — 2 Key Lockout*
0	0	1	Decoded Scan Keyboard — 2-Key Lockout
0	1	0	Encoded Scan Keyboard — N-Key Rollover
0	1	1	Decoded Scan Keyboard — N-Key Rollover
1	0	0	Encoded Scan Sensor Matrix
1	0	1	Decoded Scan Sensor Matrix
1	1	0	Strobed Input, Encoded Display Scan
1	1	1	Strobed Input, Decoded Display Scan

Program Clock

Code:	0	0	1	P	P	P	P	P

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

Code:	0	1	0	AI	X	A	A	A

X = Don't Care

The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-

*Default after reset

board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read ($A_0 = 0$) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set (AI = 1), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code:

0	1	1	AI	A	A	A	A

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set (AI = 1), this row address will be incremented after each following read *or write* to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read *or write* address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code:

1	0	0	AI	A	A	A	A

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with $A_0 = 1$, all subsequent writes with $A_0 = 0$ will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

Code:

				A	B	A	B
1	0	1	X	IW	IW	BL	BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag (IW = 1) for one of the ports, the port becomes marked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A_3 corresponds to bit D_7.

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code:

1	1	0	C_D	C_D	C_D	C_F	C_A

The C_D bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

C_D C_D C_D	
0 X	All Zeros (X = Don't Care)
1 0	AB = Hex 20 (0010 0000)
1 1	All Ones

Enable clear display when = 1 (or by C_A = 1)

During the time the Display RAM is being cleared (~160 μs), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted ($C_F = 1$), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A, the Clear All bit, has the combined effect of C_D and C_F; it uses the C_D clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code:

1	1	1	E	X	X	X	X

X = Don't care.

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset).

For the N-key rollover mode — if the E bit is programmed to "1" the chip will operate in the special Error mode (For further details, see Interface Considerations Section.)

Status Word

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A_0 is high and $\overline{CS}$ and $\overline{RD}$ are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A_0, $\overline{CS}$ and $\overline{RD}$ are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of $\overline{RD}$ will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

Data Write

Data that is written with A_0, $\overline{CS}$ and $\overline{WR}$ low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of $\overline{WR}$ occurs if AI set by the latest display command.

INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRQ will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is, the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

Scanned Keyboard — Special Error Modes

For N-key rollover mode the user can program a special error mode. This is done by the "End Interrupt/Error Mode Set" command. The debounce cycle and key-validity check are as in normal N-key mode. If during a single debounce cycle, two keys are found depressed, this is considered a simultaneous multiple depression, and sets an error flag. This flag will prevent any further writing into the FIFO and will set interrupt (if not yet set). The error flag could be read in this mode by reading the FIFO STATUS word. (See "FIFO STATUS" for further details.) The error flag is reset by sending the normal CLEAR command with $C_F = 1$.

Sensor Matrix Mode

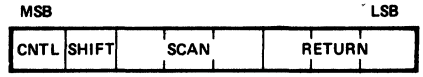
In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is inputted directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them.

The IRQ line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the Auto-Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

Note: Multiple changes in the matrix Addressed by ($SL_{0-3} = 0$) may cause multiple interrupts. ($SL_0 = 0$ in the Decoded Mode). Reset may cause the 8279 to see multiple changes.

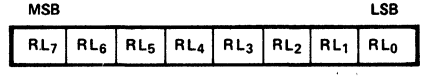
Data Format

In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.

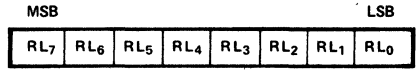


SCANNED KEYBOARD DATA FORMAT

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch postion maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.



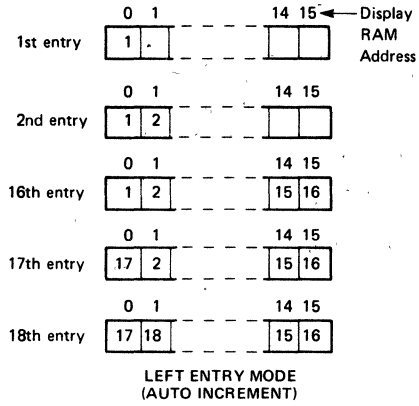
In Strobed Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input.



Display

Left Entry

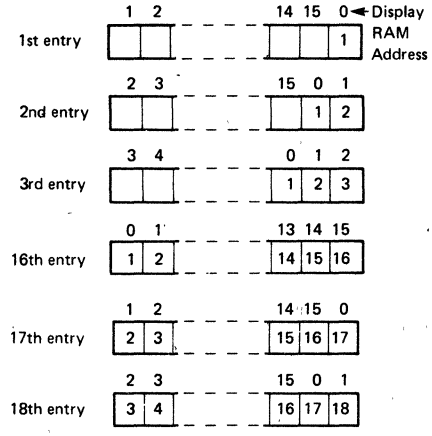
Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.



LEFT ENTRY MODE
(AUTO INCREMENT)

Right Entry

Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.

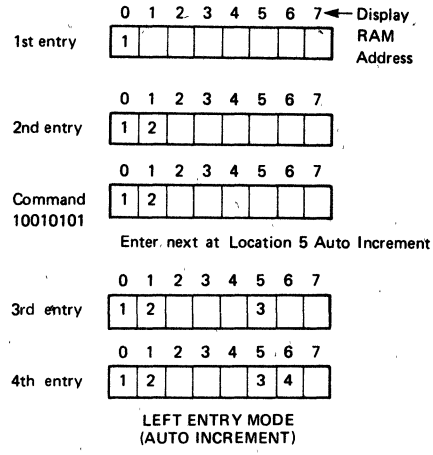


RIGHT ENTRY MODE
(AUTO INCREMENT)

Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

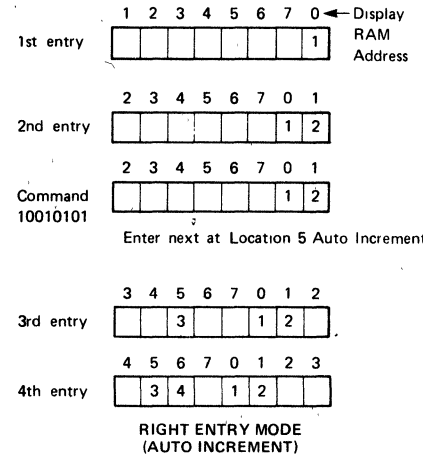
Auto Increment

In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



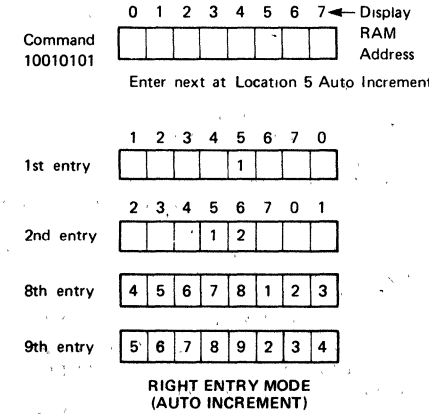
LEFT ENTRY MODE
(AUTO INCREMENT)

In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted:



RIGHT ENTRY MODE
(AUTO INCREMENT)

Starting at an arbitrary location operates as shown below:



RIGHT ENTRY MODE
(AUTO INCREMENT)

Entry appears to be from the initial entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

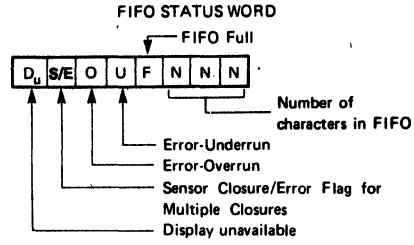
G. FIFO Status

FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.



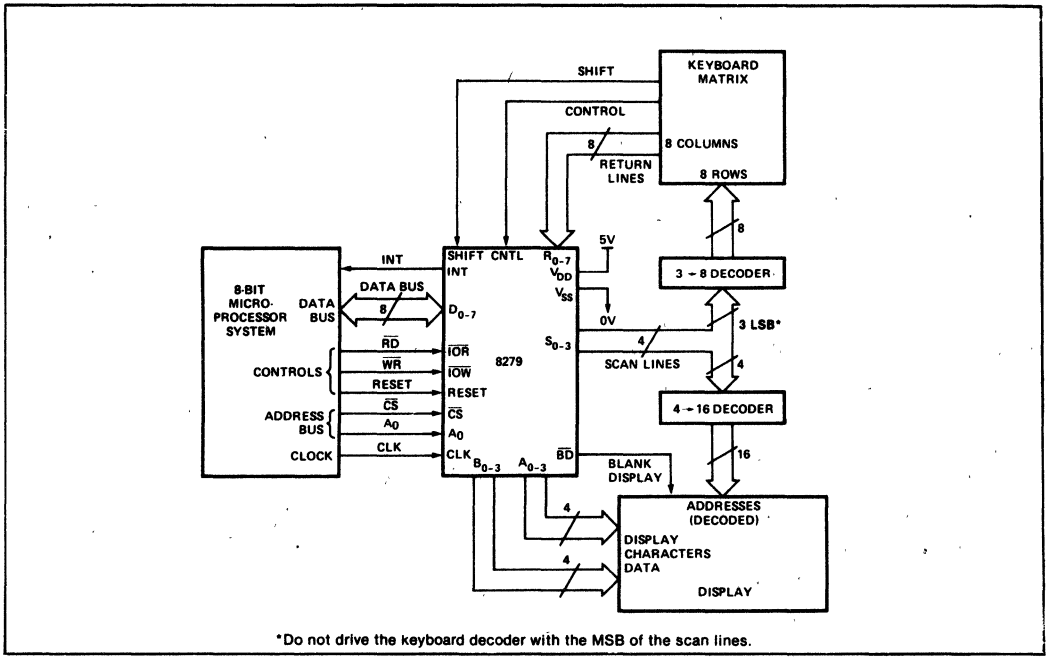


Figure 4. System Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature . . . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to 125°C
Voltage on any Pin with
Respect to Ground . . . . . -0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (NOTE 3)]*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL1}	Input Low Voltage for Return Lines	-0.5	1.4	V	
V_{IL2}	Input Low Voltage for All Others	-0.5	0.8	V	
V_{IH1}	Input High Voltage for Return Lines	2.2		V	
V_{IH2}	Input High Voltage for All Others	2.0		V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH1}	Output High Voltage on Interrupt Line	3.5		V	Note 2
V_{OH2}	Other Outputs	2.4			I_{OH} = −400 μA 8279-5 −100 μA 8279
I_{IL1}	Input Current on Shift, Control and Return Lines		+10 -100	μA μA	$V_{IN} = V_{CC}$ V_{IN} = 0V
I_{IL2}	Input Leakage Current on All Others		±10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		±10	μA	$V_{OUT} = V_{CC}$ to 0.45V
I_{CC}	Power Supply Current		120	mA	

CAPACITANCE

Symbol	Parameter	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance	5	10	pF	f_C = 1 MHz Unmeasured pins returned to V_{SS}
C_{OUT}	Output Capacitance	10	20	pF	

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{SS} = 0V, (Note 3)] *

Bus Parameters

READ CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{READ}$	50		0		ns
t_{RA}	Address Hold Time for $\overline{READ}$	5		0		ns
t_{RR}	$\overline{READ}$ Pulse Width	420		250		ns
t_{RD} [4]	Data Delay from $\overline{READ}$		300		150	ns
t_{AD} [4]	Address to Data Valid		450		250	ns
t_{DF}	$\overline{READ}$ to Data Floating	10	100	10	100	ns
t_{RCY}	Read Cycle Time	1		1		μs

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before $\overline{WRITE}$	50		0		ns
t_{WA}	Address Hold Time for $\overline{WRITE}$	20		0		ns
t_{WW}	$\overline{WRITE}$ Pulse Width	400		250		ns
t_{DW}	Data Set Up Time for $\overline{WRITE}$	300		150		ns
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		0		ns
t_{WCY}	Write Cycle Time	1		1		μs

OTHER TIMINGS

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{\phi W}$	Clock Pulse Width	230		120		nsec
t_{CY}	Clock Period	500		320		nsec

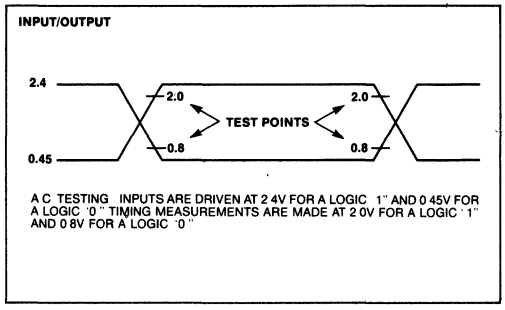
Keyboard Scan Time ..... 5.1 msec
Keyboard Debounce Time ..... 10.3 msec
Key Scan Time ..... 80 μsec
Display Scan Time ..... 10.3 msec

Digit-on Time ..... 480 μsec
Blanking Time ..... 160 μsec
Internal Clock Cycle[5] ..... 10 μsec

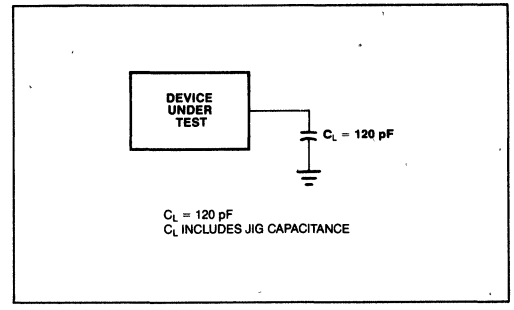
NOTES:
1. 8279, I_{OL} = 1.6mA; 8279-5, I_{OL} = 2.2mA.
2. I_{OH} = −100 μA
3. 8279, V_{CC} = +5V ±5%; 8279-5, V_{CC} = +5V ±10%.
4. 8279, C_L = 100pF; 8279-5, C_L = 150pF.
5. The Prescaler should be programmed to provide a 10 μs internal clock cycle.

* For Extended Temperature EXPRESS, use M8279A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM

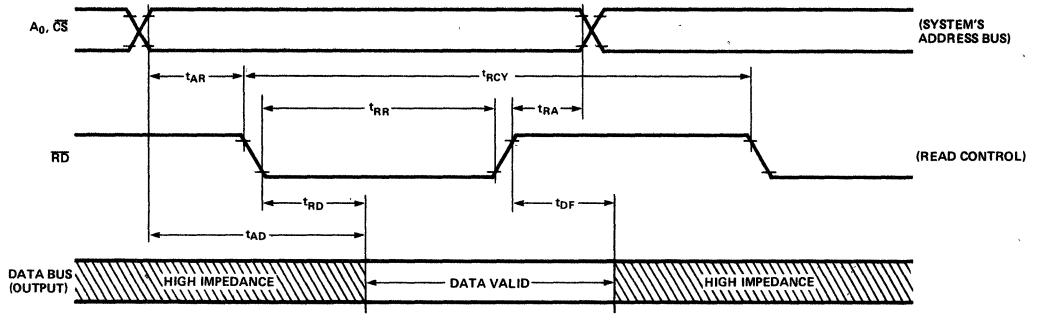


A.C. TESTING LOAD CIRCUIT

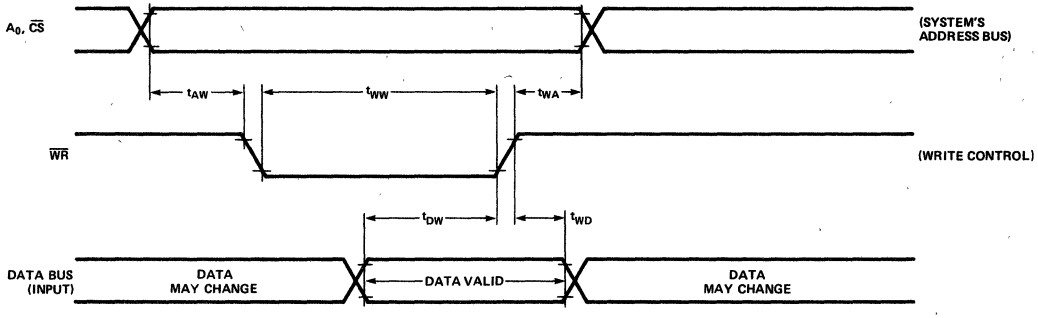


WAVEFORMS

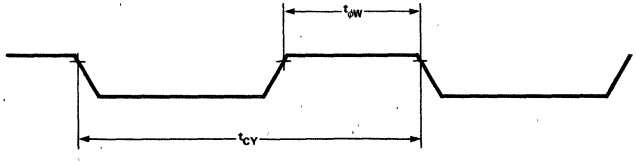
READ OPERATION



WRITE OPERATION

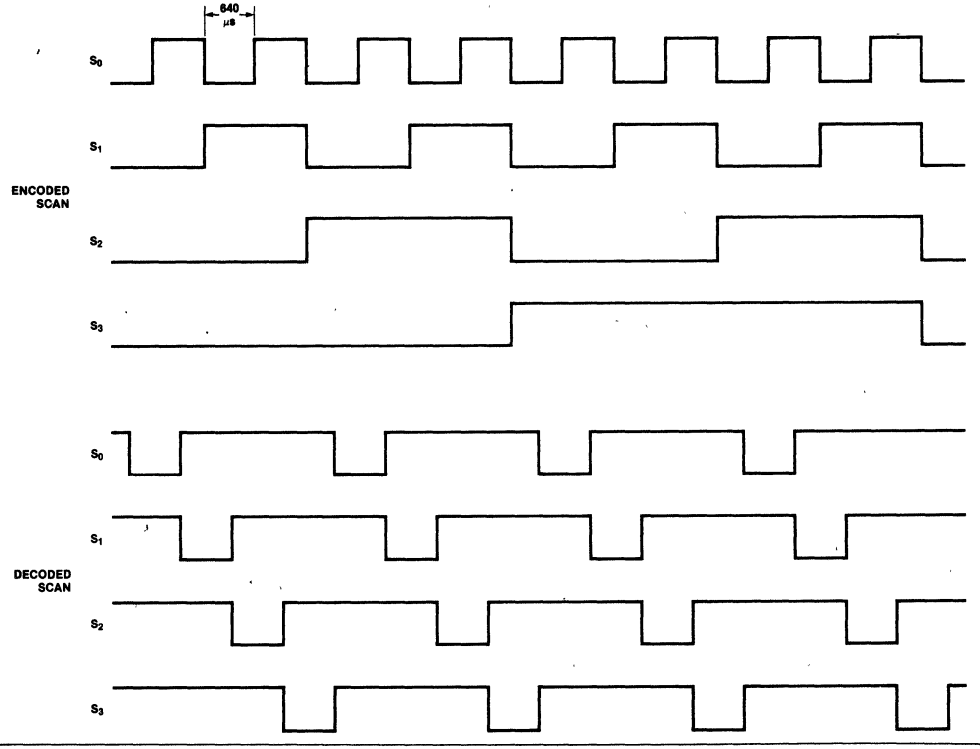


CLOCK INPUT

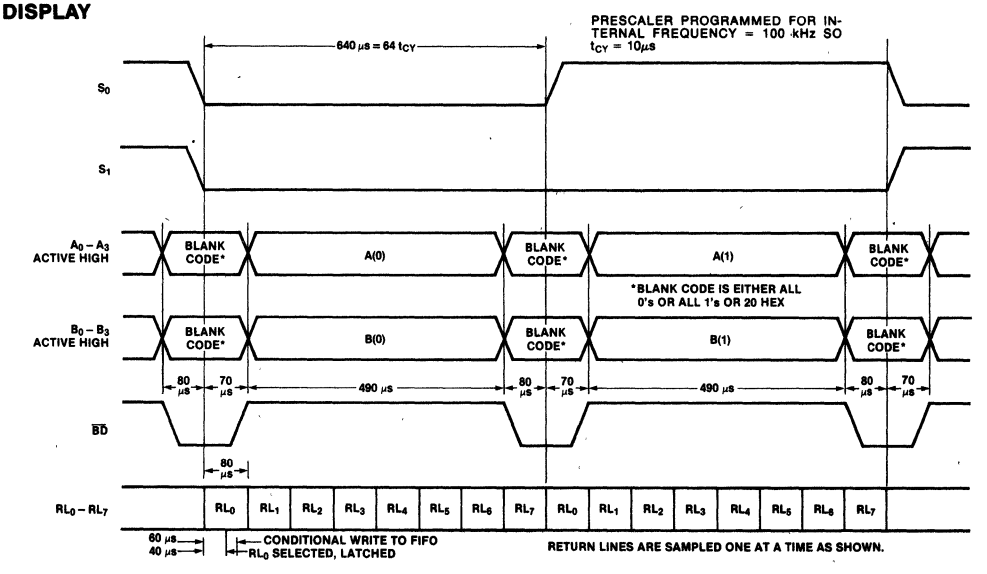


WAVEFORMS (Continued)

SCAN



DISPLAY



NOTE: SHOWN IS ENCODED SCAN LEFT ENTRY
S_2-S_3 ARE NOT SHOWN BUT THEY ARE SIMPLY S_1 DIVIDED BY 2 AND 4

82285
CLOCK GENERATOR AND READY INTERFACE FOR I/O COPROCESSORS

82285 is an 18 pin bipolar clock generator/driver designed to provide clock signals for the 82730, 82586, or other master peripherals. It also contains READY multiplexing logic to provide the required RDYO and READY timing and synchronization for the peripheral chips. RESET logic with hysteresis and synchronization is also provided.

- **Uses crystal or TTL signal for Frequency Source.**
- **Provides a 50% duty cycle peripheral clock output with MOS drive characteristics.**
- **Provides synchronous READY for peripherals from synchronous and/or asynchronous sources.**
- **Generates system reset output from Schmitt Trigger input.**
- **Capable of clock synchronization with other 82285's.**

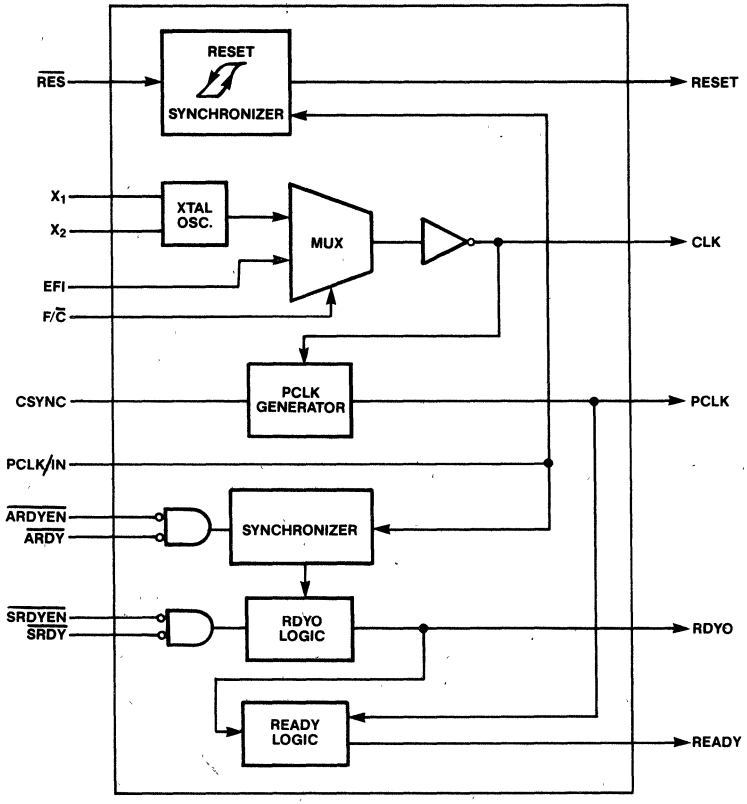


Figure 1. 82285 Block Diagram

NOVEMBER 1983
ORDER NUMBER 230813-001

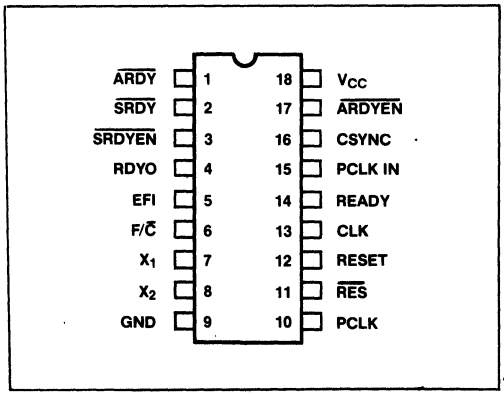


Figure 2. 82285 Pin Configuration

NOTE

1. CLK is a TTL level output and has the same frequency as either the crystal or EFI, depending on the state of F/$\overline{C}$.
2. PCLK is a MOS level output and has half the frequency of CLK.
3. $\overline{ARDY}$ and $\overline{ARDYEN}$ are interchangeable.
4. $\overline{SRDY}$ and $\overline{SRDYEN}$ are interchangeable.

FUNCTIONAL DESCRIPTION

Clock Generator

The CLK and PCLK clock outputs may be generated either by an external crystal or by an external TTL frequency input. If the frequency/crystal select input (F/$\overline{C}$) is high, the EFI input is used. If F/$\overline{C}$ is low, a crystal attached to X_1 and X_2 pins is used. CLK is a TTL output at the crystal or EFI frequency. PCLK is a MOS-level output which has a 50% duty cycle, operates at 1/2 the CLK frequency, and can be used to drive the clock inputs of the 82586, 82730, or other devices.

Reset Logic

The reset logic provides a Schmitt Trigger input ($\overline{RES}$) and two synchronization flip-flops to synchronize the reset timing. The reset signal is synchronized at the falling edge of PCLK IN. A simple RC network can be used to provide power-on reset of proper duration.

Table 1. Pin Description

Symbol	Pin Number	Type	Name and Function
$\overline{RES}$	11	I	RESET IN: $\overline{RES}$ is an active low signal which is used to generate RESET. A Schmitt trigger input is provided so that a RC connection can be used to establish the power up reset of proper duration.
RESET	12	O	RESET: RESET is an active high signal which is the synchronized version of the $\overline{RES}$ input.
X_1, X_2	7,8	I	CRYSTAL INPUT: X_1 and X_2 are attached to a parallel resonant, fundamental mode crystal. If F/$\overline{C}$ is strapped low to select the internal oscillator as the clock source, CLK will be the same frequency as the crystal, PCLK will be ½ that frequency.
CLK	13	O	CLOCK: CLK is a TTL output and has the same frequency as either the crystal or the external frequency input (EFI), dependent upon the state of F/$\overline{C}$.
PCLK	10	O	PERIPHERAL CLOCK: PCLK is a clock output at half the frequency of the crystal input or EFI, depending on F/$\overline{C}$ input. It provides MOS levels to drive the system CLK inputs of 82586 or 82730 or other device. PCLK has a 50% duty cycle.
PCLK IN	15	I	PERIPHERAL CLOCK IN: PCLK IN is a clock input which is used for clocking the RESET flip-flops and the $\overline{ARDY}$ synchronizing flip-flop. It can be driven by the PCLK output or some other system clock.
F/$\overline{C}$	6	I	FREQUENCY/CRYSTAL SELECT: F/$\overline{C}$ is a strapping option. When low, CLK and PCLK are generated from an external crystal. When high, CLK and PCLK are generated from the EFI input.

Table 1. Pin Description (Cont.)

Symbol	Pin Number	Type	Name and Function
EFI	5	I	EXTERNAL FREQUENCY IN: When F/$\overline{C}$ is strapped high, CLK and PCLK are generated from the EFI input. CLK will be the same frequency as EFI; PCLK will be half that frequency.
$\overline{ARDYEN}$	17	I	ASYNCHRONOUS READY ENABLE: $\overline{ARDYEN}$ is an asynchronous active low input which qualifies $\overline{ARDY}$. Set up and hold times are given only to guarantee recognition on that clock edge.
$\overline{ARDY}$	1	I	ASYNCHRONOUS READY: $\overline{ARDY}$ is an asynchronous active low input which will be synchronized to provide the RDYO output at the falling edge of PCLK IN. Setup and hold times are given only to guarantee recognition on that falling edge of PCLK IN. The RDYO output will also be a function of the $\overline{SRDY}$ input.
$\overline{SRDYEN}$	3	I	SYNCHRONOUS READY ENABLE: $\overline{SRDYEN}$ is a synchronous active low input which qualifies $\overline{SRDY}$.
$\overline{SRDY}$	2	I	SYNCHRONOUS READY: $\overline{SRDY}$ is a synchronous active low input. The RDYO outputs will also be a function of the $\overline{ARDY}$ input.
RDYO	4	O	SYNCHRONOUS READY OUT: RDYO is an active high output which is either the $\overline{SRDY}$ input delayed, or the $\overline{ARDY}$ input synchronized. RDYO will be inactive (low) if the ready inputs are inactive (high).
READY	14	O	READY: READY is an active high output which is the RDYO signal synchronized with the falling edge of PCLK output.
CSYNC	16	I	CLOCK SYNCHRONIZATION: CSYNC is used to provide synchronization of PCLK's among multiple 82285's. The source of CSYNC come from the PCLK output of the reference 82285. When synchronization is not used, CSYNC should be connected to V_{CC}.
GND	9	-	Ground.
V_{CC}	18	-	+5V supply.

RDYO and READY Logic

RDYO is determined by synchronous ready input $\overline{SRDY}$ qualified by $\overline{SRDYEN}$ or asynchronous ready input $\overline{ARDY}$ qualified by $\overline{ARDYEN}$. For the asynchronous input $\overline{ARDY}$, it will be clocked in at the falling edge of PCLK IN; and the RDYO output will become valid at the same falling edge of PCLK IN, provided $\overline{ARDY}$ is stable. The $\overline{ARDY}$ flip-flop is used as the first step in a two flip-flop synchronization method for RDYO. For the synchronous input $\overline{SRDY}$, the RDYO output will become valid when $\overline{SRDY}$ is stable.

The READY output is the RDYO output latched at the falling edge of PCLK out. It provides an additional ready signal in order to optimize the operation of systems using the 82730, 82586, and 8086.

WARNING:

The RDYO output is not fully synchronized when the asynchronous mode ($\overline{ARDY}$) is used.

Clock Synchronization Logic

The clock synchronization logic allows the PCLK signal of the device to be synchronized with the PCLK from other 82285's. A typical application of this synchronization logic is shown in Diagram 5. Diagram 3 and 4 illustrates typical functional sequences of 82285.

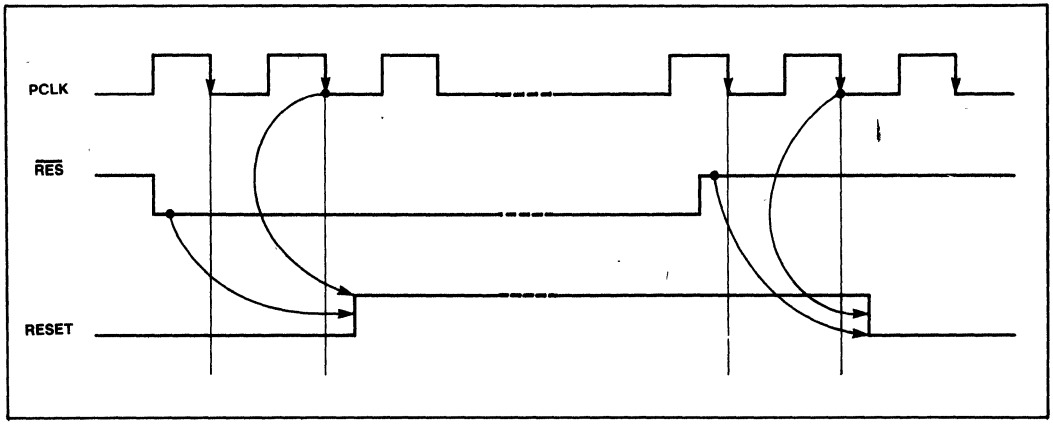


Figure 3. Reset Sequence

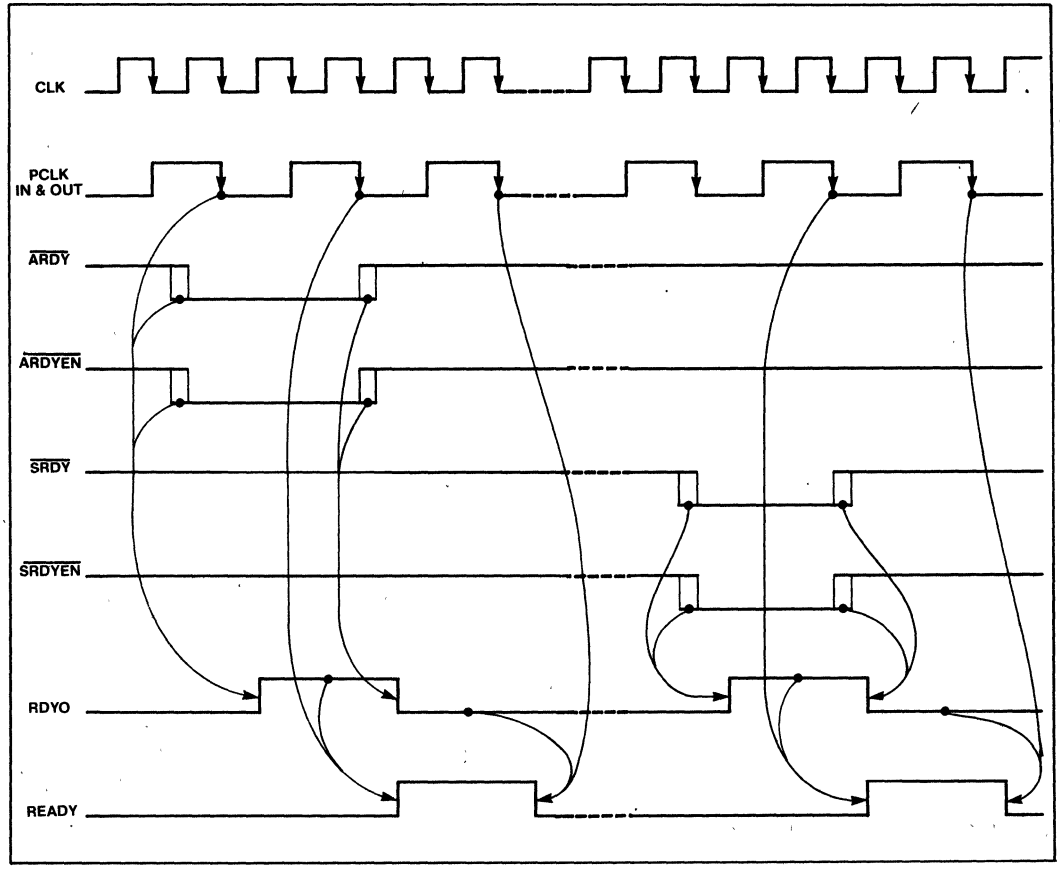


Figure 4. Ready Operation

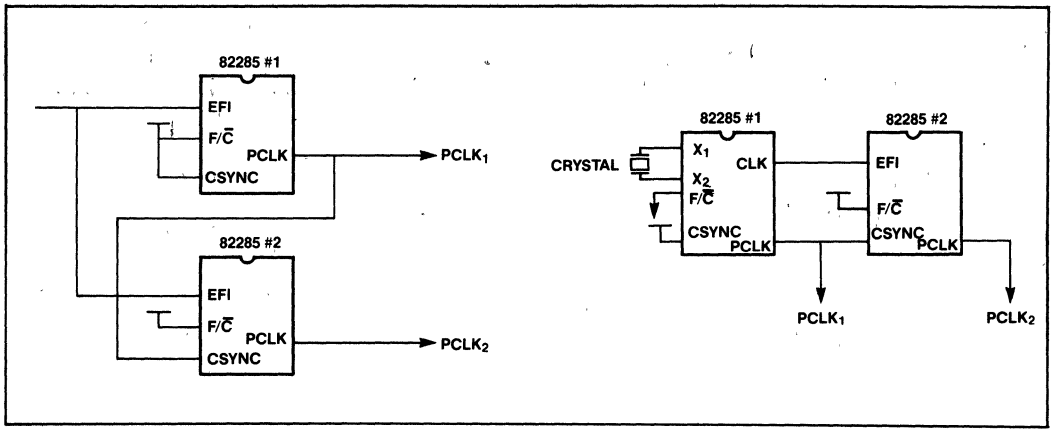


Figure 5. Typical Applictions of Clock Synchronization Among Multiple 82285's

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Voltage on any Pin with Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

Electrical Characteristics and Waveforms

D.C. Characteristics for 82285

Conditions: T_A = 0° C to 70° C; V_{CC} = 5V ± 10%

Symbol	Parameter	Min.	Max.	Units	Test Conditions
I_F	Forward Input Current		−0.5	mA	V_F = 0.45V
	For PCLK IN		−0.6	mA	V_F = 0.45V
	For $\overline{SRDYEN}$, $\overline{SRDY}$		−0.85	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	$V_R = V_{CC}$
V_C	Input Forward Clamp voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		145	mA	
V_{IL}	Input "low" voltage		0.8	V	
V_{IH}	Input "high" voltage	2.0			
V_{IHR}	Reset input "high" voltage	2.6		V	
V_{OL}	Output "low" voltage		0.45	V	I_{OL} = 5.25 mA
V_{OH}	Output "high" voltage PCLK	4.0		V	−1.05 mA
	Other outputs	2.4		V	−1.05 mA
V_{IHR}-V_{ILR}	$\overline{RES}$ Input Hysteresis	0.25			
C_I	Input Capacitance		10	pF	

82285 A.C. Characteristics (Cont.)

Condition: T_A = 0° C to 70° C; V_{CC} = %V ± 10% (Note 1)

Symbol	Parameter	Min.	Max.	Units	
t_R	CLK and PCLK rise time		10	ns	Note 2
t_F	CLK and PCLK fall time		10	ns	Note 2
t_L	PCLK IN and EFI low time	30		ns	
t_H	PCLK IN and EFI high time	30		ns	
t_1	CLK low time	1/2 t_3-15		ns	
t_2	CLK high time	1/2 t_3-15		ns	
t_3	CLK cycle time	56		ns	
t_4	PCLK low time @ 0.6V	t_3-12.5		ns	
	PCLK low time @ 1.5V	t_3-10		ns	
t_5	PCLK high time @ 3.8V	t_3-17.5		ns	
	PCLK high time @ 1.5V	t_3-10		ns	
t_6	PCLK cycle time	$2t_3$		ns	
t_7	$\overline{RES}$ setup time to PCLK IN↓	15		ns	Note 3, 4
t_8	$\overline{RES}$ hold time from PCLK IN↓	10		ns	Note 3, 4
t_9	PCLK delay from CLK low	0	40	ns	
t_{10}	RESET delay from PCLK low	0	50	ns	
t_{11}	$\overline{ARDYEN}$ setup time to $\overline{ARDY}$	0		ns	Note 4
t_{12}	$\overline{ARDYEN}$ hold time from $\overline{ARDY}$	0		ns	Note 4
t_{13}	$\overline{ARDY}$ setup time to PCLK IN↓	0		ns	Note 3, 4
t_{14}	$\overline{ARDY}$ hold time from PCLK IN↓		30	ns	Note 3, 4
t_{15}	$\overline{SRDYEN}$ setup time to $\overline{SRDY}$	0		ns	Note 4
t_{16}	$\overline{SRDYEN}$ hold time from $\overline{SRDY}$	0		ns	Note 4
t_{17}	$\overline{SRDY}$ setup time to PCLK↓		50	ns	
t_{18}	RDYO↑ delay from PCLK IN↓		55	ns	
t_{19}	RDYO↓ delay from $\overline{ARDY}$↑		30	ns	
t_{20}	RDYO delay from $\overline{SRDY}$		30	ns	
t_{21}	READY↑ delay from PCLK↓	–20	0	ns	
t_{22}	READY↓ delay from PCLK↓	–20	8	ns	
	Crystal frequency	17.6	4	MHz	Note 6
	EFI frequency	D.C.	17.6	MHz	Note 5

(see notes next page)

NOTE

1. All times are measured at the 1.5V level unless specified otherwise.
2. The rise and fall times for CLK are measured between 0.8V and 2.0V (TTL level drive characteristics). The rise and fall times for PCLK are measured between 1.0V and 3.5V (MOS level drive characteristics).
3. These are asynchronous inputs.
4. The setup and hold times are measured at the 0.8V and 2.0V levels for the inputs and at 1.5V from the PCLK signal.
5. To assure proper operation, the rise time or fall time of EFI cannot exceed 100 ns.
6. The specified timings are given in accordance with the maximum operating fre quency of 17.6 MHz. However, the device will be designed to operate to 24 MHz with all timing specs to be determined.

Loading:

For READY OUTPUT:
C_L = 30 pf, I_{OL} = 5.25 mA, I_{OH} = –1.05 mA

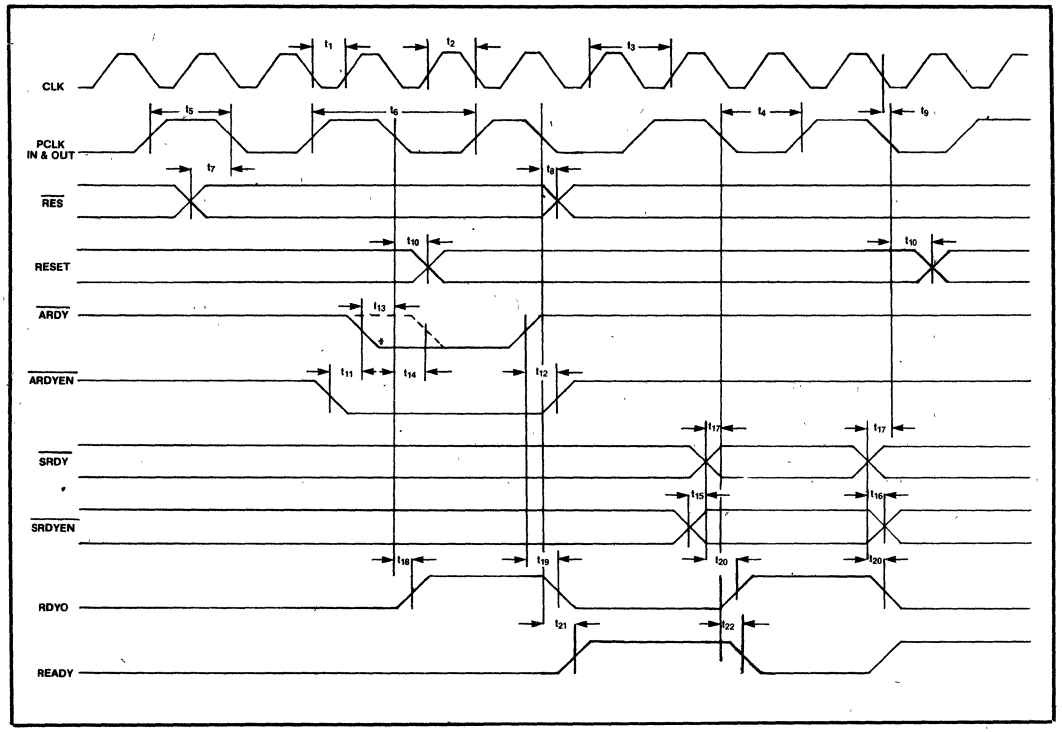
For the CLK output:
C_L = 75 pf, I_{OL} = 5.25 mA, I_{OH} = –1.05 mA

For the RDYO output:
C_L = 75 pf, I_{OL} = 5.25 mA, I_{OH} = –1.05 mA

For the PCLK output:
C_L = 175 pf, I_{OL} = 5.25 mA, I_{OH} = –1.05 mA

All input capacitance will be:
C_i = 10 pf

WAVEFORMS.



230813-001

An Intelligent Data Base System Using the 8272

Contents

1. INTRODUCTION

Most microcomputer systems in use today require low-cost, high-density removable magnetic media for information storage. In the area of removable media, a designer's choice is limited to magnetic tapes and floppy disks (flexible diskettes), both of which offer non-volatile data storage. The choice between these two technologies is relatively straight-forward for a given application. Since disk drives are designed to permit random access to stored information, they are significantly faster than tape units. For example, locating information on a disk requires less than a second, while tape movement (even at the fastest rewind or fast-forward speed) often requires several minutes. This random access ability permits the use of floppy disks in on-line storage applications (where information must be located, read, and modified/updated in real-time under program or operator control). Tapes, on the other hand, are ideally suited to archival or back-up storage due to their large storage capacities (more than 10 million bytes of data can be archived on a cartridge tape).

A sophisticated controller is required to capitalize on the abilities of the disk storage unit. In the past, disk controller designs have required upwards of 150 ICs. Today, the single-chip 8272 Floppy Disk Controller (FDC) plus approximately 30 support devices can handle up to four million bytes of on-line data storage on four floppy disk drives.

The Floppy Disk

A floppy disk is a circular piece of thin plastic material covered with a magnetic coating and enclosed in a protective jacket (Figure 1). The circular piece of plastic revolves at a fixed speed (approximately 360 rpm) within its jacket in much the same manner that a record revolves at a fixed speed on a stereo turntable. Disks are manufactured in a variety of configurations for various storage capacities. Two standard physical disk sizes are commonly used. The 8-inch disk (8 inches square) is the larger of the two sizes; the smaller size (5-1/4 inches square) is often referred to as a mini-floppy. Single-sided disks can record information on only one side of the disk, while double-sided disks increase the storage capacity by recording on both sides. In addition, disks are classified as single-density or double-density. Double-density disks use a modified recording method to store twice as much information in the same disk area as can be stored on a single-density disk. Table 1 lists storage capacities for standard floppy disk media.

A magnetic head assembly (in contact with the disk) writes information onto the disk surface and subsequently reads the data back. This head assembly can move from the outside edge of the disk toward the center in fixed increments. Once the head assembly is positioned at one of these fixed positions, the head can read or write information in a circular path as the disk revolves beneath the head assembly. This method divides the surface into a fixed number of cylinders (as shown in Figure 2). There are normally 77 cylinders on a standard disk. Once the head assembly is positioned at a given cylinder, data may be read or written on either side of the disk. The appropriate side of the disk is selected by the read/write head address (zero or one). Of course, a single-sided disk can only use head zero. The combination of cylinder address and head address uniquely specifies a single circular track on the disk. The physical beginning of a track is located by means of a small hole (physical index mark) punched through the plastic near the center of the disk. This hole is optically sensed by the drive on every revolution of the disk.

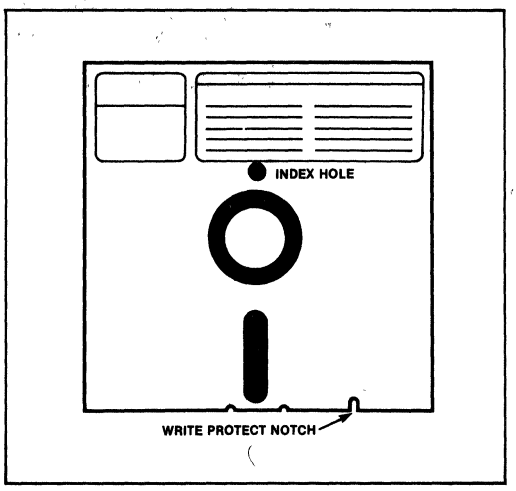


Figure 1. A Floppy Diskette

Table 1. Formatted Disk Capacities

Single-Density Format				
Byte/Sector	128	256	512	1024
Sectors/Track	26	15	8	4
Tracks/Disk	77	77	77	77
Bytes/Disk	256,256	295,680	315,392	315,392
Double-Density Format				
Bytes/Sector	128	256	512	1024
Sectors/Track	52	30	16	8
Tracks/Disk	77	77	77	77
Bytes/Disk	512,512	591,360	630,784	630,784

Each track is subdivided into a number of sectors (see detailed discussion in section 3). Sectors are generally 128, 256, 512, or 1024 data bytes in length. This track sectoring may be accomplished by one of two techniques: hard sectoring or soft sectoring. Hard sectored disks divide each track into a maximum of 32 sectors. The beginning of each sector is indicated by a sector hole punched in the disk plastic. Soft sectoring, the IBM standard method, allows software selection of sector sizes. With this technique, each data sector is preceded by a unique sector identifier that is read/written by the disk controller.

A floppy disk may also contain a write protect notch punched at the edge of the outer jacket of the disk. This notch is detected by the drive and passed to the controller as a write protect signal.

The Floppy Disk Drive

The floppy disk drive is an electromechanical device that records data on, or reads data from, the surface of a floppy disk. The disk drive contains head control electronics that move the head assembly one increment (step) forward (toward the center of the disk) or backward (toward the edge of the disk). Since the recording head must be in contact with the disk material in order to read or write information, the disk drive also contains head-load electronics. Normally the read/write head is unloaded until it is necessary to read or write information on the floppy disk. Once the head assembly has been positioned over the correct track on the disk, the head is loaded (brought into contact with the disk). This sequence prevents excessive disk wear. A small time penalty is paid when the head is loaded. Approximately thirty to fifty milliseconds are needed before data may be reliably read from, or written to, the disk. This time is known as the head load time. If desired, the head may be moved from cylinder to cylinder while loaded. In this manner, only a small time interval (head settling time) is required before data may be read from the new cylinder. The head settling time is often shorter than the head load time. Typically, disk drives also contain drive select logic that allows more than one physical drive to be connected to the same interface cable (from the controller). By means of a jumper on the drive, the drive number may be selected by the OEM or end user. The drive is enabled only when selected; when not selected, all control signals on the cable are ignored.

Finally, the drive provides additional signals to the system controller regarding the status of the drive and disk. These signals include:

Drive Ready — Signals the system that the drive door is closed and that a floppy disk is inserted into the drive.

Track Zero — Indicates that the head assembly is located over the outermost track of the disk. This signal may be used for calibration of the disk drive at system initialization and after an error condition.

Write Protect — Indicates that the floppy disk loaded into the drive is write protected.

Dual Sided — Indicates that the floppy disk in the drive is dual-sided.

Write Fault — Indicates that an error occurred during a recording operation.

Index — Informs the system that the physical index mark of the floppy disk (signifying the start of a data track) has been sensed.

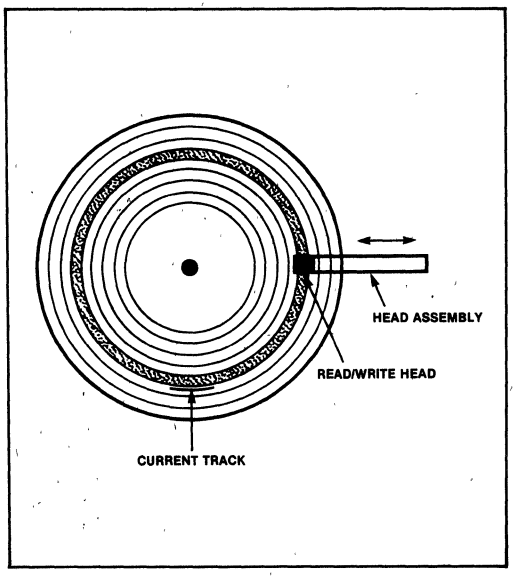


Figure 2. Concentric Cylinders on a Floppy Diskette

2. SUBSYSTEM OVERVIEW

A disk subsystem consists of the following functional electronic units:

1. Disk Controller Electronics
2. Disk Drive Electronics
3. Controller/Disk Interface (cables, drivers, terminators)
4. Controller/Microprocessor System Interface

The operation of these functional units is discussed in the following paragraphs.

Controller Electronics

The disk controller is responsible for converting high-level disk commands (normally issued by software executing on the system processor) into disk drive commands. This function includes:

1. Disk Drive Selection — Disk controllers typically manage the operations of multiple floppy disk drives. This controller function permits the system processor to specify which drive is to be used in a particular operation.
2. Track Selection — The controller issues a timed sequence of step pulses to move the head from its current location to the proper disk cylinder from which data is to be read or to which data is to be written. The controller stores the current cylinder number and computes the stepping distance from the current cylinder to the specified cylinder. The controller also manages the head select signal to select the correct side of the floppy disk.
3. Sector Selection — The controller monitors the data on a track until the requested sector is sensed.
4. Head Loading — The disk controller determines the times at which the head assembly is to be brought in contact with the disk surface in order to read or write data. The controller is also responsible for waiting until the head has settled before reading or writing information. Often the controller maintains the head loaded condition for up to 16 disk revolutions (approximately 2 seconds) after a read or write operation has been completed. This feature eliminates the head load time during periods of heavy disk I/O activity.
5. Data Separation — The actual signal recorded on a floppy disk is a combination of timing information (clock) and data. The serial READ DATA input (from the disk drive) must be converted into two signal streams: clock and data. (The READ DATA input operates at 250K bits/second for single-density disks and 500K bits/second for double-density disks.) The serial data must also be assembled into 8-bit bytes for transfer to system memory. A byte must be assembled and transferred every 32 microseconds for single-density disks and every 16 microseconds for double-density.
6. Error Checking — Information recorded on a floppy disk is subject to both hard and soft errors. Hard (permanent) errors are caused by media defects. Soft errors, on the other hand, are temporary errors caused by electromagnetic noise or mechanical interference. Disk controllers use a standard error checking technique known as a Cyclic Redundancy Check (CRC). As data is written to a disk, a 16-bit CRC character is computed and also stored on the disk. When the data is subsequently read, the CRC character allows the controller to detect data errors. Typically, when CRC errors are detected, the controlling software retries the failed operation (attempting to recover from a soft error). If data cannot reliably be read or written after a number of retries, the system software normally reports the error to the operator. Multiple CRC errors normally indicate unrecoverable media error on the current disk track. Subsequent recovery attempts must be defined by the system designers and tailored to meet system interfacing requirements.

Today, single-chip digital LSI floppy disk controllers such as the 8272 perform all the above functions with the exception of data separation. A data separation circuit (a combination of digital and analog electronics) synchronizes itself to the actual data rate of the disk drive. This data rate varies from drive to drive (due to mechanical factors such as motor tolerances) and varies from disk to disk (due to temperature effects). In order to operate reliably with both single- and double-density storage, the data separation circuit must be based on phase-locked loop (PLL) technology. The phase-locked loop data separation logic is described in section 5. The separation logic, after synchronizing with the data stream, supplies a data window to the LSI disk controller. This window differentiates data information from clock information within the serial stream. The controller uses this window to reconstruct the data previously recorded on the floppy disk.

Drive Electronics

Each floppy disk drive contains digital electronic circuits that translate TTL-compatible command signals into electromechanical operations (such as drive selection and head movement/loading) and that sense and report disk or drive status to the controller (e.g., drive ready, write fault, and write protect). In addition, the drive electronics contain analog components to sense, amplify, and shape data pulses read from, or written to, the floppy disk surface by the read/write head.

Controller/Drive Interface

The controller/drive interface consists of high-current line drivers, Schmitt triggered input gates, and flat or twisted pair cable(s) to connect the disk drive electronics to the controller electronics. Each interface signal line is resistively terminated at the end of the cable farthest from the line drivers. Eight-inch drives may be directly interfaced by means of 50-conductor flat cable. Generally, cable lengths should be less than ten feet in order to maintain noise immunity.

Normally, provisions are made for up to four disk drives to share the same interface cable. The controller may operate as many cable assemblies as practical. LSI floppy disk controllers typically operate one to four drives on a single cable.

Processor/Memory Interface

The disk controller must interface to the system processor and memory for two distinct purposes. First, the processor must specify disk control and command parameters to the controller. These parameters include the selection of the recording density and specification of disk formatting information (discussed in section 3). In addition to disk parameter specification, the processor must also send commands (e.g., read, write, seek, and scan) to the controller. These commands require the specification of the command code, drive number, cylinder address, sector address, and head address. Most LSI controllers receive commands and parameters by means of processor I/O instructions.

In addition to this I/O interface, the controller must also be designed for high-speed data transfer between memory and the disk drive. Two implementation methods may be used to coordinate this data transfer. The lowest-cost method requires direct processor intervention in the transfer. With this method, the controller issues an interrupt to the processor for each data transfer. (An equivalent method allows the processor to poll an interrupt flag in the controller status word.) In the case of a disk write operation, the processor writes a data byte (to be encoded into the serial output stream) to the disk controller following the receipt of each controller interrupt. During a disk read operation, the processor reads a data byte (previously assembled from the input data stream) from the controller after each interrupt. The processor must transfer a data byte from the controller to memory or transfer a data byte from memory to the disk controller within 16 or 32 microseconds after each interrupt (double-density and single-density response times, respectively).

If the system processor must service a variety of other interrupt sources, this interrupt method may not be practical, especially in double-density systems. In this case, the disk controller may be interfaced to a Direct Memory Access (DMA) controller. When the disk controller requires the transfer of a data byte, it simply activates the DMA request line. The DMA controller interfaces to the processor and, in response to the disk controller's request, gains control of the memory interface for a short period of time—long enough to transfer the requested data byte to/from memory. See section 6 for a detailed DMA interface description.

3. DISK FORMAT

New floppy disks must be written with a fixed format by the controller before these disks may be used to store data. Formatting is a method of taking raw media and adding the necessary information to permit the controller to read and write data without error. All formatting is performed by the disk controller on a track-by-track basis under the direction of the system processor. Generally, a track may be formatted at any time. However, since formatting "initializes" a complete disk track, all previously written data is lost (after a format operation). A format operation is normally used only when initializing new floppy disks. Since soft-sectoring in such a predominant formatting technique (due to IBM's influence), the following discussion will limit itself to soft-sectored formats.

Data Recording Techniques

Two standard data recording techniques are used to combine clock and data information for storage on a floppy disk. The single-density technique is referred to as FM encoding. In FM encoding (see Figure 3), a double frequency encoding technique is used that inserts a data bit between two adjacent clock bits. (The presence of a data bit represents a binary "one" while the absence of a data bit represents a binary "zero.") The two adjacent clock bits are referred to as a bit cell, and except for unique field identifiers, all clock bits written on the disk are binary "ones." In FM encoding, each data bit is written at the center of the bit cell and the clock bits are written at the leading edge of the bit cell.

The encoding used for double-density recording is termed MFM encoding (for "Modified FM"). In MFM encoding (Figure 3) the data bits are again written at the center of the bit cell. However, a clock bit is written at the leading edge of the bit cell only if no data bit was written in the previous bit cell and no data bit will be written in the present bit cell.

Sectors

Soft-sectored floppy disks divide each track into a number of data sectors. Typically, sector sizes of 128, 256, 512, or 1024 data bytes are permitted. The sector size is specified when the track is initially formatted by the controller. Table 1 lists the single- and double-

density data storage capacities for each of the four sector sizes. Each sector within a track is composed of the following four fields (illustrated in Figure 4):

1. Sector ID Field — This field, consisting of seven bytes, is written only when the track is formatted. The ID field provides the sector identification that is used by the controller when a sector must be read or written. The first byte of the field is the ID address mark, a unique coding that specifies the beginning of the ID field. The second, third, and fourth bytes are the cylinder, head, and sector addresses, respectively, and the fifth byte is the sector length code. The last two bytes are the 16-bit CRC character for the ID field. During formatting, the controller supplies the address mark. The cylinder, head, and sector addresses and the sector length code are supplied to the controller by the processor software. The CRC character is derived by the controller from the data in the first five bytes.
2. Post ID Field Gap — The post ID field gap (gap 2) is written initially when the track is formatted. During subsequent write operations, the drive's write circuitry is enabled within the gap and the trailing bytes of the gap are rewritten each time the sector is updated (written). During subsequent read operations, the trailing bytes of the gap are used to synchronize the data separator logic with the upcoming data field.
3. Data Field — The length (number of data bytes) of the data field is determined by software when the track is formatted. The first byte of the data field is the data address mark, a unique coding that specifies the beginning of the data field. When a sector is to be deleted, (e.g., a hard error on the disk), a deleted data address mark is written in place of the data address mark. The last two bytes of the data field comprise the CRC character.
4. Post Data Field Gap — The post data field gap (gap 3) is written when the track is formatted and separates the preceding data field from the next physical ID field on the track. Note that a post data field gap is not written following the last physical sector on a track. The gap itself contains a program-selectable number of bytes. Following a sector update (write) operation, the drive's write logic is disabled during the gap. The actual size of gap 3 is determined by the maximum number of data bits that can be recorded on a track, the number of sectors per track and the total sector size (data plus overhead information). The gap size must be adjusted so that it is large enough to contain the discontinuity generated on the floppy disk when the write current is turned on or off (at the start or completion of a disk write operation) and to contain a synchronization field for the upcoming ID field (of the next sector). On the other hand, the gaps must be small enough so that the total number of data bits required on the track (sectors plus gaps) is less than the maximum number of data bits that can be recorded on the track. The gap size must be specified for all read, write, and format operations. The gap size used during disk reads and writes must be smaller than the size used to format the disk to avoid the splice points between contiguous physical sectors. Suggested gap sizes are listed in Table 9.

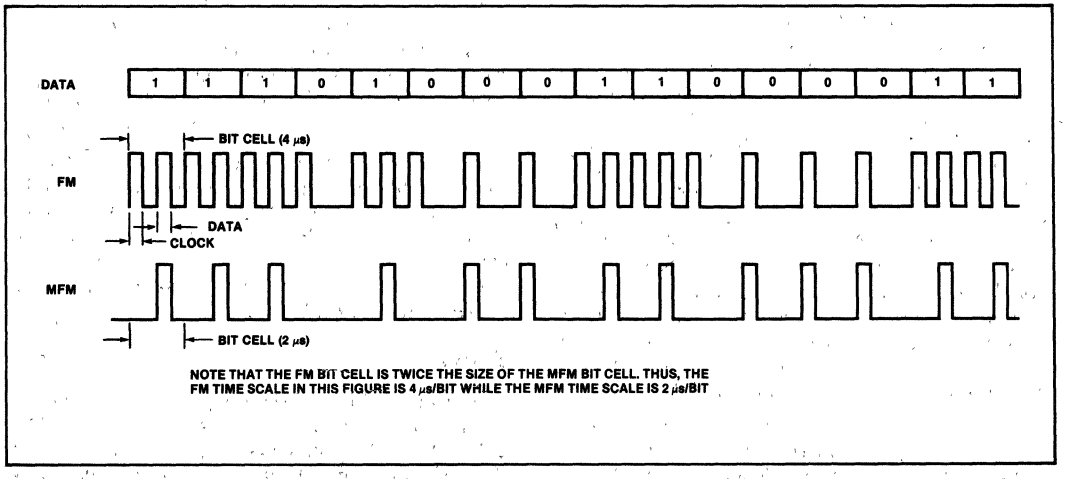


Figure 3. FM and MFM Encoding

Tracks

The overall format for a track is illustrated in Figure 4. Each track consists of the following fields:

1. Pre-Index Gap — The pre-index gap (gap 5) is written only when the track is formatted.
2. Index Address Mark — The index address mark consists of a unique code that indicates the beginning of a data track. One index mark is written on each track when the track is formatted.
3. Post Index Gap — The post index gap (gap 1) is used during disk read and write operations to synchronize the data separator logic with the data to be read from the ID field (of the first sector). The post index gap is written only when the disk is formatted.
4. Sectors — The sector information (discussed above) is repeated once for each sector on the track.
5. Final Gap — The final gap (gap 4) is written when the track is formatted and extends from the last physical data field on the track to the physical index mark. The length of this gap is dependent on the number of bytes per sector specified, the lengths of the program-selectable gaps specified, and the drive speed.

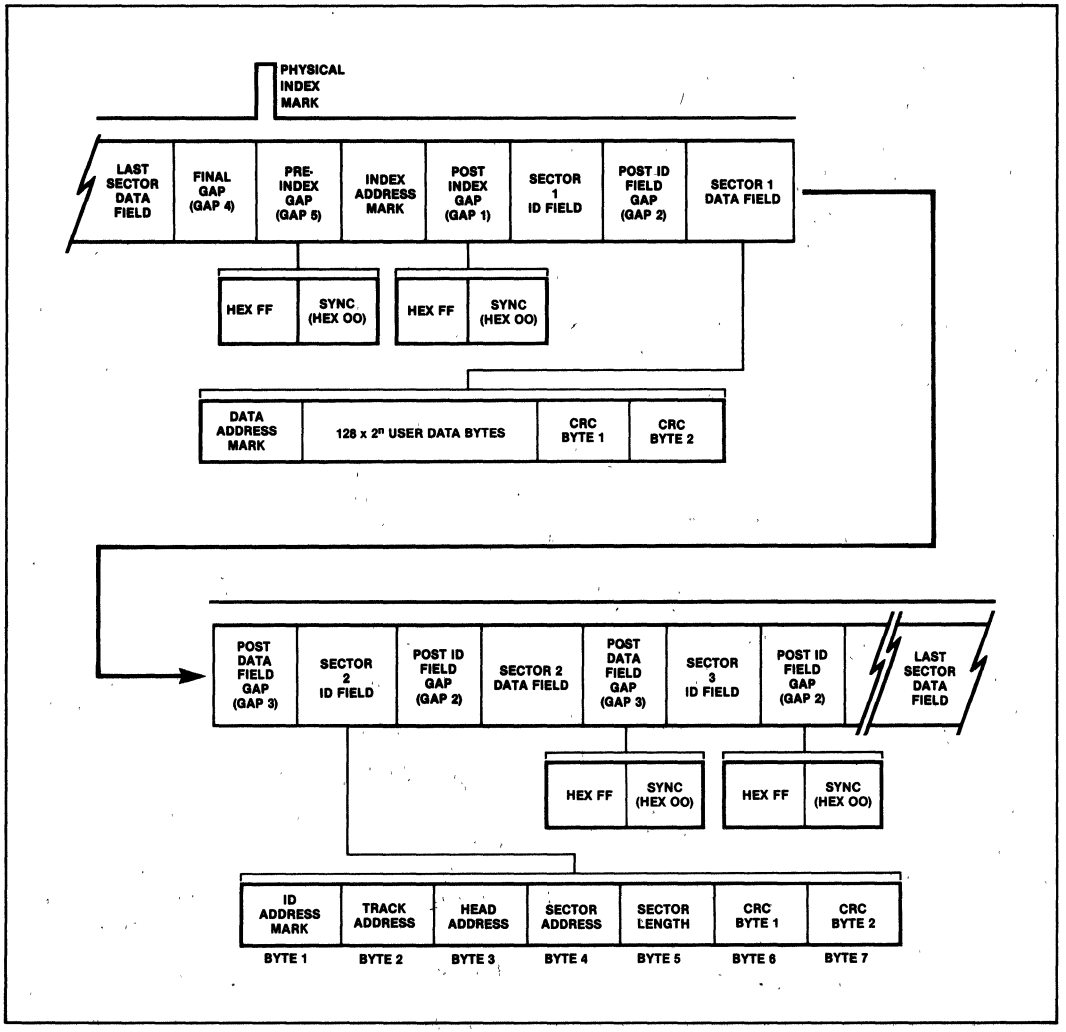


Figure 4. Standard Floppy Diskette Track Format (From SBC 204 Manual)

Sector Interleaving

The initial formatting of a floppy disk determines where sectors are located within a track. It is not necessary to allocate sectors sequentially around the track (i.e., 1,2,3,...,26). In fact, is is often advantageous to place the sectors on the track in a non-sequential order. Sequential sector ordering optimizes sector access times during multi-sector transfers (e.g., when a program is loaded) by permitting the number of sectors specified (up to an entire track) to be transferred within a single revolution of the disk. A technique known as sector interleaving optimizes access times when, although sectors are accessed sequentially, a small amount of processing must be performed between sector reads/writes. For example, an editing program performing a text search reads sectors sequentially, and after each sector is read, performs a software search. If a match is not found, the software issues a read request for the next sector. Since the floppy disk continues to rotate during the time that the software executes, the next physical sector is already passing under the read/write head when the read request is issued, and the processor must wait for another complete revolution of the disk (approximately 166 milliseconds) before the data may actually be input. With interleaving, the sectors are not stored sequentially on a track; rather, each sector is physically removed from the previous sector by some number (known as the interleave factor) of physical sectors as shown in Figure 5. This method of sector allocation provides the processor additional execution time between sectors on the disk. For example, with a 26 sector/track format, an interleave factor of 2 provides 6.4 milliseconds of processing time between sequential 128 byte sector accesses.

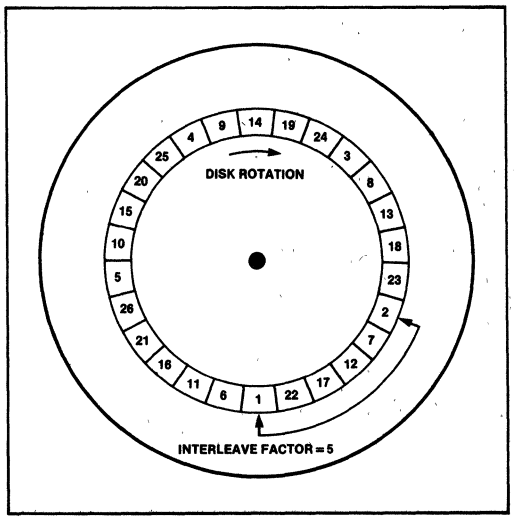


Figure 5. Interleaved Sector Allocation Within a Track

To calculate the correct interleave factor, the maximum processor time between sector operations must be divided by the time required for a complete sector to pass under the disk read/write head. After determining the interleave factor, the correct sector numbers are passed to the disk controller (in the exact order that they are to physically appear on the track) during the execution of a format operation.

4. THE 8272 FLEXIBLE DISKETTE CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that contains the circuitry necessary to implement both single-and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). With the 8272, less than 30 ICs are needed to implement a complete disk subsystem. The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, write sector, and read track. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. External logic is required only for the generation of the FDC master clock and write clock (see Section 6) and for data separation (Section 5). The FDC provides signals that control the startup and base frequency selection of the data separator. These signals greatly ease the design of a phase-locked loop data separator.

In addition to the data separator interface signals, the 8272 also provides the necessary signals to interface to microprocessor systems with or without Direct Memory Access (DMA) capabilities. In order to interface to a large number of commercially available floppy disk drives, the FDC permits software specification of the track stepping rate, the head load time, and the head unload time.

The pin configuration and internal block diagram of the 8272 is shown in Figure 6. Table 2 contains a description for each FDC interface pin.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

- Specify
- Sense Drive Status
- Sense Interrupt Status
- Seek
- Recalibrate
- Format Track
- Read Data
- Read Deleted Data
- Write Data
- Write Deleted Data
- Read Track
- Read ID
- Scan Equal
- Scan High or Equal
- Scan Low or Equal

Each command is initiated by a multi-byte transfer from the processor to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the processor. It is convenient to consider each FDC command as consisting of the following three phases:

COMMAND PHASE: The executing program transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

EXECUTION PHASE: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk (signalled by the TC input to the FDC) or when an error occurs.

RESULT PHASE: After completion of the disk operation, status and other housekeeping information are made available to the processor. After the processor reads this information, the FDC reenters the command phase and is ready to accept another command.

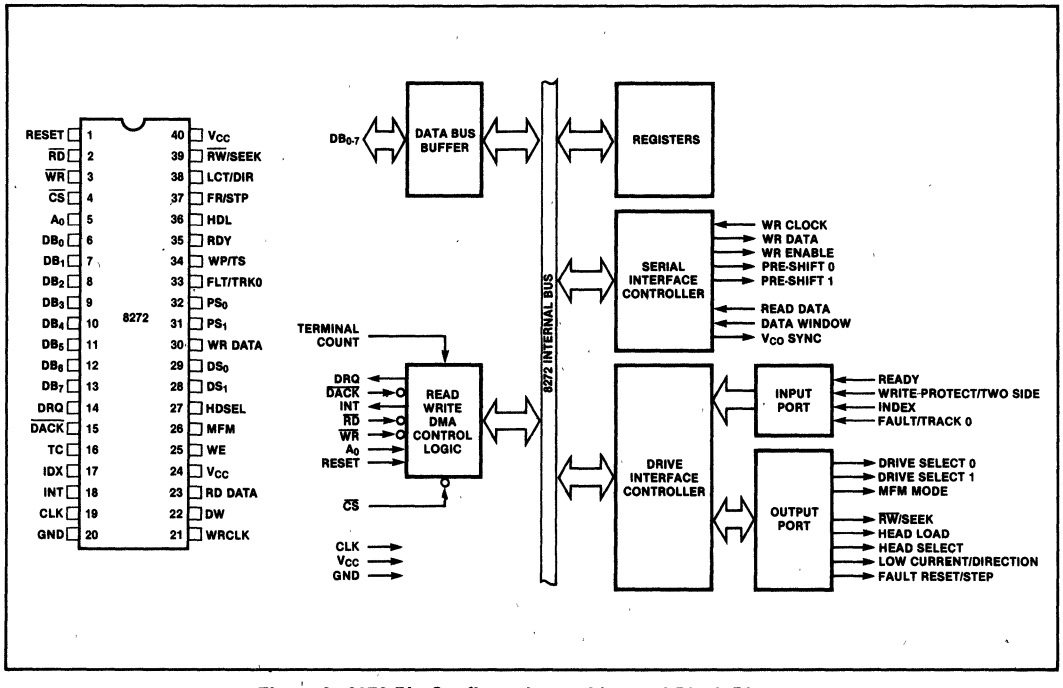


Figure 6. 8272 Pin Configuration and Internal Block Diagram

Table 2. 8272 FDC Pin Description

Number	Pin Symbol	I/O	To/From	Description
1	RST	I	uP	Reset. Active-high signal that places the FDC in the "idle" state and all disk drive output signals are forced inactive (low). This input must be held active during power on reset while the RD and WR inputs are active.
2	$\overline{RD}$	I*	uP	Read. Active-low control signal that enables data transfer from the FDC to the data bus.
3	$\overline{WR}$	I*	uP	Write. Active-low control signal that enables data transfer from the data bus into the FDC.
4	$\overline{CS}$	I	uP	Chip Select. Active-low control signal that selects the FDC. No reading or writing will occur unless the FDC is selected.
5	A_0	I*	uP	Address. Selects the Data Register or Main Status Register for input/output in conjunction with the RD and WR inputs. (See Table 3.)
6-13	DB_0-DB_7	I/O*	uP	Data Bus. Bidirectional three-state 8-bit data bus.
14	DRQ	O	DMA	DMA Request. Active-high output that indicates an FDC request for DMA services.
15	$\overline{DACK}$	I	DMA	DMA Acknowledge. Active-low control signal indicating that the requested DMA transfer is in progress.
16	TC	I	DMA	Terminal Count. Active-high signal that causes the termination of a command. Normally, the terminal count input is directly connected to the TC/EOP output from the DMA controller, signalling that the DMA transfer has been completed. In a non-DMA environment, the processor must count data transfers and supply a TC signal to the FDC.
17	IDX	I	Drive	Index. Indicates detection of the physical index mark (the beginning of a track) on the selected disk drive.
18	INT	O	uP	Interrupt Request. Active-high signal indicating an 8272 interrupt service request.
19	CLK	I		Clock. Signal phase 8 MHz clock (50% duty cycle).
20	GND			Ground. DC power return.
21	WR CLK	I		Write Clock. 500 kHz (FM) or 1 MHz (MFM) write clock with a constant pulse width of 250 ns (for both FM and MFM recording). The write clock must be present at all times.
22	DW	I	PLL	Data Window. Data sample signal from the phase-locked loop indicating that the FDC should sample input data from the disk drive.
23	RD DATA	I	Drive	Read Data. FDC input data from the selected disk drive.
24	VCO	O	PLL	VCO Sync. Active-high output that enables the phase-locked loop to synchronize with the input data from the disk drive.
25	WE	O	Drive	Write Enable. Active-high output that enables the disk drive write gate.
26	MFM	O	PLL	MFM Mode. Active-high output used by external logic to enable the MFM double-density recording mode. When the MFM output is low, single-density FM recording is indicated.
27	HDSEL	O	Drive	Head Select. Selects head 0 or head 1 on a dual-sided disk.
28,29	DS_1,DS_0	O	Drive	Drive Select. Selects one of four disk drives.
30	WR DATA	O	Drive	Write Data. Serial data stream (combination of clock and data bits) to be written on the disk.
31,32	PS_1,PS_0	O	Drive	Precompensation (pre-shift) Control. Write precompensation output control during MFM mode. Specifies early, late, and normal timing signals. See the discussion in Section 5.

Table 2. 8272 FDC Pin Description (continued)

Number	Pin Symbol	I/O	To/From	Description
33	FLT/TRKO	I	Drive	Fault/Track 0. Senses the disk drive fault condition in the Read/Write mode and the Track 0 condition in the Seek mode.
34	WP/TS	I	Drive	Write Protect/Two-Sided. Senses the disk write protect status in the Read/Write mode and the dual-sided media status in the Seek mode.
35	RDY	I	Drive	Ready. Senses the disk drive ready status.
36	HDL	O	Drive	Head Load. Loads the disk drive read/write head. (The head is placed in contact with the disk.)
37	FR/STP	O	Drive	Fault Reset/Step. Resets the fault flip-flop in the disk drive when operating in the Read/Write mode. Provides head step pulses (to move the head from one cylinder to another cylinder) in the Seek mode.
38	LCT/DIR	O	Drive	Low Current/Direction. Signals that the recording head has been positioned over the inner cylinders (44-77) of the floppy disk in the Read/Write mode. (The write current must be lowered when recording on the physically shorter inner cylinders of the disk. Most drives do not track the actual head position and require that the FDC supply this signal.) Determines the head step direction in the Seek mode. In the Seek mode, a high level on this pin steps the read/write head toward the spindle (step-in); a low level steps the head away from the spindle (step-out).
39	RW/SEEK	O	Drive	Read, Write/Seek Mode Selector. A high level selects the Seek mode; a low level selects the Read/Write mode.
40	V_{CC}			+5V DC Power.

*Disabled when CS is high.

Interface Registers

To support information transfer between the FDC and the system processor, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 4) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations. Data is read from, or written to, the FDC registers by the combination of the A0, $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ signals, as described in Table 3.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Table 3. FDC Read/Write Interface

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Function
0	0	0	1	Read Main Status Register
0	0	1	0	Illegal
0	0	0	0	Illegal
0	1	0	0	Illegal
0	1	0	1	Read from Data Register
0	1	1	0	Write into Data Register
1	X	X	X	Data Bus is three-stated

Table 4. Main Status Register Bit Definitions

Bit Number	Symbol	Description
0	D_0B	Disk Drive 0 Busy. Disk Drive 0 is in the Seek mode.
1	D_1B	Disk Drive 1 Busy. Disk Drive 1 is in the Seek mode.
2	D_2B	Disk Drive 2 Busy. Disk Drive 2 is in the Seek mode.
3	D_3B	Disk Drive 3 Busy. Disk Drive 3 is in the Seek mode.
4	CB	FDC Busy. A read or write command is in process.
5	NDM	Non-DMA Mode. The FDC is in the non-DMA mode when this bit is high. This bit is set only during the execution phase of commands in the non-DMA mode. Transition to a low level indicates that the execution phase has ended.
6	DIO	Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is high, data is read from the Data Register by the processor; when DIO is low, data is written from the processor to the Data Register.
7	RQM	Request for Master. Indicates that the Data Register is ready to send data to, or receive data from, the processor.

Command/Result Phases

Table 5 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 6, and the contents of the result status registers (ST0-ST3) are illustrated in Table 7.

The bytes of data which are sent to the 8272 during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 5. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the command is automatically ended and the 8272 is ready for a new command. A command may be aborted by simply raising the terminal count signal (pin 16). This is a convenient means of ensuring that the processor may always gain control of the 8272 (even if the disk system hangs up in an abnormal manner).

It is important to note that during the result phase all bytes shown in Table 5 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the system processor must examine the Main Status Register to determine if the Data Register is available. The DI0 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DI0 and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1. DMA mode
2. non-DMA mode

In the DMA mode, DRQ (DMA Request) is activated for each transfer request. The DMA controller responds to DRQ with $\overline{DACK}$ (DMA Acknowledge) and $\overline{RD}$ (for read commands) or $\overline{WR}$ (for write commands). DRQ is reset by the FDC during the transfer. INT is activated after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase. In the DMA mode, the terminal count (TC/EOP) output of the DMA controller should be connected to the 8272 TC input to properly terminate disk data transfer commands.

Table 5. 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after command execution
	R				H					
	R				R					
	R				N					
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EC 1					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the main-system and the FDD
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information during Execution Phase
	R				H					
	R				R					
	R				N					
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				N					Bytes/Sector
	W				SC					Sectors/Track
	W				GPL					Gap 3
	W				D					Filter Byte
Execution										FDC formats an entire track
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					In this case, the ID information has no meaning
	R				H					
	R				R					
	R				N					
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

Note: 1. $A_0 = 1$ for all operations.

Table 5. Command Set (Continued)

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R				ST 0					Status information at the end of each seek operation about the FDC
	R				C					
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SPT ◄							HUT	Timer Settings
	W	HLT ►							ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R				ST 3					Status information about the FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W				Invalid Codes					Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R				ST 0					ST 0 = 80 (16)

Table 6. Command/Result Parameter Abbreviations

Symbol	Description	Symbol	Description
C	Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.	EOT	End of Track. The final sector number of the current track.
D	Data Pattern. The pattern to be written in each sector data field during formatting.	GPL	Gap Length. The gap 3 size. (Gap 3 is the space between sectors excluding the VCO synchronization field as defined in section 3.)
DS0,DS1	Disk Drive Select. DS1 DS0 0 0 Drive 0 0 1 Drive 1 1 0 Drive 2 1 1 Drive 3	H	Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
DTL	Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the diskette) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.	HLT	Head Load Time. Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
		HUT	Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
		MFM	MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.

Table 6. Command/Result Parameter Abbreviations (continued)

Symbol	Description	Symbol	Description
MT	Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read opeation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.	SK	Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.
		SRT	Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).
N	Sector Size. The number of data bytes within a sector. (See Table 9.)	ST0 ST1 ST2	Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is
ND	Non-DMA Mode Flag. When set (high), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor is interrupted for each data transfer. When low, the FDC interfaces to a DMA controller by means of the DRQ and DACK signals.	ST3	available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 5 for each command). These registers should not be confused with the Main Status Register.
R	Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.	STP	Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.
SC	Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.		

Table 7. Status Register Definitions

Bit Number	Symbol	Description
Status Register 0		
7,6	IC	Interrupt Code. 00 — Normal termination of command. The specified command was properly executed and completed without error. 01 — Abnormal termination of command. Command execution was started but could not be successfully completed. 10 — Invalid command. The requested command could not be executed. 11 — Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	Seek End. This flag is set (high) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	Equipment Check Error. This flag is set (high) if a fault signal is received from the disk drive or if the track 0 signal fails to become active after 77 step pulses (Recalibrate command).
3	NR	Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	Head Address. The head address at the time of the interrupt.
1,0	DS1,DS0	Drive Select. The number of the drive selected at the time of the interrupt.

Table 7. Status Register Definitions (continued)

Bit Number	Symbol	Description
Status Register 1		
7	EN	End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		Not used. This bit is always low.
5	DE	Data Error. Set when the FDC detects a CRC error in either the ID field or the data field of a sector.
4	OR	Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.
3		Not used. This bit is always low.
2	ND	Sector Not Found Error. This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	Missing Address Mark Error. This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two occurrences of the physical index mark). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)
Status Register 2		
7		Not used. This bit is always low.
6	CM	Control Mark. This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	Data Error. Set (high) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.
4	WC	Cylinder Address Error. Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	Scan Hit. Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	Scan Not Satisfied. Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.
1	BC	Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

Table 7. Status Register Definitions (continued)

Bit Number	Symbol	Description
Status Register 3		
7	FT	Fault. This flag indicates the status of the fault signal from the selected disk drive.
6	WP	Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	Ready. This flag indicates the status of the ready signal from the selected disk drive.
4	TO	Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.
1,0	DS1,DS0	Drive Select. Indicates the currently selected disk drive number.

In the non-DMA mode, transfer requests are indicated by activation of both the INT output signal and the RQM flag (bit 7) in the Main Status Register. INT can be used for interrupt-driven systems and RQM can be used for polled systems. The system processor must respond to the transfer request by reading data from (activating $\overline{RD}$), or writing data to (activating $\overline{WR}$), the FDC. This response removes the transfer request (INT and RQM are set inactive). After completing the last transfer, the 8272 activates the INT output to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the TC signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the processor) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a terminal count signal is sensed or when the last sector on a track (the EOT parameter—Table 5) has been read or written. In addition, if the disk drive is in a "not ready" state at the beginning of the execution phase, the "not ready" flag (bit 3 in Status Register 0) is set (high) and the command is terminated.

If a fault signal is received from the disk drive at the end of a write operation (Write Data, Write Deleted Data, or Format), the FDC sets the "equipment check" flag (bit 4 in Status Register 0), and terminates the command after setting the interrupt code (bits 7 and 6 of Status Register 0) to "01" (bit 7 low, bit 6 high).

Multi-sector and Multi-track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally connected to the TC/EOP (terminal count) output of the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system processor or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte—Table 5) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system processor when a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DI0 (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DI0 to be set high and RQM to be set low.

The following paragraphs describe the fifteen FDC commands in detail.

Specify

The Specify command is used prior to performing any disk operations (including the formatting of a new disk) to define drive/FDC operating characteristics. The Specify command parameters set the values for three internal timers:

1. Head Load Time (HLT) — This seven-bit value defines the time interval that the FDC waits after loading the head before initiating a read or write operation. This timer is programmable from 2 to 254 milliseconds in increments of 2 ms.
2. Head Unload Time (HUT) — This four-bit value defines the time from the end of the execution phase (of a read or write command) until the head is unloaded. This timer is programmable from 16 to 240 milliseconds in increments of 16 ms. If the processor issues another command before the head unloads, the head will remain loaded and the head load wait will be eliminated.
3. Step Rate Time (SRT) — This four-bit value defines the time interval between step pulses issued by the FDC (track-to-track access time). This timer is programmable from 1 to 16 milliseconds in increments of 1 ms.

The time intervals mentioned above are a direct function of the FDC clock (CLK on pin 19). Times indicated above are for an 8 MHz clock.

The Specify command also indicates the choice of DMA or non-DMA operation (by means of the ND bit). When this bit is high the non-DMA mode is selected; when ND is low, the DMA mode is selected.

Sense Drive Status

This command may be used by the processor whenever it wishes to obtain the status of the disk drives. Status Register 3 (returned during the result phase) contains the drive status information as described in Table 7.

Sense Interrupt Status

An interrupt signal is generated by the FDC when one or more of the following events occurs:

1. The FDC enters the result phase for:
   a. Read Data command
   b. Read Track command
   c. Read ID command
   d. Read Deleted Data command
   e. Write Data command
   f. Format Track command
   g. Write Deleted Data command
   h. Scan commands
2. The ready signal from one of the disk drives changes state.
3. A Seek or Recalibrate command completes operation.
4. The FDC requires a data transfer during the execution phase of a command in the non-DMA mode.

Interrupts caused by reasons (1) and (4) above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons (2) and (3) above are uniquely identified with the aid of the Sense Interrupt Status command. This command, when issued, resets the interrupt signal and by means of bits 5, 6, and 7 of Status Register 0 (returned during the result phase) identifies the cause of the interrupt (see Table 8).

Table 8. Interrupt Codes

Seek End Bit 5	Interrupt Code		Cause
	Bit 6	Bit 7	
0	1	1	Ready Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

Neither the Seek nor the Recalibrate command has a result phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the disk head position.

When an interrupt is received by the processor, the FDC busy flag (bit 4) and the non-DMA flag (bit 5) may be used to distinguish the above interrupt causes:

bit 5	bit 4	
0	0	Asynchronous event-(2) or (3) above
0	1	Result phase-(1) above
1	1	Data transfer required-(4) above

A single interrupt request to the processor may, in fact, be caused by more than one of the above events. The processor should continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are serviced.

Seek

The Seek command causes the drive's read/write head to be positioned over the specified cylinder. The FDC determines the difference between the current cylinder address and the desired (specified) address, and issues the appropriate number of step pulses. If the desired cylinder address is larger than the current address, the direction signal (LCT/DIR, pin 38) is set high (step-in); the direction signal is set low (step-out) if the desired cylinder address is less than the current address. No head movement occurs (no step pulses are issued) if the desired cylinder is the same as the current cylinder.

The rate at which step pulses are issued is controlled by the step rate time (SRT) in the Specify command. After each step pulse is issued, the desired cylinder address is compared against the current cylinder address. When the cylinder addresses are equal, the "seek end" flag (bit 5 in Status Register 0) is set (high) and the command is terminated. If the disk drive becomes "not ready" during the seek operation, the "not ready" flag (in Status Register 0) is set (high) and the command is terminated.

During the command phase of the Seek operation the FDC is in the FDC busy state, but during the execution phase it is in the non-busy state. While the FDC is in the non-busy state, another Seek command may be issued. In this manner parallel seek operations may be in operation on up to four floppy disk drives at once. The Main Status Register contains a flag for each drive (Table 4) that indicates whether the associated drive is currently operating in the seek mode. When a drive has completed a seek operation, the FDC generates an interrupt. In response to this interrupt, the system software must issue a Sense Interrupt Status command. During the result phase of this command, Status Register 0 (containing the drive number in bits 0 and 1) is read by the processor.

Recalibrate

This command causes the read/write head of the disk drive to retract to the track 0 position. The FDC clears the contents of its internal cylinder counter, and checks the status of the track 0 signal from the disk drive. As long as the track 0 signal is low, the direction signal remains high and step pulses are issued. When the track 0 signal goes high, the seek end flag (in Status Register 0) is set (high) and the command is terminated. If the track 0 signal is still low after 77 step pulses have been issued, the seek end and equipment check flags (in Status Register 0) are both set and the Recalibrate command is terminated.

Recalibrate commands for multiple drives can be overlapped in the same manner that Seek commands are overlapped.

Format Track

The Format Track command formats or "initializes" a track on a floppy disk by writing the ID field, gaps, and address marks for each sector. Before issuing the Format command, the Seek command must be used to position the read/write head over the correct cylinder. In addition, a table of ID field values (cylinder, head, and sector addresses and sector length code) must be prepared before the command is executed. During command execution, the FDC accesses the table and, using the values supplied, writes each sector on the track. The ID field address mark originates from the FDC and is written automatically as the first byte of each sector's ID field. The cylinder, head, and sector addresses are taken, in order, from the table. The ID field CRC character (derived from the data written in the first five bytes) is written as the last two bytes of the ID field. Gaps are written automatically by the FDC, with the length of the variable gap determined by one of the Format command parameters.

The data field address mark is generated by the FDC and is written automatically as the first byte of the data field. The data pattern specified in the command phase is written into each data byte of each sector. A CRC character is derived from the data address mark and the data written in the sector's data field. The two CRC bytes are appended to the last data byte.

The formatting of a track begins at the physical index mark. As previously mentioned, the order of sector assignment is taken directly from the formatting table. Four entries are required for each sector: a cylinder address, a head address, a sector address, and a sector length code. The cylinder address in the ID field should be equal to the cylinder address of the track currently being formatted.

The sector addresses must be unique (no two equal). The order of the sector entries in the table is the sequence in which sector numbers appear on the track when it is formatted. The number of entry sets (cylinder, head, and sector address and sector length code) must equal the number of sectors allocated to the track (specified in the command phase).

Since the sector address is supplied, in order, for each sector, tracks can be formatted sequentially (the first sector following the index mark is assigned sector address 1, the adjacent sector is assigned sector address 2, and so on) or sector numbers can be interleaved (see section 3) on a track.

Table 9 lists recommended gap sizes and sectors/track for various sector sizes.

Read Data

Nine (9) bytes are required to complete the command phase specification for the Read Data command. During the execution phase, the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined in the Specify command), and begins reading ID address marks and ID fields. When the requested sector address compares with the sector address read from the disk, the FDC outputs data (from the data field) byte-by-byte to the system. The Read Data command automatically operates in the multi-sector mode described earlier. In addition, multi-track operation may be specified by means of the MT command flag (Table 5). The amount of data that can be transferred with a single command to the FDC depends on the multi-track flag, the recording density flag, and the number of bytes per sector.

During the execution of read and write commands, the special sector size parameter (DTL) is used to temporarily alter the effective disk sector size. By setting the sector size code (N) to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, only the number of bytes specified by the DTL parameter are passed to the system; the remainder of the actual disk sector is not transferred (although the data is checked for CRC errors). Multi-sector read operations are performed in the same manner as they are when the sector size code is non-zero. (The N and DTL parameters are always present in the command sequence. DTL should be set to FF hexadecimal when N is not zero.)

If the FDC detects the physical index mark twice without finding the requested sector, the FDC sets the "sector not found error" flag (bit 2 in Status Register 1) and terminates the Read Data command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01." Note that the FDC searches for each sector in a multi-sector operation. Therefore, a "sector not found" error may occur after successful transfer of one or more preceding sectors. This error could occur if a particular sector number was not included when the track was first formatted or if a hard error on the disk has invalidated a sector ID field.

After reading the ID field and data field in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in the ID field), the FDC sets the "data error" flag in Status Register 1; if a CRC error occurs in the data field, the FDC sets the "data error" flag in Status Register 2. In either error condition, the FDC terminates the Read Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

If the FDC reads a deleted data address mark from the disk, and the skip flag (specified during the command phase) is not set, the FDC sets the "control mark" flag (bit 6 in Status Register 2) and terminates the Read Data command (after reading all the data in the sector). If the skip flag is set, the FDC skips the sector with the deleted data address mark and reads the next sector. Thus, the skip flag may be used to cause the FDC to ignore deleted data sectors during a multi-sector read operation.

During disk data transfers between the FDC and the system, the FDC must be serviced by the system (processor or DMA controller) every 27 μs in the FM mode, and every 13 μs in the MFM mode. If the FDC is not

Table 9. Sector Size Relationships

Format	Sector Size	N Sector Size Code	SC Sectors/ Track	GPL[1] Gap 3 Length	GPL[2] Gap 3 Length	Remarks
FM Mode	128 bytes/Sector	00	$1A_{(16)}$	$07_{(16)}$	$1B_{(16)}$	IBM Diskette 1
	256	01	$0F_{(16)}$	$0E_{(16)}$	$2A_{(16)}$	IBM Diskette 2
	512	02	08	$1B_{(16)}$	$3A_{(16)}$	
MFM Mode	256	01	$1A_{(16)}$	$0E_{(16)}$	$36_{(16)}$	IBM Diskette 2D
	512	02	$0F_{(16)}$	$1B_{(16)}$	$54_{(16)}$	
	1024	03	08	$35_{(16)}$	$74_{(16)}$	IBM Diskette 2D

Notes: 1. Suggested values of GPL in Read or Write commands to avoid splice point between data field and ID field of contiguous sectors.
2. Suggested values of GPL in Format command.

serviced within this interval, the "overrun error" flag (bit 4 in Status Register 1) is set and the Read Data command is terminated.

If the processor terminates a read (or write) operation in the FDC, the ID information in the result phase is dependent upon the state of the multi-track flag and end of track byte. Table 11 shows the values for C, H, R, and N, when the processor terminates the command.

Write Data

Nine (9) bytes are required to complete the command phase specification for the Write Data command. During the execution phase the FDC loads the head (if it is in the unloaded state), waits the specified head load time (defined by the Specify command), and begins reading sector ID fields. When the requested sector address compares with the sector address read from the disk, the FDC reads data from the processor one byte at a time via the data bus and outputs the data to the data field of that sector. The CRC is computed on this data and two CRC bytes are written at the end of the data field.

The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID fields, it sets the "data error" flag (bit 5 in Status Register 1) and terminates the Write Data command. The interrupt code (bits 7 and 6 in Status Register 0) is set to "01."

The Write Data command operates in much the same manner as the Read Data command. The following items are the same; refer to the Read Data command for details:

- Multi-sector and Multi-track operation
- Data transfer capacity
- "End of track error" flag
- "Sector not found error" flag
- "Data error" flag
- Head unload time interval
- ID information when the processor terminates the command (see Table 11)
- Definition of DTL when $N=0$ and when $N\neq 0$

During the Write Data execution phase, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this, the FDC sets the "overrun error" flag (bit 4 in Status Register 1) and terminates the Write Data command.

Read Deleted Data

This command operates in almost the same manner as the Read Data command operates. The only difference involves the treatment of the data address mark and the skip flag. When the FDC detects a data address mark at the beginning of a data field (and the skip flag is not set), the FDC reads all the data in the sector, sets the "control mark" flag (bit 6 in Status Register 2), and terminates the command. If the skip flag is set, the FDC skips the sector with the data address mark and continues reading at the next sector. Thus, the skip flag may be used to cause the FDC to read only deleted data sectors during a multi-sector read operation.

Write Deleted Data

This command operates in the same manner as the Write Data command operates except that a deleted data address mark is written at the beginning of the data field instead of the normal data address mark. This command is used to mark a bad sector (containing a hard error) on the floppy disk.

Read Track

The Read Track command is similar to the Read Data command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the physical index mark, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID field or data field CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the values specified during the command phase. If the specified ID field information is not found on the track, the "sector not found error" flag (in Status Register 1) is set. Multi-track and skip operations are not allowed with this command.

This command terminates when the last sector on the track has been read. (The number of sectors on the track is specified by the end of track parameter byte during the command phase.) If the FDC does not find an ID address mark on the disk after it encounters the physical index mark for the second time, it sets the "missing address mark error" flag (bit 0 in Status Register 1) and terminates the command. The interrupt code (bits 7 and 6 of Status Register 0) is set to "01."

Read ID

The Read ID command transfers (reads) the first correct ID field from the current disk track (following the physical index mark) to the processor. If no correct ID address mark is found on the track, the "missing address mark error" flag is set (bit 0 in Status Register 1). If no data mark is found on the track, the "sector not found error" flag is also set (bit 2 in Status Register 1). Either error condition causes the command to be terminated.

Scan Commands

The Scan commands allow the data being read from the disk to be compared against data supplied by the system (by the processor in non-DMA mode, and by the DMA controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and searches for a sector of data that meets the conditions of "disk data equal to system data", "disk data less than or equal to system data", or "disk data greater than or equal to system data". Simple binary (ones complement) arithmetic is used for comparison (FF = largest number, 00 = smallest number). If, after a complete sector of data is compared, the conditions are not met, the sector number is incremented by the scan sector increment (specified in the command phase), and the scan operation is continued. The scan operation continues until one of the following conditions occurs; the conditions for scan are met (equal, low, or high), the last sector on the track is reached, or the terminal count signal is received.

If the conditions for scan are met, the FDC sets the "scan hit" flag (bit 3 in Status Register 2) and terminates the Scan command. If the conditions for scan are not met between the starting sector and the last sector on the track (specified in the command phase), the FDC sets the "scan not satisfied" flag (bit 2 in Status Register 2) and terminates the Scan command. The receipt of a terminal count signal from the processor or DMA controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and to terminate the command. Table 10 shows the status of the "scan hit" and "scan

Table 10. Scan Status Codes

Command	Status Register 2		Comments
	Bit 2 = SN	Bit 3 = SH	
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} \not< D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} \not> D_{Processor}$

Table 11. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase			
			C	H	R	N
0	1A	Sector 1 to 25 at Side 0	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	C+1	NC	R=01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C+1	NC	R=01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				
1	1A	Sector 1 to 25 at Side 0	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	NC	$\overline{LSB}$	R=01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C+1	$\overline{LSB}$	R=01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				

Notes: 1. NC (No Change): The same value as the one at the beginning of command execution.
2. $\overline{LSB}$ (Least Significant Bit): The least significant bit of H is complemented.

not satisfied" flags under various scan termination conditions.

If the FDC encounters a deleted data address mark in one of the sectors and the skip flag is low, it regards the sector as the last sector on the cylinder, sets the "control mark" flag (bit 6 in Status Register 2) and terminates the command. If the skip flag is high, the FDC skips the sector with the deleted address mark, and reads the next sector. In this case, the FDC also sets the "control mark" flag (bit 6 in Status Register 2) in order to show that a deleted sector had been encountered.

NOTE: During scan command execution, the last sector on the track must be read for the command to terminate properly. For example, if the scan sector increment is set to 2, the end of track parameter is set to 26, and the scan begins at sector 21, sectors 21, 23, and 25 will be scanned. The next sector, 27 will not be found on the track and an abnormal command termination will occur. The command would be completed in a normal manner if either a) the scan had started at sector 20 or b) the end of track parameter had been set to 25.

During the Scan command, data is supplied by the processor or DMA controller for comparison against the data read from the disk. In order to avoid having the "overrun error" flag set (bit 4 in Status Register 1), it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an overrun error occurs, the FDC terminates the command.

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

In some applications the user may wish to use this command as a No-Op command to place the FDC in a stand-by or no operation state.

5. THE DATA SEPARATOR

As briefly discussed in section 2, LSI disk controllers such as the 8272 require external circuitry to generate a data window signal. This signal is used within the FDC to isolate the data bits contained within the READ DATA input signal from the disk drive. (The disk READ DATA signal is a composite signal constructed from both clock and data information.) After isolating the data bits from this input signal, the FDC assembles the data bits into 8-bit bytes for transfer to the system processor or memory.

Single Density

In single-density (FM) recording (Figure 3), the bit cell is 4 microseconds wide. Each bit cell contains a clock bit at the leading edge of the cell. The data bit (if present) is always located at the center of the cell. The job of data separation is relatively straightforward for single-density; simply generate a data window 2 μs wide starting 1 μs after each clock bit. Since every cell has a clock bit, a fixed window reference is available for every data bit and because the window is 2 μs wide, a slightly shifted data bit will still remain within the data window.

A single-density data separator with these specifications may be easily generated using a digital or analog one-shot triggered by the clock bit.

Double-Density

Double-density (MFM) bit cells are reduced to 2 μs (in order to double the disk data storage capacity). Clock bits are inserted into the data stream only if data bits are not present in both the current and preceding bit cells (Figure 3). The data bit (if present) still occurs at the center of the bit cell and the clock bit (if present) still occurs at the leading edge of the bit cell.

MFM data separation has two problems. First, only some bit cells contain a clock bit. In this manner, MFM encoding loses the fixed bit cell reference pulse present in FM encoding. Second, the bit cell for MFM is one-half the size of the bit cell for FM. This shorter bit cell means that MFM cannot tolerate as large a playback data-shift (as FM can tolerate) without errors.

Since most playback data-shift is predictable, the FDC can precompensate the write data stream so that data/clock pulses will be correctly positioned for subsequent playback. This function is completely controlled by the FDC and is only required for MFM recording. During write operations, the FDC specifies an early, normal, or late bit positioning. This timing information is specified with respect to the FDC write clock. Early and late timing is typically 125 ns to 250 ns before or after the write clock transition (depending on disk drive requirements).

The data separator circuitry for double-density recording must continuously analyze the total READ DATA stream, synchronizing its operation (window generation) with the actual clock/data bits of the data stream. The data separation circuit must track the disk input data frequency very closely—unpredictable bit shifts leave less than 50 ns margin to the window edges.

Phase-Locked Loop

Only an analog phase-locked loop (PLL) can provide the reliability required for a double-density data separation circuit. (A phase-locked loop is an electronic circuit that constantly analyzes the frequency of an input signal and locks another oscillator to that frequency.) Using analog PLL techniques, a data separator can be designed with ±1 ns resolution (this would require a 100 MHz clock in a digital phase-locked loop). The analog PLL determines the clock and data bit positions by sampling each bit in the serial data stream. The phase relationship between a data bit and the PLL generated data window is constantly fed back to adjust the position of the data window, enabling the PLL to track input data frequency changes, and thereby reliably read previously recorded data from a floppy disk.

PLL Design

A block diagram of the phase-locked loop described in this application note is shown in Figure 7. Basically, the phase-locked loop operates by comparing the frequency of the input data (from the disk drive) against the frequency of a local oscillator. The difference of these frequencies is used to increase or decrease the frequency of the local oscillator in order to bring its frequency closer to that of the input. The PLL synchronizes the local oscillator to the frequency of the input during the all "zeroes" synchronization field on the floppy disk (immediately preceding both the ID field and the data field).

The PLL consists of nine ICs and is located on page 3 of the schematics in the Appendix. The 8272 VCO output essentially turns the PLL circuitry on and off. When the PLL is off, it "idles" at its center frequency. The VCO output turns the PLL on only when valid data is being received from the disk drive. The VCO turns the PLL on after the read/write head has been loaded and the head load time has elapsed. The PLL is turned off in the gap between the ID field and the data field and in the gap after the data field (before the next sector ID field). The GPL parameter in the FDC read and write commands specifies the elapsed time (number of data bytes) that the PLL is turned off in order to blank out discontinuities that appear in the gaps when the write current is turned on and off. The PLL operates with either MFM or FM input data. The MFM output from the FDC controls the PLL operation frequency.

The PLL consists of six functional blocks as follows:

1. Pulse Shaping — A 96LS02 senses a READ DATA pulse and provides a clean output signal to the FDC and to the PLL Phase Comparator and Frequency Discriminator circuitry.
2. Phase Comparator — The phase difference between the PLL oscillator and the READ DATA input is compared. Pump up (PU) and pump down (PD) error signals are derived from this phase difference and output to the filter. If there is no phase difference between the PLL oscillator and the READ DATA input, the PU and PD pulse widths are equal. If the READ DATA pulse occurs early, the PU duration is shorter than the PD duration. If the data pulse occurs late, the PU duration is longer than the PD duration.
3. Filter — This analog circuit filters the PU and PD pulses into an error voltage. This error voltage is buffered by an LM358 operational amplifier.

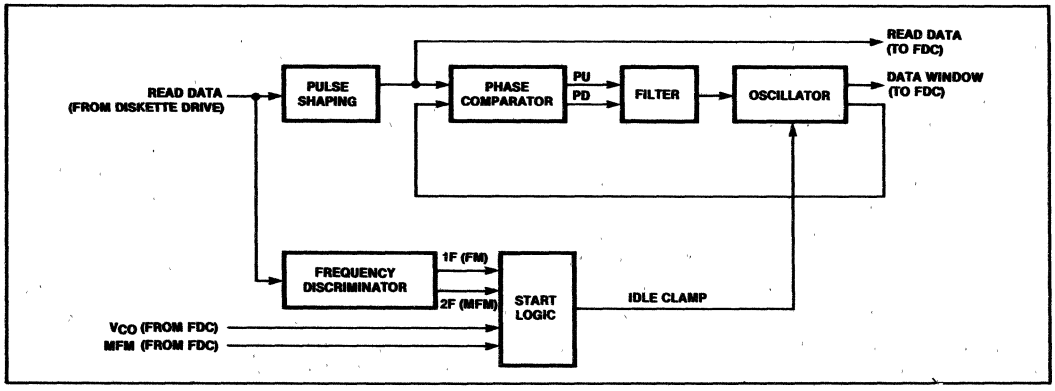


Figure 7. Phase-Locked Loop Data Separator

4. PLL Oscillator — This oscillator is composed of a 74LS393, 74LS74, and 96LS02. The oscillator frequency is controlled by the error voltage output by the filter. This oscillator also generates the data window signal to the FDC.
5. Frequency Discriminator — This logic tracks the READ DATA input from the disk drive and discriminates between the synchronization gap for FM recording (250 KHz) and the gap for MFM recording (500 KHz). Synchronization gaps immediately precede address marks.
6. Start Logic — The function of this logic is to clamp the PLL oscillator to its center frequency (2 MHz) until the FDC VCO signal is enabled and a valid data pattern is sensed by the frequency discriminator. The start logic (consisting of a 74LS393 and 74LS74) ensures that the PLL oscillator is started with zero phase error.

PLL Adjustments

The PLL must be initially adjusted to operate at its center frequency with the VCO output off and the adjustment jumper removed. The 5K trimpot should be adjusted until the frequency at the test point (Q output of the 96LS02) is 2 MHz. The jumper should then be replaced for normal operation.

PLL Design Details

The following paragraphs describe the operational and design details of the phase-locked loop data separator illustrated in the appendix. Note that the analog section is operated from a separately filtered +5V supply.

Initialization

As long as the 8272 maintains a low VCO signal, the data separator logic is "turned off". In this state, the PLL oscillator (96LS02) is not oscillating and therefore the 2XBR signal is constantly low. In addition, the pump up (PU) and pump down (PD) signals are inactive (PU low and PD high), the CNT8 signal is inactive (low), and the filter input voltage is held at 2.5 volts by two 1Mohm resistors between ground and +5 volts.

Floppy Disk Data

The data separator frequency discriminator, the input pulse shaping circuitry, and the start logic are always enabled and respond to rising edges of the READ DATA signal. The rising edge of every data bit from the disk drive triggers two pulse shaping one-shots. The first pulse shaper generates a stable and well-defined 200 ns read data pulse for input to the 8272 and other portions of the data separator logic. The second one-shot generates a 2.5 μs data pulse that is used for input data frequency discrimination.

The frequency discriminator operates as illustrated in Figure 8. The 2F output signal is active (high) during reception of valid MFM (double-density) sync fields on the disk while the 1F signal is active (high) during reception of valid FM (single-density) sync fields. A multiplexer (controlled by the 8272 MFM signal) selects the appropriate 1F or 2F signal depending on the programmed mode.

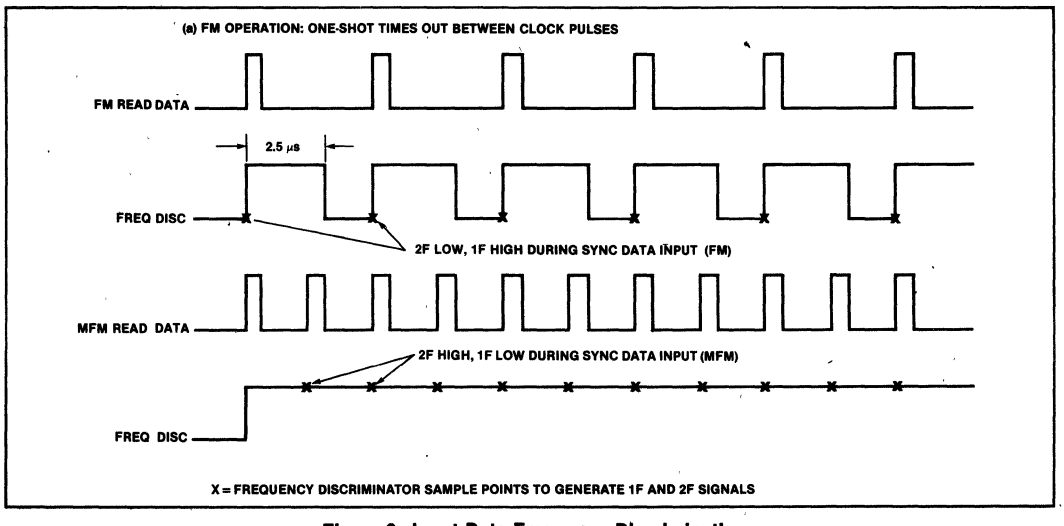


Figure 8. Input Data Frequency Discrimination

Startup

The data separator is designed to require reception of eight valid sync bits (one sync byte) before enabling the PLL oscillator and attempting to synchronize with the input data stream (see Figure 9). This delay ensures that the PLL will not erroneously synchronize outside a valid sync field in the data stream if the VCO signal is enabled slightly early. The sync bit counter is asynchronously reset by the CNTEN signal when valid sync data is not being received by the drive.

Once the VCO signal is active and eight sync bits have been counted, the CNT8 signal is enabled. This signal turns on the PLL oscillator. Note that this oscillator starts synchronously with the rising edge of the disk input data (because CNT8 is synchronous with the data rising edge) and the oscillator also starts at its center frequency of 2 MHz (because the LM348 filter input is held at its center voltage of approximately 2.5 volts). This frequency is divided by two and four to generate the 2XBR signal (1 MHz for MFM and 500 KHz for FM).

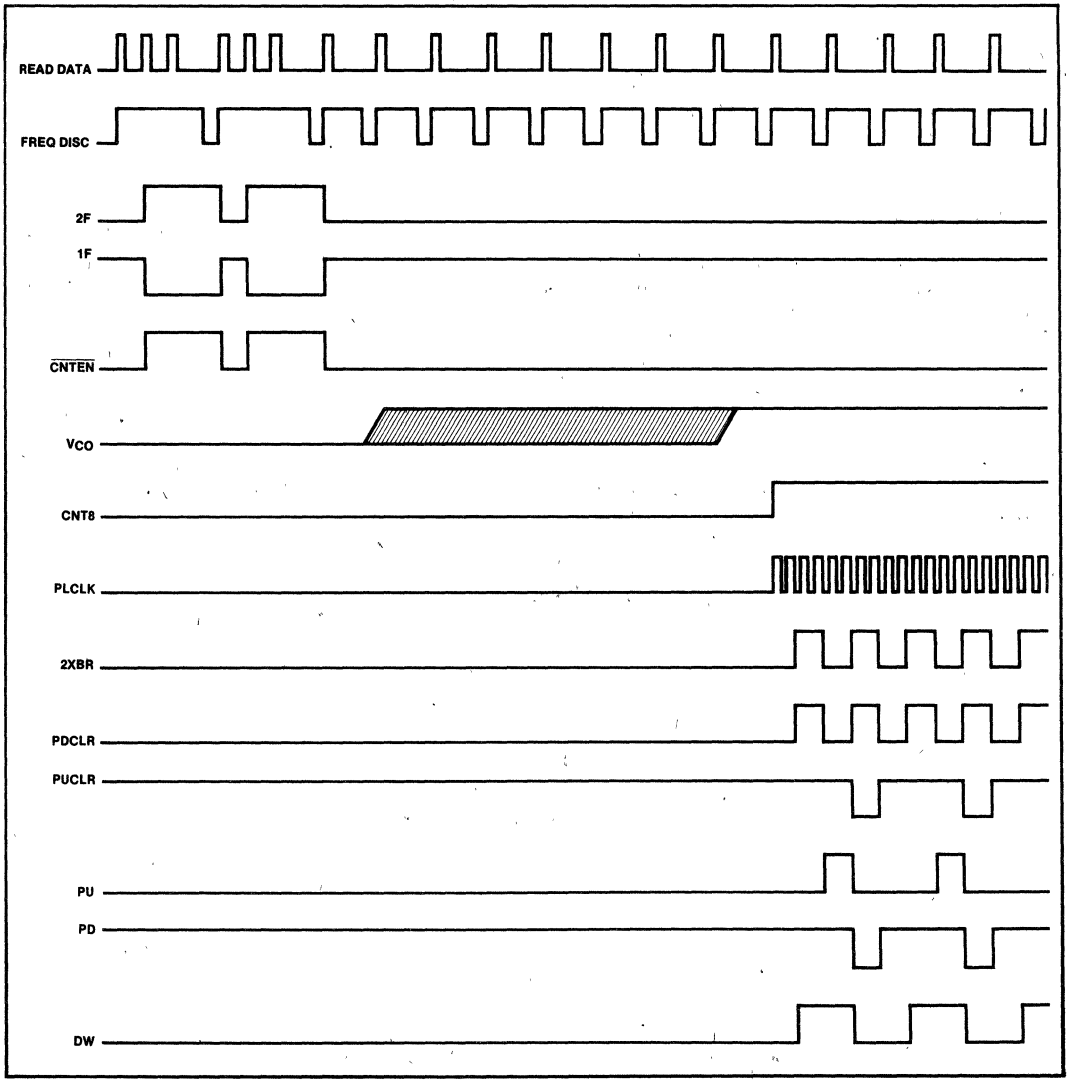


Figure 9. Typical Data Separator Startup Timing Diagram

PLL Synchronization

At this point, the PLL is enabled and begins to synchronize with the input data stream. This synchronization is accomplished very simply in the following manner. The pump up (PU) signal is enabled on the rising edge of the READ DATA from the disk drive. (When the PLL is synchronized with the data stream, this point will occur at the same time as the falling edge of the 2XBR signal as shown in Figure 9). The PU signal is turned off and the PD signal is activated on the next rising edge of the 2XBR clock. With this scheme, the difference between PU active time and the PD active time is equal to the difference between the input bit rate and the PLL clock rate. Thus, if PU is turned on longer than PD is on, the input bit rate is faster than the PLL clock.

As long as PU and PD are both inactive, no charge is transferred to or from the LM358 input holding capacitor, and the PLL output frequency is maintained (the LM358 operational amplifier has a very high input impedance). Whenever PU is turned on, current flows from the +5 volt supply through a 20K resistor into the holding capacitor. When the PD signal is turned on, current flows from the holding capacitor to ground through a 20K resistor. In this manner, both the pump up and pump down charging rates are balanced.

The change in capacitor charge (and therefore voltage) after a complete PU/PD cycle is proportional to the difference between the PU and PD pulse widths and is also proportional to the frequency difference between the incoming data stream and the PLL oscillator. As the capacitor voltage is raised (PU active longer than PD), the PLL oscillator time constant (RC of the 96LS02) is modified by the filter output (LM358) to raise the oscillator frequency. As the capacitor voltage is lowered (PD active longer than PD), the oscillator frequency is lowered. If both frequencies are equal, the voltage on the holding capacitor does not change, and the PLL oscillator frequency remains constant.

6. AN INTELLIGENT DISKETTE DATA BASE SYSTEM

The system described in this application note is designed to function as an intelligent data base controller. The schematics for this data base unit are presented in Appendix A; a block diagram of the unit is illustrated in Figure 10. As designed, the unit can access over four million bytes of mass storage on four floppy disk drives (using a single 8272 FDC); the system can easily be expanded to four FDC devices (and 16 megabytes of online disk storage). Three serial data links are also included. These data links may be used by CRT terminals or other microprocessor systems to access the data base.

Processor and Memory

A high-performance 8088 eight-bit microprocessor (operating at 5 MHz with no wait states) controls system operation. The 8088 was selected because of its memory addressing capabilities and its sophisticated string handling instructions. These features improve the speed of data base search operations. In addition, these capabilities allow the system to be easily upgraded with additional memory, disk drives, and if required, a bubble memory or winchester disk unit.

The schematics for the basic design provide 8K bytes of 2732A high-speed EPROM program storage and 8K bytes of disk directory and file buffer RAM. This memory can easily be expanded to 1 megabyte for performance upgrades.

An 8259A Programmable Interrupt Controller (PIC) is also included in the design to field interrupts from both the serial port and the FDC. This interrupt controller provides a large degree of programming flexibility for the implementation of data base functions in an asynchronous, demand driven environment. The PIC allows the system to accumulate asynchronous data base requests from all serial I/O ports while previously specified data base operations are currently in progress. This feature is made possible by the ability of the 8251A RXRDY signal to cause a processor interrupt. After receiving this interrupt, the processor can temporarily halt work on existing requests and enter the incoming information into a data base request buffer. Once the information has been entered into the buffer, the system can resume its previous processing.

In addition, the PIC permits some portions of data base requests to be processed in parallel. For example, once a disk record has been loaded into a memory buffer, a memory search can proceed in parallel with the loading of the next record. After the FDC completes the record transfer, the memory search will be interrupted and the processor can begin another disk transfer before resuming the memory search.

The bus structure of the system is split into three functional buffered units. A 20-bit address from the processor is latched by three-state transparent 74LS373 devices. When the processor is in control of the address and data busses, these devices are output enabled to the system buffered address bus. All I/O devices are placed directly on the local data bus. Finally, the memory data bus is isolated from the local data bus by an 8286 octal transceiver. The direction of this transceiver is determined by the Memory Read signal, while its output enable is activated by a Memory Read or Memory Write command.

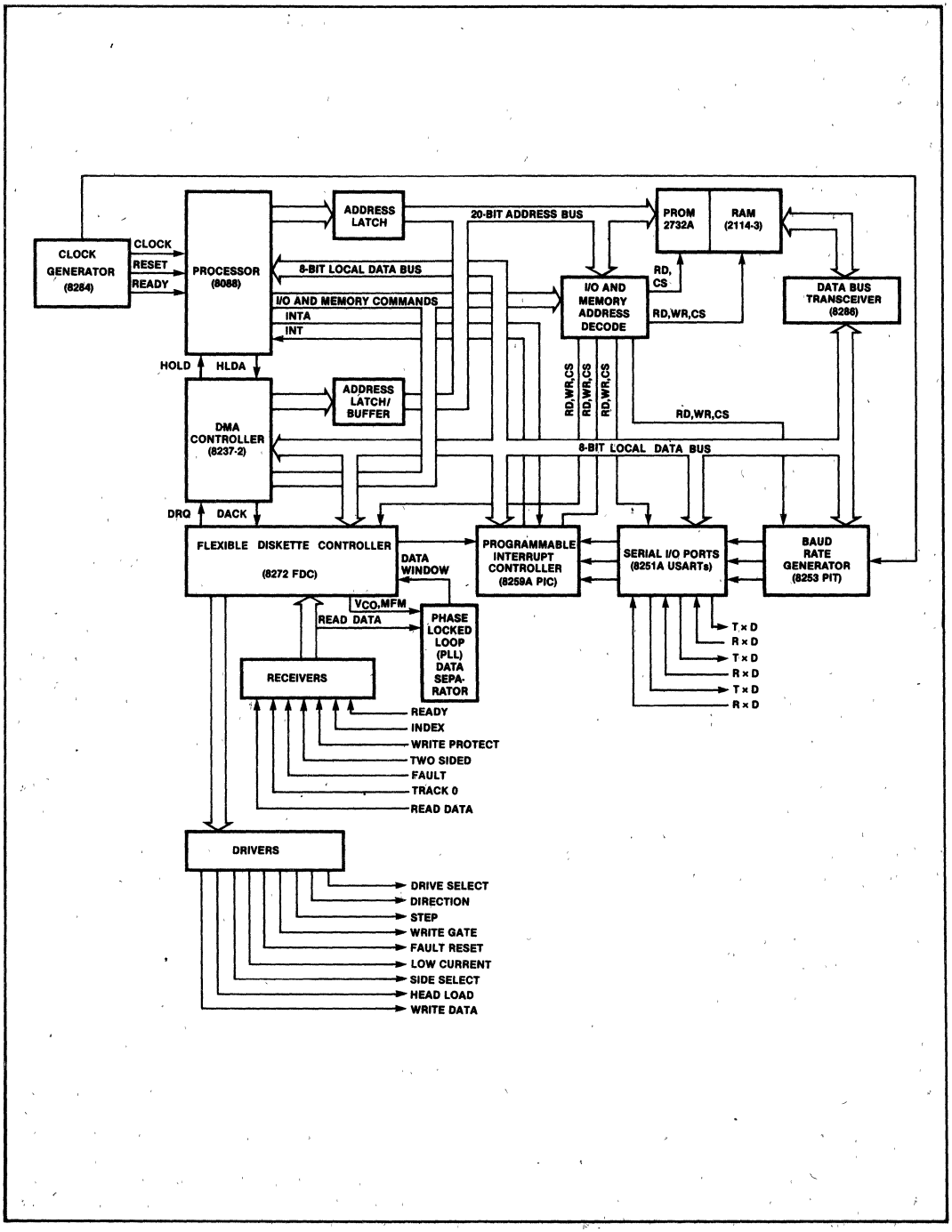


Figure 10. Intelligent Data Base Block Diagram

Serial I/O

The three RS-232-C compatible serial I/O ports operate at software-programmable baud rates to 19.2K. Each I/O port is controlled by an 8251A USART (Universal Synchronous/Asynchronous Receiver/Transmitter). Each USART is individually programmable for operation in many synchronous and asynchronous serial data transmission formats (including IBM Bi-sync). In operation, USART error detection circuits can check for parity, data overrun, and framing errors. An 8253 Programmable Interval Timer is employed to generate the baud rates for the serial I/O ports.

The Transmitter Ready and Receiver Ready output signals of the 8251As are routed to the interrupt inputs of the 8259A interrupt controller. These signals interrupt processor execution when a data byte is received by a USART and also when the USART is ready to accept another data byte for transmission.

DMA

The 8272 FDC interfaces to system memory by means of an 8237-2 high-speed DMA controller. Transfers between the disk controller and memory also operate with no wait states when 2114-3 (150 ns) or faster static RAM is used. In operation, the 8272 presents a DMA request to the 8237 for every byte of data to be transferred. This request causes the 8273 to present a HOLD request to the 8088. As soon as the 8088 is able to relinquish data/address bus control, the processor signals a HOLD acknowledge to the 8237. The 8237 then assumes control over the data and address busses. After latching the address for the DMA transfer, the 8237 generates simultaneous I/O Read and Memory Write commands (for a disk read) or simultaneous I/O Write and Memory Read commands (for a disk write). At the same time, the 8272 is selected as the I/O device by means of the DMA acknowledge signal from the 8237. After this single byte has been transferred between the FDC and memory, the DMA controller releases the data/address busses to the 8088 by deactivating the HOLD request. In a short period of time (13 μs for double-density and 27 μs for single-density) the FDC requests a subsequent data transfer. This transfer occurs in exactly the same manner as the previous transfer. After all data transfers have been completed (specified by the word count programmed into the 8237 before the FDC operation was initiated), the 8237 signals a terminal count (EOP pin). This terminal count signal informs the 8272 that the data transfer is complete. Upon reception of this terminal count signal, the 8272 halts DMA requests and initiates an "operation complete" interrupt.

Since the system is designed for 20-bit addressing, a four-bit DMA-address latch is included as a processor addressable I/O port. The processor writes the upper four DMA address bits before a data transfer. When the DMA controller assumes bus control, the contents of this latch are output enabled on the upper four bits of the address bus. The only restriction in the use of this address latch is that a single disk read or write transfer cannot cross a 64K memory boundary.

Disk Drive Interface

The 8272 FDC may be interfaced to a maximum of four eight-inch floppy disk drives. Both single- and double-density drives are accommodated using the data separation circuit described in section 5. In addition, single- or dual-sided disk drives may be used. The 8272 is driven by an 8 MHz crystal controller clock produced by an 8224 clock generator.

Drive select signals are decoded by means of a 74LS139 from the DS0, DS1 outputs of the FDC. The fault reset, step, low current, and direction outputs to the disk drives are generated from the FR/STEP, LCT/DIR, and RW/SEEK FDC output signals by means of a 74LS240. The other half of the 74LS240 functions as an input multiplexer for the disk write protect, two-sided, fault, and track zero status signals. These signals are multiplexed into the WP/TS and FLT/TRK0 inputs to the 8272.

The 8272 write clock (WR CLK) is generated by a ring counter/multiplexer combination. The write clock frequency is 1 MHz for MFM recording and 500 KHz for FM recording (selected by the MFM output of the 8272). The pulse width is a constant 250 ns. The write clock is constantly generated and input to the FDC (during both read and write operations). The FDC write enable output (WE) is transmitted directly to the write gate disk drive input.

Write data to the disk drive is preshifted (according to the PS0, PS1 FDC outputs) by the combination of a 74LS175 four-bit latch and a 74LS153 multiplexer. The amount of preshift is completely controlled within the 8272 FDC. Three cases are possible: the data may be written one clock cycle early, one clock cycle late, or with no preshift. The data preshift circuit is activated by the FDC only in the double-density mode. The preshift is required to cancel predictable playback data shifts when recorded data is later read from the floppy disk.

A single 50-conductor flat cable connects the board to the floppy disk drives. FDC outputs are driven by 7438 open collector high-current line-drivers. These drivers are resistively terminated on the last disk drive by means of a 150 ohm resistor to +5V. The line receivers are 7414 Schmitt triggered inverters with 150 ohm pull-up resistors on board.

7. SPECIAL CONSIDERATIONS

This section contains a quick review of key features and issues, most of which have been mentioned in other sections of this application note. Before designing with the 8272 FDC, it is advisable that the information in this section be completely understood.

1. Multi-Sector Transfers

The 8272 always operates in a multi-sector transfer mode. The 8272 continues to transfer data until the TC input is activated. In a DMA configuration, the TC input of the 8272 must always be connected to the EOP/TC output of the DMA controller. When multiple DMA channels are used on a single DMA controller, EOP must be gated with the select signal for the proper FDC. If the TC signal is not gated, a terminal count on another channel will abort FDC operation.

In a processor driven configuration with no DMA controller, the system must count the transfers and supply a TC signal to the FDC. In a DMA environment, ORing a programmable TC with the TC from the DMA controller is a convenient means of ensuring that the processor may always gain control of the FDC (even if the diskette system hangs up in an abnormal manner).

2. Processor Command/Result Phase Interface

In the command phase, the processor must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the processor must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. Command and result phases cannot be shortened.

During both the command and result phases, the Main Status Register must be read by the processor before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

NOTE: After the 8272 receives a command byte, the RQM flag may remain set for 12 microseconds (with an 8 MHz clock). Software should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the software will erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Write Clock

The FDC Write Clock input (WR CLK) must be present at all times.

5. Reset

The FDC Reset input (RST) must be held active during power-on reset while the RD and WR inputs are active. If the reset input becomes inactive while RD and WR are still active, the 8272 enters the test mode. Once activated, the test mode can only be deactivated by a power-down condition.

6. Drive Status

The 8272 constantly polls (starting after the power-on reset) all drives for changes in the drive ready status. At power-on, the FDC assumes that all drives are not ready. If a drive application requires that the ready line be strapped active, the FDC will generate an interrupt immediately after power is applied.

7. Gap Length

Only the gap 3 size is software programmable. All other gap sizes are fixed. In addition, different gap 3 sizes must be specified in format, read, write, and scan commands. Refer to Section 3 and Table 9 for gap size recommendations.

8. Seek Command

The drive busy flag in the Main Status Register remains set after a Seek command is issued until the Sense Interrupt Status command is issued (following reception of the seek complete interrupt).

The FDC does not perform implied seeks. Before issuing data read or write commands, the read/write head must be positioned over the correct cylinder. If the head is not positioned correctly, a cylinder address error is generated.

After issuing a step pulse, the 8272 resumes drive status polling. For correct stepper operation in this mode, the stepper motor must be constantly enabled. (Most drives provide a jumper to permit the stepper motor to be constantly enabled.)

9. Step Rate

The 8272 can emit a step pulse that is one millisecond faster than the rate programmed by the SRT parameter in the Specify command. This action may cause subsequent sector not found errors. The step rate time should be programmed to be 1 ms longer than the step rate time required by the drive.

10. Cable Length

A cable length of less than 10 feet is recommended for drive interfacing.

11. Scan Commands

The current 8272 has several problems when using the scan commands. These commands should not be used at this time.

12. Interrupts

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC busy flag (bit 4) in the Main Status Register. If the FDC busy flag is high, the interrupt is of type (a). If the FDC busy flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

13. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

14. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physical track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 67 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

15. Head Load versus Head Settle Times

The 8272 does not permit separate specification of the head load time and the head settle time. When the Specify command is issued for a given disk drive, the proper value for the HLT parameter is the maximum of the head load time and the head settle time.

APPENDIX

Power Distribution

Part	Ref Desig	+5	GND	+12	−12
8088	A2	40	1,20		
8224	I6	9,16	8		
8237-2	A6	31	20		
8251A	A9,B9,C9	26	4		
8253-5	A10	24	12		
8259A	B10	28	14		
8272	D10	40	20		
8284	A1	18	9		
8286	B6,F4	20	10		
2114	F1,F2,G1,G2,H1,H2,I1,I2	18	9		
2732A	D1,D2	24	12		
74LS00	E1	14	7		
74LS04	B2,E6,E8,F8	14	7		
74LS27	E2,E5	14	7		
74LS32	B1	14	7		
74LS74	A4,G5,H6	14	7		
74LS138	F3	16	8		
74LS139	E10	16	8		
74LS153	I3	16	8		
74LS157	F6	16	8		
74LS164	F5	14	7		
74LS173	G3	16	8		
74LS175	G4	16	8		
74LS240	G10	20	10		
74LS257	D3	16	8		
74LS367	C3,E9	16	8		
74LS373	B4,C4,D4,C6	20	10		
74LS393	I5,F7	14	7		
74S08	E4	14	7		
74S138	D6,E3	16	8		
7414	H7	14	7		
7438	H8,H9,H10	14	7		
1488	H3		7	14	1
1489	H4	14	7		
96LS02	G7	16	8		
96LS02	G6			16	8
LM358	H5			8	4

REFERENCES

1. Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.
2. Intel, *iSBC 208 Hardware Reference Manual,* Manual Order No. 143078, Intel Corporation, 1980.
3. Intel, *iSBC 204 Flexible Diskette Controller Hardware Reference Manual,* Manual Order No. 9800568A, Intel Corporation, 1978.
4. Shugart, *SA800/801 Diskette Storage Drive OEM Manual,* Part No. 50574, Shugart Associates, 1977.
5. Shugart, *SA800/801 Diskette Storage Drive Theory of Operations,* Part No. 50664, Shugart Associates, 1977.
6. Shugart, *SA800 Series Diskette Storage Drive Double Density Design Guide,* Part No. 39000, Shugart Associates, 1977.
7. Shugart, "Application Notes for Shugart Dual VFO," Part No. 39101, Shugart Associates, 1980.
8. Pertec, "Soft-sector Formatting for PERTEC Flexible Disk Drives," Pertec Application Note, 1977.
9. Austin Lesea and Rodnay Zaks, "Floppy-disc Controller Design Must Begin With the Basics," EDN, May 20, 1978.
10. John Hoeppner and Larry Wall, "Encoding/Decoding Techniques Double Floppy Disc Capacity," Computer Design, Feb 1980.
11. John Zarrella, *System Architecture,* Mirocomputer Applications, 1980.

Software Design and Implementation of Floppy Disk Subsystems

Contents

Contents (Continued)

1. Introduction

Disk interface software is a major contributor to the efficient and reliable operation of a floppy disk subsystem. This software must be a well-designed compromise between the needs of the application software modules and the capabilities of the floppy disk controller (FDC). In an effort to meet these requirements, the implementation of disk interface software is often divided into several levels of abstraction. The purpose of this application note is to define these software interface levels and describe the design and implementation of a modular and flexible software driver for the 8272 FDC. This note is a companion to AP-116, "An Intelligent Data Base System Using the 8272."

The Physical Interface Level

The software interface level closest to the FDC hardware is referred to as the physical interface level. At this level, interface modules (often called disk drivers or disk handlers) communicate directly with the FDC device. Disk drivers accept floppy disk commands from other software modules, control and monitor the FDC execution of the commands, and finally return operational status information (at command termination) to the requesting modules.

In order to perform these functions, the drivers must support the bit/byte level FDC interface for status and data transfers. In addition, the drivers must field, classify, and service a variety of FDC interrupts.

The Logical Interface Level

System and application software modules often specify disk operation parameters that are not directly compatible with the FDC device. This software incompatibility is typically caused by one of the following:

1. The change from an existing FDC to a functionally equivalent design. Replacing a TTL based controller with an LSI device is an example of a change that may result in software incompatibilities.

2. The upgrade of an existing FDC subsystem to a higher capability design. An expansion from a single-sided, single-density system to a dual-sided, double-density system to increase data storage capacity is an example of such a system change.

3. The abstraction of the disk software interface to avoid redundancy. Many FDC parameters (in particular the density, gap size, number of sectors per track and number of bytes per sector) are fixed for a floppy disk (after formatting). In fact, in many systems these parameters are never changed during the life of the system.

4. The requirement to support a software interface that is independent of the type of disk attached to the system. In this case, a system generated ("logical") disk address (drive, head, cylinder, and sector numbers) must be mapped into a physical floppy disk address. For example, to switch between single- and dual-sided disks, it may be easier and more cost-effective for the software to treat the dual-sided disk as containing twice as many sectors per track (52) rather than as having two sides. With this technique, accesses to sectors 1 through 26 are mapped onto head 0 while accesses to sectors 27 through 52 are mapped onto head 1.

5. The necessity of supporting a bad track map. Since bad tracks depend on the disk media, the bad track mapping varies from disk to disk. In general, the system and application software should not be concerned with calculating bad track parameters. Instead, these software modules should refer to cylinders logically (0 through 76). The logical interface level procedures must map these cylinders into physical cylinder positions in order to avoid the bad tracks.

The key to logical interface software design is the mapping of the "logical disk interface" (as seen by the application software) into the "physical disk interface" (as implemented by the floppy disk drivers). This logical to physical mapping is tightly coupled to system software design and the mapping serves to isolate both applications and system software from the peculiarities of the FDC device. Typical logical interface procedures are described in Table 1.

The File System Interface Level

The file system typically comprises the highest level of disk interface software used by application programs. The file system is designed to treat the disk as a collection of named data areas (known as files). These files are cataloged in the disk directory. File system interface software permits the creation of new files and the deletion of existing files under software control. When a file is created, its name and disk address are entered into the directory; when a file is deleted, its name is removed from the directory. Application software requests the use of a file by executing an OPEN function. Once opened, a file is normally reserved for use by the requesting program or task and the file cannot be reopened by other tasks. When a task no longer needs to use an open file, the task closes the file, releasing it for use by other tasks.

Most file systems also support a set of file attributes that can be specified for each file. File attributes may be used to protect files (e.g., the WRITE PROTECT attribute ensures that an existing file cannot accidentally be overwritten) and to supply system configuration information (e.g., a FORMAT attribute may specify that a file should automatically be created on a new disk when the disk is formatted).

At the file system interface level, application programs need not be explicitly aware of disk storage allocation techniques, block sizes, or file coding strategies. Only a "file name" must be presented in order to open, read or write, and subsequently close a file. Typical file system functions are listed in Table 2.

Table 1: Examples of Logical Interface Procedures

Name	Description
FORMAT DISK	Controls physical disk formatting for all tracks on a disk. Formatting adds FDC recognized cylinder, head, and sector addresses as well as address marks and data synchronization fields (gaps) to the floppy disk media.
RECALIBRATE	Moves the disk read/write head to track 0 (at the outside edge of the disk).
SEEK	Moves the disk read/write head to a specified logical cylinder. The logical and physical cylinder numbers may be different if bad track mapping is used.
READ STATUS	Indicates the status of the floppy disk drive and media. One important use of this procedure is to determine whether a floppy disk is dual-sided.
READ SECTOR	Reads one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).
WRITE SECTOR	Writes one or more complete sectors starting at a specified disk address (drive, head, cylinder, and sector).

Table 2: Disk File System Functions

Name	Description
OPEN	Prepare a file for processing. If the file is to be opened for input and the file name is not found in the directory, an error is generated. If the file is opened for output and the file name is not found in the directory, the file is automatically created.
CLOSE	Terminate processing of an open file.
READ	Transfer data from an open file to memory. The READ function is often designed to buffer one or more sectors of data from the disk drive and supply this data to the requesting program, as required.
WRITE	Transfer data from memory to an open file. The WRITE function is often designed to buffer data from the application program until enough data is available to fill a disk sector.
CREATE	Initialize a file and enter its name and attributes into the file directory.
DELETE	Remove a file from the directory and release its storage space.
RENAME	Change the name of a file in the directory.
ATTRIBUTE	Change the attributes of a file.
LOAD	Read a file of executable code into memory.
INITDISK	Initialize a disk by formatting the media and establishing the directory file, the bit map file, and other system files.

Scope of this Note

This application note directly addresses the logical and physical interface levels. A complete 8272 driver (including interrupt service software) is listed in Appendix A. In addition, examples of recalibrate, seek, format, read, and write logical interface level procedures are included as part of the exerciser program found in Appendix B. Wherever possible, specific hardware configuration dependencies are parametized to provide maximum flexibility without requiring major software changes.

2. Disk I/O Techniques

One of the most important software aspects of disk interfacing is the fixed sector size. (Sector sizes are fixed when the disk is formatted.) Individual bytes of disk storage cannot be read/written; instead, complete sectors must be transferred between the floppy disk and system memory.

Selection of the appropriate sector size involves a tradeoff between memory size, disk storage efficiency, and disk transfer efficiency. Basically, the following factors must be weighed:

1. Memory size. The larger the sector size, the larger the memory area that must be reserved for use during disk I/O transfers. For example, a 1K byte disk sector size requires that at least one 1K memory block be reserved for disk I/O.

2. Disk Storage efficiency. Both very large and very small sectors can waste disk storage space as follows. In disk file systems, space must be allocated somewhere on the disk to link the sectors of each file together. If most files are composed of many small sectors, a large amount of linkage overhead information is required. At the other extreme, when most files are smaller than a single disk sector, a large amount of space is wasted at the end of each sector.

3. Disk transfer efficiency. A file composed of a few large sectors can be transferred to/from memory more efficiently (faster and with less overhead) than a file composed of many small sectors.

Balancing these considerations requires knowledge of the intended system applications. Typically, for general purpose systems, sector sizes from 128 bytes to 1K bytes are used. For compatibility between single-density and double-density recording with the 8272 floppy disk controller, 256 byte sectors or 512 byte sectors are most useful.

FDC Data Transfer Interface

Three distinct software interface techniques may be used to interface system memory to the FDC device during sector data transfers:

1. DMA - In a DMA implementation, the software is only required to set up the DMA controller memory address and transfer count, and to initiate the data transfer. The DMA controller hardware handshakes with the processor/system bus in order to perform each data transfer.

2. Interrupt Driven - The FDC generates an interrupt when a data byte is ready to be transferred to memory, or when a data byte is needed from memory. It is the software's responsibility to perform appropriate memory reads/writes in order to transfer data from/to the FDC upon receipt of the interrupt.

3. Polling - Software responsibilities in the polling mode are identical to the responsibilities in the interrupt driven mode. The polling mode, however, is used when interrupt service overhead (context switching) is too large to support the disk data

rate. In this mode, the software determines when to transfer data by continually polling a data request status flag in the FDC status register.

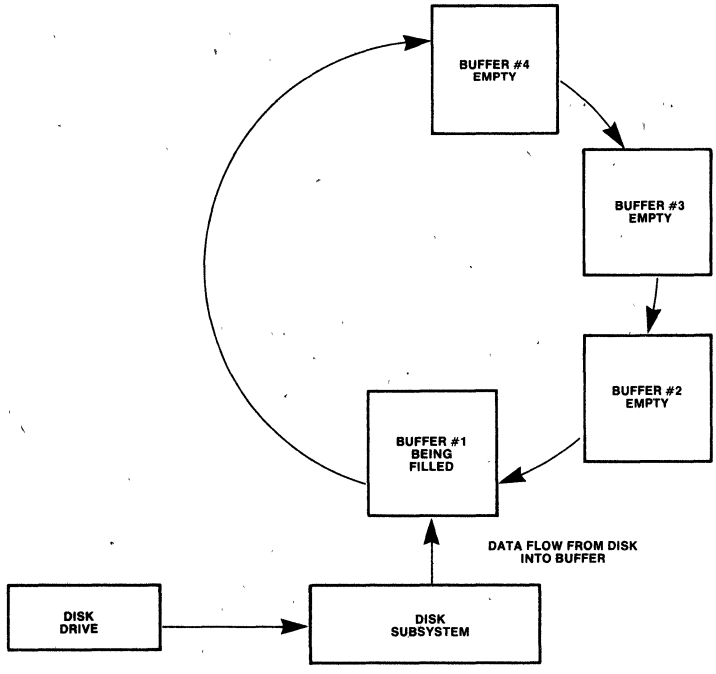
The DMA mode has the advantage of permitting the processor to continue executing instructions while a disk transfer is in progress. (This capability is especially useful in multiprogramming environments when the operating system is designed to permit other tasks to execute while a program is waiting for I/O.) Modes 2 and 3 are often combined and described as non-DMA operating modes. Non-DMA modes have the advantage of significantly lower system cost, but are often performance limited for double-density systems (where data bytes must be transferred to/from the FDC every 16 microseconds).

Overlapped Operations

Some FDC devices support simultaneous disk operations on more than one disk drive. Normally seek and recalibrate operations can be overlapped in this manner. Since seek operations on most floppy drives are extremely slow, this mode of operation can often be used by the system software to reduce overall disk access times.

Buffers

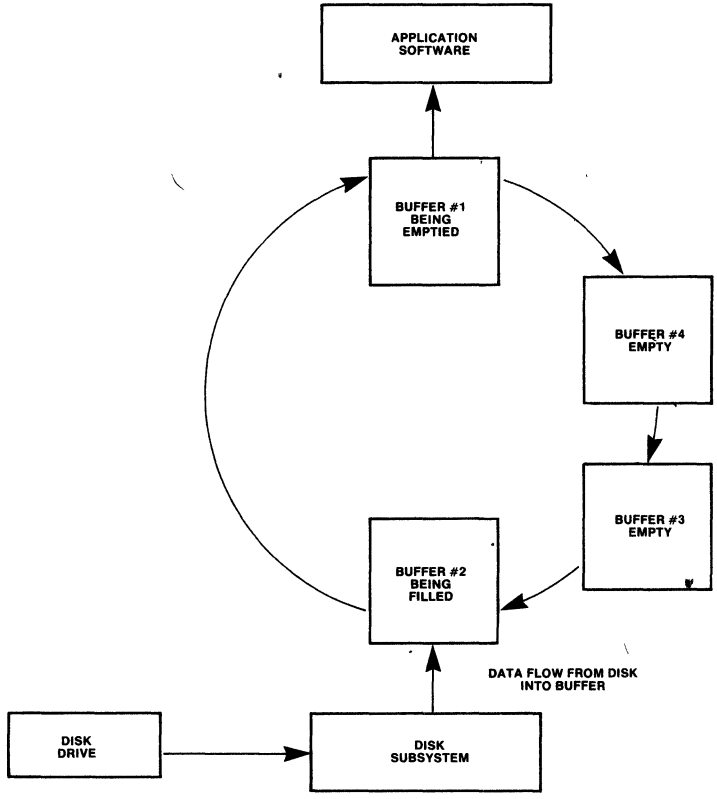
The buffer concept is an extremely important element in advanced disk I/O strategies. A buffer is nothing more than a memory area containing the same amount of data as a disk sector contains. Generally, when an application program requests data from a disk, the system software allocates a buffer (memory area) and transfers the data from the appropriate disk sector into the buffer. The address of the buffer is then returned to the application software. In the same manner, after the application program has filled a buffer for output, the buffer address is passed to the system software, which writes data from the buffer into a disk sector. In multitasking systems, multiple buffers may be allocated from a buffer pool. In these systems, the disk controller is often requested to read ahead and fill additional data buffers while the application software is processing a previous buffer. Using this technique, system software attempts to fill buffers before they are needed by the application programs, thereby eliminating program waits during I/O transfers. Figure 1 illustrates the use of multiple buffers in a ring configuration.



a) The first disk read request by the application software causes the disk subsystem to begin filling the first empty buffer. The application software must wait until the buffer is filled before it may continue execution.

AFN-01949A

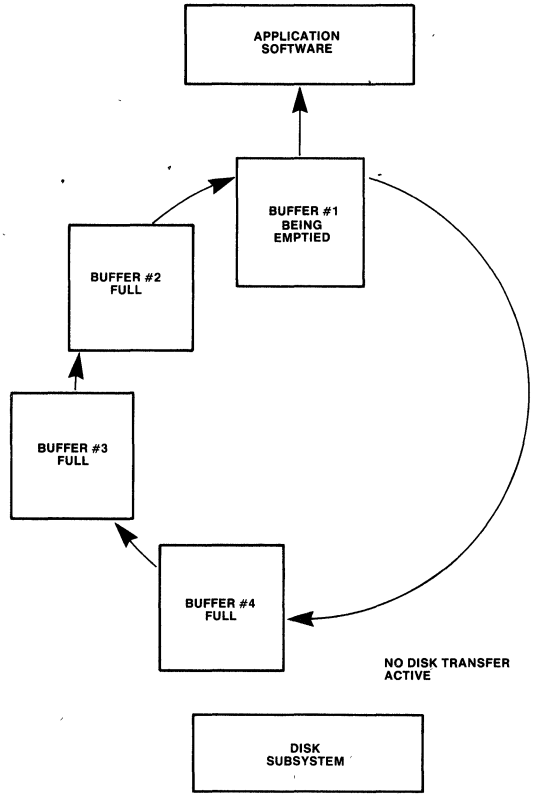
Figure 1. Using Multiple Memory Buffers for Disk I/O



b) After the first buffer is filled, the disk system continues to transfer disk data into the next buffer while the application software begins operating on the first full buffer.

AFN-01949A

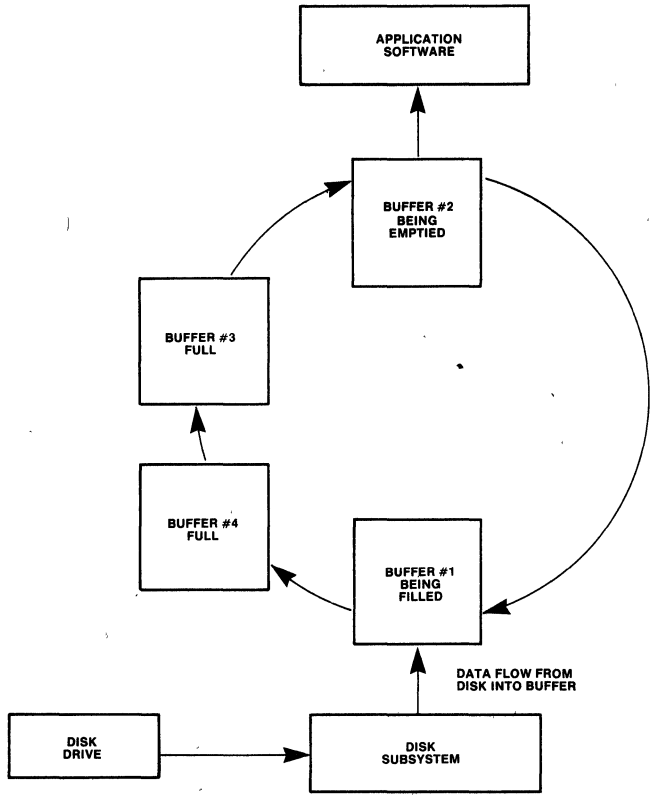
Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



c) When all empty buffers have been filled, disk activity is stopped until the application software releases one or more buffers for reuse.

AFN-01949A

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)



d) When the application software releases a buffer (for reuse), the disk subsystem begins a disk sector read to refill the buffer. This strategy attempts to anticipate application software needs by maintaining a sufficient number of full data buffers in order to minimize data transfer delays. If disk data is already in memory when the application software requests it, no disk transfer delays are incurred.

AFN-01949A

Figure 1. Using Multiple Memory Buffers for Disk I/O (Continued)

3. THE 8272 FLOPPY DISK CONTROLLER

The 8272 is a single-chip LSI Floppy Disk Controller (FDC) that implements both single- and double-density floppy disk storage subsystems (with up to four dual-sided disk drives per FDC). The 8272 supports the IBM 3740 single-density recording format (FM) and the IBM System 34 double-density recording format (MFM). The 8272 accepts and executes high-level disk commands such as format track, seek, read sector, and write sector. All data synchronization and error checking is automatically performed by the FDC to ensure reliable data storage and subsequent retrieval. The 8272 interfaces to microprocessor systems with or without Direct Memory Access (DMA) capabilities and also interfaces to a large number of commercially available floppy disk drives.

Floppy Disk Commands

The 8272 executes fifteen high-level disk interface commands:

Specify	Write Data
Sense Drive Status	Write Deleted Data
Sense Interrupt Status	Read Track
Seek	Read ID
Recalibrate	Scan Equal
Format Track	Scan High or Equal
Read Data	Scan Low or Equal
Read Deleted Data	

Each command is initiated by a multi-byte transfer from the driver software to the FDC (the transferred bytes contain command and parameter information). After complete command specification, the FDC automatically executes the command. The command result data (after execution of the command) may require a multi-byte transfer of status information back to the driver. It is convenient to consider each FDC command as consisting of the following three phases:

Command Phase: The driver transfers to the FDC all the information required to perform a particular disk operation. The 8272 automatically enters the command phase after RESET and following the completion of the result phase (if any) of a previous command.

Execution Phase: The FDC performs the operation as instructed. The execution phase is entered immediately after the last command parameter is written to the FDC in the preceding command phase. The execution phase normally ends when the last data byte is transferred to/from the disk or when an error occurs.

Result Phase: After completion of the disk operation, status and other housekeeping information are made available to the driver software. After this information is read, the FDC reenters the command phase and is ready to accept another command.

Interface Registers

To support information transfer between the FDC and the system software, the 8272 contains two 8-bit registers: the Main Status Register and the Data Register. The Main Status Register (read only) contains FDC status information and may be accessed at any time. The Main Status Register (Table 3) provides the system processor with the status of each disk drive, the status of the FDC, and the status of the processor interface. The Data Register (read/write) stores data, commands, parameters, and disk drive status information. The Data Register is used to program the FDC during the command phase and to obtain result information after completion of FDC operations.

In addition to the Main Status Register, the FDC contains four additional status registers (ST0, ST1, ST2, and ST3). These registers are only available during the result phase of a command.

Command/Result Phases

Table 4 lists the 8272 command set. For each of the fifteen commands, command and result phase data transfers are listed. A list of abbreviations used in the table is given in Table 5, and the contents of the result status registers (ST0-ST3) are illustrated in Table 6.

The bytes of data which are sent to the 8272 by the drivers during the command phase, and are read out of the 8272 in the result phase, must occur in the order shown in Table 4. That is, the command code must be sent first and the other bytes sent in the prescribed sequence. All bytes of the command and result phases must be read/written as described. After the last byte of data in the command phase is sent to the 8272 the execution phase automatically starts. In a similar fashion, when the last byte of data is read from the 8272 in the result phase, the result phase is automatically ended and the 8272 reenters the command phase.

It is important to note that during the result phase all bytes shown in Table 4 must be read. The Read Data command, for example, has seven bytes of data in the result phase. All seven bytes must be read in order to successfully complete the Read Data command. The 8272 will not accept a new command until all seven bytes have been read. The number of command and result bytes varies from command-to-command.

In order to read data from, or write data to, the Data Register during the command and result phases, the software driver must examine the Main Status Register to determine if the Data Register is available. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command word may be written into the 8272. Many of the commands require multiple bytes, and as a result, the Main Status Register must be read prior to each byte transfer to the 8272. To read status bytes during the result phase, DIO and RQM in the Main Status Register must both be high. Note, checking the Main Status Register in this manner before each byte transfer to/from the 8272 is required only in the command and result phases, and is NOT required during the execution phase.

Table 3: Main Status Register Bit Definitions

BIT NUMBER	SYMBOL	DESCRIPTION
0	D_0B	Disk Drive 0 Busy. Disk Drive 0 is seeking.
1	D_1B	Disk Drive 1 Busy. Disk Drive 1 is seeking.
2	D_2B	Disk Drive 2 Busy. Disk Drive 2 is seeking.
3	D_3B	Disk Drive 3 Busy. Disk Drive 3 is seeking.
4	CB	FDC Busy. A read or write command is in progress.
5	NDM	Non-DMA Mode. The FDC is in the non-DMA mode when this flag is set (1). This flag is set only during the execution phase of commands in the non-DMA mode. Transition of this flag to a zero (0) indicates that the execution phase has ended.
6	DIO	Data Input/Output. Indicates the direction of a data transfer between the FDC and the Data Register. When DIO is set (1), data is read from the Data Register by the processor; when DIO is reset (0), data is written from the processor to the Data Register.
7	RQM	Request for Master. When set (1), this flag indicates that the Data Register is ready to send data to, or receive data from, the processor.

Table 4: 8272 Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after command execution
	R				H					
	R				R					
	R				N					
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EC 1					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the main-system and the FDD
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and the main-system. FDC reads the complete track contents from the physical index mark to EOT
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the track is stored in Data Register
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information during Execution Phase
	R				H					
	R				R					
	R				N					
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				N					Bytes/Sector
	W				SC					Sectors/Track
	W				GPL					Gap 3
	W				D					Filter Byte
Execution										FDC formats an entire track
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					In this case, the ID information has no meaning
	R				H					
	R				R					
	R				N					
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				STP					
Execution										Data compared between the FDD and the main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

Note: 1. $A_0 = 1$ for all operations.

APPLICATIONS

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	———			C				———	Sector ID information prior Command execution
	W	———			H				———	
	W	———			R				———	
	W	———			N				———	
	W	———			EOT				———	
	W	———			GPL				———	
	W	———			STP				———	
Execution										Data compared between the FDD and the main-system
Result	R	———			ST 0				———	Status information after Command execution
	R	———			ST 1				———	
	R	———			ST 2				———	
	R	———			C				———	Sector ID information after Command execution
	R	———			H				———	
	R	———			R				———	
	R	———			N				———	
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	———			C				———	Sector ID information prior Command execution
	W	———			H				———	
	W	———			R				———	
	W	———			N				———	
	W	———			EOT				———	
	W	———			GPL				———	
	W	———			STP				———	
Execution										Data compared between the FDD and the main-system
Result	R	———			ST 0				———	Status information after Command execution
	R	———			ST 1				———	
	R	———			ST 2				———	
	R	———			C				———	Sector ID information after Command execution
	R	———			H				———	
	R	———			R				———	
	R	———			N				———	

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	———			ST 0				———	Status information at the end of each seek operation about the FDC
	R	———			C				———	
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	———	SPT					HUT	———	Timer Settings
	W	———	HLT						ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	———			ST 3				———	Status information about the FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	———			C				———	
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W	———			Invalid Codes				———	Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	———			ST 0				———	ST 0 = 80 (16)

Table 5: Command/Result Parameter Abbreviations

SYMBOL	DESCRIPTION
C	Cylinder Address. The currently selected cylinder address (0 to 76) on the disk.
D	Data Pattern. The pattern to be written in each sector data field during formatting.
DS0,DS1	Disk Drive Select. DS1 DS0 0 0 Drive 0 0 1 Drive 1 1 0 Drive 2 1 1 Drive 3
DTL	Special Sector Size. During the execution of disk read/write commands, this parameter is used to temporarily alter the effective disk sector size. By setting N to zero, DTL may be used to specify a sector size from 1 to 256 bytes in length. If the actual sector (on the disk) is larger than DTL specifies, the remainder of the actual sector is not passed to the system during read commands; during write commands, the remainder of the actual sector is written with all-zeroes bytes. DTL should be set to FF hexadecimal when N is not zero.
EOT	End of Track. The final sector number of the current track.
GPL	Gap Length. The gap 3 size. (Gap 3 is the space between sectors.)
H	Head Address. Selected head: 0 or 1 (disk side 0 or 1, respectively) as encoded in the sector ID field.
HLT	Head Load Time. Defines the time interval that the FDC waits after loading the head before initiating a read or write operation. Programmable from 2 to 254 milliseconds (in increments of 2 ms).
HUT	Head Unload Time. Defines the time interval from the end of the execution phase (of a read or write command) until the head is unloaded. Programmable from 16 to 240 milliseconds (in increments of 16 ms).
MFM	MFM/FM Mode Selector. Selects MFM double-density recording mode when high, FM single-density mode when low.
MT	Multi-Track Selector. When set, this flag selects the multi-track operating mode. In this mode (used only with dual-sided disks), the FDC treats a complete cylinder (under both read/write head 0 and read/write head 1) as a single track. The FDC operates as if this expanded track started at the first sector under head 0 and ended at the last sector under head 1. With this flag set (high), a multi-sector read operation will automatically continue to the first sector under head 1 when the FDC finishes operating on the last sector under head 0.
N	Sector Size Code. The number of data bytes within a sector.

ND	Non-DMA Mode Flag. When set (1), this flag indicates that the FDC is to operate in the non-DMA mode. In this mode, the processor participates in each data transfer (by means of an interrupt or by polling the RQM flag in the Main Status Register). When reset (0), the FDC interfaces to a DMA controller.
R	Sector Address. Specifies the sector number to be read or written. In multi-sector transfers, this parameter specifies the sector number of the first sector to be read or written.
SC	Number of Sectors per Track. Specifies the number of sectors per track to be initialized by the Format Track command.
SK	Skip Flag. When this flag is set, sectors containing deleted data address marks will automatically be skipped during the execution of multi-sector Read Data or Scan commands. In the same manner, a sector containing a data address mark will automatically be skipped during the execution of a multi-sector Read Deleted Data command.
SRT	Step Rate Interval. Defines the time interval between step pulses issued by the FDC (track-to-track access time). Programmable from 1 to 16 milliseconds (in increments of 1 ms).
ST0 ST1 ST2 ST3	Status Register 0-3. Registers within the FDC that store status information after a command has been executed. This status information is available to the processor during the Result Phase after command execution. These registers may only be read after a command has been executed (in the exact order shown in Table 4 for each command). These registers should not be confused with the Main Status Register.
STP	Scan Sector Increment. During Scan operations, this parameter is added to the current sector number in order to determine the next sector to be scanned.

Table 6: Status Register Definitions

Status Register 0		
BIT NUMBER	**SYMBOL**	**DESCRIPTION**
7,6	IC	Interrupt Code. 00 - Normal termination of command. The specified command was properly executed and completed without error. 01 - Abnormal termination of command. Command execution was started but could not be successfully completed. 10 - Invalid command. The requested command could not be executed. 11 - Abnormal termination. During command execution, the disk drive ready signal changed state.
5	SE	Seek End. This flag is set (1) when the FDC has completed the Seek command and the read/write head is positioned over the correct cylinder.
4	EC	Equipment Check Error. This flag is set (1) if a fault signal is received from the disk drive or if the track 0 signal is not received from the disk drive after 77 step pulses (Recalibrate command).
3	NR	Not Ready Error. This flag is set if a read or write command is issued and either the drive is not ready or the command specifies side 1 (head 1) of a single-sided disk.
2	H	Head Address. The head address at the time of the interrupt.
1,0	DS1,DS0	Drive Select. The number of the drive selected at the time of the interrupt.
Status Register 1		
BIT NUMBER	**SYMBOL**	**DESCRIPTION**
7	EN	End of Track Error. This flag is set if the FDC attempts to access a sector beyond the final sector of the track.
6		Undefined
5	DE	Data Error. Set when the FDC detects a CRC error in either the the ID field or the data field of a sector.
4	OR	Overrun Error. Set (during data transfers) if the FDC does not receive DMA or processor service within the specified time interval.

3		Undefined
2	ND	Sector Not Found Error. This flag is set by any of the following conditions. a) The FDC cannot locate the sector specified in the Read Data, Read Deleted Data, or Scan command. b) The FDC cannot locate the starting sector specified in the Read Track command. c) The FDC cannot read the ID field without error during a Read ID command.
1	NW	Write Protect Error. This flag is set if the FDC detects a write protect signal from the disk drive during the execution of a Write Data, Write Deleted Data, or Format Track command.
0	MA	Missing Address Mark Error. This flag is set by either of the following conditions: a) The FDC cannot detect the ID address mark on the specified track (after two rotations of the disk). b) The FDC cannot detect the data address mark or deleted data address mark on the specified track. (See also the MD bit of Status Register 2.)

Status Register 2

BIT NUMBER	SYMBOL	DESCRIPTION
7		Undefined
6	CM	Control Mark. This flag is set when the FDC encounters one of the following conditions: a) A deleted data address mark during the execution of a Read Data or Scan command. b) A data address mark during the execution of a Read Deleted Data command.
5	DD	Data Error. Set (1) when the FDC detects a CRC error in a sector data field. This flag is not set when a CRC error is detected in the ID field.
4	WC	Cylinder Address Error. Set when the cylinder address from the disk sector ID field is different from the current cylinder address maintained within the FDC.
3	SH	Scan Hit. Set during the execution of the Scan command if the scan condition is satisfied.
2	SN	Scan Not Satisfied. Set during execution of the Scan command if the FDC cannot locate a sector on the specified cylinder that satisfies the scan condition.

1	BC	Bad Track Error. Set when the cylinder address from the disk sector ID field is FF hexadecimal and this cylinder address is different from the current cylinder address maintained within the FDC. This all "ones" cylinder number indicates a bad track (one containing hard errors) according to the IBM soft-sectored format specifications.
0	MD	Missing Data Address Mark Error. Set if the FDC cannot detect a data address mark or deleted data address mark on the specified track.

Status Register 3

BIT NUMBER	SYMBOL	DESCRIPTION
7	FT	Fault. This flag indicates the status of the fault signal from the selected disk drive.
6	WP	Write Protected. This flag indicates the status of the write protect signal from the selected disk drive.
5	RDY	Ready. This flag indicates the status of the ready signal from the selected disk drive.
4	T0	Track 0. This flag indicates the status of the track 0 signal from the selected disk drive.
3	TS	Two-Sided. This flag indicates the status of the two-sided signal from the selected disk drive.
2	H	Head Address. This flag indicates the status of the side select signal for the currently selected disk drive.
1,0	DS1,DS0	Drive Select. Indicates the currently selected disk drive number.

Execution Phase

All data transfers to (or from) the floppy drive occur during the execution phase. The 8272 has two primary modes of operation for data transfers (selected by the specify command):

1) DMA mode
2) non-DMA mode

In the DMA mode, execution phase data transfers are handled by the DMA controller hardware (invisible to the driver software). The driver software, however, must set all appropriate DMA controller registers prior to the beginning of the disk operation. An interrupt is generated by the 8272 after the last data transfer, indicating the completion of the execution phase, and the beginning of the result phase.

In the non-DMA mode, transfer requests are indicated by generation of an interrupt and by activation of the RQM flag (bit 7 in the Main Status Register). The interrupt signal can be used for interrupt-driven systems and RQM can be used for polled systems. The driver software must respond to the transfer request by reading data from, or writing data to, the FDC. After completing the last transfer, the 8272 generates an interrupt to indicate the beginning of the result phase. In the non-DMA mode, the processor must activate the "terminal count" (TC) signal to the FDC (normally by means of an I/O port) after the transfer request for the last data byte has been received (by the driver) and before the appropriate data byte has been read from (or written to) the FDC.

In either mode of operation (DMA or non-DMA), the execution phase ends when a "terminal count" signal is sensed by the FDC, when the last sector on a track (the EOT parameter - Table 4) has been read or written, or when an error occurs.

Multi-sector and Multi-track Transfers

During disk read/write transfers (Read Data, Write Data, Read Deleted Data, and Write Deleted Data), the FDC will continue to transfer data from sequential sectors until the TC input is sensed. In the DMA mode, the TC input is normally set by the DMA controller. In the non-DMA mode, the processor directly controls the FDC TC input as previously described. Once the TC input is received, the FDC stops requesting data transfers (from the system software or DMA controller). The FDC, however, continues to read data from, or write data to, the floppy disk until the end of the current disk sector. During a disk read operation, the data read from the disk (after reception of the TC input) is discarded, but the data CRC is checked for errors; during a disk write operation, the remainder of the sector is filled with all-zero bytes.

If the TC signal is not received before the last byte of the current sector has been transferred to/from the system, the FDC increments the sector number by one and initiates a read or write command for this new disk sector.

The FDC is also designed to operate in a multi-track mode for dual-sided disks. In the multi-track mode (specified by means of the MT flag in the command byte - Table 4) the FDC will automatically increment the head address (from 0 to 1) when the last sector (on the track under head 0) has been read or written. Reading or writing is then continued on the first sector (sector 1) of head 1.

Drive Status Polling

After the power-on reset, the 8272 automatically enters a drive status polling mode. If a change in drive status is detected (all drives are assumed to be "not ready" at power-on), an interrupt is generated. The 8272 continues this status polling between command executions (and between step pulses in the Seek command). In this manner, the 8272 automatically notifies the system software whenever a floppy disk is inserted, removed, or changed by the operator.

Command Details

During the command phase, the Main Status Register must be polled by the driver software before each byte is written into the Data Register. The DIO (bit 6) and RQM (bit 7) flags in the Main Status Register must be low and high, respectively, before each byte of the command may be written into the 8272. The beginning of the execution phase for any of these commands will cause DIO to be set high and RQM to be set low.

Operation of the FDC commands is described in detail in Application Note AP-116, "An Intelligent Data Base System Using the 8272."

Invalid Commands

If an invalid (undefined) command is sent to the FDC, the FDC will terminate the command. No interrupt is generated by the 8272 during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both set indicating to the processor that the 8272 is in the result phase and the contents of Status Register 0 must be read. When the processor reads Status Register 0 it will find an 80H code indicating that an invalid command was received. The driver software in Appendix B checks each requested command and will not issue an invalid command to the 8272.

A Sense Interrupt Status command must be sent after a Seek or Recalibrate interrupt; otherwise the FDC will consider the next command to be an invalid command. Also, when the last "hidden" interrupt has been serviced, further Sense Interrupt Status commands will result in invalid command codes.

4. 8272 Physical Interface Software

PL/M software driver listings for the 8272 FDC are contained in Appendix A. These drivers have been designed to operate in a DMA environment (as described in Application Note AP-116, "An Intelligent Data Base System Using the 8272"). In the following paragraphs, each driver procedure is described. (A description of the driver data base variables is given in Table 7.) In addition, the modifications necessary to reconfigure the drivers for operation in a polled environment are discussed.

INITIALIZE$DRIVERS

This initialization procedure must be called before any FDC operations are attempted. This module initializes the DRIVE$READY, DRIVE$STATUS$CHANGE, OPERATION$IN$PROGRESS, and OPERATION$COMPLETE arrays as well as the GLOBAL$DRIVE$NO variable.

EXECUTE$DOCB

This procedure contains the main 8272 driver control software and handles the execution of a complete FDC command. EXECUTE$DOCB is called with two parameters: a) a pointer to a disk operation control block and b) a pointer to a result status byte. The format of the disk operation control block is illustrated in Figure 2 and the result status codes are described in Table 8.

Before starting the command phase for the specified disk operation, the command is checked for validity and to determine whether the FDC is busy. (For an overlapped operation, if the FDC BUSY flag is set — in the Main Status Register — the command cannot be started; non-overlapped operations cannot be started if the FDC BUSY flag is set, if any drive is in the process of seeking/recalibrating, or if an operation is currently in progress on the specified drive.)

After these checks are made, interrupts are disabled in order to set the OPERATIONINPROGRESS flag, reset the OPERATION$COMPLETE flag, load a pointer to the current operation control block into the OPERATION$DOCB$PTR array and set GLOBAL$DRIVE$NO (if a non-overlapped operation is to be started).

At this point, parameters from the operation control block are output to the DMA controller and the FDC command phase is initiated. After completion of the command phase, a test is made to determine the type of result phase required for the current operation. If no result phase is needed, control is immediately returned to the calling program. If an immediate result phase is required, the result bytes are input from the FDC. Otherwise, the CPU waits until the OPERATION$COMPLETE flag is set (by the interrupt service procedure).

Finally, if an error is detected in the result status code (from the FDC), an FDC operation error is reported to the calling program.

Table 7: Driver Data Base

NAME	DESCRIPTION
DRIVE$READY	A public array containing the current "ready" status of each drive.
DRIVE$STATUS$CHANGE	A public array containing a flag for each drive. The appropriate flag is set whenever the ready status of a drive changes.
OPERATION$DOCB$PTR	An internal array of pointers to the operation control block currently in progress for each drive.
OPERATIONINPROGRESS	An internal array used by the driver procedures to determine if a disk operation is in progress on a given drive.
OPERATION$COMPLETE	An internal array used by the driver procedures to determine when the execution phase of a disk operation is complete.
GLOBAL$DRIVE$NO	A data byte that records the current drive number for non-overlapped disk operations.
VALID$COMMAND	A constant flag array that indicates whether a specified FDC command code is valid.
COMMAND$LENGTH	A constant byte array specifying the number of command/parameter bytes to be transferred to the FDC during the command phase.
DRIVENOPRESENT	A constant flag array that indicates whether a drive number is encoded into an FDC command.
OVERLAP$OPERATION	A constant flag array that indicates whether an FDC command can be overlapped with other commands.
NO$RESULT	A constant flag array that is used to determine when an FDC operation does not have a result phase.
IMMED$RESULT	A constant flag array that indicates that an FDC operation has a result phase beginning immediately after the command phase is complete.
POSSIBLE$ERROR	A constant flag array that indicates if an FDC operation should be checked for an error status indication during the result phase.

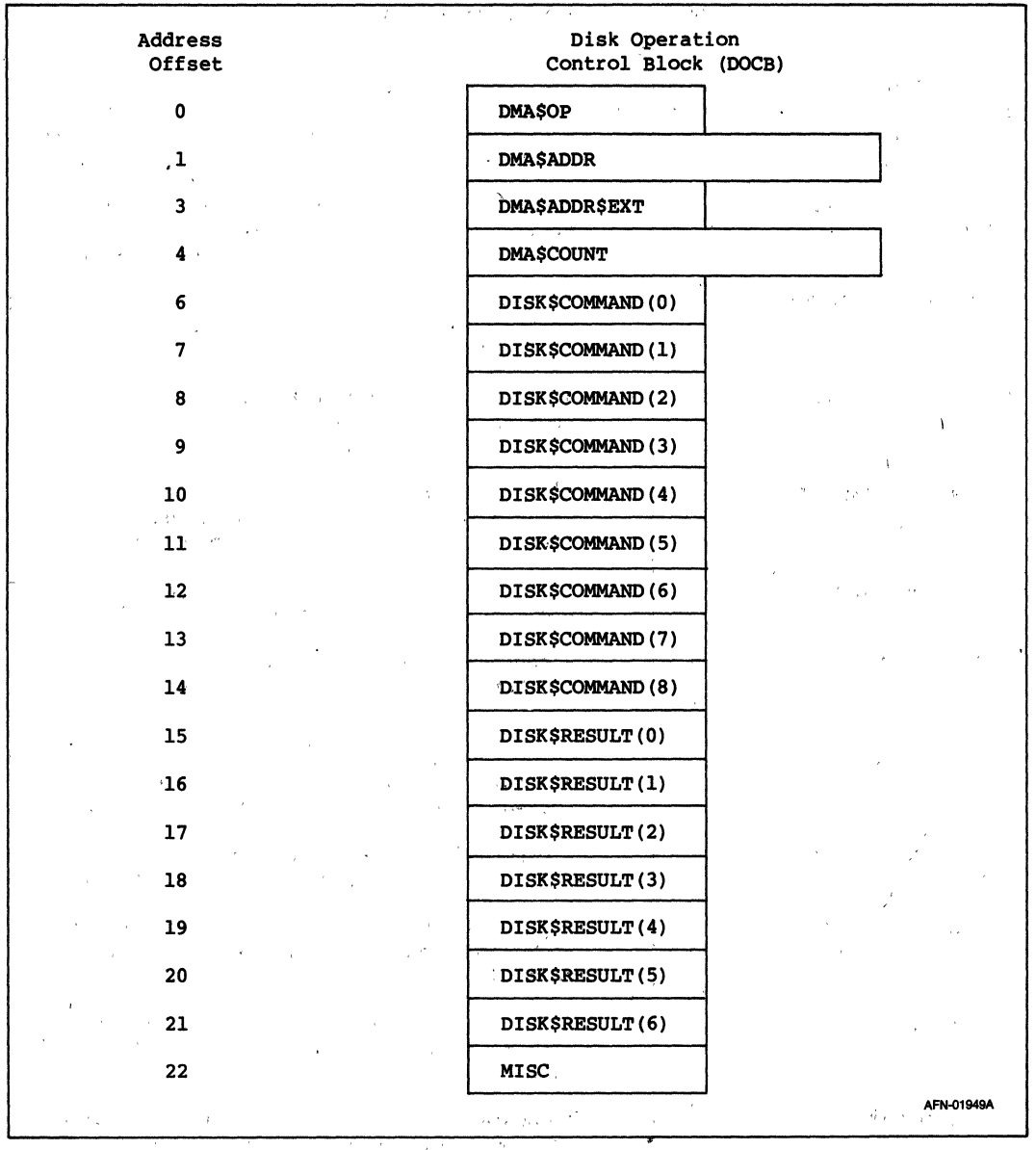


Figure 2. Disk Operation Control Block (DOCB) Format

Table 8: EXECUTE$DOCB Return Status Codes

Code	Description
0	No errors. The specified operation was completed without error.
1	FDC busy. The requested operation cannot be started. This error occurs if an attempt is made to start an operation before the previous operation is completed.
2	FDC error. An error was detected by the FDC during the execution phase of a disk operation. Additional error information is contained in the result data portion of the disk operation control block (DOCB.DISK$RESULT) as described in the 8272 data sheet. This error occurs whenever the 8272 reports an execution phase error (e.g., missing address mark).
3	8272 command interface error. An 8272 interfacing error was detected during the command phase. This error occurs when the command phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
4	8272 result interface error. An 8272 interfacing error was detected during the result phase. This error occurs when the result phase of a disk operation cannot be successfully completed (e.g., incorrect setting of the DIO flag in the Main Status Register).
5	Invalid FDC Command.

FDCINT

This procedure performs all interrupt processing for the 8272 interface drivers. Basically, two types of interrupts are generated by the 8272: (a) an interrupt that signals the end of a command execution phase and the beginning of the result phase and (b) an interrupt that signals the completion of an overlapped operation or the occurrence of an unexpected event (e.g., change in the drive "ready" status).

An interrupt of type (a) is indicated when the FDC BUSY flag is set (in the Main Status Register). When a type (a) interrupt is sensed, the result bytes are read from the 8272 and placed in the result portion of the disk operation control block, the appropriate OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

When an interrupt of type (b) is indicated (FDC not busy), a sense interrupt status command is issued (to the FDC). The upper two bits of the result status register (Status Register Zero - ST0) are used to determine the cause of the interrupt. The following four cases are possible:

1) Operation Complete. An overlapped operation is complete. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active operation control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

2) Abnormal Termination. A disk operation has abnormally terminated. The drive number is found in the lower two bits of ST0. The ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

3) Invalid Command. The execution of an invalid command (i.e., a sense interrupt command with no interrupt pending) has been attempted. This interrupt signals the successful completion of all interrupt processing.

4) Drive Status Change. A change has occurred in the "ready" status of a disk drive. The drive number is found in the lower two bits of ST0. The DRIVE$READY flag for this disk drive is set to the new drive "ready" status and the DRIVE$STATUS$CHANGE flag for the drive is also set. In addition, if a command is currently in progress, the ST0 data is transferred to the active control block, the OPERATION$COMPLETE flag is set, and the OPERATIONINPROGRESS flag is reset.

After processing a type (b) interrupt, additional sense interrupt status commands must be issued and processed until an "invalid command" result is returned from the FDC. This action guarantees that all "hidden" interrupts are serviced.

In addition to the major driver procedures described above, a number of support procedures are required. These support routines are briefly described in the following paragraphs.

OUTPUT$CONTROLS$TO$DMA

This procedure outputs the DMA mode, the DMA address, and the DMA word count to the 8237 DMA controller. In addition, the upper four bits of the 20-bit DMA address are output to the address extension latch. Finally, the disk DMA channel is started.

OUTPUT$COMMAND$TO$FDC

This software module outputs a complete disk command to the 8272 FDC. The number of required command/parameter bytes is found in the COMMAND$LENGTH table. The appropriate bytes are output one at a time (by calls to OUTPUT$BYTE$TO$FDC) from the command portion of the disk operation control block.

INPUT$RESULT$FROM$FDC

This procedure is used to read result phase status information from the disk controller. At most, seven bytes are read. In order to read each byte, a call is made to INPUT$BYTE$FROM$FDC. When the last byte has been read, a check is made to insure that the FDC is no longer busy.

OUTPUT$BYTE$TO$FDC

This software is used to output a single command/parameter byte to the FDC. This procedure waits until the FDC is ready for a command byte and then outputs the byte to the FDC data port.

INPUT$BYTE$FROM$FDC

This procedure inputs a single result byte from the FDC. The software waits until the FDC is ready to transfer a result byte and then reads the byte from the FDC data port.

FDC$READY$FOR$COMMAND

This procedure assures that the FDC is ready to accept a command/parameter byte by performing the following three steps. First, a small time interval (more than 20 microseconds) is inserted to assure that the RQM flag has time to become valid (after the last byte transfer). Second, the master request flag (RQM) is polled until it is activated by the FDC. Finally, the DIO flag is checked to ensure that it is properly set for FDC input (from the processor).

FDC$READY$FOR$RESULT

The operation of this procedure is similar to the FDC$READY$FOR$COMMAND with the following exception. If the FDC BUSY flag (in the Main Status Register) is not set, the result phase is complete and no more data is available from the FDC. Otherwise, the procedure waits for the RQM flag and checks the DIO flag for FDC output (to the processor).

OPERATION$CLEAN$UP

This procedure is called after the execution of a disk operation that has no result phase. OPERATION$CLEAN$UP resets the OPERATIONINPROGRESS flag and the GLOBAL$DRIVE$NO variable if appropriate. This procedure is also called to clean up after some disk operation errors.

Modifications for Polling Operation

To operate in the polling mode, the following modifications should be made to the previous routines:

1. The OUTPUT$CONTROLS$TO$DMA routine should be deleted.

2. In EXECUTE$DOCB, immediately prior to WAITFOROP$COMPLETE, a polling loop should be inserted into the code. The loop should test the RQM flag (in the Main Status Register). When RQM is set, a data byte should be written to, or read from, the 8272. The buffer address may be computed from the base address contained in DOCB.DMA$ADDR and DOCB.DMA$ADDR$EXT. After the correct number of bytes have been transferred, an operation complete interrupt will be issued by the FDC. During data transfer in the non-DMA mode, the NON-DMA MODE flag (bit 5 of the Main Status Register) will be set. This flag will remain set for the complete execution phase. When the transfer is finished, the NON-DMA MODE flag is reset and the result phase interrupt is issued by the FDC.

5. 8272 Logical Interface Software

Appendix B of this Application Note contains a PL/M listing of an exerciser program for the 8272 drivers. This program illustrates the design of logical interface level procedures to specify disk parameters, recalibrate a drive, seek to a cylinder, format a disk, read data, and write data.

The exerciser program is written to operate a standard single-sided 8" floppy disk drive in either the single- or double-density recording mode. Only the eight parameters listed in Table 9 must be specified. All other parameters are derived from these 8 basic variables.

Each of these logical interface procedures is described in the following paragraphs (refer to the listing in Appendix B).

SPECIFY

This procedure sets the FDC signal timing so that the FDC will interface correctly to the attached disk drive. The SPECIFY procedure requires four parameters, the step rate (SRT), head load time (HLT), head unload time (HUT), and the non-DMA mode flag (ND). This procedure builds a disk operation control block (SPECIFY$DOCB) and passes the control block to the FDC driver module (EXECUTE$DOCB) for execution. (Note carefully the computation required to transform the step rate (SRT) into the correct 8272 parameter byte.)

RECALIBRATE

This procedure causes the floppy disk read/write head to retract to track 0. The RECALIBRATE procedure requires only one parameter — the drive number on which the recalibrate operation is to be performed. This procedure builds a disk operation control block (RECALIBRATE$DOCB) and passes the control block to the FDC driver for execution.

SEEK

This procedure causes the disk read/write head (on the selected drive) to move to the desired cylinder position. The SEEK procedure is called with three parameters: drive number (DRV), head/side number (HD), and cylinder number (CYL). This software module builds a disk operation control block (SEEK$DOCB) that is executed by the FDC driver.

FORMAT

The FORMAT procedure is designed to initialize a complete floppy disk so that sectors can subsequently be read and written by system and application programs. Three parameters must be supplied to this procedure: the drive number (DRV), the recording density (DENS), and the interleave factor (INTLVE). The FORMAT procedure generates a data block (FMTBLK) and a disk operation control block (FORMAT$DOCB) for each track on the floppy disk (normally 77).

Table 9: Basic Disk Parameters

Name	Description
DENSITY	The recording mode (FM or MFM).
FILLER$BYTE	The data byte to be written in all sectors during formatting.
TRACKSPERDISK	The number of cylinders on the floppy disk.
BYTESPERSECTOR	The number of bytes in each disk sector. The exerciser accepts 128, 256, and 512 in FM mode, and 256, 512, and 1024 in MFM mode.
INTERLEAVE	The sector interleave factor for each disk track.
STEP$RATE	The disk drive step rate (1-16 milliseconds).
HEAD$LOAD$TIME	The disk drive head load time (2-254 milliseconds).
HEAD$UNLOAD$TIME	The head unload time (16-240 milliseconds).

The format data block specifies the four sector ID field parameters (cylinder, head, sector, and bytes per sector) for each sector on the track. The sector numbers need not be sequential; the interleave factor (INTLVE parameter) is used to compute the logical to physical sector mapping.

After both the format data block and the operation control block are generated for a given cylinder, control is passed to the 8272 drivers for execution. After the format operation is complete, a SEEK to the next cylinder is performed, a new format table is generated, and another track formatting operation is executed by the drivers. This track formatting continues until all tracks on the diskette are formatted.

In some systems, bad tracks must also be specified when a disk is formatted. For these systems, the existing FORMAT procedure should be modified to format bad tracks with a cylinder number of OFFH.

WRITE

The WRITE procedure transfers a complete sector of data to the disk drive. Five parameters must be supplied to this software module: the drive number (DRV), the cylinder number (CYL), the head/side number (HD), the sector number (SEC) and the recording density (DENS). This procedure generates a disk operation control block (WRITE$DOCB) from these parameters and passes the control block to the 8272 driver for execution. When control returns to the calling program, the data has been transferred to disk.

READ

This procedure is identical to the WRITE procedure except the direction of data transfer is reversed. The READ procedure transfers a sector of data from the floppy disk to system memory.

Coping With Errors

In actual practice all logical disk interface routines would contain error processing mechanisms. (Errors have been ignored for the sake of simplicity in the exerciser programs listed in Appendix B.) A typical error recovery technique consists of a two-stage procedure. First, when an error is detected, a recalibrate operation is performed followed by a retry of the failed operation. This procedure forces the drive to seek directly to the requested cylinder (lowering the probability of a seek error) and attempts to perform the requested operation an additional time. Soft (temporary) errors caused by mechanical or electrical interference do not normally recur during the retry operation; hard errors (caused by media or drive failures), on the other hand, will continue to occur during retry operations. If, after a number of retries (approximately 10), the operation continues to fail, an error message is displayed to the system operator. This error message lists the drive number, type of operation, and failure status (from the FDC). It is the operator's responsibility to take additional action as required.

6. File Systems

The file system provides the disk I/O interface level most familiar to users of interactive microcomputer and minicomputer systems. In a file system, all data is stored in named disk areas called files. The user and applications programs need not be concerned with the exact location of a file on the disk — the disk file system automatically determines the file location from the file name. Files may be created, read, written, modified, and finally deleted (destroyed) when they are no longer needed. Each floppy disk typically contains a directory that lists all the files existing on the disk. A directory entry for a file contains information such as file name, file size, and the disk address (track and sector) of the beginning of the file.

File Allocation

File storage is actually allocated on the disk (by the file system) in fixed size areas called blocks. Normally a block is the same size as a disk sector. Files are created by finding and reserving enough unused blocks to contain the data in the file. Two file allocation methods are currently in widespread use. The first method allocates blocks (for a file) from a sequential pool of unused blocks. Thus, a file is always contained in a set of sequential blocks on the disk. Unfortunately, as files are created, updated, and deleted, these free-block pools become fragmented (separated from one another). When this fragmentation occurs, it often becomes impossible for the file system to create a file even though there is a sufficient number of free blocks on the disk. At this point, special programs must be run to "squeeze" or compact the disk, in order to re-create a single contiguous free-block pool.

The second file allocation method uses a more flexible technique in which individual data blocks may be located anywhere on the disk (with no restrictions). With this technique, a file directory entry contains the disk address of a file pointer block rather than the disk address of the first data block of the file. This file pointer block contains pointers (disk addresses) for each data block in the file. For example, the first pointer in the file pointer block contains the track and sector address of the first data block in the file, the second pointer contains the disk address of the second data block, etc.

In practice, pointer blocks are usually the same size as data blocks. Therefore, some files will require multiple pointer blocks. To accommodate this requirement without loss of flexibility, pointer blocks are linked together, that is, each pointer block contains the disk address of the following pointer block. The last pointer block of the file is signalled by an illegal disk address (e.g., track 0, sector 0 or track OFFH, sector OFFH).

The Intel File System

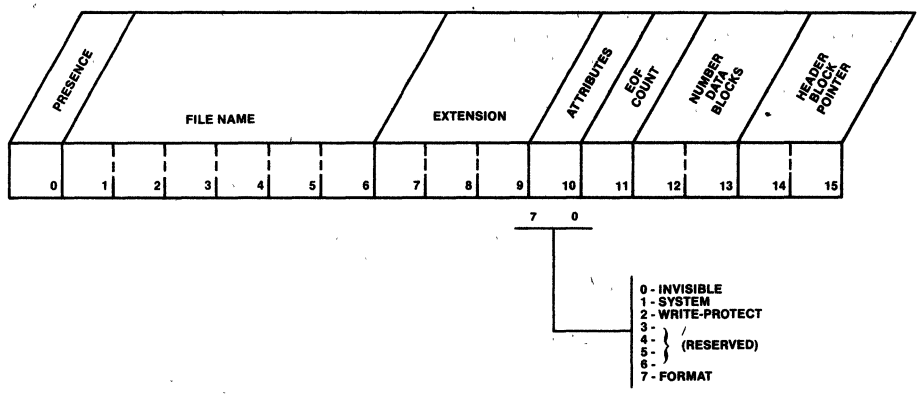
The Intel file system (described in detail in the RMX-80 Users Guide) uses the second disk file allocation method (previously discussed). In order to lower the system overhead involved in finding free data blocks, the Intel file system incorporates a free space management data structure known as a bit map. Each disk sector is represented by a single bit in the bit map. If a bit in the bit map is set to 1, the corresponding disk sector has been allocated. A zero in the bit map indicates that the corresponding sector is free. With this technique, the process of allocating or freeing a sector is accomplished by simply altering the bit map.

File names consist of a basic file name (up to six characters) and a file extension (up to three characters). The basic file name and the file extension are separated by a period (.). Examples of valid file names are: DRIV72.OBJ, XX.TMP, and FILE.CS. In addition, four file attributes are supported (see Figure 3 for attribute definitions).

The bit map and the file directory are placed on prespecified disk tracks (reserved for system use) beginning at track zero.

Disk File System Functions

Table 2 illustrates the typical functions implemented by a disk file system. As an example, the disk directory function (DIR) lists disk file information on the console display terminal. Figure 3 details the contents of a display entry in the Intel file system. The PL/M procedure outlined in Figure 4 illustrates a disk directory algorithm that displays the file name, the file attributes, and the file size (in blocks) for each file in the directory.



AFN-01949A

Directory Entry

Presence is a flag that can contain one of three values:

000H - The file associated with this entry is present on the disk.

07FH - No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 07FH marks the current logical end of the directory and directory searches stop at this entry.

0FFH - The file named in this entry once existed on the disk but is currently deleted. The next file added to the directory will be placed in the first entry marked 0FFH. This flag cannot, therefore, be used to (reliably) find a file that has been deleted. A value of 0FFH should be thought of as simply marking an open directory entry.

File Name is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the file name is shorter than six characters, the remaining bytes contain binary zeros. For example, the name ALPHA would be stored as: 414C50484100H.

Extension is a string of up to 3 non-blank ASCII characters that specifies an extension to the file name. Extensions often identify the type of data in the file such as OBJ (object module), or PLM (PL/M source module). As with the file name, unused positions in the extension field are filled with binary zeros.

Figure 3. Intel Directory Entry Format

Attributes are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low-order or rightmost bit, bit 7 is the leftmost bit):

0: Invisible. Files with this attribute are not listed by the ISIS-II DIR command unless the I switch is used. All system files are invisible.

1: System. Files with this attribute are copied to the disk in drive 1 when the S switch is specified with the ISIS-II FORMAT command.

2: Write-Protect. Files with this attribute cannot be opened for output or update, nor can they be deleted or renamed.

3-6: These positions are reserved for future use.

7: Format. Files with this attribute are treated as though they are write-protected. In addition, these files are created on a new diskette when the ISIS-II FORMAT command is issued. The system files all have the FORMAT attribute and it should not be given to any other files.

EOF Count contains the number of the last byte in the last data block of the file. If the value of this field is 080H, for example, the last byte in the file is byte number 128 in the last data block (the last block is full).

Number of Data Blocks is an address variable that indicates the number of data blocks currently used by the file. ISIS-II and the RMX/80 Disk File system both maintain a counter called LENGTH that is the current number of bytes in the file. This is calculated as:

((NUMBER OF DATA BLOCKS - 1) x 128 + EOF COUNT.

Header Block Pointer is the address of the file's header block. The high byte of the field is the sector number and the low byte is the track number. The system "finds" a disk file by searching the directory for the name and then using the header block pointer to seek to the beginning of the file.

Figure 3. Intel Directory Entry Format (Continued)

```
dir: procedure(drv,dens)   public;
  declare  drv             byte,
           dens            byte,
           sector          byte,
           i               byte,
           dir$ptr         byte,
           dir$entry       based rdbptr structure (presence byte,
                           file$name(6) byte,extension(3) byte,
                           attribute byte,eof$count byte,
                           data$blocks address,header$ptr address),
           size (5)        byte,

           invisible$flag  literally '1',
           system$flag     literally '2',
           protected$flag  literally '4',
           format$flag     literally '80H';


  /* The disk directory starts at cylinder 1, sector 2 */
  call seek(drv,1,0);
  do sector=2 to 26;
    call read(drv,1,0,sector,dens);
    do dir$ptr=0 to 112 by 4;
      if dir$entry.presence=7FH then return;
      if dir$entry.presence=0
        then do;
          do i=0 to 5; call co(dir$entry.file$name(i)); end;
          call co(period);
          do i=0 to 2; call co(dir$entry.extension(i)); end;
          do i=0 to 4; call co(space); end;
          call convert$to$decimal(@size,dir$entry.data$blocks);
          do i=0 to 4; call co(size(i)); end;
          If (dir$entry.attribute and invisible$flag) <> 0 then call co('I');
          If (dir$entry.attribute and system$flag) <> 0 then call co('S');
          If (dir$entry.attribute and protected$flag) <> 0 then call co('W');
          If (dir$entry.attribure and format$flag) <> 0 then call co('F');
        end;
    end;
  end;

end dir;
```

AFN-01949A

Figure 4. Sample PL/M Directory Procedure

7. Key 8272 Software Interfacing Considerations

This section contains a quick review of Key 8272 Software design features and issues. (Most items have been mentioned in other sections of this application note.) Before designing 8272 software drivers, it is advisable that the information in this section be thoroughly understood.

1. Non-DMA Data Transfers

In systems that operate without a DMA controller (in the polled or interrupt driven mode), the system software is responsible for counting data transfers to/from the 8272 and generating a TC signal to the FDC when the transfer is complete.

2. Processor Command/Result Phase Interface

In the command phase, the driver software must write the exact number of parameters in the exact order shown in Table 5. During the result phase, the driver must read the complete result status. For example, the Format Track command requires six command bytes and presents seven result bytes. The 8272 will not accept a new command until all result bytes are read. Note that the number of command and result bytes varies from command-to-command. **Command and result phases cannot be shortened.**

During both the command and result phases, the Main Status Register must be read by the driver before each byte of information is read from, or written to, the FDC Data Register. Before each command byte is written, DIO (bit 6) must be low (indicating a data transfer from the processor) and RQM (bit 7) must be high (indicating that the FDC is ready for data). During the result phase, DIO must be high (indicating a data transfer to the processor) and RQM must also be high (indicating that data is ready for the processor).

Note: After the 8272 receives a command byte, the RQM flag may remain set for approximately 16 microseconds (with an 8 MHz clock). The driver should not attempt to read the Main Status Register before this time interval has elapsed; otherwise, the driver may erroneously assume that the FDC is ready to accept the next byte.

3. Sector Sizes

The 8272 does not support 128 byte sectors in the MFM (double-density) mode.

4. Drive Status Changes

The 8272 constantly polls all drives for changes in the drive ready status. This polling begins immediately following RESET. An interrupt is generated every time the FDC senses a change in the drive ready status. After reset, the FDC assumes that all drives are "not ready". If a drive is ready immediately after reset, the 8272 generates a drive status change interrupt.

5. Seek Commands

The 8272 FDC does not perform implied seeks. Before issuing a data read or write command, the read/write head must be positioned over the correct cylinder by means of an explicit seek command. If the head is not positioned correctly, a cylinder address error is generated.

6. Interrupt Processing

When the processor receives an interrupt from the FDC, the FDC may be reporting one of two distinct events:

a) The beginning of the result phase of a previously requested read, write, or scan command.

b) An asynchronous event such as a seek/recalibrate completion, an attention, an abnormal command termination, or an invalid command.

These two cases are distinguished by the FDC BUSY flag (bit 4) in the Main Status Register. If the FDC BUSY flag is high, the interrupt is of type (a). If the FDC BUSY flag is low, the interrupt was caused by an asynchronous event (b).

A single interrupt from the FDC may signal more than one of the above events. After receiving an interrupt, the processor must continue to issue Sense Interrupt Status commands (and service the resulting conditions) until an invalid command code is received. In this manner, all "hidden" interrupts are ferreted out and serviced.

7. Skip Flag (SK)

The skip flag is used during the execution of Read Data, Read Deleted Data, Read Track, and various Scan commands. This flag permits the FDC to skip unwanted sectors on a disk track.

When performing a Read Data, Read Track, or Scan command, a high SK flag indicates that the FDC is to skip over (not transfer) any sector containing a deleted data address mark. A low SK flag indicates that the FDC is to terminate the command (after reading all the data in the sector) when a deleted data address mark is encountered.

When performing a Read Deleted Data command, a high SK flag indicates that sectors containing normal data address marks are to be skipped. Note that this is just the opposite situation from that described in the last paragraph. When a data address mark is encountered during a Read Deleted Data command (and the SK flag is low), the FDC terminates the command after reading all the data in the sector.

8. Bad Track Maintenance

The 8272 does not internally maintain bad track information. The maintenance of this information must be performed by system software. As an example of typical bad track operation, assume that a media test determines that track 31 and track 66 of a given floppy disk are bad. When the disk is formatted for use, the system software formats physical track 0 as logical cylinder 0 (C=0 in the command phase parameters), physical track 1 as logical track 1 (C=1), and so on, until physical track 30 is formatted as logical cylinder 30 (C=30). Physcial track 31 is bad and should be formatted as logical cylinder FF (indicating a bad track). Next, physical track 32 is formatted as logical cylinder 31, and so on, until physical track 65 is formatted as logical cylinder 64. Next, bad physical track 66 is formatted as logical cylinder FF (another bad track marker), and physical track 67 is formatted as logical cylinder 65. This formatting continues until the last physical track (77) is formatted as logical cylinder 75. Normally, after this formatting is complete, the bad track information is stored in a prespecified area on the floppy disk (typically in a sector on track 0) so that the system will be able to recreate the bad track information when the disk is removed from the drive and reinserted at some later time.

To illustrate how the system software performs a transfer operation on a disk with bad tracks, assume that the disk drive head is positioned at track 0 and the disk described above is loaded into the drive. If a command to read track 36 is issued by an application program, the system software translates this read command into a seek to physical track 37 (since there is one bad track between 0 and 36, namely 31) followed by a read of logical cylinder 36. Thus, the cylinder parameter C is set to 37 for the Seek command and 36 for the Read Sector command.

REFERENCES

1. Intel, "8272 Single/Double Density Floppy Disk Controller Data Sheet," Intel Corporation, 1980.

2. Intel, "An Intelligent Data Base System Using the 8272," Intel Application Note, AP-116, 1981.

3. Intel, iSBC 208 Hardware Reference Manual, Manual Order No. 143078, Intel Corporation, 1980.

4. Intel, RMX/80 User's Guide, Manual Order No. 9800522, Intel Corporation, 1978

5. Brinch Hansen, P., Operating System Principles, Prentice-Hall, Inc., New Jersey, 1973.

6. Flores, I., Computer Software: Programming Systems for Digital Computers, Prentice-Hall, Inc., New Jersey, 1965.

7. Knuth, D. E., Fundamental Algorithms, Addison-Wesley Publishing Company, Massachusetts, 1975.

8. Shaw, A. C., The Logical Design of Operating Systems, Prentice-Hall, Inc., New Jersey, 1974.

9. Watson, R. W., Time Sharing System Design Concepts, McGraw-Hill, Inc., New York, 1970.

10. Zarrella, J., Operating Systems: Concepts and Principles, Microcomputer Applications, California, 1979.

APPENDIX A
8272 FDC DEVICE DRIVER SOFTWARE

```
PL/M-86 COMPILER    8272 FLOPPY DISK CONTROLLER DEVICE DRIVERS

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE DRIVERS
OBJECT MODULE PLACED IN :F1:driv72.OBJ
COMPILER INVOKED BY:  plm86 :F1:driv72.p86 DEBUG

                $title('8272 floppy disk controller device drivers')
                $nointvector
                $optimize(2)
                $large

  1             drivers: do;

  2   1         declare
                  /* floppy disk port definitions */
                  fdc$status$port       literally '30H',        /* 8272 status port */
                  fdc$data$port         literally '31H';        /* 8272 data port */

  3   1         declare
                  /* floppy disk commands */
                  sense$int$status      literally '08H';

  4   1         declare
                  /* interrupt definitions */
                  fdc$int$level         literally '33';     /* fdc interrupt level */

  5   1         declare
                  /* return status and error codes */
                  error                 literally '0',
                  ok                    literally '1',
                  complete              literally '3',
                  false                 literally '0',
                  true                  literally '1',
                  error$in              literally 'not',
                  propagate$error       literally 'return error',

                  stat$ok               literally '0',          /* fdc operation completed without errors */
                  stat$busy             literally '1',          /* fdc is busy, operation cannot be started */
                  stat$error            literally '2',          /* fdc operation error */
                  stat$command$error    literally '3',          /* fdc not ready for command phase */
                  stat$result$error     literally '4',          /* fdc not ready for result phase */
                  stat$invalid          literally '5';          /* invalid fdc command */

  6   1         declare
                  /* masks */
                  busy$mask             literally '10H',
                  DIO$mask              literally '40H',
                  RQM$mask              literally '80H',
                  seek$mask             literally '0FH',
                  result$error$mask     literally '0C0H',
                  result$drive$mask     literally '03H',
                  result$ready$mask     literally '08H';

  7   1         declare
                  /* drive numbers */
                  max$no$drives         literally '3',
                  fdc$general           literally '4';

  8   1         declare
                  /* miscellaneous control */
                  any$drive$seeking     literally '((input(fdc$status$port) and seek$mask) <> 0)',
                  command$code          literally '(docb.disk$command(0) and 1FH)',
                  DIO$set$for$input     literally '((input(fdc$status$port) and DIO$mask)=0)',
                  DIO$set$for$output    literally '((input(fdc$status$port) and DIO$mask)<>0)',
                  extract$drive$no      literally '(docb.disk$command(1) and 03H)',
                  fdc$busy              literally '((input(fdc$status$port) and busy$mask) <> 0)',
                  no$fdc$error          literally 'possible$error(command$code) and ((docb.disk$result(0)
                                                  and result$error$mask) = 0)',
                  wait$for$op$complete  literally 'do while not operation$complete(drive$no); end',
                  wait$for$RQM          literally 'do while (input(fdc$status$port) and RQM$mask) = 0; end;';

  9   1         declare
                  /* structures */
                  docb$type             literally               /* disk operation control block */
                        '(dma$op byte,dma$addr word, dma$addr$ext byte,dma$count word,
                          disk$command(9) byte,disk$result(7) byte,misc byte)';

                $eject
 10   1         declare
                  drive$status$change(4) byte public,           /* when set - indicates that drve status changed */
                  drive$ready(4) byte public;                   /* current status of drives */
```

```
11  1     declare
            operation$in$progress(5) byte,              /* internal flags for operation with multiple drives */
            operation$complete(5) byte,                 /* fdc execution phase completed */
            operation$docb$ptr(5) pointer,              /* pointers for operations in progress */
            interrupt$docb structure docb$type,         /* temporary docb for interrupt processing */
            global$drive$no byte;                       /* drive number of non-overlapped operation
                                                           in progress - if any */

12  1     declare
            /* internal vectors that contain command operational information */
            no$result(32) byte                          /* no result phase to command */
              data(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
            immed$result(32) byte                       /* immediate result phase for command */
              data(0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
            overlap$operation(32) byte                  /* command permits overlapped operation of drvies */
              data(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
            drive$no$present(32) byte                   /* drive number present in command information */
              data(0,0,1,0,1,1,1,1,0,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0),
            possible$error(32) byte                     /* determines if command can return with an error */
              data(0,0,1,0,0,1,1,1,1,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0),
            command$length(32) byte                     /* contains number of command bytes for each command */
              data(0,0,9,3,2,9,9,2,1,9,2,0,9,6,0,3,0,9,0,0,0,0,0,0,0,0,9,0,0,0,9,0,0),
            valid$command(32) byte                      /* flags invalid command codes */
              data(0,0,1,1,1,1,1,1,1,1,1,0,1,1,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0);

          $eject

          /**** initialization for the 8272 fdc driver software.  This procedure must
                be called prior to execution of any driver software.    ****/

13  1     initialize$drivers: procedure public;
          /* initialize 8272 drivers */
14  2       declare drv$no byte;

15  2       do drv$no=0 to max$no$drives;
16  3         drive$ready(drv$no)=false;
17  3         drive$status$change(drv$no)=false;
18  3         operation$in$progress(drv$no)=false;
19  3         operation$complete(drv$no)=false;
20  3       end;

21  2       operation$in$progress(fdc$general)=false;
22  2       operation$complete(fdc$general)=false;
23  2       global$drive$no=0;

24  2     end initialize$drivers;


          /**** wait until the 8272 fdc is ready to receive command/parameter bytes
                in the command phase.  The 8272 is ready to receive command bytes
                when the RQM flag is high and the DIO flag is low.    ****/

25  1     fdc$ready$for$command: procedure byte;

            /* wait for valid flag settings in status register */
26  2       call time(1);

            /* wait for "master request" flag */
27  2       wait$for$RQM;

            /* check data direction flag */
30  2       if DIO$set$for$input
              then return ok;
32  2         else return error;

33  2     end fdc$ready$for$command;


          /**** wait until the 8272 fdc is ready to return data bytes in the result
                phase.  The 8272 is ready to return a result byte when the RQM and DIO
                flags are both high.  The busy flag in the main status register will
                remain set until the last data byte of the result phase has been read
                by the processor.    ****/

34  1     fdc$ready$for$result: procedure byte;

            /* wait for valid settings in status register */
35  2       call time(1);

            /* result phase has ended when the 8272 busy flag is reset */
36  2       if not fdc$busy
              then return complete;
```

```
                  /* wait for "master request" flag */
38   2            wait$for$RQM;

                  /* check data direction flag */
41   2            if DIO$set$for$output
                    then return ok;
43   2              else return error;

44   2          end fdc$ready$for$result;


                /**** output a single command/parameter byte to the 8272 fdc.  The "data$byte"
                      parameter is the byte to be output to the fdc.    ****/

45   1          output$byte$to$fdc: procedure(data$byte) byte;
46   2            declare data$byte byte;

                  /* check to see if fdc is ready for command */
47   2            if not fdc$ready$for$command
                    then propagate$error;

49   2            output(fdc$data$port)=data$byte;

50   2            return ok;
51   2          end output$byte$to$fdc;


                /**** input a single result byte from the 8272 fdc.  The "data$byte$ptr"
                      parameter is a pointer to the memory location that is to contain
                      the input byte.    ****/

52   1          input$byte$from$fdc: procedure(data$byte$ptr) byte;
53   2            declare data$byte$ptr pointer;
54   2            declare
                    data$byte based data$byte$ptr byte,
                    status byte;

                  /* check to see if fdc is ready */
55   2            status=fdc$ready$for$result;
56   2            if error$in status
                    then propagate$error;

                  /* check for result phase complete */
58   2            if status=complete
                    then return complete;

60   2            data$byte=input(fdc$data$port);
61   2            return ok;
62   2          end input$byte$from$fdc;

                $eject

                /**** output the dma mode, the dma address, and the dma word count to the
                      8237 dma controller.  Also output the high order four bits of the
                      address to the address extension latch.  Finally, start the disk
                      dma channel.  The "docb$ptr" parameter is a pointer to the appropriate
                      disk operation control block.    ****/

63   1          output$controls$to$dma: procedure(docb$ptr);
64   2            declare docb$ptr pointer;
65   2            declare docb based docb$ptr structure docbtype;

66   2            declare
                    /* dma port definitions */
                    dma$upper$addr$port   literally '10H',      /* upper 4 bits of current address */
                    dma$disk$addr$port    literally '00H',      /* current address port */
                    dma$disk$word$count   literally '01H',      /* word count port */
                    dma$command$port      literally '08H',      /* command port */
                    dma$mode$port         literally '0BH',      /* mode port */
                    dma$mask$sr$port      literally '0AH',      /* mask set/reset port */
                    dma$clear$ff$port     literally '0CH',      /* clear first/last flip-flop port */
                    dma$master$clear$port literally '0DH',      /* dma master clear port */
                    dma$mask$port         literally '0FH',      /* parallel mask set port*/

                    dma$disk$chan$start   literally '00H',      /* dma mask to start disk channel */
                    dma$extended$write    literally 'shl(1,5)', /* extended write flag */
                    dma$single$transfer   literally 'shl(1,6)'; /* single transfer flag */

67   2            if docb.dma$op < 3
                    then do;
                      /* set dma mode and clear first/last flip-flop */
69   3                output(dma$mode$port)=shl(docb.dma$op,2) or 40H;
70   3                output(dma$clear$ff$port)=0;
```

```
                    /* set dma address */
 71   3             output(dma$disk$addr$port)=low(docb.dma$addr);
 72   3             output(dma$disk$addr$port)=high(docb.dma$addr);
 73   3             output(dma$upper$addr$port)=docb.dma$addr$ext;

                    /* output disk transfer word count to dma controller */
 74   3             output(dma$disk$word$count)=low(docb.dma$count);
 75   3             output(dma$disk$word$count)=high(docb.dma$count);

                    /* start dma channel 0 for fdc */
 76   3             output(dma$mask$sr$port)=dma$disk$chan$start;
 77   3           end;

 78   2       end output$controls$to$dma;


                /**** output a high-level disk command to the 8272 fdc.  The number of bytes
                      required for each command is contained in the "command$length" table.
                      The "docb$ptr" parameter is a pointer to the appropriate disk operation
                      control block.    ****/

 79   1       output$command$to$fdc: procedure(docb$ptr) byte;
 80   2         declare docb$ptr pointer;

 81   2         declare
                    docb based docb$ptr structure docb$type,
                    cmd$byte$no byte;

 82   2         disable;

                  /* output all command bytes to the fdc */
 83   2         do cmd$byte$no=0 to command$length(command$code)-1;
 84   3           if error$in output$byte$to$fdc(docb.disk$command(cmd$byte$no))
                      then do; enable; propagate$error; end;
 89   3         end;

 90   2         enable;
 91   2         return ok;
 92   2       end output$command$to$fdc;


                /**** input the result data from the 8272 fdc during the result phase (after
                      command execution).  The "docb$ptr" parameter is a pointer to the
                      appropriate disk operation control block.    ****/

 93   1       input$result$from$fdc: procedure(docb$ptr) byte;
 94   2         declare docb$ptr pointer;
 95   2         declare
                    docb based docb$ptr structure docb$type,
                    result$byte$no byte,
                    temp byte,
                    status byte;

 96   2         disable;

 97   2         do result$byte$no=0 to 7;
 98   3           status=input$byte$from$fdc(@temp);
 99   3           if error$in status
                      then do; enable; propagate$error; end;
104   3           if status=complete
                      then do; enable; return ok; end;
109   3           docb.disk$result(result$byte$no)=temp;
110   3         end;

111   2         enable;
112   2         if fdc$busy
                    then return error;
114   2           else return ok;
115   2       end input$result$from$fdc;


                /**** cleans up after the execution of a disk operation that has no result
                      phase.  The procedure is also used after some disk operation errors.
                      "drv" is the drive number, and "cc" is the command code for the
                      disk operation.    ****/

116   1       operation$clean$up: procedure(drv,cc);
117   2         declare (drv,cc) byte;

118   2         disable;
119   2         operation$in$progress(drv)=false;
```

```
120  2        if not overlap$operation(cc)
                then global$drive$no=0;
122  2        enable;

123  2      end operation$clean$up;

            $eject

            /**** execute the disk operation control block specified by the pointer
                  parameter "docb$ptr".  The "status$ptr" parameter is a pointer to
                  a byte variable that is to contain the status of the requested
                  operation when it has been completed.  Six status conditions are
                  possible on return:

                    0    The specified operation was completed without error.
                    1    The fdc is busy and the requested operation cannot be started.
                    2    Fdc error (further information is contained in the result
                         storage portion of the disk operation control block - as
                         described in the 8272 data sheet).
                    3    Transfer error during output of the command bytes to the fdc.
                    4    Transfer error during input of the result bytes from the fdc.
                    5    Invalid fdc command.    ****/

124  1      execute$docb: procedure(docb$ptr,status$ptr) public;
            /* execute a disk operation control block */

125  2        declare docb$ptr pointer, status$ptr pointer;
126  2        declare
               docb based docb$ptr structure docb$type,
               status based status$ptr byte,
               drive$no byte;

             /* check command validity */
127  2        if not valid$command(command$code)
               then do; status=stat$invalid; return; end;

             /* determine if command has a drive number field - if not, set the drive
                number for a general fdc command */
132  2        if drive$no$present(command$code)
               then drive$no=extract$drive$no;
134  2          else drive$no=fdc$general;

             /* an overlapped operation can not be performed if the fdc is busy */
135  2        if overlap$operation(command$code) and fdc$busy
               then do; status=stat$busy; return; end;

             /* for a non-overlapped operation, check fdc busy or any drive seeking */
140  2        if not overlap$operation(command$code) and (fdc$busy or any$drive$seeking)
               then do; status=stat$busy; return; end;

             /* check for drive operation in progress - if none, set flag and start operation */
145  2        disable;
146  2        if operation$in$progress(drive$no)
               then do; enable; status=stat$busy; return; end;
152  2          else operation$in$progress(drive$no)=true;

             /* at this point, an fdc operation is about to begin, so:
                1. reset the operation complete flag
                2. set the docb pointer for the current operation
                3. if this is not an overlapped operation, set the global drive
                   number for the subsequent result phase interrupt.  */
153  2        operation$complete(drive$no)=0;
154  2        operation$docb$ptr(drive$no)=docb$ptr;

155  2        if not overlap$operation(command$code)
               then global$drive$no=drive$no+1;
157  2        enable;

158  2        call output$controls$to$dma(docb$ptr);
159  2        if error$in output$command$to$fdc(docb$ptr)
               then do;
161  3            call operation$clean$up(drive$no,command$code);
162  3            status=stat$command$error;
163  3            return;
164  3          end;

             /* return immediately if the command has no result phase or completion interrupt - specify */
165  2        if no$result(command$code)
               then do;
167  3            call operation$clean$up(drive$no,command$code);
168  3            status=stat$ok;
169  3            return;
170  3          end;
```

```
171  2        if immed$result(command$code)
                then do;
173  3          if error$in input$result$from$fdc(docb$ptr)
                  then do;
175  4            call operation$clean$up(drive$no,command$code);
176  4            status=stat$result$error;
177  4            return;
178  4          end;
179  3        end;
180  2        else do;
181  3          wait$for$op$complete;
183  3          if docb.misc = error
                  then do; status=stat$result$error; return; end;
188  3        end;

189  2        if no$fdc$error
                then status=stat$ok;
191  2          else status=stat$error;

192  2      end execute$docb;

            $eject

            /**** copy disk command results from the interrupt control block to the
                  currently active disk operation control block if a disk operation is
                  in progress.    ****/

193  1      copy$int$result: procedure(drv);
194  2        declare drv byte;
195  2        declare
                i byte,
                docb$ptr pointer,
                docb based docb$ptr structure docb$type;

196  2        if operation$in$progress(drv)
                then do;
198  3          docb$ptr=operation$docb$ptr(drv);
199  3          do i=1 to 6; docb.disk$result(i)=interrupt$docb.disk$result(i); end;
202  3          docb.misc=ok;
203  3          operation$in$progress(drv)=false;
204  3          operation$complete(drv)=true;
205  3        end;

206  2      end copy$int$result;


            /**** interrupt processing for 8272 fdc drivers.  Basically, two types of
                  interrupts are generated by the 8272: (a)when the execution phase of
                  an operation has been completed, an interrupt is generated to signal
                  the beginning of the result phase (the fdc busy flag is set
                  when this interrupt is received), and (b) when an overlapped operation
                  is completed or an unexpected interrupt is received (the fdc busy flag
                  is not set when this interrupt is received).

                  When interrupt type (a) is received, the result bytes from the operation
                  are read from the 8272 and the operation complete flag is set.

                  When an interrupt of type (b) is received, the interrupt result code is
                  examined to determine which of the following four actions are indicated:

                    1. An overlapped option (recalibrate or seek) has been completed.  The
                       result data is read from the 8272 and placed in the currently active
                       disk operation control block.
                    2. An abnormal termination of an operation has occurred.  The result
                       data is read and placed in the currently active disk operation
                       control block.
                    3. The execution of an invalid command has been attempted.  This
                       signals the successful completion of all interrupt processing.
                    4. The ready status of a drive has changed.  The "drive$ready" and
                       "drive$ready$status" change tables are updated.  If an operation
                       is currently in progress on the affected drive, the result data
                       is placed in the currently active disk operation control block.

                  After an interrupt is processed, additional sense interrupt status commands
                  must be issued and processed until an invalid command result is returned
                  from the fdc.  This action guarantees that all "hidden" interrupts
                  are serviced.    ****/
```

```
207   1      fdcint: procedure public interrupt fdc$int$level;
208   2        declare
                 invalid byte,
                 drive$no byte,
                 docb$ptr pointer,
                 docb based docb$ptr structure docb$type;

209   2        declare
                 /* interrupt port definitions */
                 ocw2                    literally '70H',
                 nseoi                   literally 'sh1(1,5)';

210   2        declare
                 /* miscellaneous flags */
                 result$code             literally 'shr(interrupt$docb.disk$result(0) and result$error$mask,6)',
                 result$drive$ready      literally '((interrupt$docb.disk$result(0) and result$ready$mask) = 0)',
                 extract$result$drive$no literally '(interrupt$docb.disk$result(0) and result$drive$mask)',
                 end$of$interrupt        literally 'output(ocw2)=nseoi';

               /* if the fdc is busy when an interrupt is received, then the result
                  phase of the previous non-overlapped operation has begun */
211   2         if fdc$busy
                   then do;
                     /* process interrupt if operation in progress */
213   3              if global$drive$no <> 0
                       then do;
215   4                  docb$ptr=operation$docb$ptr(global$drive$no-1);
216   4                  if error$in input$result$from$fdc(docb$ptr)
                           then docb.misc=error;
                           else docb.misc=ok;
218   4                  operation$in$progress(global$drive$no-1)=false;
219   4                  operation$complete(global$drive$no-1)=true;
220   4                  global$drive$no=0;
221   4                end;
222   4              end;
223   3            end;

               /* if the fdc is not busy, then either an overlapped operation has been
                  completed or an unexpected interrupt has occurred (e.g., drive status
                  change) */
224   2          else do;
225   3            invalid=false;
226   3            do while not invalid;

                   /* perform a sense interrupt status operation - if errors are detected,
                      in the actual fdc interface, interrupt processing is discontinued */
227   4              if error$in output$byte$to$fdc(sense$int$status) then go to ignore;
229   4              if error$in input$result$from$fdc(@interrupt$docb) then go to ignore;

231   4              do case result$code;

                       /* case 0 - operation complete */
232   5                do;
233   6                  drive$no=extract$result$drive$no;
234   6                  call copy$int$result(drive$no);
235   6                end;

                       /* case 1 - abnormal termination */

236   5                do;
237   6                  drive$no=extract$result$drive$no;
238   6                  call copy$int$result(drive$no);
239   6                end;

                       /* case 2 - invalid command */
240   5                invalid=true;

                       /* case 3 - drive ready change */
241   5                do;
242   6                  drive$no=extract$result$drive$no;
243   6                  call copy$int$result(drive$no);
244   6                  drive$status$change(drive$no)=true;
245   6                  if result$drive$ready
                             then drive$ready(drive$no)=true;
247   6                      else drive$ready(drive$no)=false;
248   6                end;
249   5              end;
250   4            end;
251   3          end;

252   2        ignore: end$of$interrupt;
253   2      end fdcint;

254   1      end drivers;
```

```
MODULE INFORMATION:
     CODE AREA SIZE     = 0615H    1557D
     CONSTANT AREA SIZE = 0000H       0D
     VARIABLE AREA SIZE = 0050H      80D
     MAXIMUM STACK SIZE = 0032H      50D
     564 LINES READ
     0 PROGRAM ERROR(S)
END OF PL/M-86 COMPILATION
```

APPENDIX B
8272 FDC EXERCISER PROGRAM

```
PL/M-86 COMPILER    8272 FLOPPY DISK DRIVER EXERCISE PROGRAM

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE RUN72
OBJECT MODULE PLACED IN :F1:run72.OBJ
COMPILER INVOKED BY:  plm86 :F1:run72.p86 DEBUG


             $title ('8272 floppy disk driver exercise program')
             $nointvector
             $optimize(2)
             $large
   1         run72: do;

   2   1       declare
                 docb$type                literally          /* disk operation control block */
                   '(dma$op byte,dma$addr word,dma$addr$ext byte,dma$count word,
                   disk$command(9) byte,disk$result(7) byte,misc byte)';

   3   1       declare
                 /* 8272 fdc commands */
                 fm                       literally '0',
                 mfm                      literally '1',
                 dma$mode                 literally '0',
                 non$dma$mode             literally '1',
                 recalibrate$command      literally '7',
                 specify$command          literally '3',
                 read$command             literally '6',
                 write$command            literally '5',
                 format$command           literally '0DH',
                 seek$command             literally '0FH';

   4   1       declare
                 dma$verify               literally '0',
                 dma$read                 literally '1',
                 dma$write                literally '2',
                 dma$noop                 literally '3';

   5   1       declare
                 /* disk operation control blocks */
                 format$docb              structure docb$type,
                 seek$docb                structure docb$type,
                 recalibrate$docb         structure docb$type,
                 specify$docb             structure docb$type,
                 read$docb                structure docb$type,
                 write$docb               structure docb$type;

   6   1       declare
                 step$rate                byte,
                 head$load$time           byte,
                 head$unload$time         byte,
                 filler$byte              byte,
                 operation$status         byte,
                 interleave               byte,
                 format$gap               byte,
                 read$write$gap           byte,
                 index                    byte,
                 drive                    byte,
                 density                  byte,
                 multitrack               byte,
                 sector                   byte,
                 cylinder                 byte,
                 head                     byte,              /* disk drive head */
                 tracks$per$disk          byte,
                 sectors$per$track        byte,
                 bytes$per$sector$code    byte,
                 bytes$per$sector         word;              /* number of bytes in a sector on the disk */

   7   1       declare
                 /* read and write buffers */
                 fmtblk(104)              byte public,
                 wrbuf(1024)              byte public,
                 rdbuf(1024)              byte public;

   8   1       declare
                 /* disk format initialization tables */
                 sec$trk$table(3)         byte data(26,15,8),
                 fmt$gap$table(8)         byte data(1BH,2AH,3AH,0,0,36H,54H,74H),
                 rd$wr$gap$table(8)       byte data(07H,0EH,1BH,0,0,0EH,1BH,35H);
```

```
 9   1        declare
                /* external pointer tables and interrupt vector */
                rdbptr(2)               word external,
                wrbptr(2)               word external,
                fbptr(2)                word external,
                intptr(2)               word external,
                intvec(80H)             word external;


10   1        execute$docb: procedure(docb$ptr,status$ptr) external;
11   2          declare docb$ptr pointer, status$ptr pointer;
12   2        end execute$docb;

13   1        initialize$drivers: procedure external;
14   2        end initialize$drivers;

               $eject

               /**** specify step rate ("srt"), head load time ("hlt"), head unload time ("hut"),
                     and dma or non-dma operation ("nd").   ****/

15   1          specify: procedure(srt,hlt,hut,nd);
16   2            declare (srt,hlt,hut,nd) byte;

17   2            specify$docb.dma$op=dma$noop;
18   2            specify$docb.disk$command(0)=specify$command;
19   2            specify$docb.disk$command(1)=shl((not srt)+1,4) or shr(hut,4);
20   2            specify$docb.disk$command(2)=(hlt and 0FEH) or (nd and 1);
21   2            call execute$docb(@specify$docb,@operation$status);

22   2          end specify;


               /****  recalibrate disk drive
                     8272 automatically steps out until the track 0 signal is activated
                     by the disk drive.   ****/

23   1          recalibrate: procedure(drv);
24   2            declare drv byte;

25   2            recalibrate$docb.dma$op=dma$noop;
26   2            recalibrate$docb.disk$command(0)=recalibrate$command;
27   2            recalibrate$docb.disk$command(1)=drv;
28   2            call execute$docb(@recalibrate$docb,@operation$status);

29   2          end recalibrate;


               /**** seek drive "drv", head (side) "hd" to cylinder "cyl".   ****/

30   1          seek: procedure(drv,cyl,hd);
31   2            declare (drv,cyl,hd) byte;

32   2            seek$docb.dma$op=dma$noop;
33   2            seek$docb.disk$command(0)=seek$command;
34   2            seek$docb.disk$command(1)=drv or shl(hd,2);
35   2            seek$docb.disk$command(2)=cyl;
36   2            call execute$docb(@seek$docb,@operation$status);

37   2          end seek;


               /**** format a complete side ("head") of a single floppy disk in drive "drv".  The density,
                     (single or double) is specified by flag "dens".   ****/

38   1          format: procedure(drv,dens,intlve);
                /*  format disk */
39   2            declare (drv,dens,intlve) byte;
40   2            declare physical$sector byte;

41   2            call recalibrate(drv);
42   2            do cylinder=0 to tracks$per$disk-1;
                    /* set sector numbers in format block to zero before computing interleave */

43   3              do physical$sector=1 to sectors$per$track; fmtblk((physical$sector-1)*4+2)=0; end;
                    /* physical sector 1 equals logical sector 1 */
46   3              physical$sector=1;

                    /* assign interleaved sectors */
47   3              do sector=1 to sectors$per$track;
48   4                index=(physical$sector-1)*4;
```

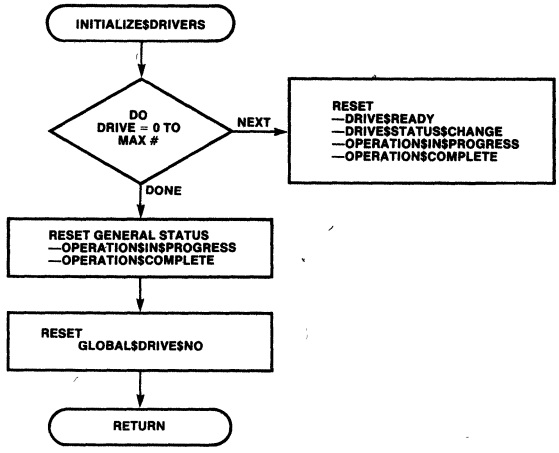
```
                    /* change sector and index if sector has already been assigned */
 49   4             do while fmtblk(index+2) <> 0; index=index+4; physical$sector=physical$sector+1; end;

                    /* set cylinder, head, sector, and size code for current sector into table */
 53   4             fmtblk(index)=cylinder;
 54   4             fmtblk(index+1)=head;
 55   4             fmtblk(index+2)=sector;
 56   4             fmtblk(index+3)=bytes$per$sector$code;

                    /* update physical sector number by interleave */
 57   4             physical$sector=physical$sector+intlve;
 58   4             if physical$sector > sectors$per$track
                      then physical$sector=physical$sector-sectors$per$track;
 60   4           end;

                  /* seek to next cylinder */
 61   3           call seek(drv,cylinder,head);

                  /* set up format control block */
 62   3           format$docb.dma$op=dma$write;
 63   3           format$docb.dma$addr=fbptr(0)+shl(fbptr(1),4);
 64   3           format$docb.dma$addr$ext=0;
 65   3           format$docb.dma$count=sectors$per$track*4-1;
 66   3           format$docb.disk$command(0)=format$command or shl(dens,6);
 67   3           format$docb.disk$command(1)=drv or shl(head,2);
 68   3           format$docb.disk$command(2)=bytes$per$sector$code;
 69   3           format$docb.disk$command(3)=sectors$per$track;
 70   3           format$docb.disk$command(4)=format$gap;
 71   3           format$docb.disk$command(5)=filler$byte;
 72   3           call execute$docb(@format$docb,@operation$status);
 73   3         end;

 74   2       end format;


                /**** write sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
                      disk recording density is specified by the "dens" flag.  Data is expected to be
                      in the global write buffer ("wrbuf").    ****/

 75   1       write: procedure(drv,cyl,hd,sec,dens);
 76   2         declare (drv,cyl,hd,sec,dens) byte;

 77   2         write$docb.dma$op=dma$write;
 78   2         write$docb.dma$addr=wrbptr(0)+shl(wrbptr(1),4);
 79   2         write$docb.dma$addr$ext=0;
 80   2         write$docb.dma$count=bytes$per$sector-1;
 81   2         write$docb.disk$command(0)=write$command or shl(dens,6) or shl(multitrack,7);
 82   2         write$docb.disk$command(1)=drv or shl(hd,2);
 83   2         write$docb.disk$command(2)=cyl;
 84   2         write$docb.disk$command(3)=hd;
 85   2         write$docb.disk$command(4)=sec;
 86   2         write$docb.disk$command(5)=bytes$per$sector$code;
 87   2         write$docb.disk$command(6)=sectors$per$track;
 88   2         write$docb.disk$command(7)=read$write$gap;
 89   2         if bytes$per$sector$code = 0
                  then write$docb.disk$command(8)=bytes$per$sector;
 91   2           else write$docb.disk$command(8)=0FFH;
 92   2         call execute$docb(@write$docb,@operation$status);

 93   2       end write;


                /**** read sector "sec" on drive "drv" at head "hd" and cylinder "cyl".  The
                      disk recording density is defined by the "dens" flag.  Data is read into
                      the global read buffer ("rdbuf").    ****/

 94   1       read: procedure(drv,cyl,hd,sec,dens);
 95   2         declare (drv,cyl,hd,sec,dens) byte;

 96   2         read$docb.dma$op=dma$read;
 97   2         read$docb.dma$addr=rdbptr(0)+shl(rdbptr(1),4);
 98   2         read$docb.dma$addr$ext=0;
 99   2         read$docb.dma$count=bytes$per$sector-1;
100   2         read$docb.disk$command(0)=read$command or shl(dens,6) or shl(multitrack,7);
101   2         read$docb.disk$command(1)=drv or shl(hd,2);
102   2         read$docb.disk$command(2)=cyl;
103   2         read$docb.disk$command(3)=hd;
104   2         read$docb.disk$command(4)=sec;
105   2         read$docb.disk$command(5)=bytes$per$sector$code;
106   2         read$docb.disk$command(6)=sectors$per$track;
107   2         read$docb.disk$command(7)=read$write$gap;
```

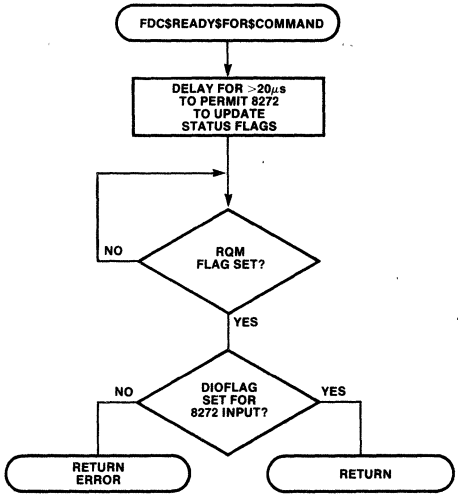
```
108   2          if bytes$per$sector$code = 0
                   then read$docb.disk$command(8)=bytes$per$sector;
110   2          else read$docb.disk$command(8)=0FFH;
111   2        call execute$docb(@read$docb,@operation$status);

112   2      end read;

            $eject

            /**** initialize system by setting up 8237 dma controller and 8259A interrupt
                  controller.    ****/

113   1      initialize$system: procedure;
114   2        declare
                   /* I/O ports */
                   dma$disk$addr$port       literally '00H',       /* current address port */
                   dma$disk$word$count$port literally '01H',       /* word count port */
                   dma$command$port         literally '08H',       /* command port */
                   dma$mode$port            literally '0BH',       /* mode port */
                   dma$mask$sr$port         literally '0AH',       /* mask set/reset port */
                   dma$clear$ff$port        literally '0CH',       /* clear first/last flip-flop port */
                   dma$master$clear$port    literally '0DH',       /* dma master clear port */
                   dma$mask$port            literally '0FH',       /* parallel mask set port*/
                   dma$c1$addr$port         literally '02H',
                   dma$c1$word$count$port   literally '03H',
                   dma$c2$addr$port         literally '04H',
                   dma$c2$word$count$port   literally '05H',
                   dma$c3$addr$port         literally '06H',
                   dma$c3$word$count$port   literally '07H',
                   icw1                     literally '70H',
                   icw2                     literally '71H',
                   icw4                     literally '71H',
                   ocw1                     literally '71H',
                   ocw2                     literally '70H',
                   ocw3                     literally '70H';

115   2        declare
                   /* misc masks and literals */
                   dma$extended$write       literally 'shl(1,5)',  /* extended write flag */
                   dma$single$transfer      literally 'shl(1,6)',  /* single transfer flag */
                   dma$disk$mode            literally '40H',
                   dma$c1$mode              literally '41H',
                   dma$c2$mode              literally '42H',
                   dma$c3$mode              literally '43H',
                   mode$8088                literally '1',
                   interrupt$base           literally '20H',
                   single$controller        literally 'shl(1,1)',
                   level$sensitive          literally 'shl(1,3)',
                   control$word$4$required  literally '1',
                   base$icwl                literally '10H',
                   mask$all                 literally '0FFH',
                   disk$interrupt$mask      literally '1';

116   2        output(dma$master$clear$port)=0;            /* master reset */
117   2        output(dma$mode$port)=dma$extended$write;   /* set dma command mode */

               /* set all dma registers to valid values */
118   2        output(dma$mask$port)=mask$all;             /* mask all channels */

               /* set all addresses to zero */
119   2        output(dma$clear$ff$port)=0;                /* reset first/last flip-flop */
120   2        output(dma$disk$addr$port)=0;
121   2        output(dma$disk$addr$port)=0;
122   2        output(dma$c1$addr$port)=0;
123   2        output(dma$c1$addr$port)=0;
124   2        output(dma$c2$addr$port)=0;
125   2        output(dma$c2$addr$port)=0;
126   2        output(dma$c3$addr$port)=0;
127   2        output(dma$c3$addr$port)=0;

               /* set all word counts to valid values */
128   2        output(dma$clear$ff$port)=0;                /* reset first/last flip-flop */
129   2        output(dma$disk$word$count$port)=1;
130   2        output(dma$disk$word$count$port)=1;
131   2        output(dma$c1$word$count$port)=1;
132   2        output(dma$c1$word$count$port)=1;
133   2        output(dma$c2$word$count$port)=1;
134   2        output(dma$c2$word$count$port)=1;
135   2        output(dma$c3$word$count$port)=1;
136   2        output(dma$c3$word$count$port)=1;
```

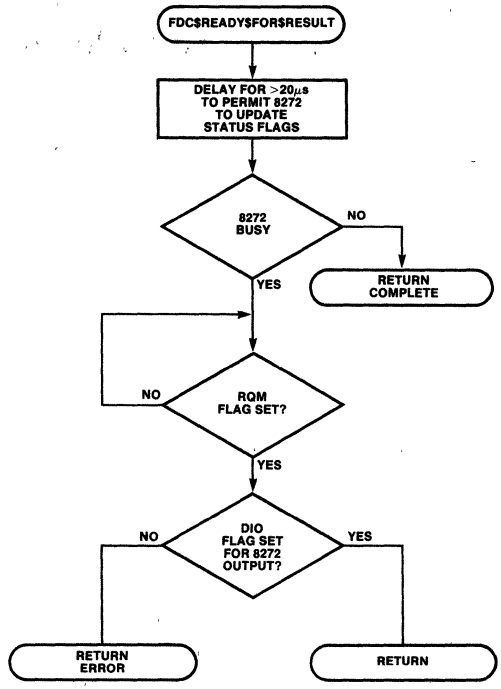
```
                  /* initialize all dma channel modes */
137   2           output(dma$mode$port)=dma$disk$mode;
138   2           output(dma$mode$port)=dma$c1$mode;
139   2           output(dma$mode$port)=dma$c2$mode;
140   2           output(dma$mode$port)=dma$c3$mode;

                  /* initialize 8259A interrupt controller */
141   2           output(icw1)=single$controller or level$sensitive or control$word4$required or base$icw1;
142   2           output(icw2)=interrupt$base;
143   2           output(icw4)=mode$8088;                    /* set 8088 interrupt mode */
144   2           output(ocw1)=not disk$interrupt$mask;      /* mask all interrupts except disk */

                  /* initialize interrupt vector for fdc */
145   2           intvec(40H)=intptr(0);
146   2           intvec(41H)=intptr(1);

147   2         end initialize$system;

                $eject

                /**** main program:   first format disk (all tracks on side (head) 0.  Then
                      read each sector on every track of the disk forever.   ****/

148   1         declare drive$ready(4) byte external;

                  /* disable until interrupt vector setup and initialization complete */
149   1         disable;

                  /* set initial floppy disk parameters */
150   1         density=mfm;                               /* double-density */
151   1         head=0;                                    /* single sided */
152   1         multitrack=0;                              /* no multitrack operation */
153   1         filler$byte=55H;                           /* for format */
154   1         tracks$per$disk=77;                        /* normal floppy disk drive */
155   1         bytes$per$sector=1024;                     /* 1024 bytes in each sector */
156   1         interleave=6;                              /* set track interleave factor */
157   1         step$rate=11;                              /* 10ms for SA800 plus 1 for uncertainty */
158   1         head$load$time=40;                         /* 40ms head load for SA800 */
159   1         head$unload$time=240;                      /* keep head loaded as long as possible */

                  /* derive dependent parameters from those above */
160   1         bytes$per$sector$code=shr(bytes$per$sector,7);
161   1         do index=0 to 3;
162   2           if (bytes$per$sector$code and 1) <> 0
                      then do; bytes$per$sector$code=index; go to donebc; end;
167   2              else bytes$per$sector$code=shr(bytes$per$sector$code,1);
168   2         end;

169   1         donebc:
                sectors$per$track=sec$trk$table(bytes$per$sector$code-density);
170   1         format$gap=fmt$gap$table(shl(density,2)+bytes$per$sector$code);
171   1         read$write$gap=rd$wr$gap$table(shl(density,2)+bytes$per$sector$code);

                  /* initialize system and drivers */
172   1         call initialize$system;
173   1         call initialize$drivers;

                  /* reenable interrupts and give 8272 a chance to report on drive status
                     before proceeding */
174   1         enable;
175   1         call time(10);

                  /* specify disk drive parameters */
176   1         call specify(step$rate,head$load$time,head$unload$time,dma$mode);

177   1         drive=0;                                   /* run single disk drive #0 */

                  /* wait until drive ready */
178   1         do while 1;
179   2           if drive$ready(drive)
                      then go to start;
181   2         end;

182   1         start:
                call format(drive,density,interleave);

183   1         do while 1;
184   2           do cylinder=0 to tracks$per$disk-1;
185   3             call seek(drive,cylinder,head);
186   3             do sector=1 to sectors$per$track;

                      /* set up write buffer */
187   4                 do index=0 to bytes$per$sector-1; wrbuf(index)=index+sector+cylinder; end;
```

```
 190   4              call write(drive,cylinder,head,sector,density);
 191   4              call read(drive,cylinder,head,sector,density);

                      /* check read buffer against write buffer */
 192   4              if cmpw(@wrbuf,@rdbuf,shr(bytes$per$sector,1)) <> 0FFFFH
                        then halt;
 194   4            end;
 195   3          end;
 196   2        end;

 197   1      end run72;

MODULE INFORMATION:

     CODE AREA SIZE     = 0570H   1392D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0907H   2311D
     MAXIMUM STACK SIZE = 0022H     34D
     412 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```

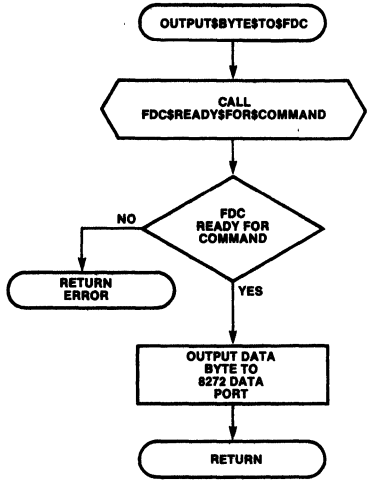
APPENDIX C
8272 DRIVER FLOWCHARTS

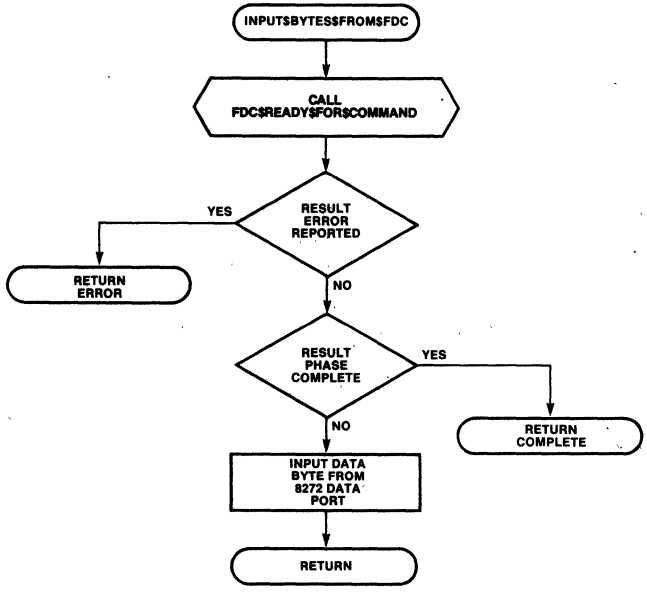

INITIALIZE$DRIVERS
DO
DRIVE = 0 TO
MAX #
NEXT
RESET
—DRIVE$READY
—DRIVE$STATUS$CHANGE
—OPERATIONINPROGRESS
—OPERATION$COMPLETE
DONE
RESET GENERAL STATUS
—OPERATIONINPROGRESS
—OPERATION$COMPLETE
RESET
GLOBAL$DRIVE$NO
RETURN


FDC$READY$FOR$COMMAND
DELAY FOR >20μs
TO PERMIT 8272
TO UPDATE
STATUS FLAGS
NO
RQM
FLAG SET?
YES
NO
DIOFLAG
SET FOR
8272 INPUT?
YES
RETURN
ERROR
RETURN

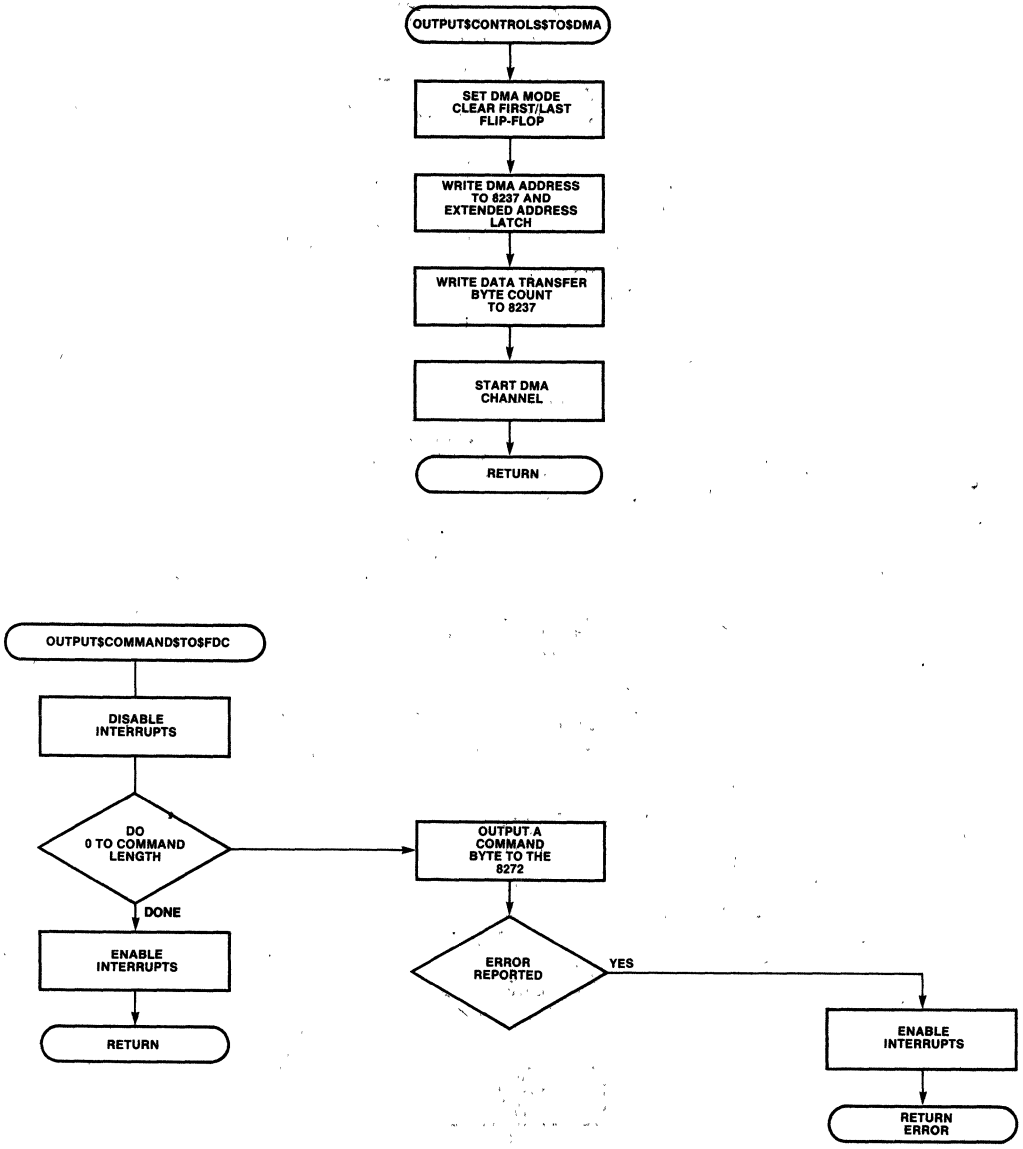

FDC$READY$FOR$RESULT
DELAY FOR >20μs
TO PERMIT 8272
TO UPDATE
STATUS FLAGS
8272
BUSY
NO
RETURN
COMPLETE
YES
NO
RQM
FLAG SET?
YES
NO
DIO
FLAG SET
FOR 8272
OUTPUT?
YES
RETURN
ERROR
RETURN

AFN-01949A

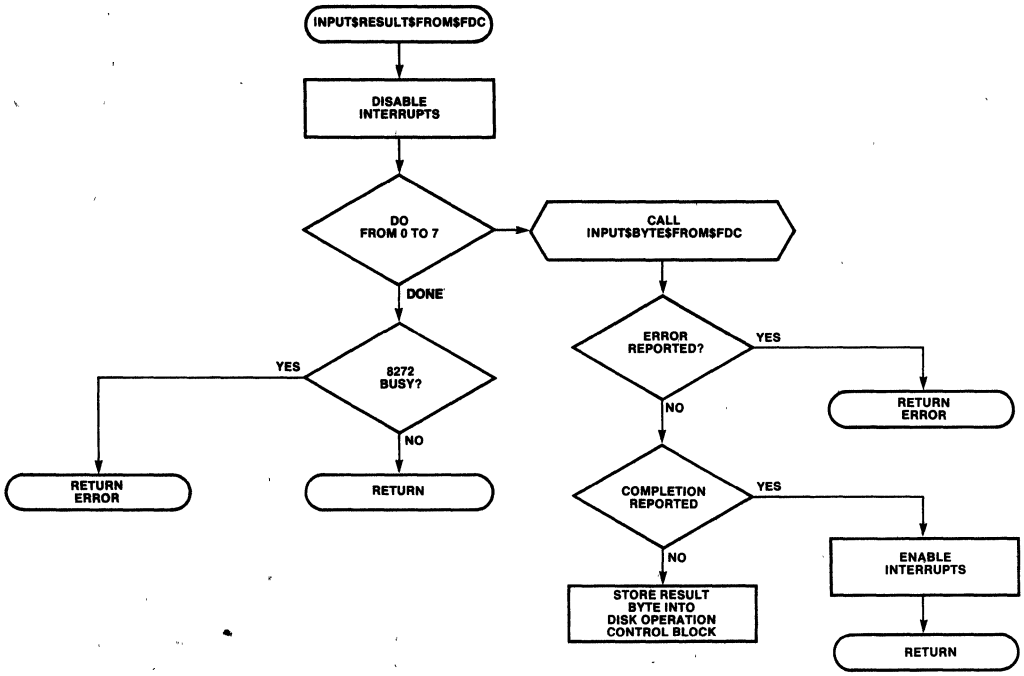

OUTPUT$BYTE$TO$FDC
CALL FDC$READY$FOR$COMMAND
FDC READY FOR COMMAND
NO
RETURN ERROR
YES
OUTPUT DATA BYTE TO 8272 DATA PORT
RETURN

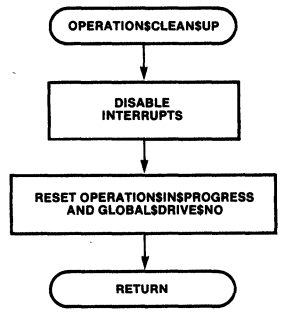

INPUT$BYTES$FROM$FDC
CALL FDC$READY$FOR$COMMAND
RESULT ERROR REPORTED
YES
RETURN ERROR
NO
RESULT PHASE COMPLETE
YES
RETURN COMPLETE
NO
INPUT DATA BYTE FROM 8272 DATA PORT
RETURN

AFN-01949A

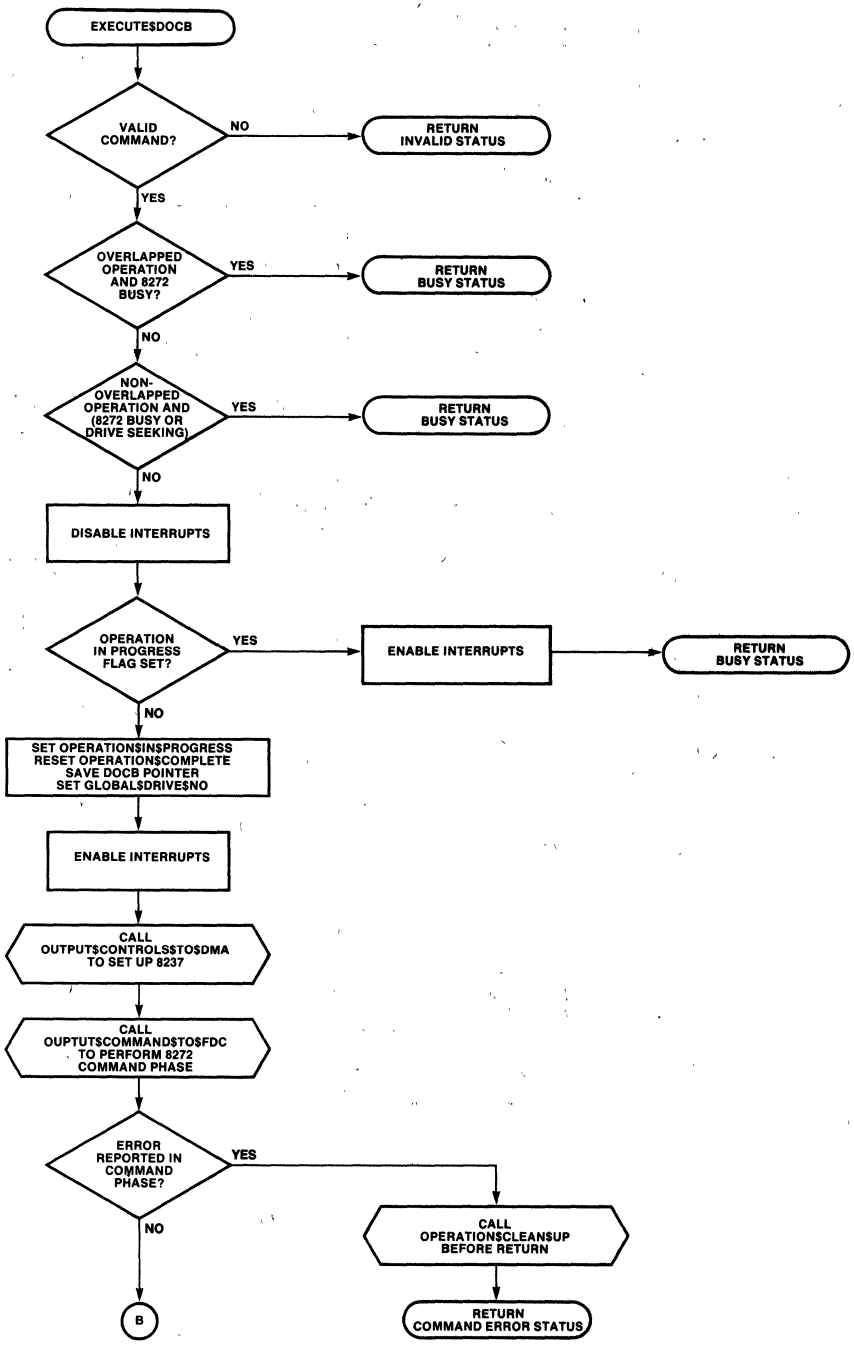

OUTPUT$CONTROLS$TO$DMA
SET DMA MODE
CLEAR FIRST/LAST
FLIP-FLOP
WRITE DMA ADDRESS
TO 8237 AND
EXTENDED ADDRESS
LATCH
WRITE DATA TRANSFER
BYTE COUNT
TO 8237
START DMA
CHANNEL
RETURN
OUTPUT$COMMAND$TO$FDC
DISABLE
INTERRUPTS
DO
0 TO COMMAND
LENGTH
DONE
ENABLE
INTERRUPTS
RETURN
OUTPUT A
COMMAND
BYTE TO THE
8272
ERROR
REPORTED
YES
ENABLE
INTERRUPTS
RETURN
ERROR

AFN-01949A

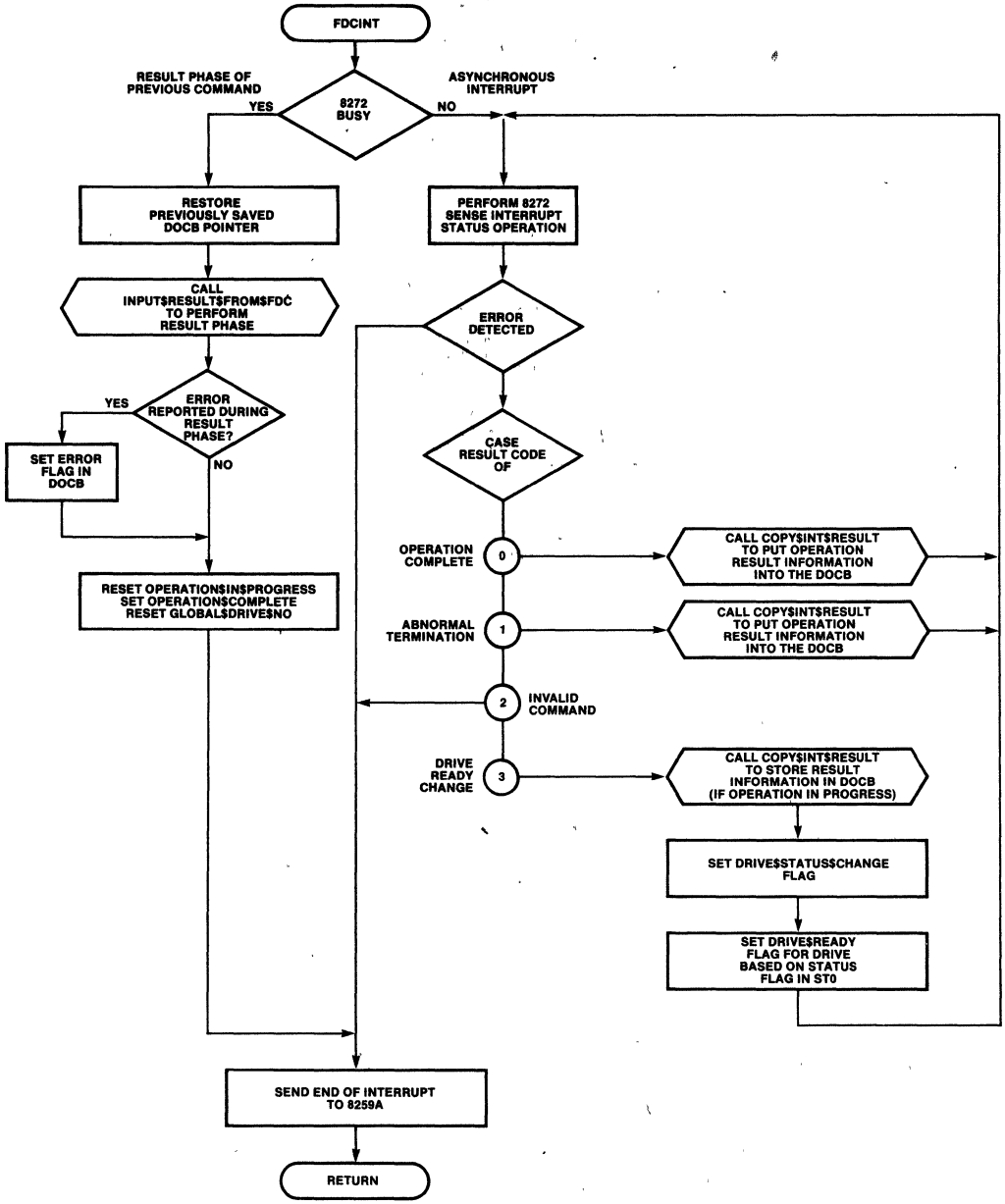

INPUT$RESULT$FROM$FDC
DISABLE INTERRUPTS
DO FROM 0 TO 7
CALL INPUT$BYTE$FROM$FDC
DONE
8272 BUSY?
YES
NO
RETURN ERROR
RETURN
ERROR REPORTED?
YES
NO
RETURN ERROR
COMPLETION REPORTED
YES
NO
ENABLE INTERRUPTS
STORE RESULT BYTE INTO DISK OPERATION CONTROL BLOCK
RETURN


OPERATION$CLEAN$UP
DISABLE INTERRUPTS
RESET OPERATIONINPROGRESS AND GLOBAL$DRIVE$NO
RETURN

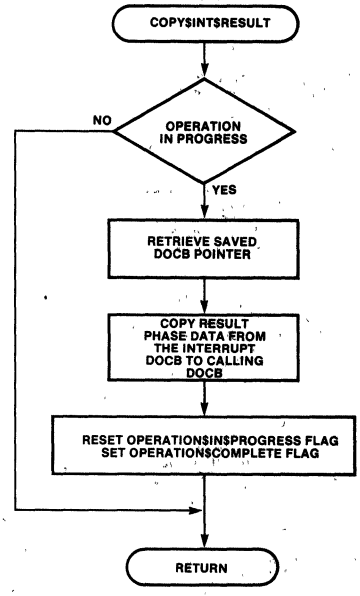
AFN-01949A

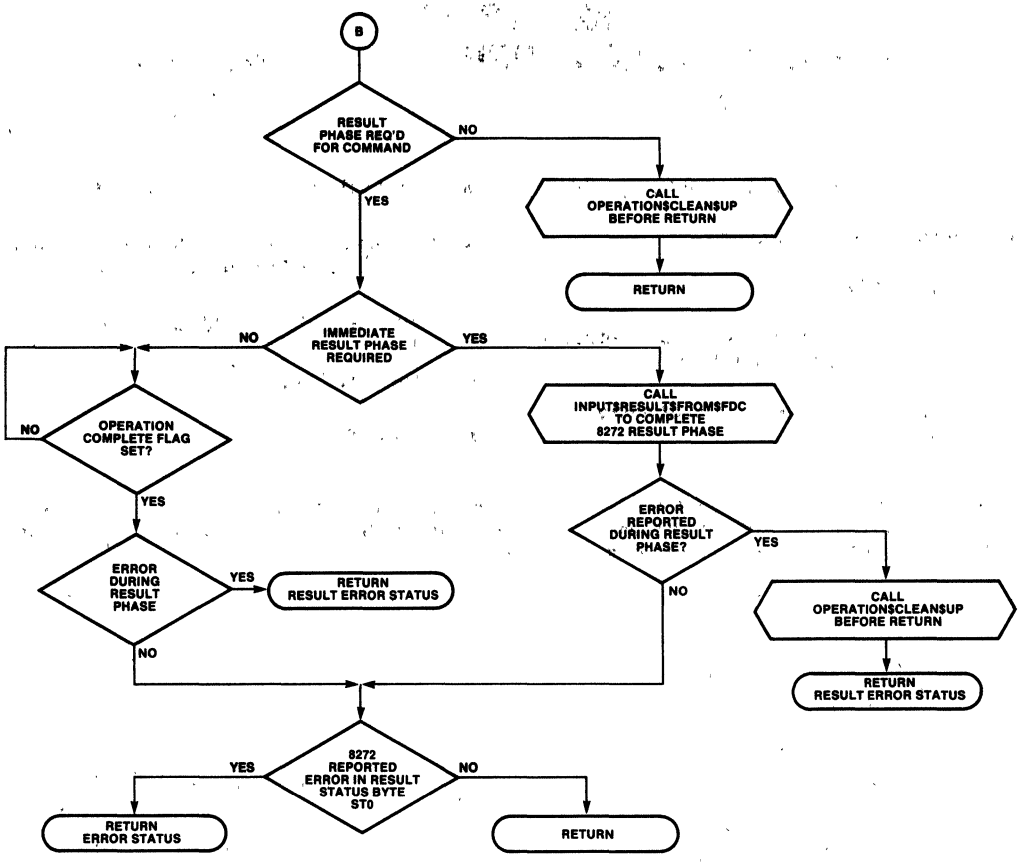

EXECUTE$DOCB
VALID COMMAND?
NO
RETURN INVALID STATUS
YES
OVERLAPPED OPERATION AND 8272 BUSY?
YES
RETURN BUSY STATUS
NO
NON-OVERLAPPED OPERATION AND (8272 BUSY OR DRIVE SEEKING)
YES
RETURN BUSY STATUS
NO
DISABLE INTERRUPTS
OPERATION IN PROGRESS FLAG SET?
YES
ENABLE INTERRUPTS
RETURN BUSY STATUS
NO
SET OPERATIONINPROGRESS
RESET OPERATION$COMPLETE
SAVE DOCB POINTER
SET GLOBAL$DRIVE$NO
ENABLE INTERRUPTS
CALL OUTPUT$CONTROLS$TO$DMA TO SET UP 8237
CALL OUPTUT$COMMAND$TO$FDC TO PERFORM 8272 COMMAND PHASE
ERROR REPORTED IN COMMAND PHASE?
YES
NO
CALL OPERATION$CLEAN$UP BEFORE RETURN
RETURN COMMAND ERROR STATUS
B

AFN-01949A


FDCINT
RESULT PHASE OF PREVIOUS COMMAND
YES
8272 BUSY
NO
ASYNCHRONOUS INTERRUPT
RESTORE PREVIOUSLY SAVED DOCB POINTER
CALL INPUT$RESULT$FROM$FDC TO PERFORM RESULT PHASE
YES
ERROR REPORTED DURING RESULT PHASE?
NO
SET ERROR FLAG IN DOCB
RESET OPERATIONINPROGRESS
SET OPERATION$COMPLETE
RESET GLOBAL$DRIVE$NO
PERFORM 8272 SENSE INTERRUPT STATUS OPERATION
ERROR DETECTED
CASE RESULT CODE OF
OPERATION COMPLETE
0
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
ABNORMAL TERMINATION
1
CALL COPYINTRESULT TO PUT OPERATION RESULT INFORMATION INTO THE DOCB
2
INVALID COMMAND
DRIVE READY CHANGE
3
CALL COPYINTRESULT TO STORE RESULT INFORMATION IN DOCB (IF OPERATION IN PROGRESS)
SET DRIVE$STATUS$CHANGE FLAG
SET DRIVE$READY FLAG FOR DRIVE BASED ON STATUS FLAG IN ST0
SEND END OF INTERRUPT TO 8259A
RETURN

AFN-01949A


COPYINTRESULT
OPERATION IN PROGRESS
NO
YES
RETRIEVE SAVED DOCB POINTER
COPY RESULT PHASE DATA FROM THE INTERRUPT DOCB TO CALLING DOCB
RESET OPERATIONINPROGRESS FLAG
SET OPERATION$COMPLETE FLAG
RETURN


B
RESULT PHASE REQ'D FOR COMMAND
NO
YES
CALL OPERATION$CLEANUP BEFORE RETURN
RETURN
IMMEDIATE RESULT PHASE REQUIRED
NO
YES
OPERATION COMPLETE FLAG SET?
NO
YES
CALL INPUT$RESULTS$FROM$FDC TO COMPLETE 8272 RESULT PHASE
ERROR REPORTED DURING RESULT PHASE?
YES
NO
ERROR DURING RESULT PHASE
YES
RETURN RESULT ERROR STATUS
NO
CALL OPERATION$CLEANUP BEFORE RETURN
RETURN RESULT ERROR STATUS
8272 REPORTED ERROR IN RESULT STATUS BYTE ST0
YES
NO
RETURN ERROR STATUS
RETURN

AFN-01949A

8271/8271-6
PROGRAMMABLE FLOPPY DISK CONTROLLER

- **IBM 3740 Soft Sectored Format Compatible**
- **Programmable Record Lengths**
- **Multi-Sector Capability**
- **Maintain Dual Drives with Minimum Software Overhead Expandable to 4 Drives**
- **Automatic Read/Write Head Positioning and Verification**
- **Internal CRC Generation and Checking**
- **Programmable Step Rate, Settle-Time, Head Load Time, Head Unload Index Count**
- **Fully MCS-80™ and MCS-85™ Compatible**
- **Single +5V Supply**
- **40-Pin Package**

The Intel® 8271 Programmable Floppy Disk Controller (FDC) is an LSI component designed to interface one to 4 floppy disk drives to an 8-bit microcomputer system. Its powerful control functions minimize both hardware and software overhead normally associated with floppy disk controllers.

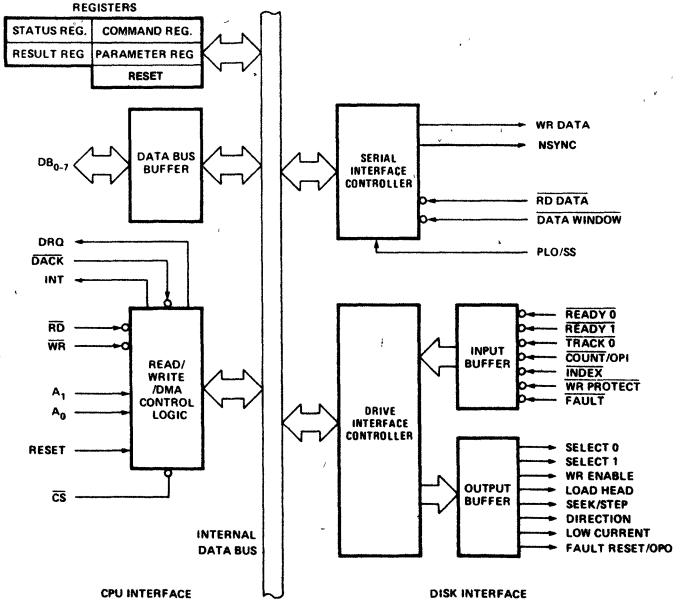


Figure 1. Block Diagram

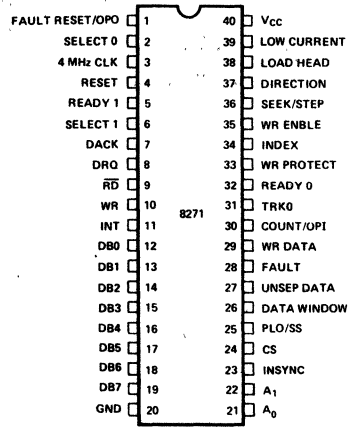


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	40		**+5V Supply.**
GND	20		**Ground.**
Clock	3	I	**Clock:** A square wave clock.
Reset	4	I	**Reset:** A high signal on the reset input forces the 8271 to an idle state. The 8271 remains idle until a command is issued by the CPU. The output signals of the drive interface are forced inactive (LOW). Reset must be active for 10 or more clock cycles.
$\overline{CS}$	24	I	**Chip Select:** The I/O Read and I/O Write inputs are enabled by the chip select signal.
DB_7-DB_0	19-12	I/O	**Data Bus:** The Data Bus lines are bidirectional, three-state lines (8080 data bus compatible).
$\overline{WR}$	10	I	**Write:** The Write signal is used to signal the control logic that a transfer of data from the data bus to the 8271 is required.
$\overline{RD}$	9	I	**Read:** The Read signal is used to signal the control logic that a transfer of data from the 8271 to the data bus is required.
INT	11	O	**Interrupt:** The interrupt signal indicates that the 8271 requires service.
A_1-A_0	22-21	I	**Address Line:** These two lines are CPU Interface Register select lines.
DRQ	8	O	**Data Request:** The DMA request signal is used to request a transfer of data between the 8271 and memory.
$\overline{DACK}$	7	I	**Data Acknowledge:** The DMA acknowledge signal notifies the 8271 that a DMA cycle has been granted. For non-DMA transfers, this signal should be driven in the manner of a "Chip Select."
Select 1– Select 0	6 2	O	**Selected Drive:** These lines are used to specify the selected drive. These lines are set by the command byte.

Symbol	Pin No.	Type	Name and Function
Fault Reset/ OPO	1	O	**Fault Reset:** The optional fault reset output line is used to reset an error condition which is latched by the drive. If this line is not used for a fault reset it can be used as an optional output line. This line is set with the write special register command.
Write Enable	35	O	**Write Enable:** This signal enables the drive write logic.
Seek/Step	36	O	**Seek/Step:** This multifunction line is used during drive seeks.
Direction	37	O	**Direction:** The direction line specifies the seek direction. A high level on this pin steps the R/W head toward the spindle (step-in), a low level steps the head away from the spindle (step-out).
Load Head	38	O	**Load Head:** The load head line causes the drive to load the Read/Write head against the diskette.
Low Current	39	O	**Low Current:** This line notifies the drive that track 43 or greater is selected.
$\overline{Ready\ 1}$, $\overline{Ready\ 0}$	5 32	I	**Ready 1:** These two lines indicate that the specified drive is ready.
$\overline{Fault}$	28	I	**Fault:** This line is used by the drive to specify a file unsafe condition.
$\overline{Count/OPI}$	30	I	**Count/OPI:** If the optional seek/direction/count seek mode is selected, the count pin receives pulses to step the R/W head to the desired track. Otherwise, this line can be used as an optional input.
$\overline{Write\ Protect}$	33	I	**Write Protect:** This signal specifies that the diskette inserted is write protected.
$\overline{TRK0}$	31	I	**Track Zero:** This signal indicates when the R/W head is positioned over track zero.
$\overline{Index}$	34	I	**Index:** The index signal gives an indication of the relative position of the diskette.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
PLO/SS	25	I	**Phase-Locked Oscillator/ Single Shot:** This pin is used to specify the type of data separator used.
Write Data	29	O	**Write Data:** Composite write data.
$\overline{\text{Unseparated Data}}$	27	I	**Unseparated Data:** This input is the unseparated data and clocks.
Data Window	26	I	**Data Window:** This is a data window established by a single-shot or phase-locked oscillator data separator.
INSYNC	23	O	**Input Synchronization:** This line is high when 8271 has attained input data synchronization, by detecting 2 bytes of zeros followed by an expected Address Mark. It will stay high until the end of the ID or data field.

FUNCTIONAL DESCRIPTION

General

The 8271 Floppy Disk Controller (FDC) interfaces either two single or one dual floppy drive to an eight bit microprocessor and is fully compatible with Intel's new high performance MCS-85 microcomputer system. With minimum external circuitry, this innovative controller supports most standard, commonly-available flexible disk drives including the mini-floppy.

The 8271 FDC supports a comprehensive soft sectored format which is IBM 3740 compatible and includes provision for the designating and handling of bad tracks. It is a high level controller that relieves the CPU (and user) of many of the control tasks associated with implementing a floppy disk interface. The FDC supports a variety of high level instructions which allow the user to store and retrieve data on a floppy disk without dealing with the low level details of disk operation.

In addition to the standard read/write commands, a scan command is supported. The scan command allows the user program to specify a data pattern and instructs the FDC to search for that pattern on a track. Any application that is required to search the disk for information (such as point of sale price lookup, disk directory search, etc.), may use the scan command to reduce the CPU overhead. Once the scan operation is initiated, no CPU intervention is required.

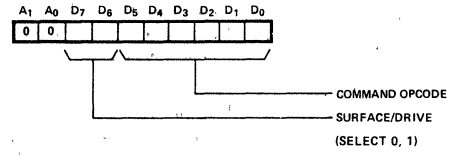
CPU Interface Description

This interface minimizes CPU involvement by supporting a set of high level commands and both DMA and non-DMA type data transfers and by providing hierarchical status information regarding the result of command execution.

The CPU utilizes the control interface (see the Block diagram) to specify the FDC commands and to determine the result of an executed command. This interface is supported by five Registers which are addressed by the CPU via the A_1, A_0, $\overline{RD}$ and $\overline{WR}$ signals. If an 8080 based system is used, the $\overline{RD}$ and $\overline{WR}$ signals can be driven by the 8228's $\overline{I/OR}$ and $\overline{I/OW}$ signals. The registers are defined as follows:

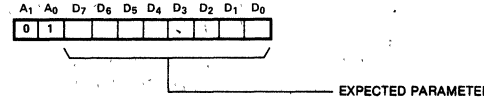
Command Register

The CPU loads an appropriate command into the Command Register which has the following format:



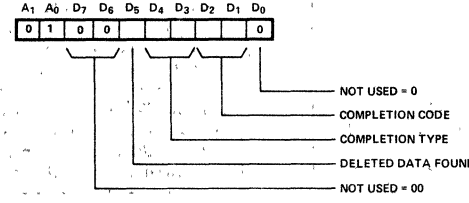
Parameter Register

Accepts parameters of commands that require further description; up to five parameters may be required, example:



Result Register

The Result Register is used to supply the outcome of FDC command execution (such as a good/bad completion) to the CPU. The standard Result byte format is:



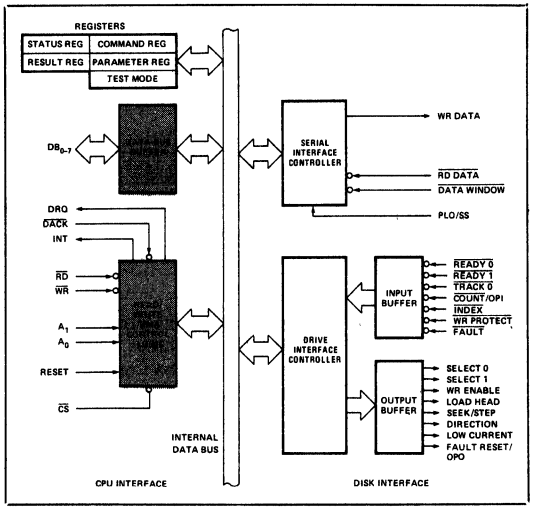
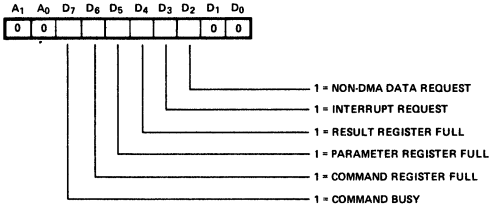


Figure 3. 8271 Block Diagram Showing CPU Interface Functions

Status Register

Reflects the state of the FDC.



Reset Register

Allows the 8271 to be reset by the program. Reset must be active for 11 or more chip clocks.

INT (Interrupt Line)

Another element of the control interface is the Interrupt line (INT). This line is used to signal the CPU that an FDC operation has been completed. It remains active until the result register is read.

DMA Operation

The 8271 can transfer data in either DMA or non DMA mode. The data transfer rate of a floppy disk drive is high enough (one byte every 32 µsec) to justify DMA transfer. In DMA mode the elements of the DMA interface are:

DRQ: DMA Request:
The DMA request signal is used to request a transfer of data between the 8271 and memory.

$\overline{\text{DACK}}$: DMA Acknowledge:
The DMA acknowledge signal notifies the 8271 that a DMA cycle has been granted.

$\overline{\text{RD}}$, $\overline{\text{WR}}$: Read, Write
The read and write signals are used to specify the direction of the data transfer.

DMA transfers require the use of a DMA controller such as the Intel®8257. The function of the DMA controller is to provide sequential addresses and timing for the transfer at a starting address determined by the CPU. Counting of data block lengths is performed by the FDC.

To request a DMA transfer, the FDC raises DRQ. DACK and RD enable DMA data onto the bus (independently of CHIP SELECT). DACK and WR transfer DMA data to the FDC. If a data transfer request (read or write) is not serviced within 31 µsec, the command is cancelled, a late DMA status is set, and an interrupt is generated. In DMA mode, an interrupt is generated at the completion of the data block transfer.

When configured to transfer data in non-DMA mode, the CPU must pass data to the FDC in response to the non-DMA data requests indicated by the status word. The data is passed to and from the chip by asserting the DACK and the RD or WR signals. Chip select should be inactive (HIGH).

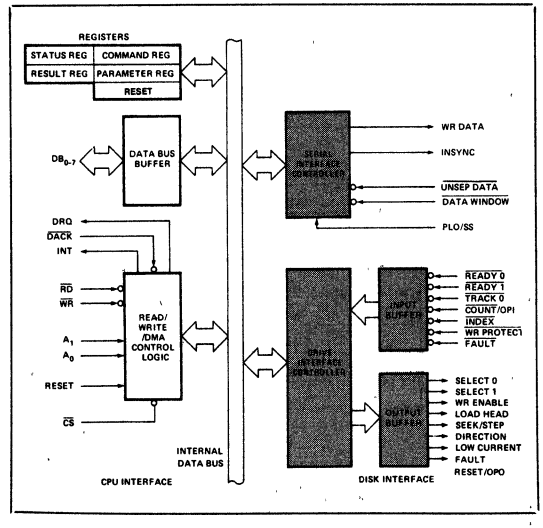


Figure 4. 8271 Block Diagram Showing Disk Interface Functions

Disk Drive Interface

The 8271 disk drive interface supports the high level command structure described in the Command Description section. The 8271 maintains the location of bad tracks and the current track location for two drives. However, with minor software support, this interface can support four drives by expanding the two drive select lines (select 0, select 1) with the addition of minimal support hardware.

The FDC Disk Drive Interface has the following major functions.

READ FUNCTIONS

Utilize the user supplied data window to obtain the clock and data patterns from the unseparated read data.

Establish byte synchronization.

Compute and verify the ID and data field CRCs.

WRITE FUNCTIONS

Encode composite write data.

Compute the ID and data field CRCs and append them to their respective fields.

CONTROL FUNCTIONS

Generate the programmed step rate, head load time, head settling time, head unload delay, and monitor drive functions.

Data Separation

The 8271 needs only a data window to separate the data from the composite read data as well as to detect missing clocks in the Address Marks.

The window generation logic may be implemented using either a single-shot separator or a phase-locked oscillator.

Single-Shot Separator

The single-shot separator approach is the lowest cost solution.

The FDC samples the value of Data Window on the leading edge of Unseparated Data and determines whether the delay from the previous pulse was a half or full bit-cell (high input = full bit-cell, low input = half bit-cell). PLO/SS should be tied to Ground.

Insync Pin

This pin gives an indication of whether the 8271 is synchronized with the serial data stream during read operations. This pin can be used with a phase-locked oscillator for soft and hard locking.

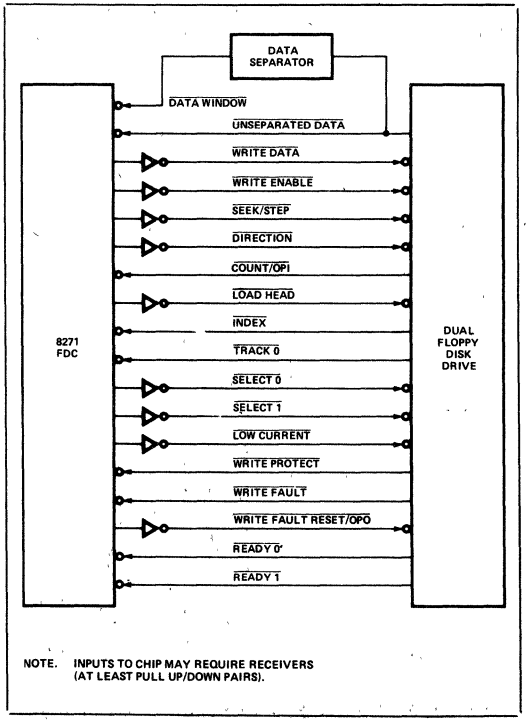
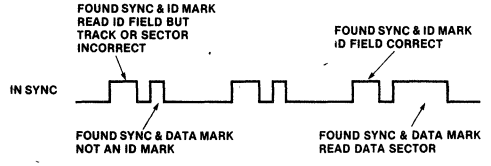


Figure 5. 8271 Disk Drive Interface



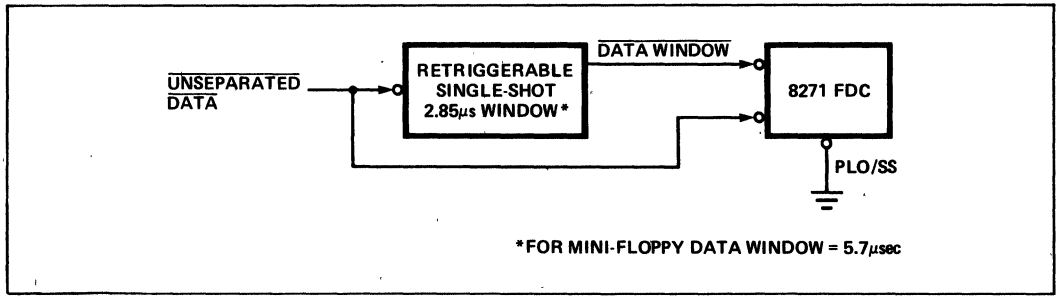


Figure 6. Single-Shot Data Separator Block Diagram

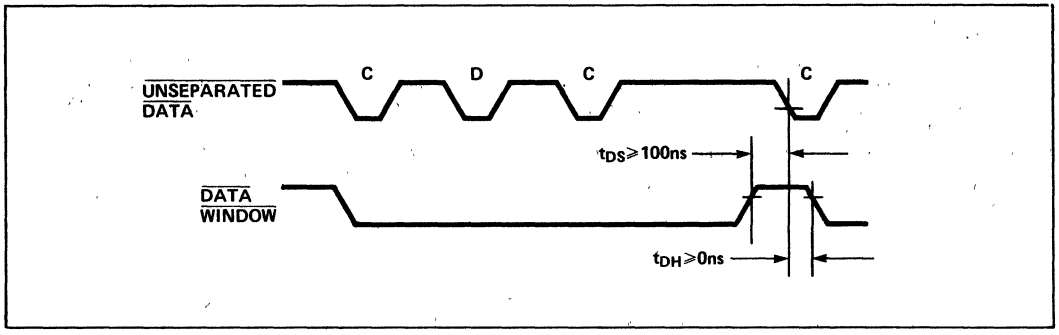


Figure 7. Single-Shot Data Window Timing

Phase-Locked Oscillator Separator

The FDC samples the value of Data Window on the leading edge of Unseparated Data and determines whether the pulse represents a Clock or Data Pulse.

PLO/SS should be tied to V_{CC} (+5V).

Insync may be used to provide soft and hard locking control for the phase-locked oscillator.

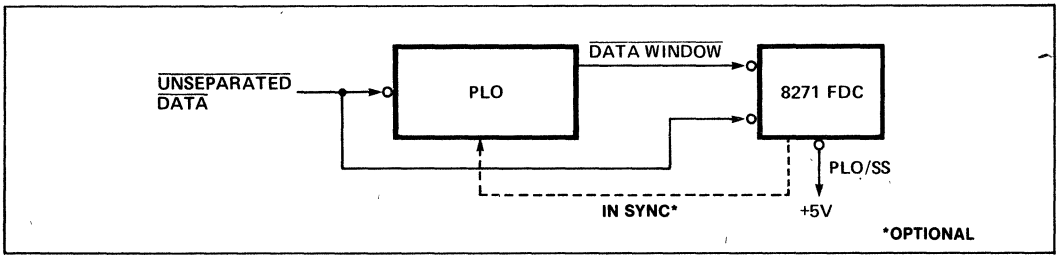
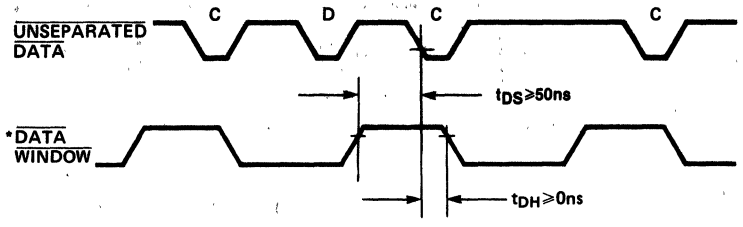


Figure 8. PLO Data Separator Block Diagram



*DATA WINDOW MAY BE 180° OUT OF PHASE IN PLO DATA SEPARATION MODE.

Figure 9. PLO Data Window Timing

Disk Drive Control Interface

The disk drive control interface performs the high level and programmable flexible disk drive operations. It custom tailors many varied drive performance parameters such as the step rate, settling time, head load time, and head unload index count. The following is the description of the control interface.

Write Enable

The Write Enable controls the read and write functions of a flexible disk drive. When Write Enable is a logical one, it enables the drive write electronics to pass current through the Read/Write head. When Write Enable is a logical zero, the drive Write circuitry is disabled and the Read/Write head detects the magnetic flux transitions recorded on a diskette. The write current turn-on is as follows.

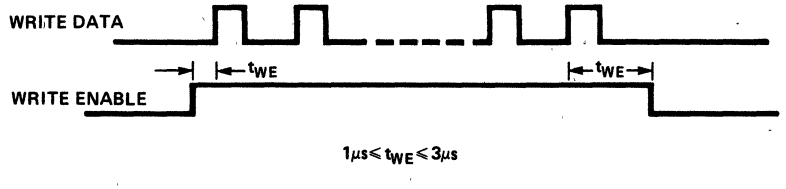


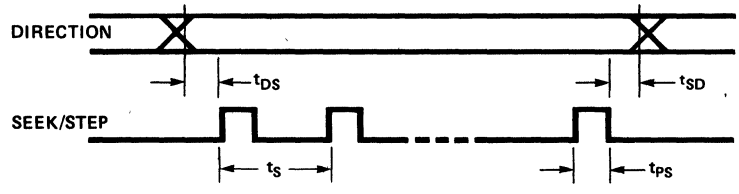
Figure 10. Write Enable Timing

Seek Control

Seek Control is accomplished by Seek/Step, Direction, and Count pins and can be implemented two ways to provide maximum flexibility in the subsystem design. One instance is when the programmed step rate is not equal to zero. In this case, the 8271 uses the Seek/Step and Direction pins (the Seek/Step pin becomes a Step pin). Programmable Step timing parameters are shown.

Another instance is when the programmable step rate is equal to zero, in which case the 8271 holds the seek line high until the appropriate number of user-supplied step pulses have been counted on the count input pin.

The Direction pin is a control level indicating the direction in which the R/W head is stepped. A logic high level on this line moves the head toward the spindle (step-in). A logic low level moves the head away from the spindle (step-out).

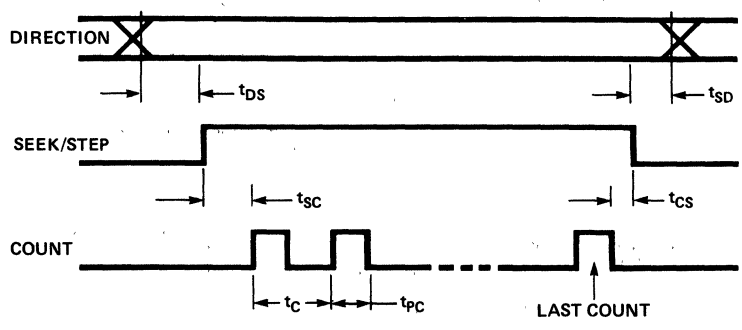


$t_{PS}=t_{DS}=t_{SD}=10\mu s$

STANDARD: $1ms \leq t_S \leq 255ms$

MINI-FLOPPY: $2ms \leq t_s \leq 510ms$

Figure 11. Seek Timing



$t_{DS}=t_{SD}=t_{CS}=10\mu s$

$t_{SC} \geq 1\mu s$

$t_{PC} \geq 20\mu s$

$t_C \geq 1ms$

Figure 12. Seek/Step/Count Timing

Head Seek Settling Time

The 8271 allows the head settling time to be programmed from 0 to 255ms, in increments of 1ms.

The head settling time is defined as the interval of time from completion of the last step to the time when reading or writing on the diskette is possible (R/W Enable). The R/W head is assumed loaded.

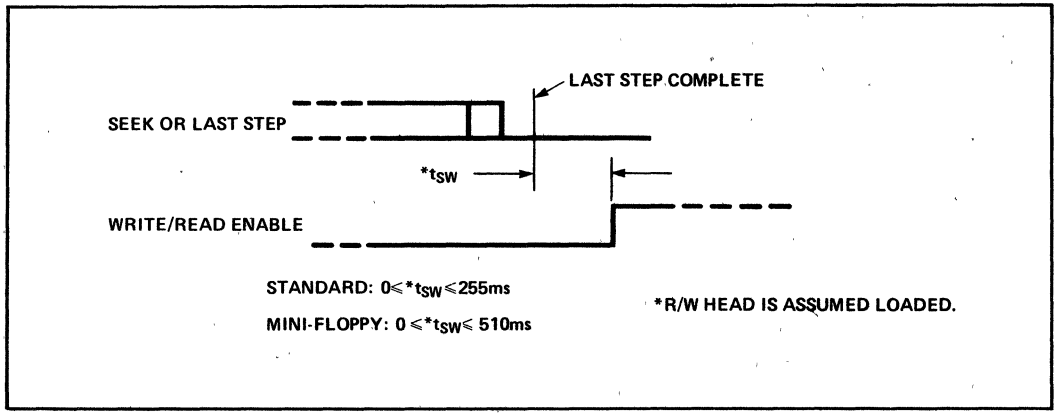


Figure 13. Head Load Settling Timing

Load Head

When active, load head output pin causes the drive's read/write head to be loaded on the diskette. When the head is initially loaded, there is a programmed delay (0 to 60ms in 4ms increments) prior to any read or write operation. Provision is also made to unload the head following an operation within a programmed number of diskette revolutions.

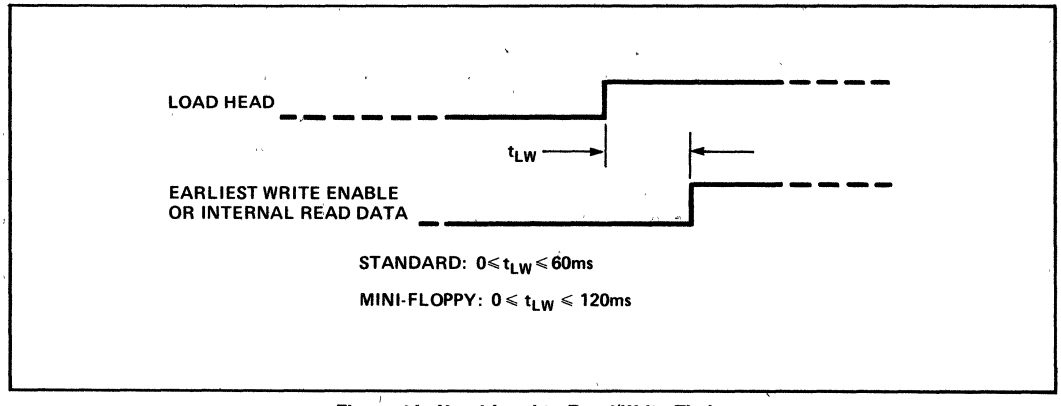


Figure 14. Head Load to Read/Write Timing

Index

The Index input is used to determine "Sector not found" status and to initiate format track/read ID commands and head unload Index and Count operations.

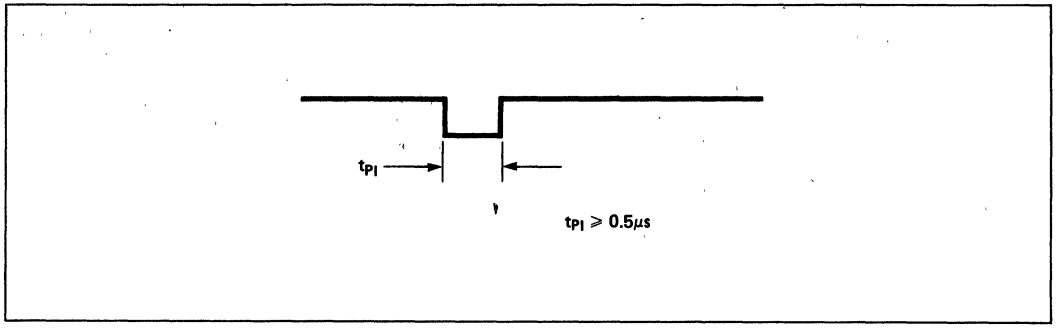


Figure 15. Index Timing

Track 0

This input pin indicates that the diskette is at track 0. During any seek operation, the stepping out of the actuator ceases when the track 0 pin becomes active.

Select 1, 0

Only one drive may be selected at a time. The Input/Output pins that must be externally qualified with Select 0 and Select 1 are:

Unseparated Data
Data Window
Write Enable
Seek/Step
Count/Optional Input
Load Head
Track 0
Low Current
Write Protect
Write Fault
Fault Reset/Optional Output
Index

When a new set of select bits is specified by a new command or the FDC finishes the index count before head unload, the following pins will be set to the 0 state:

Write Enable (35)
Seek/Step (36)
Direction (37)
Load Head (38)
Low Head Current (39)

The select pins will be set to the state specified by the command or both are set to zero following the index count before head unload.

Low Current

This output pin is active whenever the physical track location of the selected drive is greater than 43. Generally this signal is used to enable compensation for the lower velocities encountered while recording on the inner tracks.

Write Protect

The 8271 will not write to a disk when this input pin is active and will interrupt the CPU if a Write attempt is made. Operations which check Write Protect are aborted if the Write Protect line is active.

This signal normally originates from a sensor which detects the presence or absence of the Write Protect hole in the diskette jacket.

Write Fault and Write Fault Reset

The Write Fault input is normally latched by the drive and indicates any condition which could endanger data integrity. The 8271 interrupts the CPU anytime Write Fault is detected during an operation and immediately resets the Write Enable, Seek/Step, Direction, and Low Current signals. The write fault condition can be cleared by using the write fault reset pin. If the drive being used does not support write fault, then this pin should be connected to V_{CC} through a pull-up resistor.

Ready 1, 0

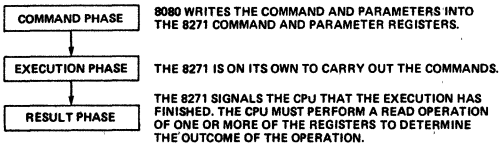
These two pins indicate the functional status of the disk drives. Whenever an operation is attempted on a drive which is not ready, an interrupt is generated. The interface continually monitors this input during an operation and if a Not Ready condition occurs, immediately terminates the operation. Note that the 8271 latches the Not Ready condition and it can only be reset by the execution of a Read Drive Status command. For drives that do not support a ready signal, either one can be derived with a one shot and the index pulse, or the ready inputs can be grounded and Ready determined through some software means.

PRINCIPLES OF OPERATION

As an 8080 peripheral device, the 8271 accepts commands from the CPU, executes them and provides a RESULT back to the 8080 CPU at the end of command execution. The communication with the CPU is established by the activation of $\overline{CS}$ and $\overline{RD}$ or $\overline{WR}$. The A_1, A_0 inputs select the appropriate registers on the chip:

$\overline{DACK}$	$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	Operation
1	0	0	0	0	1	Read Status
1	0	0	0	1	0	Write Command
1	0	0	1	0	1	Read Result
1	0	0	1	1	0	Write Parameter
1	0	1	0	1	0	Write Reset Reg.
0	1	X	X	1	0	Write Data
0	1	X	X	0	1	Read Data
0	0	X	X	X	X	Not Allowed

The FDC operation is composed of the following sequence of events.



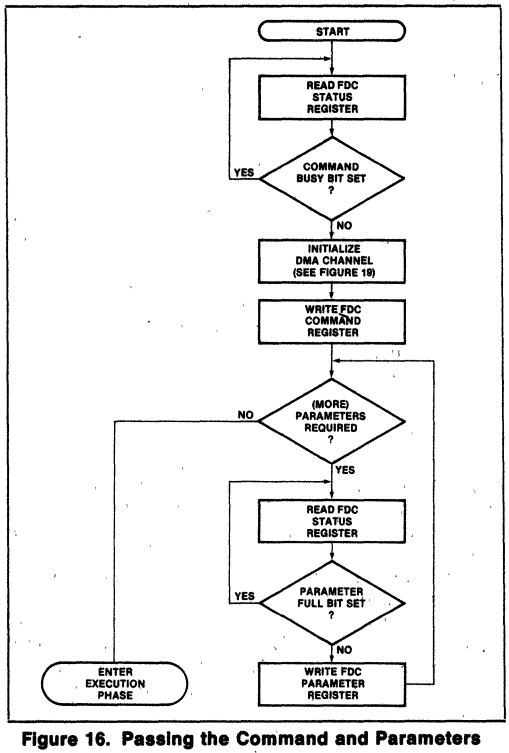


Figure 16. Passing the Command and Parameters to the 8271

The Command Phase

The software writes a command to the command register. As a function of the command issued, from zero to five parameters are written to the parameter register. Refer to diagram showing a flow chart of the command phase. Note that the flow chart shows that a command may not be issued if the FDC status register indicates that the device is busy. Issuing a command while another command is in progress is illegal. The flow chart also shows a parameter buffer full check. The FDC status indicates the state of the parameter buffer. If a parameter is issued while the parameter buffer is full, the previous parameter is over written and lost.

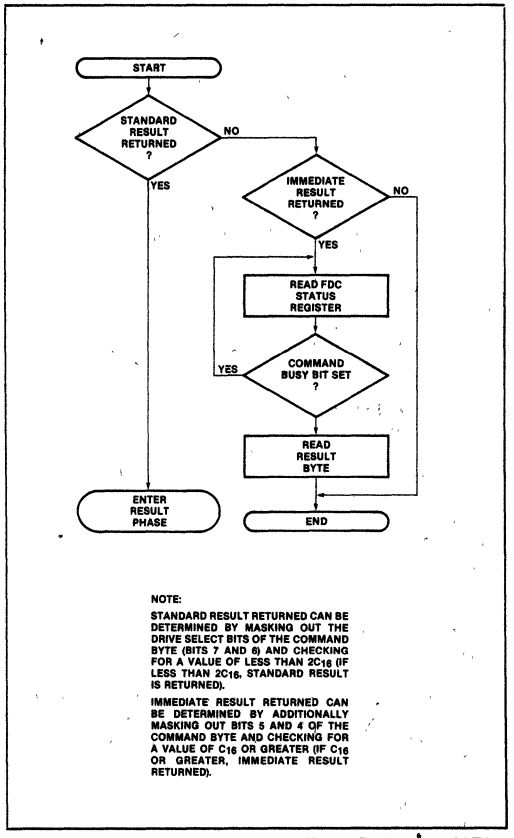


Figure 17. Checking for Result Type Following 8271 Command and Parameters

The Execution Phase

During the execution phase the operation specified during the command phase is performed. During this phase, there is no CPU involvement if the system utilizes DMA for the data transfers. The execution phase of each command is discussed within the detailed command descriptions. The following table summarizes many of the basic execution phase characteristics.

EXECUTION PHASE BASIC CHARACTERISTICS

The following table summarizes the various commands with corresponding execution phase characteristics.

Table 2. Execution Phase Basic Characteristics

COMMANDS	1 Deleted Data	2 Head	3 Ready	4 Write/ Protect	5 Seek	6 Seek Check	7 Result	8 Completion Interrupt
SCAN DATA	SKIP	LOAD	√	x	YES	YES	YES	YES
SCAN DATA AND DEL DATA	XFER	LOAD	√	x	YES	YES	YES	YES
WRITE DATA	x	LOAD	√	√	YES	YES	YES	YES
WRITE DEL DATA	x	LOAD	√	√	YES	YES	YES	YES
READ DATA	SKIP	LOAD	√	x	YES	YES	YES	YES
READ DATA AND DEL DATA	XFER	LOAD	√	x	YES	YES	YES	YES
READ ID	x	LOAD	√	x	YES	NO	YES	YES
VERIFY DATA AND DEL DATA	XFER	LOAD	√	x	YES	YES	YES	YES
FORMAT TRACK	x	LOAD	√	√	YES	NO	YES	YES
SEEK	x	LOAD	y	x	YES	NO	YES	YES
READ DRIVE STATUS	x	–	x	x	NO	NO	NOTE 5	NO
SPECIFY	x	–	x	x	NO	NO	NO	NO
RESET	x	UNLOAD	x	x	NO	NO	NO	NO
R SP REGISTERS	x	–	x	x	NO	NO	NOTE 6	NO
W SP REGISTERS	x	–	x	x	NO	NO	NO	NO

Note: 1. "x" → DON'T CARE 2. "√" → check 3. "–" → No change 4. "y" → Check at end of operation 5. See "READ DRIVE STATUS" command. 6. See "READ SPECIAL REGISTER" command.

Explanation of the execution phase characteristics table.

1. Deleted Data Processing

   If deleted data is encountered during an operation that is marked skip in the table, the deleted data record is not transferred into memory, but the record is counted. For example, if the command and parameters specify a read of five records and one of the records was written with a deleted data mark, four records are transferred to memory. The deleted data flag is set in the result byte. However, if the operation is marked transfer, all data is transferred to memory regardless of the type of data mark.

2. Head

   The Head column in the table specifies whether the Read/Write head will be loaded or not. If the table specifies load, the head is loaded after it is positioned over the track. The head loaded by a command remains loaded until the user specified number of index pulses have occurred.

3. Ready

   The Ready column indicates if the ready line (Ready 1, Ready 0) associated with the selected drive is checked. A not ready state is latched by the 8271 until the user executes a read status command.

4. Write Protect

   The operations that are marked check Write Protect are immediately aborted if Write Protect line is active at the beginning of an operation.

5. Seek

   Many of the 8271 commands cause a seek to the desired track. A current track register is maintained for each drive or surface.

6. Seek Check

   Operations that perform Seek Check verify that selected data in the ID field is correct before the 8271 accesses the data field.

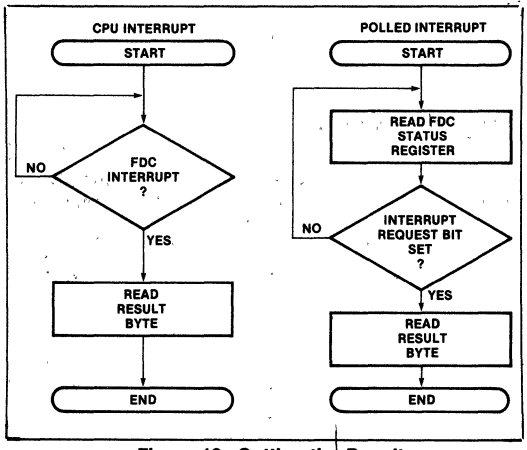


Figure 18. Getting the Result

The Result Phase

During the Result Phase, the FDC notifies the CPU of the outcome of the command execution. This phase may be initiated by:

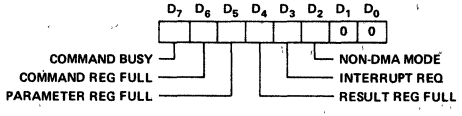
1. The successful completion of an operation.
2. An error detected during an operation.

PROGRAMMING

A_1	A_0	CS RD	CS WR
0	0	Status Reg	Command Reg
0	1	Result Reg	Parameter Reg
1	0	—	Reset Reg
1	1	—	—

STATUS REGISTER

FDC Status



Bit 7: Command Busy

The command busy bit is set on writing to the command register. Whenever the FDC is busy processing a command, the command busy bit is set to a one. This bit is set to zero after the command is completed.

Bit 6: Command Full

The command full bit is set on writing to the command buffer and cleared when the FDC begins processing the command.

Bit 5: Parameter Full

This bit indicates the state of the parameter buffer. This bit is set when a parameter is written to the FDC and reset after the FDC has accepted the parameter.

Bit 4: Result Full

This bit indicates the state of the result buffer. It is valid only after Command Busy bit is low. This bit is set when the FDC finishes a command and is reset after the result byte is read by the CPU. The data in the result buffer is valid only after the FDC has completed a command. Reading the result buffer while a command is in progress yields no useful information.

Bit 3: Interrupt Request

This bit reflects the state of the FDC INT pin. It is set when FDC requests attention as a result of the completion of an operation or failure to complete an intended operation. This bit is cleared by reading the result register.

Bit 2: Non-DMA Data Request

When the FDC is utilized without a DMA controller, this bit is used to indicate FDC data requests. Note that in the non-DMA mode, an interrupt is generated (interrupt request bit is set) with each data byte written to or read from the diskette.

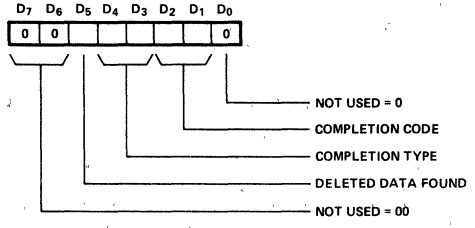
Bits 1 and 0:

Not used (zero returned).

After reading the Status Register, the CPU then reads the Result Register for more information.

THE RESULT REGISTER

This byte format facilitates the use of an address table to look up error routines and messages. The standard result byte format is:



Bits 7 and 6:

Not used (zero returned).

Bit 5:

Deleted Data Found: This bit is set when deleted data is encountered during a transaction.

Bits 4 and 3: Completion Type

The completion type field provides general information regarding the outcome of an operation.

The completion type field provides general information regarding the outcome of an operation.

Completion Type	Event
00	Good Completion — No Error
01	System Error — recoverable errors;
10	operator intervention probably required for recovery.
11	Command/Drive Error — either a program error or drive hardware failure.

Bits 2 and 1: Completion Code

The completion code field provides more detailed information about the completion type (See Table).

Completion Type	Completion Code	Event
00	00	Good Completion/ Scan Not Met
00	01	Scan Met Equal
00	10	Scan Met Not Equal
00	11	—
01	00	Clock Error
01	01	Late DMA
01	10	ID CRC Error
01	11	Data CRC Error
10	00	Drive Not Ready
10	01	Write Protect
10	10	Track 0 Not Found
10	11	Write Fault
11	00	Sector Not Found
11	01	—
11	10	—
11	11	—

It is important to note the hierarchical structure of the result byte. In very simple systems where only a GO-NO GO result is required, the user may simply branch on a zero result (a zero result is a good completion). The next level of complexity is at the completion type interface. The completion type supplies enough information so that the software may distinguish between fatal and non-fatal errors. If a completion type 01 occurs, ten retries should be performed before the error is considered unrecoverable.

The Completion Type/Completion Code interface supplies the greatest detail about each type of completion. This interface is used when detailed information about the transaction completion is required.

Bit 0:

Not used (zero returned).

Table 3. Completion Code Interpretation

Definition	Interpretation
Successful Completion/ Scan Not Met	The diskette operation specified was completed without error If scan operation was specified, the pattern scanned was not found on the track addressed.
Scan Met Equal	The data pattern specified with the scan command was found on the track addressed with the specified comparison, and the equality was met.
Scan Met Not Equal	The data pattern specified with the scan command was found with the specified comparison on the track addressed, but the equality was not met.
Clock Error	During a diskette read operation, a clock bit was missing (dropped). Note that this function is disabled when reading any of the ID address marks (which contain missing clock pulses). If this error occurs, the operation is terminated immediately and an interrupt is generated.
Late DMA	During either a diskette read or write operation, the data channel did not respond within the allotted time interval to prevent data from being overwritten or lost This error immediately terminates the operation and generates an interrupt
ID Field CRC Error	The CRC word (two bytes) derived from the data read in an ID field did not match the CRC word written in the ID field when the track was formatted If this error occurs, the associated diskette operation is prevented and no data is transferred.
Data Field CRC Error	During a diskette read operation, the CRC word derived from the data field read did not match the data field CRC word previously written If this error occurs, the data read from the sector should be considered invalid
Drive Not Ready	The drive addressed was not ready. This indication is caused by any of the following conditions: 1. Drive not powered up 2. Diskette not loaded 3. Non-existent drive addressed 4 Drive went not ready during an operation Note that this completion code is cleared only through an FDC read drive status command.
Write Protect	A diskette write operation was specified on a write protected diskette The intended write operation is prevented and no data is written on the diskette.
Track 00 Not Found	During a seek to track 00 operation, the drive failed to provide a track 00 indication after being stepped 255 times.
Write Fault	This error is dependent on the drive supported and indicates that the fault input to the FDC has been activated by the drive
Sector Not Found	Either the sector addressed could not be found within one complete revolution of the diskette (two index marks encountered) or the track address specified did not match the track address contained in the ID field Note that when the track address specified and the track address read do not match, the FDC automatically increments its track address register (stepping the drive to the next track) and again compares the track addresses. If the track addresses still do not match, the track address register is incremented a second time and another comparison is made before the sector not found completion code is set.

INITIALIZATION

Reset Command

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
PAR:	1	0	0	0	0	0	0	0	0	1
PAR:	1	0	0	0	0	0	0	0	0	0

Function: The Reset command emulates the action of the reset pin. It is issued by outputting a one followed by a zero to the Reset register.

1. The drive control signals are forced low.
2. An in-progress command is aborted
3. The FDC status register flags are cleared.
4. The FDC enters an idle state until the next command is issued.

Reset must be active for 10 or more clock cycles.

SPECIFY COMMAND

Many of the interface characteristics of the FDC are specified by the systems software. Prior to initiating any drive operation command, the software must execute the three specify commands. There are two types of specify commands selectable by the first parameter issued.

First Parameter	Specify Type
$0D_H$	Initialization
10_H	Load bad Tracks Surface '0'
18_H	Load bad Tracks Surface '1'

The Specify command is used prior to performing any diskette operation (including formatting of a diskette) to define the drive's inherent operating characteristics and also is used following a formatting operation or installation of another diskette to define the locations of bad tracks. Since the Specify command only loads internal registers within the 8271 and does not involve an actual diskette operation, command processing is limited to only Command Phase. Note that once the operating characteristics and bad tracks have been specified for a given drive and diskette, redefining these values need only be done if a diskette with unique bad tracks is to be used or if the system is powered down.

Initialization:

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
* CMD:	0	0	0	0	1	1	0	1	0	1
PAR:	0	1	0	0	0	0	1	1	0	1
PAR:	0	1	STEP RATE*							
PAR	0	1	HEAD SETTLING TIME*							
PAR:	0	1	INDEX CNT BEFORE HEAD UNLOAD*				HEAD LOAD TIME*			

*Note: Mini-floppy parameters are doubled.

Parameter 0 — $0D_H$ = Select Specify Initialization.
Parameter 1 — D_7-D_0 = Step Rate (0-255ms in 1ms steps).
Parameter 2 — D_7-D_0 = Head Settling Time (0-255ms in 1 ms steps). {0 – 510ms in 2ms steps} () = standard, {} = mini
Parameter 3 — D_7-D_4 = Index Count — Specifies the number of Revolutions (0-14) which are to occur before the FDC automatically unloads the R/W head. If 15 is specified, the head remains loaded.
D_3-D_0 = Head Load Time (0-60ms in steps of 4ms). {0 – 120ms in 8ms steps} () = standard, {} = mini

Load Bad Tracks

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	0	1	1	0	1	0	1
PAR:	0	1	0	0	0	1	1/0	0	0	0
PAR:	0	1	BAD TRACK NO 1							
PAR	0	1	BAD TRACK NO 2							
PAR	0	1	CURRENT TRACK							

Parameter 0: 10_H = Load Surface zero bad tracks
18_H = Load Surface one bad track

Parameter 1:
Bad track address number 1 (Physical Address).

It is recommended to program both bad tracks and current track to FF_H during initialization.

SEEK COMMAND

The seek command moves the head to the specified track without loading the head or verifying the track.

The seek operation uses the specified bad tracks to compute the physical track address. This feature insures that the seek operation positions the head over the correct track.

When a seek to track zero is specified, the FDC steps the head until the track 00 signal is detected.

If the track 00 signal is not detected within $(FF)_H$ steps, a track 0 not found error status is returned.

A seek to track zero is used to position the read/write head when the current head position is unknown (such as after a power up).

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	SEL 1	SEL 0	1	0	1	0	0	1
PAR	0	1	TRACK ADDRESS 0-255							

Seek operations are not verified. A subsequent read or write operation must be performed to determine if the correct track is located.

READ DRIVE STATUS COMMAND

This command is used to interrogate the drive status. Upon completion the result register will hold the final drive status.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	SEL 1	SEL 0	1	0	1	1	0	0

RESULT — EACH BIT INDICATES CURRENT STATE OF INPUT PINS

A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	—	RDY 1*	WR FAULT	INDEX	WR PROT	RDY 0*	TRACK 0	CNT / OPI

IF A DRIVE NOT READY RESULT IS RETURNED, THE READ STATUS MUST BE ISSUED TO CLEAR THE CONDITION

*Note the two ready bits are zero latching. Therefore, to clear the drive not ready condition, assuming the drive is ready, and to detect it via software, one must issue this command twice.

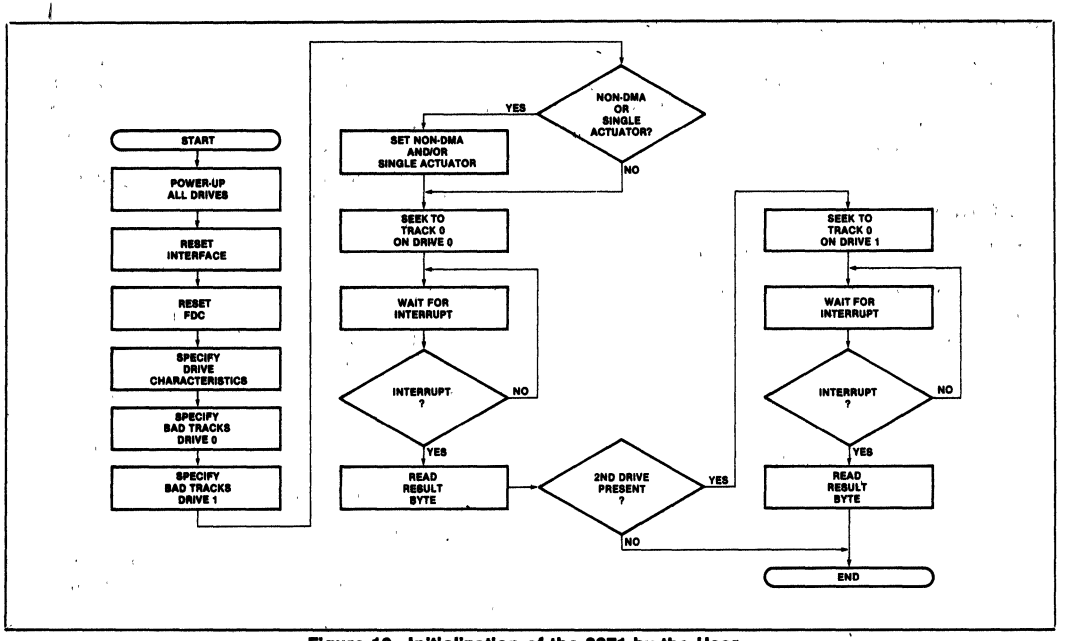


Figure 19. Initialization of the 8271 by the User

Read/Write Special Registers

This command is used to access special registers within the 8271.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	SEL 1	SEL 0	COMMAND OPCODE					
PAR:	0	1	REGISTER ADDRESS							

Command code:

$3D_H$ Read Special Register
$3A_H$ Write Special Register

For both commands, the first parameter is the register address; for Write commands a second parameter specifies data to be written. Only the Read Special Register command supplies a result.

Table 4. Special Registers

Description	Register Address in Hex	Comment
Scan Sector Number	06	See Scan Description
Scan MSB of Count	14	See Scan Description
Scan LSB of Count	13	See Scan Description
Surface 0 Current Track	12	
Surface 1 Current Track	1A	
Mode Register	17	See Mode Register Description
Drive Control Output Port	23	See Drive Output Port Description
Drive Control Input Port	22	See Drive Input Port Description
Surface 0 Bad Track 1	10	
Surface 0 Bad Track 2	11	
Surface 1 Bad Track 1	18	
Surface 1 Bad Track 2	19	

Mode Register Write Parameter Format

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	0	0	0		

D_0 = 0 DMA MODE, = 1 NON DMA
D_1 = 0 DOUBLE, = 1 SINGLE ACTUATOR

Bits 6 & 7

Must be one.

Bits 5-2

(Not used). Must be set to zero.

*Bit 1

Double/Single Actuator: Selects single or double actuator mode. If the single actuator mode is selected, the FDC assumes that the physical track location of both disks is always the same. This mode facilitates control of a drive which has a single actuator mechanism to move two heads.

*Bit 0

Data Transfer Mode: This bit selects the data transfer mode. If this bit is a zero, the FDC operates in the DMA mode (DMA Request/ACK). If this bit is a one, the FDC operates in non-DMA mode. When the FDC is operating in DMA mode, interrupts are generated at the completion of commands. If the non-DMA mode is selected, the FDC generates an interrupt for every data byte transferred.

*Bits 0 and 1 are initialized to zero.

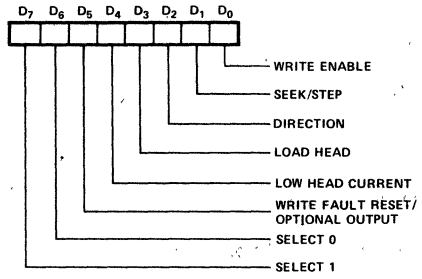
Non-DMA Transfers in DMA Mode

If the user desires, he may retain the use of interrupts generated upon command completions. This mode is accomplished by selecting the DMA capability, but using the DMA REQ/ACK pins as effective INT and CS signals, respectively.

Drive Control Input Port

Reading this port will give the CPU exactly the data that the FDC sees at the corresponding pins. Reading this port will update the drive not ready status, but will not clear the status. (See Read Drive Status Command for Bit locations.)

Drive Control Output Port Format



Each of these signals correspond to the chip pin of the same name. On standard-sized drives with write fault detection logic, bit 5 is set to generate the write fault reset signal. This signal is used to clear a write fault indication within the drive. On mini-sized drives, this bit can be used to turn on or off the drive motor prior to initiating a drive operation. A time delay after turn on may be necessary for the drive to come up to speed. The register must be read prior to writing the register in order to save the states of the remaining bits. When the register is subsequently written to modify bit 5, the remaining bits must be restored to their previous states.

IBM DISKETTE GENERAL FORMAT INFORMATION

The IBM Flexible Diskette used for data storage and retrieval is organized into concentric circular paths or TRACKS. There are 77 tracks on either one or both sides (surfaces) of the diskette. On double-sided diskettes, the corresponding top and bottom tracks are referred to as a CYLINDER. Each track is further divided into fixed length sections or SECTORS. The number of sectors per track — 26, 15 or 8 — is determined when a track is formatted and is dependent on the sector length — 128, 256 or 512 bytes respectively — specified.

All tracks on the diskette are referenced to a physical index mark (a small hole in the diskette). Each time the hole passes a photodetector cell (one revolution of the diskette), an Index pulse is generated to indicate the logical beginning of a track. This index pulse is used to initiate a track formatting operation.

Track Format

Each Diskette Surface is divided into 77 tracks with each track divided into fixed length sectors. A sector can hold a whole record or a part of a record. If the record is shorter than the sector length, the unused bytes are filled with binary zeros. If a record is longer than the sector length, the record is written over as many sectors as its length requires. The sector size that provides the most efficient use of diskette space can be chosen depending upon the record length required.

Tracks are numbered from 00 (outer-most) to 76 (inner-most) and are used as follows:

TRACK 00 reserved as System Label Track
TRACKS 01 through 74 used for data
TRACKS 75 and 76 used as alternates.

Each sector consists of an ID field (which holds a unique address for the sector) and a data field.

The ID field is seven bytes long and is written for each sector when the track is formatted. Each ID field consists of an ID field Address Mark, a Cylinder Number byte which identifies the track number, a Head Number byte which specifies the head used (top or bottom) to access the sector, a Record Number byte identifying the sector number (1 through 26 for 128 byte sectors), an N-byte specifying the byte length of the sector and two CRC (Cyclic Redundancy Check) bytes.

The Gaps separating the index mark and the ID and data fields are written on a track when it is formatted. These gaps provide both an interval for switching the drive electronics from reading or writing and compensation for rotational speed and other diskette-to-diskette and drive-to-drive manufacturing tolerances to ensure that data written on a diskette by one system can be read by another (diskette interchangeability).

IBM Format Implementation Summary

Track Format

The disk has 77 tracks, numbered physically from 00 to 76, with track 00 being the outermost track. There are logically 75 data tracks and two alternate tracks. Any two tracks may be initialized as bad tracks. The data tracks are numbered logically in sequence from 00 to 74, skipping over bad tracks (alternate tracks replace bad tracks). Note: In IBM format track 00 cannot be a bad track.

Sector Format

Each track is divided into 26, 15, or 8 sectors of 128, 256, or 512 bytes length respectively. The first sector is numbered 01, and is physically the first sector after the physical index mark. The logical sequence of the remaining sectors may be nonsequential physically. The location of these is determined at initialization by CPU software.

Each sector consists of an ID field and a data field. All fields are separated by gaps. The beginning of each field is indicated by 6 bytes of $(00)_H$ followed by a one byte address mark.

Address Marks

Address Marks are unique bit patterns one byte in length which are used to identify the beginning of ID and Data fields. Address Mark bytes are unique from all other data

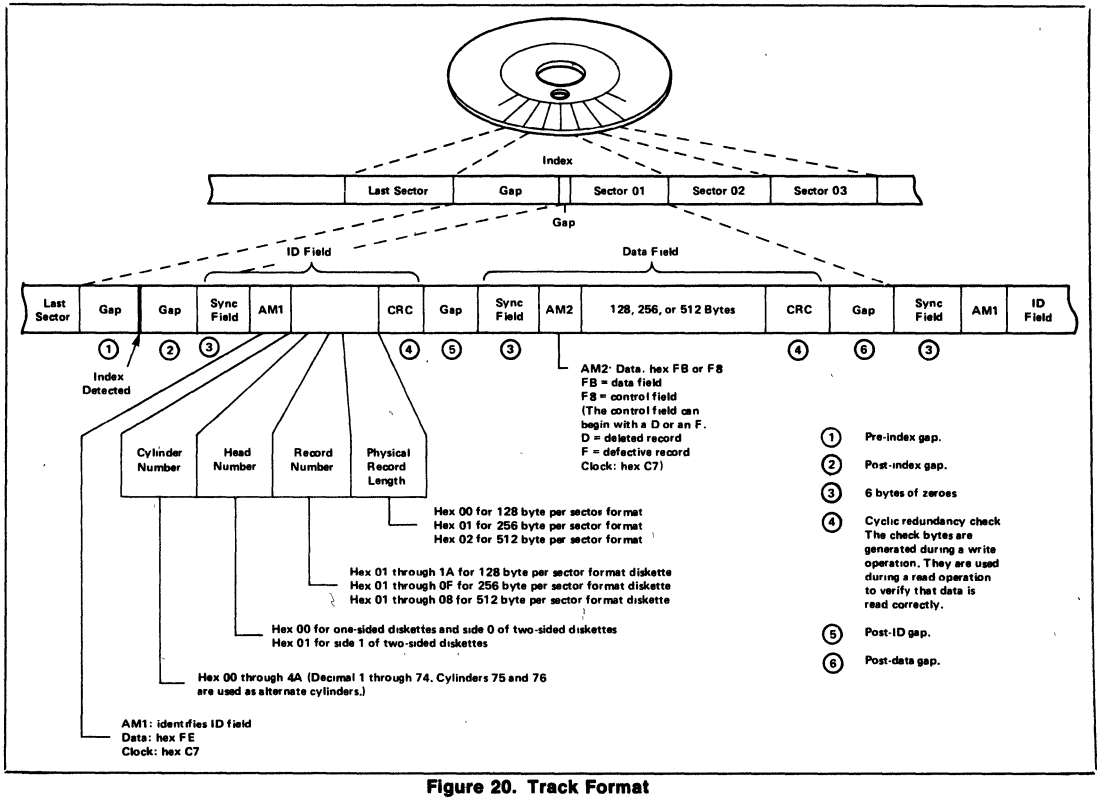


Figure 20. Track Format

bytes in that certain bit cells do not contain a clock bit (all other data bytes have clock bits in every bit cell.) There are four different types of Address Marks used. Each of these is used to identify different types of fields.

Index Address Mark

The Index Address Mark is located at the beginning of each track and is a fixed number of bytes in front of the first record.

ID Address Mark

The ID Address Mark byte is located at the beginning of each ID field on the diskette.

Data Address Mark

The Data Address Mark byte is located at the beginning of each non-deleted Data Field on the diskette.

Deleted Data Address Mark

The Deleted Data Address Mark byte is located at the beginning of each deleted Data Field on the diskette.

Address Mark Summary	Clock Pattern	Data Pattern
Index Address Mark	D7	FC
ID Address Mark	C7	FE
Data Address Mark	C7	FB
Deleted Data Address Mark	C7	F8
Bad Track ID Address Mark	C7	FE

ID Field

MARK	C	H	R	N	CRC	CRC

C = Cylinder (Track) Address, 00-74
H = Head Address
R = Record (Sector) Address, 01-26
N = Record (Sector) Length, 00-02
Note: Sector Length = 128 x 2^N bytes
CRC = 16 Bit CRC Character (See Below)

Data Field

MARK	DATA	CRC	CRC

Data is 128, 256, or 512 bytes long.

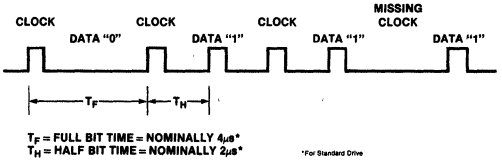
Note: All marks, data, ID characters and CRC characters are recorded and read most significant bit first.

CRC Character

The 16-bit CRC character is generated using the generator polynominal $X^{16} + X^{12} + X^5 + 1$, normally initialized to $(FF)_H$. It is generated from all characters (except the CRC in the ID or data field), including the data (not the clocks) in the address mark. It is recorded and read most significant bit first.

Data Format

Data is written (general case) in the following manner:



References

"The IBM Diskette for Standard Data Interchange," IBM Document GA21-9182-0. "System 32," Chapter 8, IBM Document GA21-9176-0.

Bad Track Format

The Bad Track Format is the same as the good track format except that the bad track ID field is initialized as follows:

$$C = H = R = N = (FF)_H$$

When formatting, bad track registers should be set to FF_H for the drive during the formatting, thus specifying no bad tracks. Thus, all tracks are left available for formatting.

The track following the bad track(s) should be one higher in number than track before the bad track(s).

Upon completion of the format the bad tracks should be set up using the write special register command. The 8271 will then generate an extra step pulse to cross the bad track, locating a new track that now happens to be an extra track out.

Format Track

Format Command

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	SEL 1	SEL 0	1	0	0	0	1	1
PAR	0	1	TRACK ADDRESS							
PAR	0	1	GAP 3 SIZE MINUS 6							
PAR	0	1	RECORD LENGTH			NO. OF SECTORS/TRACK				
PAR.	0	1	GAP 5 SIZE MINUS 6							
PAR	0	1	GAP 1 SIZE MINUS 6							

The format command can be used to initialize a disk track compatible with the IBM 3740 format. A Shugart "IBM Type" mini-floppy format may also be generated.

The Format command can be used to initialize a diskette, one track at a time. When format command is used, the program must supply ID fields for each sector on the track. During command execution, the supplied ID fields (track head sector addresses and the sector length) are written sequentially on the diskette. The ID address marks originate from the 8271 and are written automatically as the first byte of each ID field. The CRC character is written in the last two bytes of the ID field and is derived from the data written in the first five bytes. During the formatting operation, the data field of each sector is filled with data pattern $(E5)_H$. The CRC, derived from the data pattern is also appended to the last byte.

1. The parameter 2 ($D_7 - D_5$) of the Format command specify record length, the bits are coded the same way as in the Read Data commands.
2. The programmable gap sizes (gap 3, gap 5, and gap 1) must be programmed such that the 6 bytes of zero (sync) are subtracted from the intended gap size i.e., if gap 1 is intended to be 16 bytes long, programmed length must be 16 – 6 = 10 bytes (of FF_H's).

Mini-Floppy Disk Format

The mini-floppy disk format differs from the standard disk format in the following ways:

1. Gap 5 and the Index Address mark have been eliminated.
2. There are fewer sectors/tracks.

GAPS

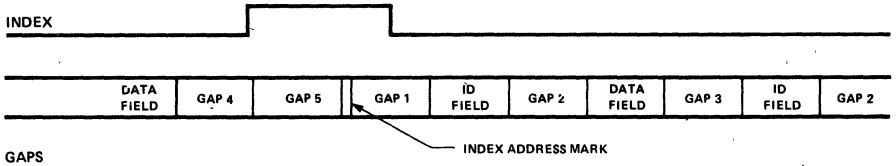
The following is the gap size and description summary:

Gap 1 Programmable
Gap 2 17 Bytes
Gap 3 Programmable
Gap 4 Variable
Gap 5 Programmable

The last six bytes of gaps 1,2,3 and 5 are $(00)_H$, all other bytes in the gaps are $(FF)_H$. The Gap 1,3 and 5 count specified by the user are the number of bytes of $(FF)_H$. Gap 4 is written until the leading edge of the index pulse. If a Gap 5 size of zero is specified, the Index Mark is not written.

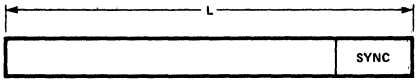
Gap 1: N bytes FF's 6 bytes 0's for sync	This gap separates the index address mark of the index pulse from the first ID mark. It is used to protect the first ID field from a write on the last physical sector of the current track.
Gap 2: 11 bytes FF's 6 bytes 0's for sync	This gap separates the ID field from the data mark and field such that during a write only the data field will be changed even if the write gate turns on early, due to drive speed changes.
Gap 3: N bytes FF's 6 bytes 0's for sync	This gap separates a data area from the next ID field. It is used so that during drive speed changes the next ID mark will not be overwritten, thus causing loss of data.
Gap 4: FF's only	This gap fills out the rest of the disk and is used for slack during formatting. During drive speed variations this gap will shrink or grow if the disk is re-formatted.
Gap 5: N bytes FF's 6 bytes 0's for sync	This gap separates the last sector from the Index Address mark and is used to assure that the index address mark is not destroyed by writing on the last physical data sector on the track.

The number of FF bytes is programmable for gaps 1, 3 and 5.

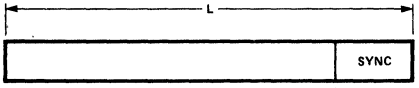


GAPS

GAP 1: POST INDEX GAP



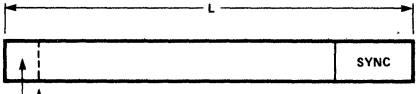
GAP 2: POST ID FIELD GAP



WRITE GATE TURN-ON FOR UPDATE OF NEXT DATA FIELD.

NOTE THE WRITE GATE TURN-ON SHOULD BE TIMED TO WITHIN ± ≈ 1 BIT BY COUNTING THE BYTES IN THE GAP UNTIL 1 BYTE BEFORE THE TURN-ON

GAP 3: POST DATA FIELD GAP

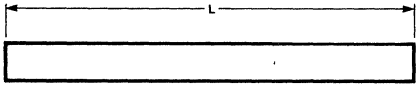


WRITE GATE TURN-OFF FROM UPDATE OF PREVIOUS DATA FIELD

NOTE IBM FORMAT REQUIRES AT LAT 2 BINARY "1" BITS AS A DATA FIELD POSTAMBLE.

2-BITS

GAP 4: FINAL GAP



GAP 5: INITIAL GAP

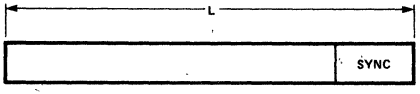
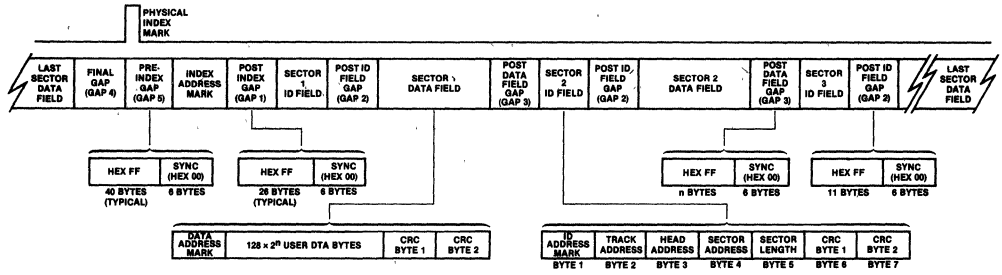


Figure 21. Track Format

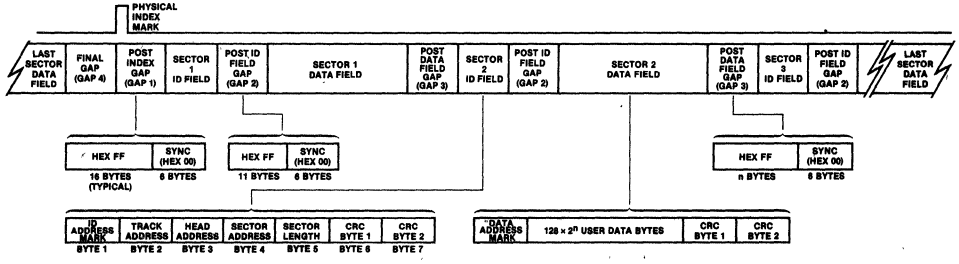


NUMBER OF SECTORS	NUMBER OF BYTES										
	GAP 1		ID FIELD	GAP 2		DATA FIELD	GAP 3		GAP 4	GAP 5	
	*ONES	SYNC		ONES	SYNC		*ONES	SYNC		*ONES	SYNC
26	26	6	7	11	6	131	27	6	275	40	6
15	26	6	7	11	6	259	48	6	129	40	6
8	26	6	7	11	6	515	90	6	146	40	6
4	26	6	7	11	6	1027	224	6	236	40	6
2	26	6	7	11	6	2051	255	6	719	40	6
1	26	6	7	11	6	4099	0	0	1007	40	6

*Program Specified

5208 Bytes Per Track

Figure 22. Standard Diskette Track Format



NUMBER OF SECTORS	NUMBER OF BYTES								
	GAP 1		ID FIELD	GAP 2		DATA FIELD	GAP 3		GAP 4
	*ONES	SYNC		ONES	SYNC		*ONES	SYNC	
18	16	6	7	11	6	131	11	6	24
10	16	6	7	11	6	259	21	6	30
5	16	6	7	11	6	515	74	6	88
2	16	6	7	11	6	1027	255	6	740
1	16	6	7	11	6	2051	0	0	1028

*Program Specified

3125 Bytes Per Track

Figure 23. Mini-Diskette Track Format

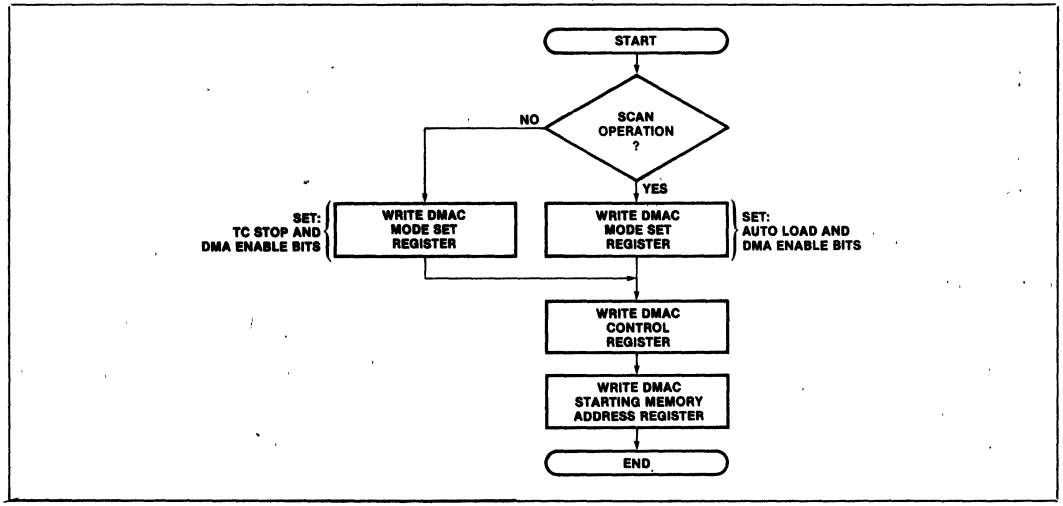


Figure 24. User DMA Channel Initialization Flowchart

Read ID Command

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	SEL 1	SEL 0	0	1	1	0	1	1
PAR:	0	1	TRACK ADDRESS							
PAR:	0	1	0	0	0	0	0	0	0	0
PAR:	0	1	NUMBER OF ID FIELDS							

The Read ID command transfers the specified number of ID fields into memory (beginning with the first ID field after Index). The CRC character is checked but not transferred.

These fields are entered into memory in the order in which they are physically located on the disk, with the first field being the one starting at the index pulse.

Data Processing Commands

All the routine Read/Write commands examine specific drive status lines before beginning execution, perform an implicit seek to the track address and load the drive's read/write head. Regardless of the type of command (i.e., read, write or verify), the 8271 first reads the ID field(s) to verify that the correct track has been located (see sector not found completion code) and also to locate the addressed sector. When a transfer is complete (or cannot be completed), the 8271 sets the interrupt request bit in the status register and provides an indication of the outcome of the operation in the result register.

If a CRC error is detected during a multisector transfer, processing is terminated with the sector in error. The address of the failing sector number can be determined by examining the Scan Sector Number register using the Read Special Register command.

Full power of the multisector read/write commands can be realized by doing DMA transfer using Intel® 8257 DMA Controller, For example, in a 128 byte per sector multisector write command, the entire data block (containing 128 bytes times the number of sectors) can be located in a disk memory buffer. Upon completion of the command phase, the 8271 begins execution by accessing the desired track, verifying the ID field, and locating the data field of the first record to be written. The 8271 then DMA-accesses the first sector and starts counting and writing one byte at a time until all 128 bytes are written. It then locates the data field of the next sector and repeats the procedure until all the specified sectors have been written. Upon completion of the execution phase the 8271 enters into the result phase and interrupts the CPU for availability of status and completion results. Note that all read/write commands, single or multisector are executed without CPU intervention.

Note, execution of multi-sector operations are faster if the sectors are *not* interleaved.

128 Byte Single Record Format

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	SEL 1	SEL 0	COMMAND OPCODE					
PAR:	0	1	TRACK ADDR 0-255							
PAR:	0	1	SECTOR 0-255							

Commands	Opcode
READ DATA	12
READ DATA AND DELETED DATA	16
WRITE DATA	0A
WRITE DELETED DATA	0E
VERIFY DATA AND DELETED DATA	1E

Variable Length/Multi-Record Format

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	SEL 1	SEL 0	COMMAND OPCODE					
PAR:	0	1	TRACK ADDR 0-255							
PAR.	0	1	SECTOR 0-255							
PAR:	0	1	LENGTH				NO. OF SECTORS			

D_7-D_5 of Parameter 2 determine the length of the disk record.

0 0 0	128 Bytes
0 0 1	256 Bytes
0 1 0	512 Bytes
0 1 1	1024 Bytes
1 0 0	2048 Bytes
1 0 1	4096 Bytes
1 1 0	8192 Bytes
1 1 1	16,384 Bytes

Commands	Opcode
READ DATA	13
READ DATA AND DELETED DATA	17
WRITE DATA	0B
WRITE DELETED DATA	0F
VERIFY DATA AND DELETED DATA	1F
SCAN DATA	00
SCAN DATA AND DELETED DATA	04

Read Commands

Read Data, Read Data and Deleted Data.

Function

The read command transfers data from a specified disk record or group of records to memory. The operation of this command is outlined in execution phase table.

Write Commands

Write Data, Write Deleted Data.

Function

The write command transfers data from memory to a specified disk record or group of records.

Verify Command

Verify Data and Deleted Data.

Function

The verify command is identical to the read data and deleted data command except that the data is not transferred to memory. This command is used to check that a record or a group of records has been written correctly by verifying the CRC character.

Scan Commands

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	SEL 1	SEL 0	0	0	0	S DATA S DELD	0	0
PAR.	0	1	TRACK ADDR 0 255							
PAR	0	1	SECTOR 0 255							
PAR.	0	1	LENGTH.				NO OF SECTORS			
PAR	0	1	SCAN TYPE		STEP SIZE					
PAR:	0	1	FIELD LENGTH (KEY)							

Command $D_2 = 0$ Scan Data
$D_2 = 1$ Scan Data and Deleted Data

Scan Commands, Scan Data and Scan Data and Deleted Data, are used to search a specific data pattern or "key" from memory. The 8271 FDC operation during a scan is unique in that data is read from memory and from the diskette simultaneously.

During the scan operation, the key is compared repetitively (using the 8257 DMA Controller in auto load mode) with the data read from the diskette (e.g., an eight byte key would be compared with the first eight bytes (1-8) read from the diskette, the second eight bytes (9-16), the third eight bytes (17-24), etc.). The scan operation is concluded when the key is located or when the specified number of sectors have been searched without locating the key. When concluded, the 8271 FDC requests an interrupt. The program must then read the result register to determine if the scan was successful (if the key was located). If successful, several of the FDC's special registers can be examined (read special registers command) to determine more specific information relating to the scan (i.e., the sector number in which the key was located, and the number of bytes within the sector that were not compared when the key was located).

The 8271 does not do a sliding scan, it does a fixed block linear search. This means the key in memory is compared to an equal length block in a sector; when these blocks meet the scan conditions the scan will stop. Otherwise, the scan continues until all the sectors specified have been searched.

The following factors regarding key length must be considered when establishing a key in memory.

1. When searching multiple sectors, the length of the key must be evenly divisible into the sector length to prevent the key from being split at subsequent sector boundaries. Since the character FF_H is not compared, the key in memory can be padded to the required length using this character. For example, if the actual pattern compared on the diskette is twelve characters in length, the field length should be sixteen and four bytes of FF_H

would be appended to the key. Consequently, the last block of sixteen bytes compared within the first sector would end at the sector boundary and the first byte of the next sector would be compared with the first byte of the key. Splitting data over sector boundarys will not work properly since the FDC expects the start of key at each sector boundary.

2. Since the first byte of the key is compared with the first byte of the sector, when the pattern does not begin with the first byte of the sector, the key must be offset using the character FF_{16}. For example, if the first byte of a nine byte pattern begins on the fifth byte of the sector, four bytes of FF_{16} are prefixed to the key (and three bytes of FF_{16} are appended to the key to meet the length requirement) so that the first actual comparison begins on the fifth byte.

The Scan Commands require five parameters:

Parameter 0, Track Address

Specifies the track number containing the sectors to be scanned. Legal values range from 00_H to $4C_H$ (0 to 76) for a standard diskette and from 00_H to 22_H (0 to 34) for a mini-sized diskette.

Parameter 1, Sector Address

Specifies the first sector to be scanned. The number of sectors scanned is specified in parameter 2, and the order in which sectors are scanned is specified in parameter 3.

Parameter 2, Sector Length/Number of Sectors

The sector length field (bits 7-5) specifies the number of data bytes allocated to each sector (see parameter 2, routine read and write commands for field interpretation). The number of sectors field (bits 4-0) specifies the number of sectors to be scanned. The number specified ranges from one sector to the physical number of sectors on the track.

Parameter 3

D_7-D_6: Indicate scan type

00-EQ Scan for each character within the field length (key) equal to the corresponding character within the disk sector. The scan stops after the first equal condition is met.

01-GEQ Scan for each character within the disk sector greater than or equal to the corresponding character within the field length (key). The scan stops after the first greater than or equal condition is met.

10-LEQ Scan for each character within the disk sector less than or equal to the corresponding character within the field length (key). The scan stops after the first less than or equal condition is met.

D_5-D_0: Step Size: The Step Size field specifies the offset to the next sector in a multisector scan. In this case, the next sector address is generated by adding the Step Size to the current sector address.

Parameter 4, Field Length

Specifies the number of bytes to be compared (length of key). While the range of legal values is from 1 to 255, the field length specified should be evenly divisible into the sector length to prevent the key from being split at sector boundaries, if the multisector scan commands are used.

Scan Command Results

More detailed information about the completion of Scan Commands may be obtained by executing Read Special Register commands.

Read Special Register

Parameter (Hex)	Results
06	The sector number of the sector in which the specified scan data pattern was located.
14	MSB Count — The number of 128 byte blocks remaining to be compared in the current sector when the scan data pattern was located. This register is decremented with each 128 byte block read.
13	LSB Count — The number of bytes remaining to be compared in the current sector when the scan data pattern is located. This register is initialized to 128 and is decremented with each byte compared.

Upon a scan met condition, the equation below can be used to determine the last byte in the located pattern.

Pointer = sector length – ((Register 14H)*128 + (Register 13H))

8271 Scan Command Example

Assume there are only 2 records on track 0 with the following data:

Record 01: 01 02 03 04 05 06 07 08 000....00
Record 02: 01 02 AA 55 00 00 00 00 00

Command	Field Length [1]	Starting Sector #	# of Sectors	Key [2]	Completion Code [3]	Special Registers [4] R06	R14	R13	Comment
* SCAN EQ	2	1	1	01,02	SME	01	0	127D	Met in first field
SCAN EQ	2	1	1	02,03	SNM	X	X	X	Not met
SCAN EQ	2	1	1	FF[5],05	SNM	X	X	X	Not met with don't care
* SCAN EQ	2	1	1	FF[5],06	SME	01	0	123D	Met with don't care
* SCAN EQ	2	1	2	AA,55	SME	02	0	125D	Met in Record 02
* SCAN EQ	2	2	1	01,02	SME	02	0	127D	Starting sector ≠ 1
* SCAN EQ	4	1	1	05,06,07,08	SME	01	0	121D	Field, Key length = 4
* SCAN GEQ	4	1	1	05,06,07,08	SME	01	0	121D	GEQ-SME
* SCAN GEQ	4	1	1	05,04,07,08	SMNE	01	0	121D	GEQ-SMNE
* SCAN GEQ	4	1	2	00,03,AA,44 [6]	SNM	X	X	X	GEQ-SNM
* SCAN LEQ	4	1	1	01,03,FF,04	SMNE	01	0	125D	LEQ-SMNE
* SCAN LEQ	4	1	1	01,02,FF,04	SME	01	0	125D	LEQ-SME

NOTES:

1. Field Length — Each record is partitioned into a number of fields equal to the record size divided by the field length. Note that the record size should be evenly divisable by the field length to insure proper operation of multi record scan. Also, maximum field length = 256 bytes.

2. Key — The key is a string of bytes located in the user system memory. The key length should equal the field length. By programming the 8257 DMA Controller into the auto load mode, the key will be recursively read in by the chip (once per field).

3. Completion Code — Shows how Scan command was met or not met.
   SNM — SCAN Not Met — 0 0 (also Good Complete)
   SME — SCAN Met Equal — 0 1
   SMNE — SCAN Met Not Equal — 1 0

4. Special Registers
   R06 — This register contains the record number where the scan was met.
   R14 — This register contains the MSB count and is decremented every 128 characters.

Length (ℓ) (D7-D5 of PAR 2)	Record Size	R14 = $2^{\ell} - 1$ (Initialize at Beginning of Record)
000	128 Bytes	0
001	256 Bytes	1
010	512 Bytes	3
011	1024 Bytes	7
•	•	•
•	•	•
•	•	•

   R13 — This register contains a modulo 128 LSB count which is initialized to 128 at beginning of each record. This count is decremented after each character is compared except for the last character in a pattern match situation.

5. The OFF_H character in the key is treated as a don't care character position.

6. The Scan comparison is done on a byte by byte basis. That is, byte 1 of each field is compared to byte 1 of the key, byte 2 of each field is compared to byte 2 of the key, etc.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to 70°C[1]
Storage Temperature.....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground.....−0.5V to +7V
Power Dissipation.....1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = +5.0V ±5%
8721: T_A = 0°C to 70°C; 8271-6: T_A = 0°C to 50°C)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	(V_{CC} + 0.5)	V	
V_{OLD}	Output Low Voltage (Data Bus)		0.45	V	I_{OL} = 2.0 mA
V_{OLI}	Output Low Voltage (Interface Pins)		0.5	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −220 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OZ}	Off-State Output Current		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		180	mA	

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	t_c = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured Pins Returned to GND

NOTE: 1. Ambient temperature under bias for 8271-6 is 0°C to 50°C.

A.C. CHARACTERISTICS (V_{CC} = +5.0V ±5%)
(8271: T_A = 0°C to 70°C; 8271-6: T_A = 0°C to 50°C)

READ CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{RD}$	0		ns	Note 2
t_{CA}	Select Hold from $\overline{RD}$	0		ns	Note 2
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	Data Delay from Address		250	ns	Note 2
t_{RD}	Data Delay from $\overline{RD}$		150	ns	C_L = 150 pF, Note 2
t_{DF}	Output Float Delay	20	100	ns	C_L = 20 pF for Minimum; 150 pF for Maximum
t_{DC}	DACK Setup to $\overline{RD}$	25		ns	
t_{CD}	DACK Hold from $\overline{RD}$	25		ns	
t_{KD}	Data Delay from DACK		250	ns	

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{WR}$	0		ns	
t_{CA}	Select Hold from $\overline{WR}$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$	150		ns	
t_{WD}	Data Hold from $\overline{WR}$	0		ns	
t_{DC}	DACK Setup to $\overline{WR}$	25		ns	
t_{CD}	DACK Hold from $\overline{WR}$	25		ns	

DMA

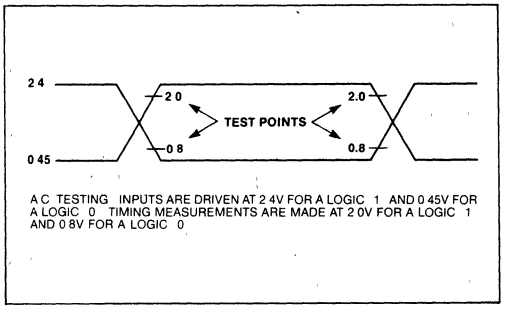
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CQ}	Request Hold from $\overline{WR}$ or $\overline{RD}$ (for Non-Burst Mode)		150	ns	

OTHER TIMINGS

Symbol	Parameter	8271/8271-6		Unit	Test Conditions
		Min.	Max.		
t_{RSTW}	Reset Pulse Width	10		t_{CY}	
t_r	Input Signal Rise Time		20	ns	
t_f	Input Signal Fall Time		20	ns	
t_{RSTS}	Reset to First IOWR	2		t_{CY}	
t_{CY}	Clock Period	250			Note 3
t_{CL}	Clock Low Period	110		ns	
t_{CH}	Clock High Period	125		ns	
t_{DS}	Data Window Setup to Unseparated Clock and Data	50		ns	
t_{DH}	Data Window Hold from Unseparated Clock and Data	0		ns	

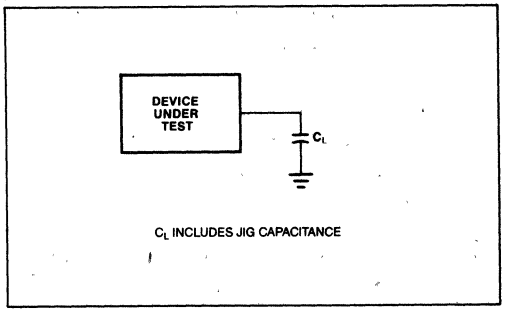
NOTES:
1. All timing measurements are made at the reference voltages unless otherwise specified: Input "1" at 2.0V, "0" at 0.8V
   Output "1" at 2.0V, "0" at 0.8V
2. t_{AD}, t_{RD}, t_{AC}, and t_{CA} are not concurrent specs.
3. Standard Floppy: $t_{CY} = 250$ ns $\pm 0.4\%$ Mini-Floppy: $t_{CY} = 500$ ns $\pm 0.4\%$

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

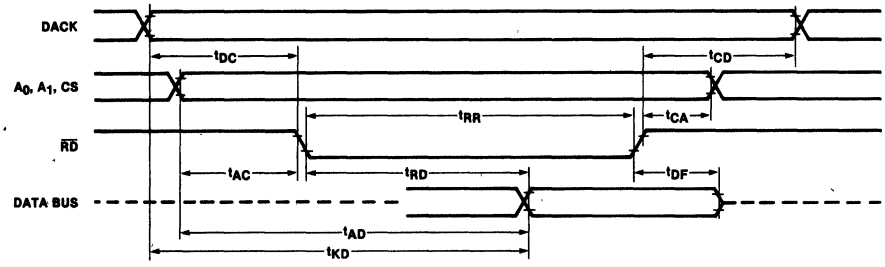
A.C. TESTING LOAD CIRCUIT



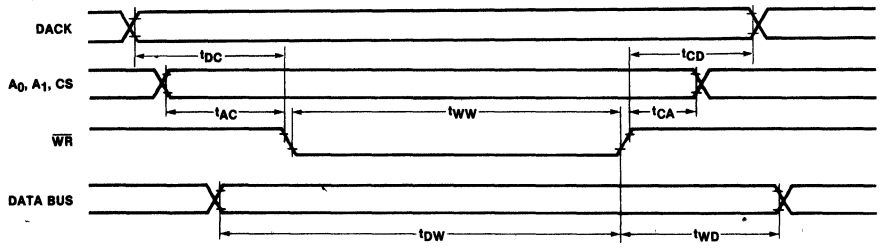
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

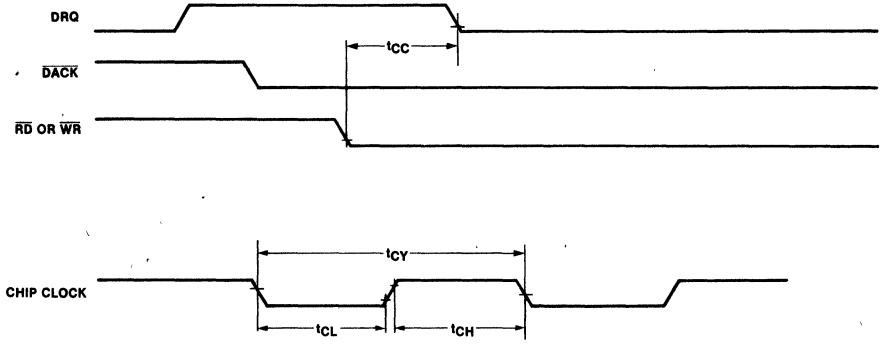
READ



WRITE

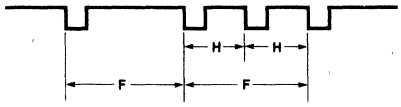


DMA



WAVEFORMS (Continued)

READ DATA

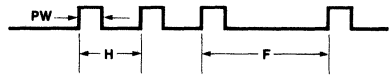


*t_{CY} = 250 ns **t_{CY} = 500 ns

F = 16 t_{CY} ±8 t_{CY}
H = 8 t_{CY} ±4 t_{CY}

*STANDARD FLEXIBLE DISK DRIVE TIMING
**MINI-FLOPPY TIMING

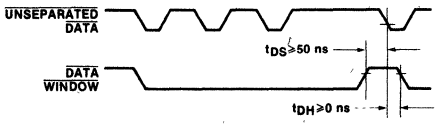
WRITE DATA



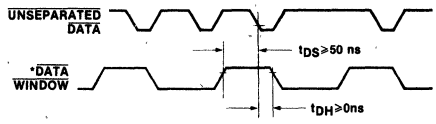
PULSE WIDTH PW = t_{CY} ±30 ns
H (HALF BIT CELL) = 8 t_{CY}
F (FULL BIT CELL) = 16 t_{CY}

*t_{CY} =		**t_{CY} =	
250 ns	±0.4%	500 ns	±0.4%
250 ns	±30 ns	500 ns	±30 ns
2.0 μs	± 8 ns	4.0 μs	±16 ns
4.0 μs	±16 ns	8.0 μs	±32 ns

SINGLE-SHOT DATA SEPARATOR



PLO DATA SEPARATOR



*DATA WINDOW MAY BE 180° OUT OF PHASE IN PLO DATA SEPARATION MODE.

8272A
SINGLE/DOUBLE DENSITY FLOPPY DISK CONTROLLER

- **IBM Compatible in Both Single and Double Density Recording Formats**
- **Programmable Data Record Lengths: 128, 256, 512, or 1024 Bytes/Sector**
- **Multi-Sector and Multi-Track Transfer Capability**
- **Drives Up to 4 Floppy or Mini-Floppy Disks**
- **Data Transfers in DMA or Non-DMA Mode**
- **Parallel Seek Operations on Up to Four Drives**
- **Compatible with all Intel and Most Other Microprocessors**
- **Single-Phase 8 MHz Clock**
- **Single +5 Volt Power Supply (±10%)**

The 8272A is an LSI Floppy Disk Controller (FDC) Chip, which contains the circuitry and control functions for interfacing a processor to 4 Floppy Disk Drives. It is capable of supporting either IBM 3740 single density format (FM), or IBM System 34 Double Density format (MFM) including double sided recording. The 8272A provides control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The FDC simplifies and handles most of the burdens associated with implementing a Floppy Disk Drive Interface. The 8272A is a pin-compatible upgrade to the 8272.

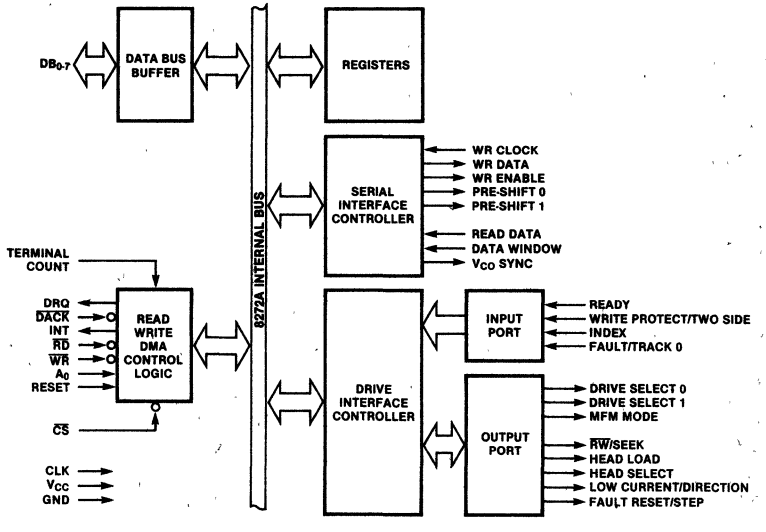


Figure 1. 8272A Internal Block Diagram

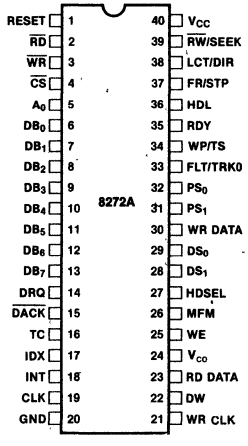


Figure 2. Pin Configuration

ORDER NUMBER 210606-001

Table 1. Pin Description

Symbol	Pin No.	Type	Connection To	Name and Function
$\overline{RESET}$	1	I	μP	**Reset:** Places FDC in idle state. Resets output lines to FDD to "0" (low). Does not clear the last specify command.
$\overline{RD}$	2	I[1]	μP	**Read:** Control signal for transfer of data from FDC to Data Bus, when "0" (low).
$\overline{WR}$	3	I[1]	μP	**Write:** Control signal for transfer of data to FDC via Data Bus, when "0" (low).
$\overline{CS}$	4	I	μP	**Chip Select:** IC selected when "0" (low), allowing $\overline{RD}$ and $\overline{WR}$ to be enabled.
A_0	5	I[1]	μP	**Data/Status Register Select:** Selects Data Reg (A_0 = 1) or Status Reg (A_0 = 0) contents to be sent to Data Bus.
DB_0-DB_7	6–13	I/O[1]	μP	**Data Bus:** Bidirectional 8-Bit Data Bus.
DRQ	14	O	DMA	**Data DMA Request:** DMA Request is being made by FDC when DRQ "1."[3]
$\overline{DACK}$	15	I	DMA	**DMA Acknowledge:** DMA cycle is active when "0" (low) and Controller is performing DMA transfer.
TC	16	I	DMA	**Terminal Count:** Indicates the termination of a DMA transfer when "1" (high)[2].
IDX	17	I	FDD	**Index:** Indicates the beginning of a disk track.
INT	18	O	μP	**Interrupt:** Interrupt Request Generated by FDC.
CLK	19	I		**Clock:** Single Phase 8 MHz (4 MHz for mini floppies) Squarewave Clock.
GND	20			**Ground:** D.C. Power Return.
V_{CC}	40			**D.C. Power:** +5V
$\overline{RW}$/SEEK	39	O	FDD	**Read Write / SEEK:** When "1" (high) Seek mode selected and when "0" (low) Read/Write mode selected.
LCT/DIR	38	O	FDD	**Low Current/Direction:** Lowers Write current on inner tracks in Read/Write mode, determines direction head will step in Seek mode.
FR/STP	37	O	FDD	**Fault Reset/Step:** Resets fault FF in FDD in Read/Write mode, provides step pulses to move head to another cylinder in Seek mode.
HDL	36	O	FDD	**Head Load:** Command which causes read/write head in FDD to contact diskette.
RDY	35	I	FDD	**Ready:** Indicates FDD is ready to send or receive data. Must be tied high (gated by the index pulse) for mini floppies which do not normally have a Ready line.
WP/TS	34	I	FDD	**Write Protect / Two-Side:** Senses Write Protect status in Read/Write mode, and Two Side Media in Seek mode.
FLT/TRK0	33	I	FDD	**Fault/Track 0:** Senses FDD fault condition in Read/Write mode and Track 0 condition in Seek mode.
PS_1,PS_0	31,32	O	FDD	**Precompensation (preshift):** Write precompensation status during MFM mode. Determines early, late, and normal times.
WR DATA	30	O	FDD	**Write Data:** Serial clock and data bits to FDD.
DS_1,DS_0	28,29	O	FDD	**Drive Select:** Selects FDD unit.
HDSEL	27	O	FDD	**Head Select:** Head 1 selected when "1" (high) Head 0 selected when "0" (low).

Note 1. Disabled when $\overline{CS}$=1
Note 2. TC must be activated to terminate the Execution Phase of any command
Note 3. DRQ is also an input for certain test modes. It should have a 5kΩ pull-up resistor to prevent activation

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Connection To	Name and Function
MFM	26	O	PLL	**MFM Mode:** MFM mode when "1," FM mode when "0."
WE	25	O	FDD	**Write Enable:** Enables write data into FDD.
VCO	24	O	PLL	**VCO Sync:** Inhibits VCO in PLL when "0" (low), enables VCO when "1."
RD DATA	23	I	FDD	**Read Data:** Read data from FDD, containing clock and data bits.
DW	22	I	PLL	**Data Window:** Generated by PLL, and used to sample data from FDD.
WR CLK	21	I		**Write Clock:** Write data rate to FDD FM = 500 kHz, MFM = 1 MHz, with a pulse width of 250 ns for both FM and MFM. Must be enabled for all operations, both Read and Write.

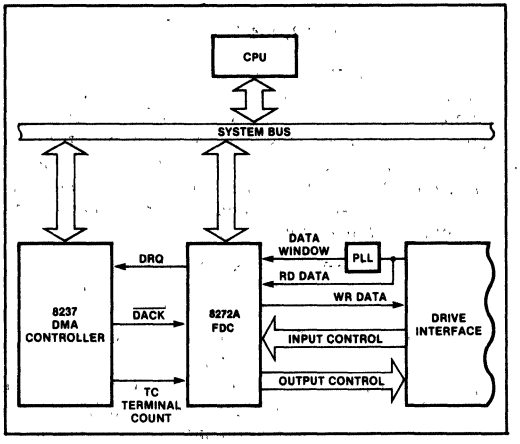


Figure 3. 8272A System Block Diagram

DESCRIPTION

Hand-shaking signals are provided in the 8272A which make DMA operation easy to incorporate with the aid of an external DMA Controller chip, such as the 8237A. The FDC will operate in either DMA or Non-DMA mode. In the Non-DMA mode, the FDC generates interrupts to the processor for every transfer of a data byte between the CPU and the 8272A. In the DMA mode, the processor need only load a command into the FDC and all data transfers occur under control of the 8272A and DMA controller.

There are 15 separate commands which the 8272A will execute. Each of these commands require multiple 8-bit bytes to fully specify the operation which the processor wishes the FDC to perform. The following commands are available.

Read Data	Write Data
Read ID	Format a Track
Read Deleted Data	Write Deleted Data
Read a Track	Seek
Scan Equal	Recalibrate (Restore to Track 0)
Scan High or Equal	Sense Interrupt Status
Scan Low or Equal	Sense Drive Status
Specify	

For more information see the Intel Application Notes AP-116 and AP-121.

FEATURES

Address mark detection circuitry is internal to the FDC which simplifies the phase locked loop and read electronics. The track stepping rate, head load time, and head unload time may be programmed by the user. The 8272A offers many additional features such as multiple sector transfers in both read and write modes with a single command, and full IBM compatibility in both single (FM) and double density (MFM) modes.

8272A ENHANCEMENTS

On the 8272A, after detecting the Index Pulse, the VCO Sync output stays low for a shorter period of time. See Figure 4A.

On the 8272 there can be a problem reading data when Gap 4A is 00 and there is no IAM. This occurs on some older floppy formats. The 8272A cures this problem by adjusting the VCO Sync timing so that it is not low during the data field. See Figure 4B.

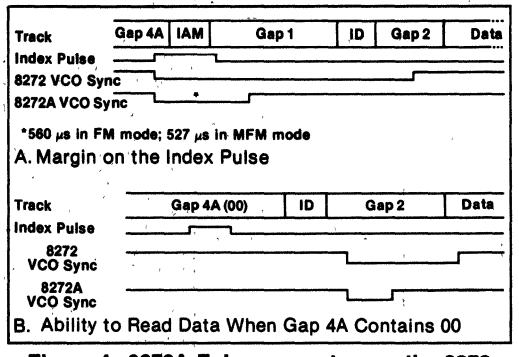


Figure 4. 8272A Enhancements over the 8272

8272A REGISTERS — CPU INTERFACE

The 8272A contains two registers which may be accessed by the main system processor; a Status Register and a Data Register. The 8-bit Main Status Register contains the status information of the FDC, and may be accessed at any time. The 8-bit Data Register (actually consists of several registers in a stack with only one register presented to the data bus at a time), stores data, commands, parameters, and FDD status information. Data bytes are read out of, or written into, the Data Register in order to program or obtain the results after execution of a command. The Status Register may only be read and is used to facilitate the transfer of data between the processor and 8272A.

The relationship between the Status/Data registers and the signals $\overline{RD}$, $\overline{WR}$, and A_0 is shown in Table 2.

Table 2. A_0, $\overline{RD}$, $\overline{WR}$ decoding for the selection of Status/Data register functions.

A_0	$\overline{RD}$	$\overline{WR}$	FUNCTION
0	0	1	Read Main Status Register
0	1	0	Illegal (see note)
0	0	0	Illegal (see note)
1	0	0	Illegal (see note)
1	0	1	Read from Data Register
1	1	0	Write into Data Register

Note: Design must guarantee that the 8272A is not subjected to illegal inputs.

The Main Status Register bits are defined in Table 3.

Table 3. Main Status Register bit description.

BIT NUMBER	NAME	SYMBOL	DESCRIPTION
D_0	FDD 0 Busy	D_0B	FDD number 0 is in the Seek mode.
D_1	FDD 1 Busy	D_1B	FDD number 1 is in the Seek mode.
D_2	FDD 2 Busy	D_2B	FDD number 2 is in the Seek mode.
D_3	FDD 3 Busy	D_3B	FDD number 3 is in the Seek mode.
D_4	FDC Busy	CB	A read or write command is in process.
D_5	Non-DMA mode	NDM	The FDC is in the non-DMA mode. This bit is set only during the execution phase in non-DMA mode. Transition to "0" state indicates execution phase has ended.
D_6	Data Input/Output	DIO	Indicates direction of data transfer between FDC and Dta Register. If DIO = "1" then transfer is from Data Register to the Processor. If DIO = "0", then transfer is from the Processor to Data Register.
D_7	Request for Master	RQM	Indicates Data Register is ready to send or receive data to or from the Processor. Both bits DIO and RQM should be used to perform the handshaking functions of "ready" and "direction" to the processor.

The DIO and RQM bits in the Status Register indicate when Data is ready and in which direction data will be transferred on the Data Bus.

Note: There is a 12μS or 24μS RQM flag delay when using an 8 or 4 MHz clock respectively.

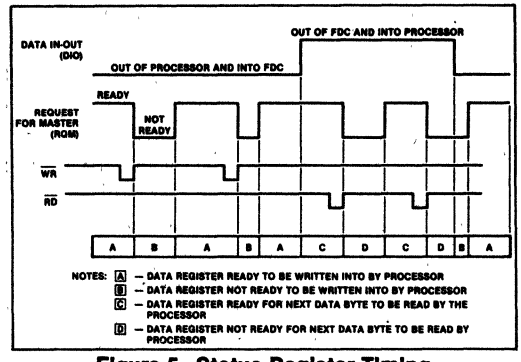


Figure 5. Status Register Timing

The 8272A is capable of executing 15 different commands. Each command is initiated by a multi-byte transfer from the processor, and the result after execution of the command may also be a multi-byte transfer back to the processor. Because of this multi-byte interchange of information between the 8272A and the processor, it is convenient to consider each command as consisting of three phases:

Command Phase: The FDC receives all information required to perform a particular operation from the processor.

Execution Phase: The FDC performs the operation it was instructed to do.

Result Phase: After completion of the operation, status and other housekeeping information are made available to the processor.

During Command or Result Phases the Main Status Register (described in Table 3) must be read by the processor before each byte of information is written into or read from the Data Register. Bits D6 and D7 in the Main Status Register must be in a 0 and 1 state, respectively, before each byte of the command word may be written into the 8272A. Many of the commands require multiple bytes, and as a result the Main Status Register must be read prior to each byte transfer to the 8272A. On the other hand, during the Result Phase, D6 and D7 in the Main Status Register must both be 1's (D6 = 1 and D7 = 1) before reading each byte from the Data Register. Note, this reading of the Main Status Register before each byte transfer to the 8272A is required in only the Command and Result Phases, and NOT during the Execution Phase.

During the Execution Phase, the Main Status Register need not be read. If the 8272A is in the non-DMA Mode, then the receipt of each data byte (if 8272A is reading data from FDD) is indicated by an interrupt signal on pin 18 (INT = 1). The generation of a Read signal (RD = 0) will reset the Interrupt as well as output the Data onto

the Data Bus. For example, if the processor cannot handle Interrupts fast enough (every 13 μs for MFM mode) then it may poll the Main Status Register and then bit D7 (RQM) functions just like the Interrupt signal. If a Write Command is in process, then the WR signal performs the reset to the Interrupt signal.

The 8272A always operates in a multi-sector transfer mode. It continues to transfer data until the TC input is active. In Non-DMA Mode, the system must supply the TC input.

If the 8272A is in the DMA Mode, no Interrupts are generated during the Execution Phase. The 8272A generates DRQ's (DMA Requests) when each byte of data is available. The DMA Controller responds to this request with both a $\overline{DACK} = 0$ (DMA Acknowledge) and a $\overline{RD} = 0$ (Read signal). When the DMA Acknowledge signal goes low ($\overline{DACK} = 0$) then the DMA Request is reset (DRQ = 0). If a Write Command has been programmed then a $\overline{WR}$ signal will appear instead of $\overline{RD}$. After the Execution Phase has been completed (Terminal Count has occurred) then an Interrupt will occur (INT = 1). This signifies the beginning of the Result Phase. When the first byte of data is read during the Result Phase, the Interrupt is automatically reset (INT = 0).

It is important to note that during the Result Phase all bytes shown in the Command Table must be read. The Read Data Command, for example, has seven bytes of data in the Result Phase. All seven bytes must be read in order to successfully complete the Read Data Command. The 8272A will not accept a new command until all seven bytes have been read. Other commands may require fewer bytes to be read during the Result Phase.

The 8272A contains five Status Registers. The Main Status Register mentioned above may be read by the processor at any time. The other four Status Registers (ST0, ST1, ST2, and ST3) are only available during the Result Phase, and may be read only after successfully completing a command. The particular command which has been executed determines how many of the Status Registers will be read.

The bytes of data which are sent to the 8272A to form the Command Phase, and are read out of the 8272A in the Result Phase, must occur in the order shown in the Table 4. That is, the Command Code must be sent first and the other bytes sent in the prescribed sequence. No foreshortening of the Command or Result Phases are allowed. After the last byte of data in the Command Phase is sent to the 8272A, the Execution Phase

Table 4. 8272A Command Set

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ DATA										
Command	W	MT	MFM	SK	0	0	1	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after command execution
	R				H					
	R				R					
	R				N					
READ DELETED DATA										
Command	W	MT	MFM	SK	0	1	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
WRITE DATA										
Command	W	MT	MFM	0	0	0	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the main-system and FDD
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					
WRITE DELETED DATA										
Command	W	MT	MFM	0	0	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W				C					Sector ID information prior to Command execution
	W				H					
	W				R					
	W				N					
	W				EOT					
	W				GPL					
	W				DTL					
Execution										Data transfer between the FDD and main-system
Result	R				ST 0					Status information after Command execution
	R				ST 1					
	R				ST 2					
	R				C					Sector ID information after Command execution
	R				H					
	R				R					
	R				N					

Note: 1. Symbols used in this table are described at the end of this section.
2. $A_0 = 1$ for all operations.
3. X = Don't care, usually made to equal binary 0.

Table 4. 8272A Command Set (Continued)

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
READ A TRACK										
Command	W	0	MFM	SK	0	0	0	1	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	DTL								
Execution										Data transfer between the FDD and main-system. FDC reads all of cylinders contents from index hole to EOT
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
READ ID										
Command	W	0	MFM	0	0	1	0	1	0	Commands
	W	0	0	0	0	0	HDS	DS1	DS0	
Execution										The first correct ID information on the Cylinder is stored in Data Register
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information during Execution Phase
	R	H								
	R	R								
	R	N								
FORMAT A TRACK										
Command	W	0	MFM	0	0	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	N								Bytes/Sector
	W	SC								Sectors/Cylinder
	W	GPL								Gap 3
	W	D								Filler Byte
Execution										FDC formats an entire cylinder
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								In this case, the ID information has no meaning
	R	H								
	R	R								
	R	N								
SCAN EQUAL										
Command	W	MT	MFM	SK	1	0	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior to Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								

PHASE	R/W	DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	REMARKS
SCAN LOW OR EQUAL										
Command	W	MT	MFM	SK	1	1	0	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
SCAN HIGH OR EQUAL										
Command	W	MT	MFM	SK	1	1	1	0	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	C								Sector ID information prior Command execution
	W	H								
	W	R								
	W	N								
	W	EOT								
	W	GPL								
	W	STP								
Execution										Data compared between the FDD and main-system
Result	R	ST 0								Status information after Command execution
	R	ST 1								
	R	ST 2								
	R	C								Sector ID information after Command execution
	R	H								
	R	R								
	R	N								
RECALIBRATE										
Command	W	0	0	0	0	0	1	1	1	Command Codes
	W	0	0	0	0	0	0	DS1	DS0	
Execution										Head retracted to Track 0
SENSE INTERRUPT STATUS										
Command	W	0	0	0	0	1	0	0	0	Command Codes
Result	R	ST 0								Status information at the end of each seek operation about the FDC
	R	PCN								
SPECIFY										
Command	W	0	0	0	0	0	0	1	1	Command Codes
	W	SRT							HUT	
	W	HLT							ND	
SENSE DRIVE STATUS										
Command	W	0	0	0	0	0	1	0	0	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
Result	R	ST 3								Status information about FDD
SEEK										
Command	W	0	0	0	0	1	1	1	1	Command Codes
	W	0	0	0	0	0	HDS	DS1	DS0	
	W	NCN								
Execution										Head is positioned over proper Cylinder on Diskette
INVALID										
Command	W	Invalid Codes								Invalid Command Codes (NoOp — FDC goes into Standby State)
Result	R	ST 0								ST 0 = 80 (16)

Table 5. Command Mneumonics

SYMBOL	NAME	DESCRIPTION
A_0	Address Line 0	A_0 controls selection of Main Status Register (A_0 = 0) or Data Register (A_0 = 1).
C	Cylinder Number	C stands for the current selected Cylinder track number 0 through 76 of the medium.
D	Data	D stands for the data pattern which is going to be written into a Sector.
D_7-D_0	Data Bus	8-bit Data Bus where D_7 is the most significant bit, and D_0 is the least significant bit.
DS0, DS1	Drive Select	DS stands for a selected drive number 0 or 1.
DTL	Data Length	When N is defined as 00, DTL stands for the data length which users are going to read out or write into the Sector.
EOT	End of Track	EOT stands for the final Sector number of a Cylinder.
GPL	Gap Length	GPL stands for the length of Gap 3 (spacing between Sectors excluding VCO Sync Field).
H	Head Address	H stands for head number 0 or 1, as specified in ID field.
HDS	Head Select	HDS stands for a selected head number 0 or 1 (H = HDS in all command words).
HLT	Head Load Time	HLT stands for the head load time in the FDD (2 to 254 ms in 2 ms increments).
HUT	Head Unload Time	HUT stands for the head unload time after a read or write operation has occurred (16 to 240 ms in 16 ms increments).
MFM	FM or MFM Mode	If MF is low, FM mode is selected and if it is high, MFM mode is selected.
MT	Multi-Track	If MT is high, a multi-track operation is to be performed (a cylinder under both HD0 and HD1 will be read or written).
N	Number	N stands for the number of data bytes written in a Sector.
NCN	New Cylinder Number	NCN stands for a new Cylinder number, which is going to be reached as a result of the Seek operation. Desired position of Head.
ND	Non-DMA Mode	ND stands for operation in the Non-DMA Mode.
PCN	Present Cylinder Number	PCN stands for the Cylinder number at the completion of SENSE INTERRUPT STATUS Command. Position of Head at present time.
R	Record	R stands for the Sector number, which will be read or written.
R/W	Read/Write	R/W stands for either Read (R) or Write (W) signal.
SC	Sector	SC indicates the number of Sectors per Cylinder.
SK	Skip	SK stands for Skip Deleted Data Address Mark.
SRT	Step Rate Time	SRT stands for the Stepping Rate for the FDD (1 to 16 ms in 1 ms increments). The same Stepping Rate applies to all drives (F=1 ms, E=2 ms, etc.).
ST 0 ST 1 ST 2 ST 3	Status 0 Status 1 Status 2 Status 3	ST 0-3 stand for one of four registers which store the status information after a command has been executed. This information is available during the result phase after command execution. These registers should not be confused with the main status register (selected by A_0 = 0). ST 0-3 may be read only after a command has been executed and contain information relevant to that particular command.
STP		During a Scan operation, if STP = 1, the data in contiguous sectors is compared byte by byte with data sent from the processor (or DMA), and if STP = 2, then alternate sectors are read and compared.

automatically starts. In a similar fashion, when the last byte of data is read out in the Result Phase, the command is automatically ended and the 8272A is ready for a new command. A command may be aborted by simply sending a Terminal Count signal to pin 16 (TC = 1). This is a convenient means of ensuring that the processor may always get the 8272A's attention even if the disk system hangs up in an abnormal manner.

POLLING FEATURE OF THE 8272A

After power-up RESET, the Drive Select Lines DS0 and DS1 will automatically go into a polling mode. In between commands (and between step pulses in the SEEK command) the 8272A polls all four FDDs looking for a change in the Ready line from any of the drives. If the Ready line changes state (usually due to a door opening or closing) then the 8272A will generate an interrupt. When Status Register 0 (ST0) is read (after Sense Interrupt Status is issued), Not Ready (NR) will be indicated. The polling of the Ready line by the 8272A occurs continuously between instructions, thus notifying the processor which drives are on or off line. Approximate scan timing is shown in Table 6.

Table 6. Scan Timing

DS1	DS0	APPROXIMATE SCAN TIMING
0	0	220μS
0	1	220μS
1	0	220μS
1	1	440μS

COMMAND DESCRIPTIONS

During the Command Phase, the Main Status Register must be polled by the CPU before each byte is written into the Data Register. The DIO (DB6) and RQM (DB7) bits in the Main Status Register must be in the "0" and "1" states respectively, before each byte of the command may be written into the 8272A. The beginning of the execution phase for any of these commands will cause DIO and RQM to switch to "1" and "0" states respectively.

READ DATA

A set of nine (9) byte words are required to place the FDC into the Read Data Mode. After the Read Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Address Marks and ID fields. When the current sector number ("R") stored in the ID Register (IDR) compares with the sector number read off the diskette, then the FDC outputs data (from the data field) byte-by-byte to the main system via the data bus.

After completion of the read operation from the current sector, the Sector Number is incremented by one, and the data from the next sector is read and output on the data bus. This continuous read function is called a "Multi-Sector Read Operation." The Read Data Command must be terminated by the receipt of a Terminal Count signal. Upon receipt of this signal, the FDC stops outputting data to the processor, but will continue to read data from the current sector, check CRC (Cyclic Redundancy Count) bytes, and then at the end of the sector terminate the Read Data Command.

The amount of data which can be handled with a single command to the FDC depends upon MT (multi-track), MFM (MFM/FM), and N (Number of Bytes/Sector). Table 7 on the next page shows the Transfer Capacity.

Table 7. Transfer Capacity

Multi-Track MT	MFM/FM MFM	Bytes/Sector N	Maximum Transfer Capacity (Bytes/Sector) (Number of Sectors)	Final Sector Read from Diskette
0	0	00	(128) (26) = 3,328	26 at Side 0
0	1	01	(256) (26) = 6,656	or 26 at Side 1
1	0	00	(128) (52) = 6,656	26 at Side 1
1	1	01	(256) (52) = 13,312	
0	0	01	(256) (15) = 3,840	15 at Side 0
0	1	02	(512) (15) = 7,680	or 15 at Side 1
1	0	01	(256) (30) = 7,680	15 at Side 1
1	1	02	(512) (30) = 15,360	
0	0	02	(512) (8) = 4,096	8 at Side 0
0	1	03	(1024) (8) = 8,192	or 8 at Side 1
1	0	02	(512) (16) = 8,192	8 at Side 1
1	1	03	(1024) (16) = 16,384	

The "multi-track" function (MT) allows the FDC to read data from both sides of the diskette. For a particular cylinder, data will be transferred starting at Sector 1, Side 0 and completing at Sector L, Side 1 (Sector L = last sector on the side). Note, this function pertains to only one cylinder (the same track) on each side of the diskette.

When N = 0, then DTL defines the data length which the FDC must treat as a sector. If DTL is smaller than the actual data length in a Sector, the data beyond DTL in the Sector is not sent to the Data Bus. The FDC reads (internally) the complete Sector performing the CRC check, and depending upon the manner of command termination, may perform a Multi-Sector Read Operation. When N is non-zero, then DTL has no meaning and should be set to 0FFH.

At the completion of the Read Data Command, the head is not unloaded until after Head Unload Time Interval (specified in the Specify Command) has elapsed. If the processor issues another command before the head unloads then the head settling time may be saved between subsequent reads. This time out is particularly valuable when a diskette is copied from one drive to another.

If the FDC detects the Index Hole twice without finding the right sector, (indicated in "R"), then the FDC sets the ND (No Data) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

After reading the ID and Data Fields in each sector, the FDC checks the CRC bytes. If a read error is detected (incorrect CRC in ID field), the FDC sets the DE (Data Error) flag in Status Register 1 to a 1 (high), and if a CRC error occurs in the Data Field the FDC also sets the DD (Data Error in Data Field) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)

If the FDC reads a Deleted Data Address Mark off the diskette, and the SK bit (bit D5 in the first Command Word) is not set (SK = 0), then the FDC sets the CM (Control Mark) flag in Status Register 2 to a 1 (high), and terminates the Read Data Command, after reading all the data in the Sector. If SK = 1, the FDC skips the sector with the Deleted Data Address Mark and reads the next sector.

During disk data transfers between the FDC and the processor, via the data bus, the FDC must be serviced by the processor every 27 μs in the FM Mode, and every 13 μs in the MFM Mode, or the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Read Data Command.

If the processor terminates a read (or write) operation in the FDC, then the ID Information in the Result Phase is dependent upon the state of the MT bit and EOT byte. Table 5 shows the values for C, H, R, and N, when the processor terminates the Command.

Table 8. ID Information When Processor Terminates Command

MT	EOT	Final Sector Transferred to Processor	ID Information at Result Phase C	H	R	N
0	1A	Sector 1 to 25 at Side 0	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	C+1	NC	R=01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C+1	NC	R=01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				
1	1A	Sector 1 to 25 at Side 0	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 0				
	08	Sector 1 to 7 at Side 0				
	1A	Sector 26 at Side 0	NC	LSB	R=01	NC
	0F	Sector 15 at Side 0				
	08	Sector 8 at Side 0				
	1A	Sector 1 to 25 at Side 1	NC	NC	R+1	NC
	0F	Sector 1 to 14 at Side 1				
	08	Sector 1 to 7 at Side 1				
	1A	Sector 26 at Side 1	C+1	LSB	R=01	NC
	0F	Sector 15 at Side 1				
	08	Sector 8 at Side 1				

Notes: 1 NC (No Change): The same value as the one at the beginning of command execution.

2. LSB (Least Significant Bit). The least significant bit of H is complemented

WRITE DATA

A set of nine (9) bytes are required to set the FDC into the Write Data mode. After the Write Data command has been issued the FDC loads the head (if it is in the unloaded state), waits the specified head settling time (defined in the Specify Command), and begins reading ID Fields. When the current sector number ("R"), stored in the ID Register (IDR) compares with the sector

number read off the diskette, then the FDC takes data from the processor byte-by-byte via the data bus, and outputs it to the FDD.
After writing data into the current sector, the Sector Number stored in "R" is incremented by one, and the next data field is written into. The FDC continues this "Multi-Sector Write Operation" until the issuance of a Terminal Count signal. If a Terminal Count signal is sent to the FDC it continues writing into the current sector to complete the data field. If the Terminal Count signal is received while a data field is being written then the remainder of the data field is filled with 00 (zeros).
The FDC reads the ID field of each sector and checks the CRC bytes. If the FDC detects a read error (incorrect CRC) in one of the ID Fields, it sets the DE (Data Error) flag of Status Register 1 to a 1 (high), and terminates the Write Data Command. (Status Register 0 also has bits 7 and 6 set to 0 and 1 respectively.)
The Write Command operates in much the same manner as the Read Command. The following items are the same; refer to the Read Data Command for details:

- Transfer Capacity
- EN (End of Cylinder) Flag
- ND (No Data) Flag
- Head Unload Time Interval
- ID Information when the processor terminates command (see Table 2)
- Definition of DTL when N = 0 and when N ≠ 0

In the Write Data mode, data transfers between the processor and FDC must occur every 31 μs in the FM mode, and every 15 μs in the MFM mode. If the time interval between data transfers is longer than this then the FDC sets the OR (Over Run) flag in Status Register 1 to a 1 (high), and terminates the Write Data Command.
For mini-floppies, multiple track writes are usually not permitted. This is because of the turn-off time of the erase head coils—the head switches tracks before the erase head turns off. Therefore the system should typically wait 1.3 mS before attempting to step or change sides.

WRITE DELETED DATA

This command is the same as the Write Data Command except a Deleted Data Address Mark is written at the beginning of the Data Field instead of the normal Data Address Mark.

READ DELETED DATA

This command is the same as the Read Data Command except that when the FDC detects a Data Address Mark at the beginning of a Data Field (and SK = 0 (low)), it will read all the data in the sector and set the CM flag in Status Register 2 to a 1 (high), and then terminate the command. If SK = 1, then the FDC skips the sector with the Data Address Mark and reads the next sector.

READ A TRACK

This command is similar to READ DATA Command except that the entire data field is read continuously from each of the sectors of a track. Immediately after encountering the INDEX HOLE, the FDC starts reading all data fields on the track as continuous blocks of data. If the FDC finds an error in the ID or DATA CRC check bytes, it continues to read data from the track. The FDC compares the ID information read from each sector with the value stored in the IDR, and sets the ND flag of Status Register 1 to a 1 (high) if there is no comparison. Multi-track or skip operations are not allowed with this command.
This command terminates when EOT number of sectors have been read. If the FDC does not find an ID Address Mark on the diskette after it encounters the INDEX HOLE for the second time, then it sets the MA (missing address mark) flag in Status Register 1 to a 1 (high), and terminates the command. (Status Register 0 has bits 7 and 6 set to 0 and 1 respectively.)

READ ID

The READ ID Command is used to give the present position of the recording head. The FDC stores the values from the first ID Field it is able to read. If no proper ID Address Mark is found on the diskette, before the INDEX HOLE is encountered for the second time then the MA (Missing Address Mark) flag in Status Register 1 is set to a 1 (high), and if no data is found then the ND (No Data) flag is also set in Status Register 1 to a 1 (high) and the command is terminated.

FORMAT A TRACK

The Format Command allows an entire track to be formatted. After the INDEX HOLE is detected, Data is written on the Diskette: Gaps, Address Marks, ID Fields and Data Fields, all per the IBM System 34 (Double Density) or System 3740 (Single Density) Format are recorded. The particular format which will be written is controlled by the values programmed into N (number of bytes/sector), SC (sectors/cylinder), GPL (Gap Length), and D (Data Pattern) which are supplied by the processor during the Command Phase. The Data Field is filled with the Byte of data stored in D. The ID Field for each sector is supplied by the processor; that is, four data requests per sector are made by the FDC for C (Cylinder Number), H (Head Number), R (Sector Number) and N (Number of Bytes/Sector). This allows the diskette to be formatted with nonsequential sector numbers, if desired.

After formatting each sector, the processor must send new values for C, H, R, and N to the 8272A for each sector on the track. The contents of the R Register is incremented by one after each sector is formatted, thus, the R register contains a value of R + 1 when it is read during the Result Phase. This incrementing and formatting continues for the whole track until the FDC encounters the INDEX HOLE for the second time, whereupon it terminates the command.

If a FAULT signal is received from the FDD at the end of a write operation, then the FDC sets the EC flag of Status Register 0 to a 1 (high), and terminates the command after setting bits 7 and 6 of Status Register 0 to 0 and 1 respectively. Also the loss of a READY signal at the beginning of a command execution phase causes command termination.

Table 9 shows the relationship between N, SC, and GPL for various sector sizes:

Table 9. Sector Size Relationships.

	8" STANDARD FLOPPY						5¼" MINI FLOPPY				
FORMAT	SECTOR SIZE	N	SC	GPL[1]	GPL[2]	REMARKS	SECTOR SIZE	N	SC	GPL[1]	GPL[2]
FM Mode	128 bytes/Sector	00	1A	07	1B	IBM Diskette 1	128 bytes/Sector	00	12	07	09
	256	01	0F	0E	2A	IBM Diskette 2	128	00	10	10	19
	512	02	08	1B	3A		256	01	08	18	30
	1024	03	04	47	8A		512	02	04	46	87
	2048	04	02	C8	FF		1024	03	02	C8	FF
	4096	05	01	C8	FF		2048	04	01	C8	FF
MPM Mode	256	01	1A	0E	36	IBM Diskette 2D	256	01	12	0A	0C
	512	02	0F	1B	54		256	01	10	20	32
	1024	03	08	35	74	IBM Diskette 2D	512	02	08	2A	50
	2048	04	04	99	FF		1024	03	04	80	F0
	4096	05	02	C8	FF		2048	04	02	C8	FF
	8192	06	01	C8	FF		4096	05	01	C8	FF

Note: 1. Suggested values of GPL in Read or Write Commands to avoid splice point between data field and ID field of contiguous sections.
2. Suggested values of GPL in format command.

SCAN COMMANDS

The SCAN Commands allow data which is being read from the diskette to be compared against data which is being supplied from the main system (Processor in NON-DMA mode, and DMA Controller in DMA mode). The FDC compares the data on a byte-by-byte basis, and looks for a sector of data which meets the conditions of $D_{FDD} = D_{Processor}$, $D_{FDD} \leq D_{Processor}$, or $D_{FDD} \geq D_{Processor}$. Ones complement arithmetic is used for comparison (FF = largest number, 00 = smallest number). After a whole sector of data is compared, if the conditions are not met, the sector number is incremented (R + STP → R), and the scan operation is continued. The scan operation continues until one of the following conditions occur; the conditions for scan are met (equal, low, or high), the last sector on the track is reached (EOT), or the terminal count signal is received.

If the conditions for scan are met then the FDC sets the SH (Scan Hit) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. If the conditions for scan are not met between the starting sector (as specified by R) and the last sector on the cylinder (EOT), then the FDC sets the SN (Scan Not Satisfied) flag of Status Register 2 to a 1 (high), and terminates the Scan Command. The receipt of a TERMINAL COUNT signal from the Processor or DMA Controller during the scan operation will cause the FDC to complete the comparison of the particular byte which is in process, and then to terminate the command. Table 10 shows the status of bits SH and SN under various conditions of SCAN.

Table 10. Scan Status Codes

COMMAND	STATUS REGISTER 2		COMMENTS
	BIT 2 = SN	BIT 3 = SH	
Scan Equal	0	1	$D_{FDD} = D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan Low or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} < D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$
Scan High or Equal	0	1	$D_{FDD} = D_{Processor}$
	0	0	$D_{FDD} > D_{Processor}$
	1	0	$D_{FDD} \neq D_{Processor}$

If the FDC encounters a Deleted Data Address Mark on one of the sectors (and SK = 0), then it regards the sector as the last sector on the cylinder, sets CM (Control Mark) flag of Status Register 2 to a 1 (high) and terminates the command. If SK = 1, the FDC skips the sector with the Deleted Address Mark, and reads the next sector. In the second case (SK = 1), the FDC sets the CM (Control Mark) flag of Status Register 2 to a 1 (high) in order to show that a Deleted Sector had been encountered.

When either the STP (contiguous sectors STP = 01, or alternate sectors STP = 02 sectors are read) or the MT (Multi-Track) are programmed, it is necessary to remember that the last sector on the track must be read. For example, if STP = 02, MT = 0, the sectors are numbered sequentially 1 through 26, and we start the Scan Command at sector 21; the following will happen. Sectors 21, 23, and 25 will be read, then the next sector (26) will be skipped and the Index Hole will be encountered before the EOT value of 26 can be read. This will result in an abnormal termination of the command. If the EOT had been set at 25 or the scanning started at sector 20, then the Scan Command would be completed in a normal manner.

During the Scan Command data is supplied by either the processor or DMA Controller for comparison against the data read from the diskette. In order to avoid having the OR (Over Run) flag set in Status Register 1, it is necessary to have the data available in less than 27 μs (FM Mode) or 13 μs (MFM Mode). If an Overrun occurs the FDC terminates the command.

SEEK

The read/write head within the FDD is moved from cylinder to cylinder under control of the Seek Command. The FDC compares the PCN (Present Cylinder Number) which is the current head position with the NCN (New Cylinder Number), and performs the following operation if there is a difference:

PCN < NCN: Direction signal to FDD set to a 1 (high), and Step Pulses are issued. (Step In.)

PCN > NCN: Direction signal to FDD set to a 0 (low), and Step Pulses are issued. (Step Out.)

The rate at which Step Pulses are issued is controlled by SRT (Stepping Rate Time) in the SPECIFY Command. After each Step Pulse is issued NCN is compared against PCN, and when NCN = PCN, then the SE (Seek End) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

During the Command Phase of the Seek operation the FDC is in the FDC BUSY state, but during the Execution Phase it is in the NON BUSY state. While the FDC is in the NON BUSY state, another Seek Command may be issued, and in this manner parallel seek operations may be done on up to 4 Drives at once.

If an FDD is in a NOT READY state at the beginning of the command execution phase or during the seek operation, then the NR (NOT READY) flag is set in Status Register 0 to a 1 (high), and the command is terminated.

Note that the 8272A Read and Write Commands do not have implied Seeks. Any R/W command should be preceded by: 1) Seek Command; 2) Sense Interrupt Status; and 3) Read ID.

RECALIBRATE

This command causes the read/write head within the FDD to retract to the Track 0 position. The FDC clears the contents of the PCN counter, and checks the status of the Track 0 signal from the FDD. As long as the Track 0 signal is low, the Direction signal remains 1 (high) and Step Pulses are issued. When the Track 0 signal goes high, the SE (SEEK END) flag in Status Register 0 is set to a 1 (high) and the command is terminated. If the Track 0 signal is still low after 77 Step Pulses have been issued, the FDC sets the SE (SEEK END) and EC (EQUIPMENT CHECK) flags of Status Register 0 to both 1s (highs), and terminates the command.

The ability to overlap RECALIBRATE Commands to multiple FDDs, and the loss of the READY signal, as described in the SEEK Command, also applies to the RECALIBRATE Command.

SENSE INTERRUPT STATUS

An Interrupt signal is generated by the FDC for one of the following reasons:

1. Upon entering the Result Phase of:
   a. Read Data Command
   b. Read a Track Command
   c. Read ID Command
   d. Read Deleted Data Command
   e. Write Data Command
   f. Format a Cylinder Command
   g. Write Deleted Data Command
   h. Scan Commands
2. Ready Line of FDD changes state
3. End of Seek or Recalibrate Command
4. During Execution Phase in the NON-DMA Mode

Interrupts caused by reasons 1 and 4 above occur during normal command operations and are easily discernible by the processor. However, interrupts caused by reasons 2 and 3 above may be uniquely identified with the aid of the Sense Interrupt Status Command. This command when issued resets the interrupt signal and via bits 5, 6, and 7 of Status Register 0 identifies the cause of the interrupt.

Neither the Seek or Recalibrate Command have a Result Phase. Therefore, it is mandatory to use the Sense Interrupt Status Command after these commands to effectively terminate them and to provide verification of the head position (PCN).

Table 11. Seek, Interrupt Codes

SEEK END BIT 5	INTERRUPT CODE		CAUSE
	BIT 6	BIT 7	
0	1	1	Ready Line changed state, either polarity
1	0	0	Normal Termination of Seek or Recalibrate Command
1	1	0	Abnormal Termination of Seek or Recalibrate Command

SPECIFY

The Specify Command sets the initial values for each of the three internal timers. The HUT (Head Unload Time) defines the time from the end of the Execution Phase of one of the Read/Write Commands to the head unload state. This timer is programmable from 16 to 240 ms in increments of 16 ms (01 = 16 ms, 02 = 32 ms .... OF = 240 ms). The SRT (Step Rate Time) defines the time interval between adjacent step pulses. This timer is programmable from 1 to 16 ms in increments of 1 ms (F = 1 ms, E = 2 ms, D = 3 ms, etc.). The HLT (Head Load Time) defines the time between when the Head Load signal goes high and when the Read/Write operation starts. This timer is programmable from 2 to 254 ms in increments of 2 ms (01 = 2 ms, 02 = 4 ms, 03 = 6 ms .... FE = 254 ms).

The step rate should be programmed 1 mS longer than the minimum time required by the drive.

The time intervals mentioned above are a direct function of the clock (CLK on pin 19). Times indicated above are for an 8 MHz clock, if the clock was reduced to 4 MHz (mini-floppy application) then all time intervals are increased by a factor of 2.

The choice of DMA or NON-DMA operation is made by the ND (NON-DMA) bit. When this bit is high (ND = 1) the NON-DMA mode is selected, and when ND = 0 the DMA mode is selected.

SENSE DRIVE STATUS

This command may be used by the processor whenever it wishes to obtain the status of the FDDs. Status Register 3 contains the Drive Status information.

INVALID

If an invalid command is sent to the FDC (a command not defined above), then the FDC will terminate the command. No interrupt is generated by the 8272A during this condition. Bit 6 and bit 7 (DIO and RQM) in the Main Status Register are both high ("1") indicating to the processor that the 8272A is in the Result Phase and the contents of Status Register 0 (ST0) must be read. When the processor reads Status Register 0 it will find an 80H indicating an invalid command was received.

A Sense Interrupt Status Command must be sent after a Seek or Recalibrate interrupt, otherwise the FDC will consider the next command to be an Invalid Command.

In some applications the user may wish to use this command as a No-Op command, to place the FDC in a standby or no operation state.

Table 12. Status Registers

BIT NO.	BIT NAME	BIT SYMBOL	DESCRIPTION
STATUS REGISTER 0			
D_7	Interrupt Code	IC	$D_7 = 0$ and $D_6 = 0$ Normal Termination of Command, (NT). Command was completed and properly executed.
D_6			$D_7 = 0$ and $D_6 = 1$ Abnormal Termination of Command, (AT). Execution of Command was started, but was not successfully completed.
			$D_7 = 1$ and $D_6 = 0$ Invalid Command issue, (IC). Command which was issued was never started.
			$D_7 = 1$ and $D_6 = 1$ Abnormal Termination because during command execution the ready signal from FDD changed state.
D_5	Seek End	SE	When the FDC completes the SEEK Command, this flag is set to 1 (high).
D_4	Equipment Check	EC	If a fault Signal is received from the FDD, or if the Track 0 Signal fails to occur after 77 Step Pulses (Recalibrate Command) then this flag is set.
D_3	Not Ready	NR	When the FDD is in the not-ready state and a read or write command is issued, this flag is set. If a read or write command is issued to Side 1 of a single sided drive, then this flag is set.
D_2	Head Address	HD	This flag is used to indicate the state of the head at Interrupt.
D_1	Unit Select 1	US 1	These flags are used to indicate a
D_0	Unit Select 0	US 0	Drive Unit Number at Interrupt
STATUS REGISTER 1			
D_7	End of Cylinder	EN	When the FDC tries to access a Sector beyond the final Sector of a Cylinder, this flag is set.
D_6			Not used This bit is always 0 (low).
D_5	Data Error	DE	When the FDC detects a CRC error in either the ID field or the data field, this flag is set.
D_4	Over Run	OR	If the FDC is not serviced by the main-systems during data transfers, within a certain time interval, this flag is set.
D_3			Not used. This bit always 0 (low).
D_2	No Data	ND	During execution of READ DATA, WRITE DELETED DATA or SCAN Command, if the FDC cannot find the Sector specified in the IDR Register, this flag is set.
			During executing the READ ID Command, if the FDC cannot read the ID field without an error, then this flag is set.
			During the execution of the READ A Cylinder Command, if the starting sector cannot be found, then this flag is set.

BIT NO.	BIT NAME	BIT SYMBOL	DESCRIPTION
STATUS REGISTER 1 (CONT.)			
D_1	Not Writable	NW	During execution of WRITE DATA, WRITE DELETED DATA or Format A Cylinder Command, if the FDC detects a write protect signal from the FDD, then this flag is set.
D_0	Missing Address Mark	MA	If the FDC cannot detect the ID Address Mark after encountering the index hole twice, then this flag is set.
			If the FDC cannot detect the Data Address Mark or Deleted Data Address Mark, this flag is set. Also at the same time, the MD (Missing Address Mark in Data Field) of Status Register 2 is set.
STATUS REGISTER 2			
D_7			Not used. This bit is always 0 (low).
D_6	Control Mark	CM	During executing the READ DATA or SCAN Command, if the FDC encounters a Sector which contains a Deleted Data Address Mark, this flag is set.
D_5	Data Error in Data Field	DD	If the FDC detects a CRC error in the data field then this flag is set.
D_4	Wrong Cylinder	WC	This bit is related with the ND bit, and when the contents of C on the medium is different from that stored in the IDR, this flag is set.
D_3	Scan Equal Hit	SH	During execution, the SCAN Command, if the condition of "equal" is satisfied, this flag is set.
D_2	Scan Not Satisfied	SN	During executing the SCAN Command, if the FDC cannot find a Sector on the cylinder which meets the condition, then this flag is set.
D_1	Bad Cylinder	BC	This bit is related with the ND bit, and when the content of C on the medium is different from that stored in the IDR and the content of C is FF, then this flag is set.
D_0	Missing Address Mark in Data Field	MD	When data is read from the medium, if the FDC cannot find a Data Address Mark or Deleted Data Address Mark, then this flag is set.
STATUS REGISTER 3			
D_7	Fault	FT	This bit is used to indicate the status of the Fault signal from the FDD.
D_6	Write Protected	WP	This bit is used to indicate the status of the Write Protected signal from the FDD.
D_5	Ready	RDY	This bit is used to indicate the status of the Ready signal from the FDD.
D_4	Track 0	T0	This bit is used to indicate the status of the Track 0 signal from the FDD.
D_3	Two Side	TS	This bit is used to indicate the status of the Two Side signal from the FDD.
D_2	Head Address	HD	This bit is used to indicate the status of Side Select signal to the FDD.
D_1	Unit Select 1	US 1	This bit is used to indicate the status of the Unit Select 1 signal to the FDD.
D_0	Unit Select 0	US 0	This bit is used to indicate the status of the Unit Select 0 signal to the FDD.

ABSOLUTE MAXIMUM RATINGS*

Operating Temperature ..... 0°C to +70°C
Storage Temperature ..... −40°C to +125°C
All Output Voltages ..... −0.5 to +7 Volts
All Input Voltages ..... −0.5 to +7 Volts
Supply Voltage V_{CC} ..... −0.5 to +7 Volts
Power Dissipation ..... 1 Watt

*T_A = 25°C

NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5V ±10%)

Symbol	Parameter	Limits		Unit	Test Conditions
		Min.	Max.		
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	V	I_{OH} = −400 μA
I_{CC}	V_{CC} Supply Current		120	mA	
I_{IL}	Input Load Current (All Input Pins)		10 −10	μA μA	V_{IN} = V_{CC} V_{IN} = 0V
I_{LOH}	High Level Output Leakage Current		10	μA	V_{OUT} = V_{CC}
I_{OFL}	Output Float Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}

CAPACITANCE (T_A = 25°C, f_c = 1 MHz, V_{CC} = 0V)

Symbol	Parameter	Limits		Unit	Test Conditions
		Min.	Max.		
$C_{IN(\Phi)}$	Clock Input Capacitance		20	pF	All Pins Except Pin Under Test Tied to AC Ground
C_{IN}	Input Capacitance		10	pF	
$C_{I/O}$	Input/Output Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5.0V ±10%)

CLOCK TIMING

Symbol	Parameter	Min.	Max.	Unit	Notes
t_{CY}	Clock Period	120	500	ns	Note 5
t_{CH}	Clock High Period	40		ns	Note 4, 5
t_{RST}	Reset Width	14		t_{CY}	

READ CYCLE

t_{AR}	Select Setup to $\overline{RD}\downarrow$	0		ns	
t_{RA}	Select Hold from $\overline{RD}\uparrow$	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{RD}	Data Delay from $\overline{RD}\downarrow$		200	ns	
t_{DF}	Output Float Delay	20	100	ns	

A.C. CHARACTERISTICS (Continued) ($T_A = 0°C$ to $+70°C$, $V_{CC} = +5.0V \pm 10\%$)

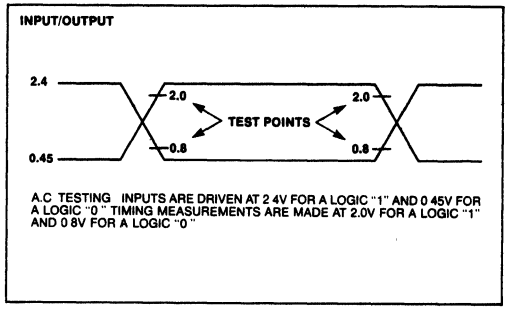
WRITE CYCLE

Symbol	Parameter	Typ.[1]	Min.	Max.	Unit	Notes
tAW	Select Setup to $\overline{WR}\downarrow$		0		ns	
tWA	Select Hold from $\overline{WR}\uparrow$		0		ns	
tWW	$\overline{WR}$ Pulse Width		250		ns	
tDW	Data Setup to $\overline{WR}\uparrow$		150		ns	
tWD	Data Hold from $\overline{WR}\uparrow$		5		ns	
INTERRUPTS						
tRI	INT Delay from $\overline{RD}\uparrow$			500	ns	Note 6
tWI	INT Delay from $\overline{WR}\uparrow$			500	ns	Note 6
DMA						
tRQCY	DRQ Cycle Period		13		µs	Note 6
tAKRQ	$\overline{DACK}\downarrow$ to DRQ↓			200	ns	
tRQR	DRQ↑ to $\overline{RD}\downarrow$		800		ns	Note 6
tRQW	DRQ↑ to $\overline{WR}\downarrow$		250		ns	Note 6
tRQRW	DRQ↑ to $\overline{RD}\uparrow$ or $\overline{WR}\uparrow$			12	µs	Note 6
FDD INTERFACE						
tWCY	WCK Cycle Time	2 or 4 1 or 2			µs	MFM = 0 MFM = 1 Note 2
tWCH	WCK High Time	250	80	350	ns	
tCP	Pre-Shift Delay from WCK↑		20	100	ns	
tCD	WDA Delay from WCK↑		20	100	ns	
tWDD	Write Data Width		tWCH − 50		ns	
tWE	WE↑ to WCK↑ or WE↓ to WCK↓ Delay		20	100	ns	
tWWCY	Window Cycle Time	2 1			µs	MFM = 0 MFM = 1
tWRD	Window Setup to RDD↑		15		ns	
tRDW	Window Hold from RDD↓		15		ns	
tRDD	RDD Active Time (HIGH)		40		ns	
FDD SEEK/DIRECTION/STEP						
tUS	$US_{0,1}$ Setup to $\overline{RW}$/SEEK↑		12		µs	Note 6
tSU	$US_{0,1}$ Hold after $\overline{RW}$/SEEK↓		15		µs	Note 6
tSD	$\overline{RW}$/SEEK Setup to LCT/DIR		7		µs	Note 6
tDS	$\overline{RW}$/SEEK Hold from LCT/DIR		30		µs	Note 6
tDST	LCT/DIR Setup to FR/STEP↑		1		µs	Note 6
tSTD	LCT/DIR Hold from FR/STEP↓		24		µs	Note 6
tSTU	$DS_{2,1}$ Hold from FR/Step↓		5		µs	Note 6
tSTP	STEP Active Time (High)	5			µs	Note 6
tSC	STEP Cycle Time		33		µs	Note 3, 6
tFR	FAULT RESET Active Time (High)		8	10	µs	Note 6
tIDX	INDEX Pulse Width	10			tCY	
tTC	Terminal Count Width		1		tCY	

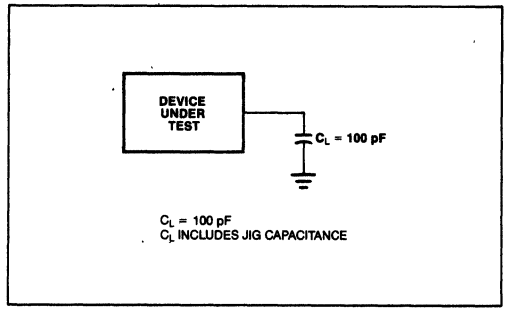
NOTES:

1. Typical values for $T_A = 25°C$ and nominal supply voltage.
2. The former values are used for standard floppy and the latter values are used for mini-floppies.
3. $tSC = 33$ µs min. is for different drive units. In the case of same unit, tSC can be ranged from 1 ms to 16 ms with 8 MHz clock period, and 2 ms to 32 ms with 4 MHz clock, under software control.
4. From 2.0V to +2.0V.
5. At 4 MHz, the clock duty cycle may range from 16% to 76%. Using an 8 MHz clock the duty cycle can range from 32% to 52%. Duty cycle is defined as: D.C. = 100 (tCH ÷ tCY) with typical rise and fall times of 5 ns.
6. The specified values listed are for an 8 MHz clock period. Multiply timings by 2 when using a 4 MHz clock period.

A.C. TESTING INPUT, OUTPUT WAVEFORM

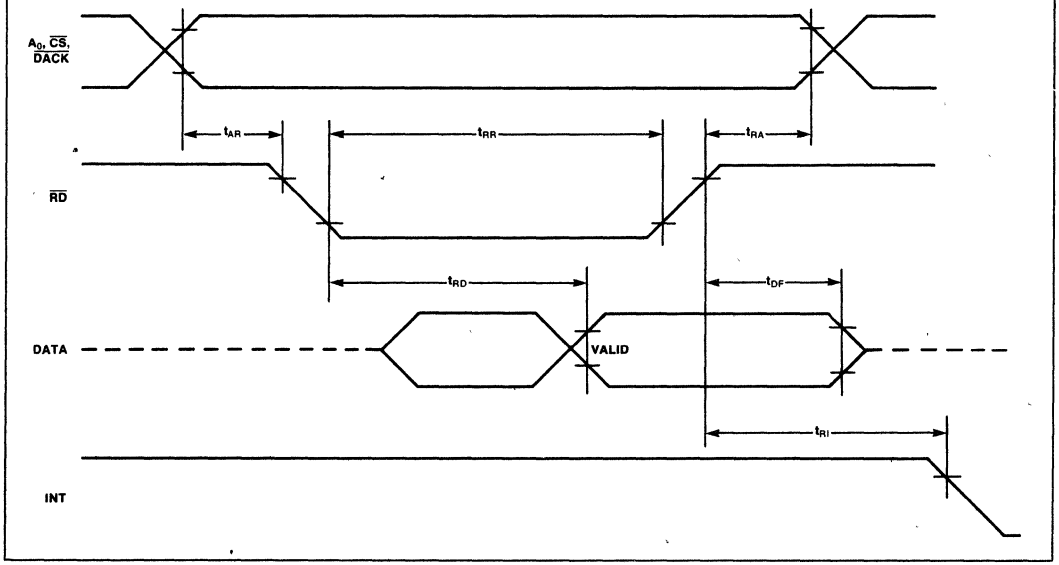


A.C. TESTING LOAD CIRCUIT



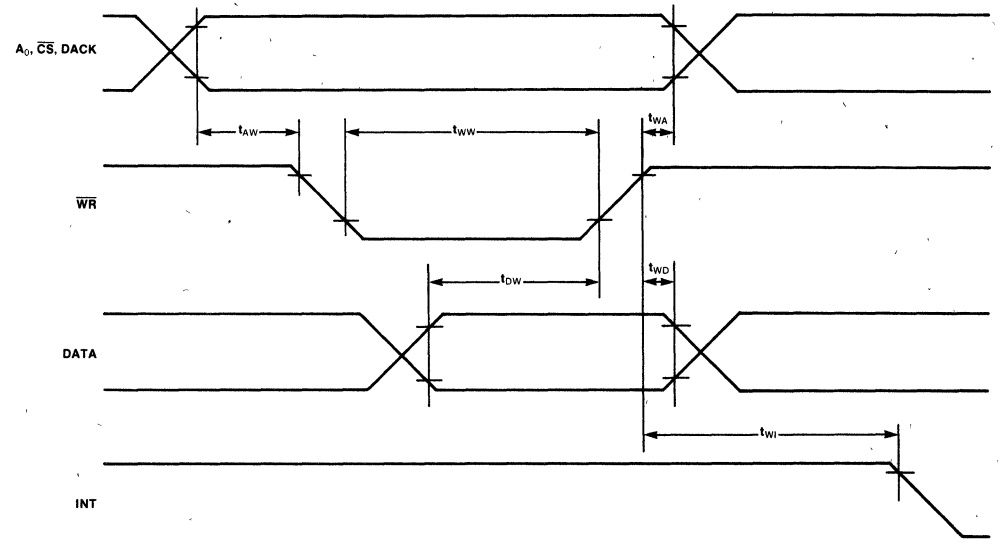
WAVEFORMS

PROCESSOR READ OPERATION

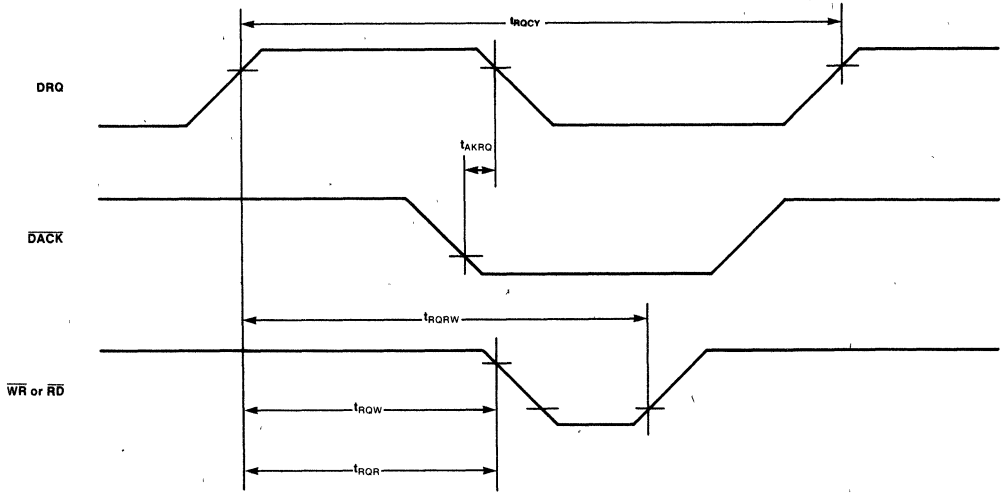


WAVEFORMS (Continued)

PROCESSOR WRITE OPERATION

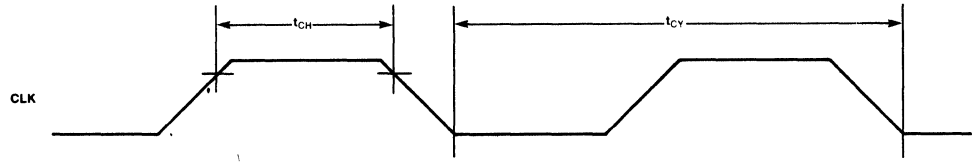


DMA OPERATION

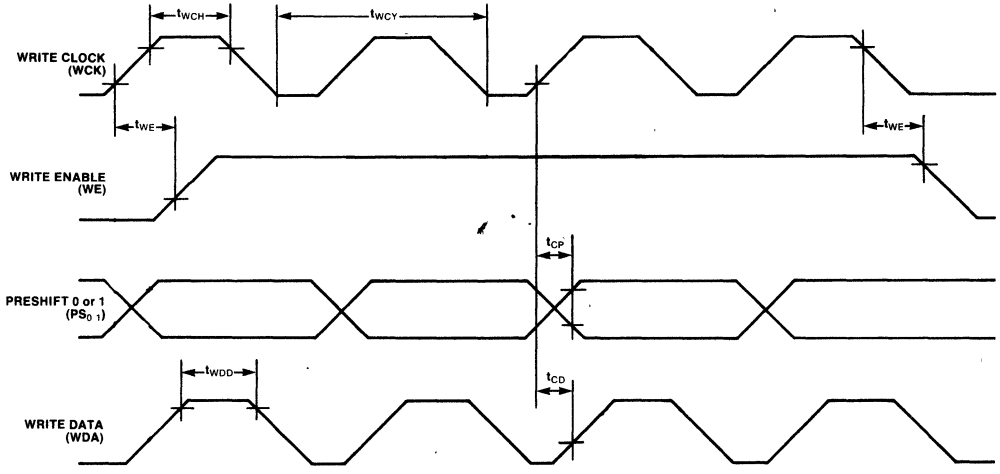


WAVEFORMS (Continued)

CLOCK TIMING



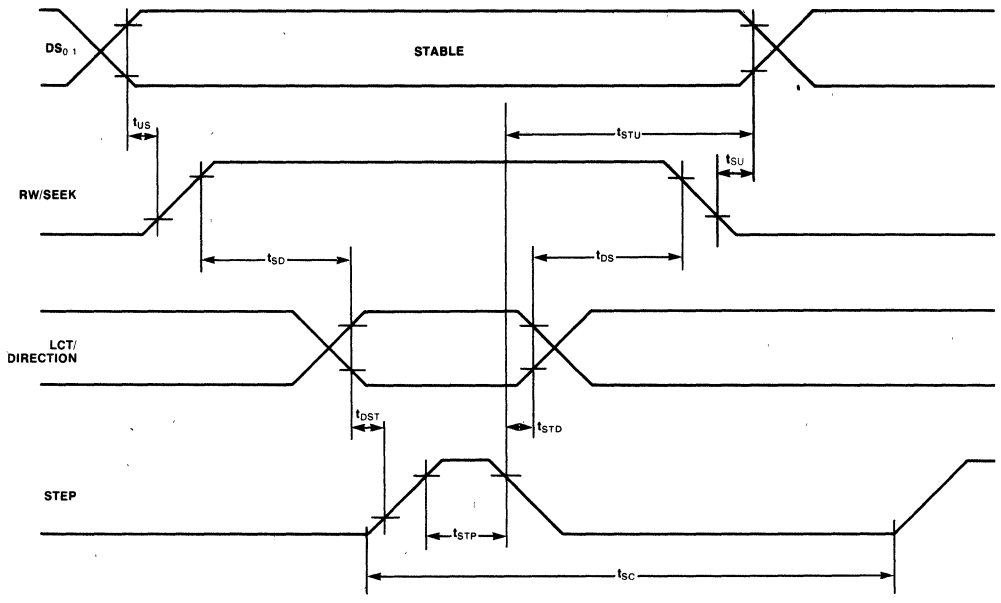
FDD WRITE OPERATION



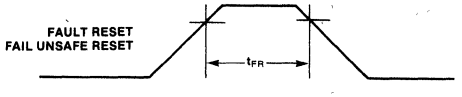
	PRESHIFT 0	PRESHIFT 1
NORMAL	0	0
LATE	0	1
EARLY	1	0
INVALID	1	1

WAVEFORMS (Continued)

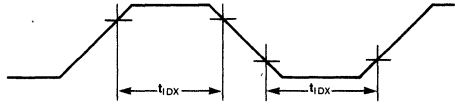
SEEK OPERATION



FLT RESET

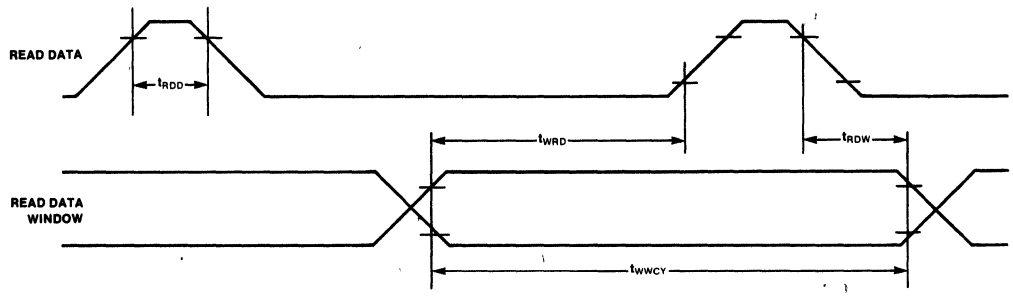


INDEX

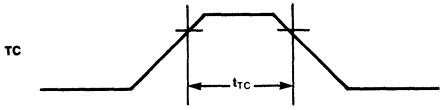


WAVEFORMS (Continued)

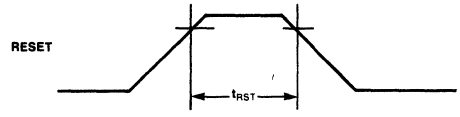
FDD READ OPERATION



TERMINAL COUNT



RESET



82062
WINCHESTER DISK CONTROLLER

- **Controls ST506/ST412 Interface Winchester Drives**
- **5 MBit/Sec Transfer Rate**
- **128, 256, 512, and 1024 Byte Sector Lengths**
- **Six High-Level Commands: Restore, Seek, Read Sector, Write Sector, Scan ID, and Write Format**
- **Multiple Sector Transfer Capability**
- **Implied Seek With Read/Write Commands**
- **7 Byte Sector Length Extension For External Error Correction Code**
- **Single +5 Volt Power Supply**

The 82062 Winchester Disk Controller (WDC) device interfaces microprocessor systems to Winchester Disks that use the Seagate Technology ST506/ST412 interface. Examples include the Seagate ST506 and ST412, Shugart SA604 and SA606, Tandon 600, and Computer Memories CM5206 and CM5412. The device translates parallel data from the microprocessor to a 5 mbit/sec, MFM-encoded serial bit stream. It provides all of the drive control logic and, in addition, control signals which simplify the design of an external phase locked loop and write precompensation circuitry. The 82062 is designed to interface to the host controller through an external sector buffer.

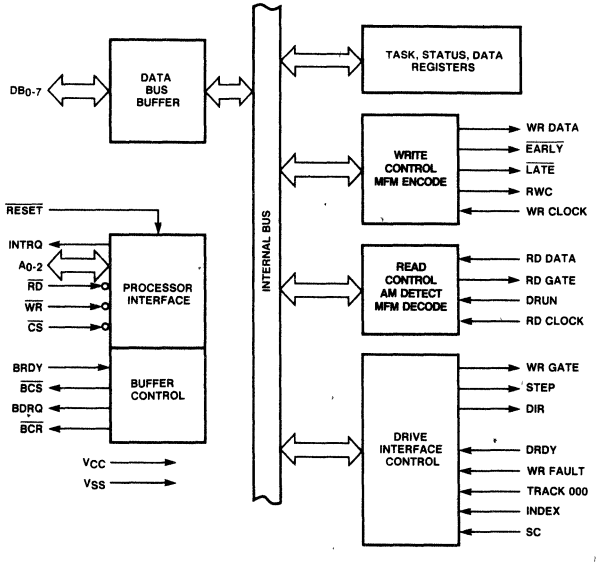


Figure 1. 82062 Block Diagram

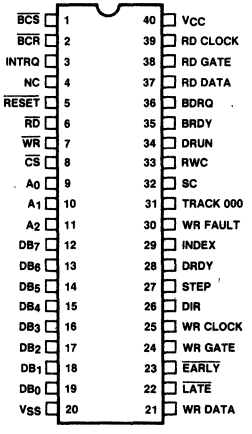


Figure 2. Pin Configuration

NOVEMBER 1983
Order Number: 210446-002

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{BCS}$	1	O	**Buffer Chip Select:** Output used to enable reading or writing of the external sector buffer.
$\overline{BCR}$	2	O	**Buffer Counter Reset:** Output that is strobed by the WDC prior to read/write operation. This pin is strobed whenever $\overline{BCS}$ changes state. It can be optionally used to reset the address counter of the buffer memory.
INTRQ	3	O	**Interrupt Request:** Interrupt generated by the WDC upon command termination. It is reset when the status register is read.
$\overline{RESET}$	5	I	**Reset:** Initializes the controller and clears all status flags.
$\overline{RD}$	6	I/O	**Read:** As an input, $\overline{RD}$ controls the transfer of status information from the WDC to the host. $\overline{RD}$ is an output when the WDC is reading data from the sector buffer.
$\overline{WR}$	7	I/O	**Write:** As an input, $\overline{WR}$ controls the transfer of command or task information into the WDC register file. $\overline{WR}$ is an output when the WDC is writing data to the sector buffer.
$\overline{CS}$	8	I	**Chip Select:** Enables $\overline{RD}$ or $\overline{WR}$ as inputs
A_0-A_2	9-11	I	**Address:** Used to select a register from the task register file
DB_7-DB_0	12-19	I/O	**Data Bus:** Bidirectional 8-bit Data Bus
GND	20	I	**Ground**
WR DATA	21	O	**Write Data:** Open drain output that shifts out MFM data at a rate determined by the Write Clock input
$\overline{LATE}$	22	O	**Late:** Open drain output used to derive a delay value for write precompensation Valid when the WR GATE output is high.
$\overline{EARLY}$	23	O	**Early:** Open drain output used to derive a delay value for write precompensation Valid when the WR GATE output is high
WR GATE	24	O	**Write Gate:** High when write data is valid WR GATE goes low if the WF input is high. This output is used by the drive to enable head write current
WR CLOCK	25	I	**Write Clock:** Clock input used to derive the write data rate Frequency = 5MHz for the ST506 interface, 4.34MHz for the SA 1000 interface.
DIR	26	O	**Direction:** High level on this output tells the drive to move the head inward (increasing cylinder number) The signal is determined by the WDC commands.
STEP	27	O	**Step:** Provides 8.4 microsecond pulses to move the drive head to another cylinder
DRDY	28	I	**Drive Ready:** If DRDY from the drive goes low, all commands will be deactivated
INDEX	29	I	**Index:** Signal from the drive indicating the beginning of a track. It is used by the WDC during formatting, and for counting retries
WR FAULT	30	I	**Write Fault:** An error input to the WDC which indicates a fault condition at the drive If WR FAULT from the drive goes low, all commands will be deactivated.
TRACK 000	31	I	**Track Zero:** Used by the Restore command to verify that the head is at the outermost cylinder.
SC	32	I	**Seek Complete:** Signal from the drive indicating that reads or writes can be made
RWC	33	O	**Reduced Write Current:** Signal goes high for all cylinder numbers above the value programmed to the Write Precomp Cylinder register It is used by the precompensation logic and by the drive
DRUN	34	I	**Data Run:** Looks for a string of zeros or ones in the read data, indicating the beginning of an ID field If the zeros are detected, RD GATE is brought high
BRDY	35	I	**Buffer Ready:** Input used by the buffer memory to signal the controller that it is ready for reading (full) or writing (empty). BRDY is checked during Read and Write commands.
BDRQ	36	O	**Buffer Data Request:** Optionally activated during Read or Write commands if BRDY is high. Can be used as a DMA Request line.
RD DATA	37	I	**Read Data:** Single ended input that accepts MFM data from the drive.
RD GATE	38	O	**Read Gate:** Output that is high for data and ID fields.
RD CLOCK	39	I	**Read Clock:** Clock input derived from the external data recovery circuits.
V_{cc}	40	I	**D.C. Power:** +5V

FUNCTIONAL DESCRIPTION

The Intel 82062 Winchester Disk Controller (WDC) integrates much of the logic needed to implement Winchester Disk controller subsystems. It provides MFM-encoded data and all the control lines required by hard disks using the Seagate Technology ST506 or Shugart Associates SA1000 interface standard. Currently, most 5¼ inch and many 8 inch Winchester Drives use this interface.

Due to the higher data rates required by these drives—1 byte every 1.6 usec—the 82062 is designed to interface with the host CPU or I/O controller through an external buffer RAM. The 82062 WDC has four pins that minimize the logic required to design a buffer interface.

Figure 3 shows a block diagram of an 82062 subsystem. The WDC is controlled by the host CPU through six commands:

Restore
Seek
Read Sector
Write Sector
Scan ID
Write Format

These commands use information stored by six task registers. Command execution starts immediately after the command register is loaded—therefore commands require only one byte from the CPU after the WDC has been initialized.

The 82062 adds all the required track formatting to the data field, including two bytes of CRC. Optionally, these two bytes can be replaced by seven bytes of ECC information for external error correction.

INTERNAL ARCHITECTURE

The internal architecture of the 82062 WDC is shown in more detail in Figure 4. The major functional blocks are:

PLA Controller

The PLA interprets commands and provides all control functions. It is synchronized with WR CLOCK.

Magnitude Comparator

A 10-bit magnitude comparator is used for the calculation of drive step, present and desired cylinder position.

CRC Logic

Generates and checks the cyclic redundancy check characters appended to the ID and data fields. The polynomial used is:

$$X^{16} + X^{12} + X^5 + 1.$$

MFM Encode/Decode

Encodes and decodes MFM data to be written/read from the drive. The MFM encoder operates from WR CLOCK, a clock having a frequency equivalent to the bit rate. The MFM decoder operates from RD CLOCK, a bit rate clock generated from the external data separator. RD CLOCK and WR CLOCK need not be synchronized.

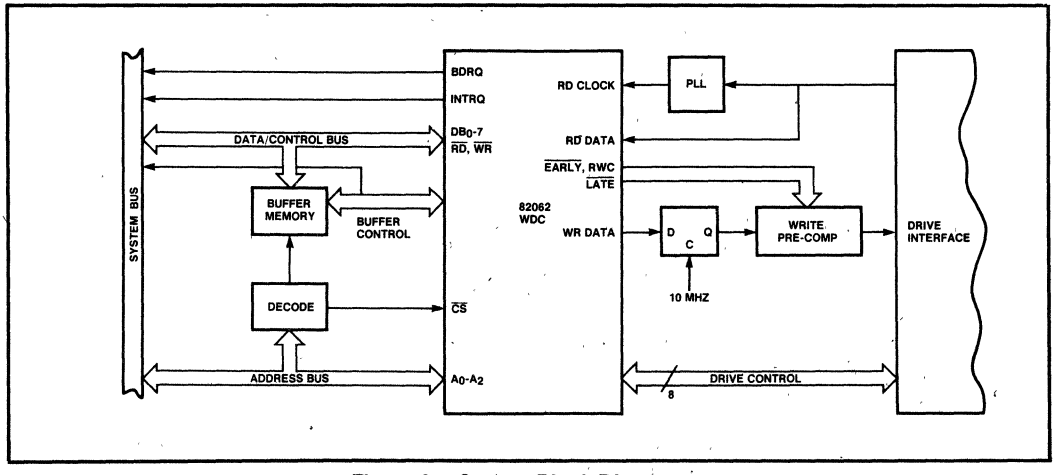


Figure 3. System Block Diagram

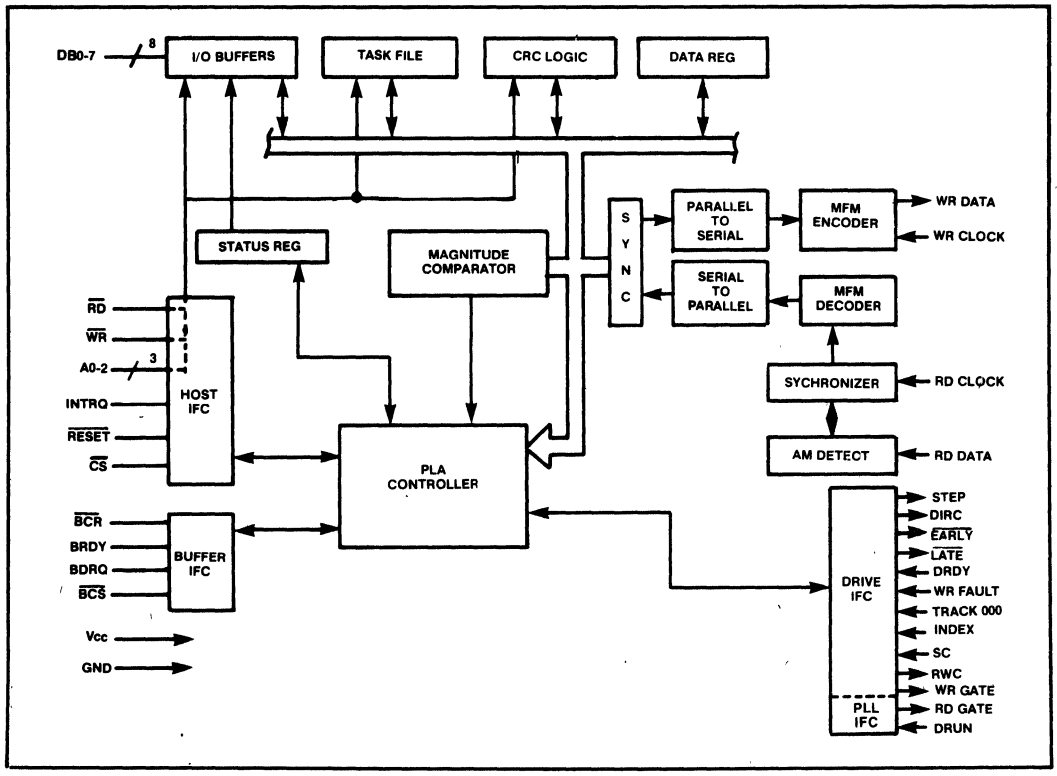


Figure 4. 82062 Detailed Block Diagram

AM Detect

The address mark detector checks the incoming data stream for a unique missing clock pattern (Data = A1H, Clock = 0AH) used in each ID and data field.

Host/Buffer Interface Control

The Host/Buffer IFC logic contains all of the necessary circuitry to communicate with the 8-bit bus from the host processor.

Drive Interface Control

The Drive IFC logic controls and monitors all lines from the drive, with the exception of read and write data.

DRIVE INTERFACE

The drive side of the 82062 WDC requires three sections of external logic. These are buffer/receivers, data separator, and write precompensation. Figure 5 illustrates a drive side interface.

The buffer/receivers condition the control lines to be driven down the cable to the drive. The control lines are typically single-ended, resistor terminated TTL levels. The data lines to and from the drive also require buffering, but are differential RS-422 levels. The interface specification to the drive can be found in the manufacturers' OEM manual. The WDC supplies TTL compatible signals, and will interface to most buffer/driver devices.

The data recovery circuits consist of a phase-lock loop data separator and associated components. The 82062 WDC interacts with the data separator thru the DATA RUN (DRUN) and RD GATE signals. A block diagram of a typical data separator circuit is shown in Figure 6. Read data from the drive is presented to the RD DATA input of the WDC, the reference multiplexor, and a retriggerable one-shot. The RD GATE (Pin 38) output will be low when the WDC is not inspecting data. The PLL at this time should remain locked to the reference clock.

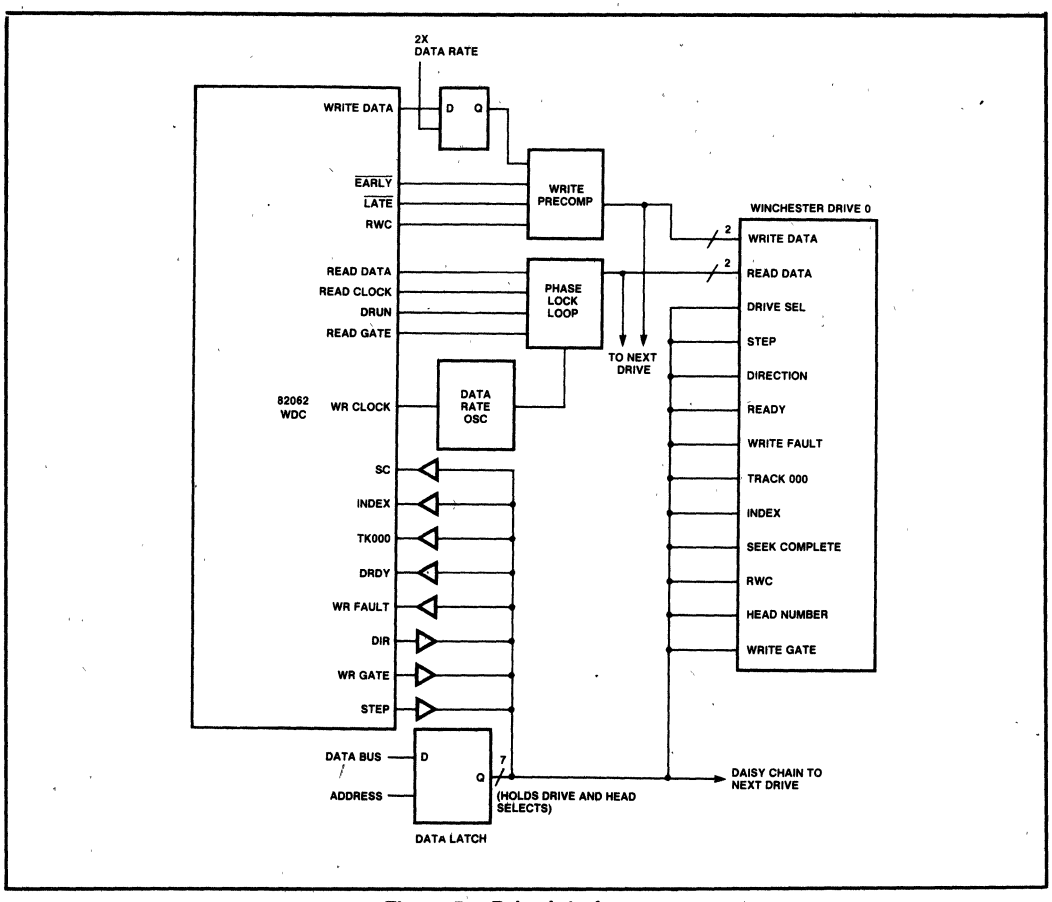


Figure 5. Drive Interface

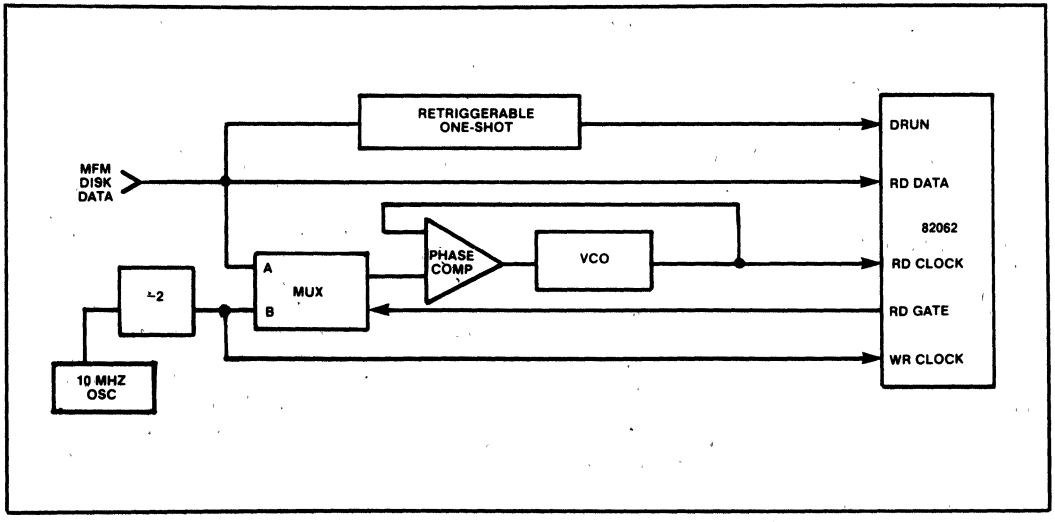


Figure 6. Data Recovery Circuit

When any Read/Write command is initiated and a search for address mark begins, the DRUN input is examined. The DRUN one-shot is set for slightly greater than one bit time, allowing it to retrigger constantly on a field of ones and zeros. An internal counter times out to see that DRUN is high for 16 bits (2 byte times). RD GATE is set by the WDC, switching the data separator to lock onto the incoming data stream. If DRUN falls prior to 72 bit times, RD GATE is lowered and the process is repeated. RD GATE will remain active high until a non-zero, non-address mark byte is detected. It will then lower RD GATE for two byte times (to allow the PLL to lock back on to the reference clock), and start the DRUN search again. If an address mark is detected, RD GATE will be held high and the command will continue searching for the proper ID field. This sequence is shown in the flow chart in Figure 7.

The write precompensation logic is controlled by the signals REDUCE WRITE CURRENT (RWC), $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$. The cylinder in which the RWC line becomes active is controlled by the REDUCE WRITE CURRENT register in the Task Register File. It can be used to turn on the precomp circuitry on a predetermined cylinder. If the REDUCE WRITE CURRENT register contents are FFH, then RWC will always be low.

The signals $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are used to tell the precomp circuitry how much delay is required on the WR DATA pulse about to be sent. The amount of delay is determined externally through a digital delay line or equivalent circuitry. Since the $\overline{\text{EARLY}}$ signal occurs after the fact, WR DATA should be delayed by one interval when both $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ are low, two intervals when $\overline{\text{LATE}}$ is high, and no delay when $\overline{\text{EARLY}}$ is high. An interval is, for example, 12-15 ns. for the ST506 interface. $\overline{\text{EARLY}}$ or $\overline{\text{LATE}}$ will be active slightly ahead of the WR DATA pulse. $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ will never be high at the same time. Regardless of the contents of the RWC register, $\overline{\text{EARLY}}$ and $\overline{\text{LATE}}$ will always be active.

HOST PROCESSOR INTERFACE

The primary interface between the host processor and the 82062 WDC is through an 8-bit bi-directional data bus. This bus is used to transmit/receive data to both the WDC and a sector buffer. The sector buffer is constructed with either FIFO memory, or static RAM and a counter. Since the WDC will use the data bus when accessing the sector buffer, a transceiver must be used to isolate the host during this time. Figure 8 shows a typical connection to a sector buffer implemented with RAM memory. Whenever the WDC is not using the sector buffer, The BUFFER CHIP SELECT ($\overline{\text{BCS}}$) is high (disabled). This allows the host to access the WDC's Task Register File, and

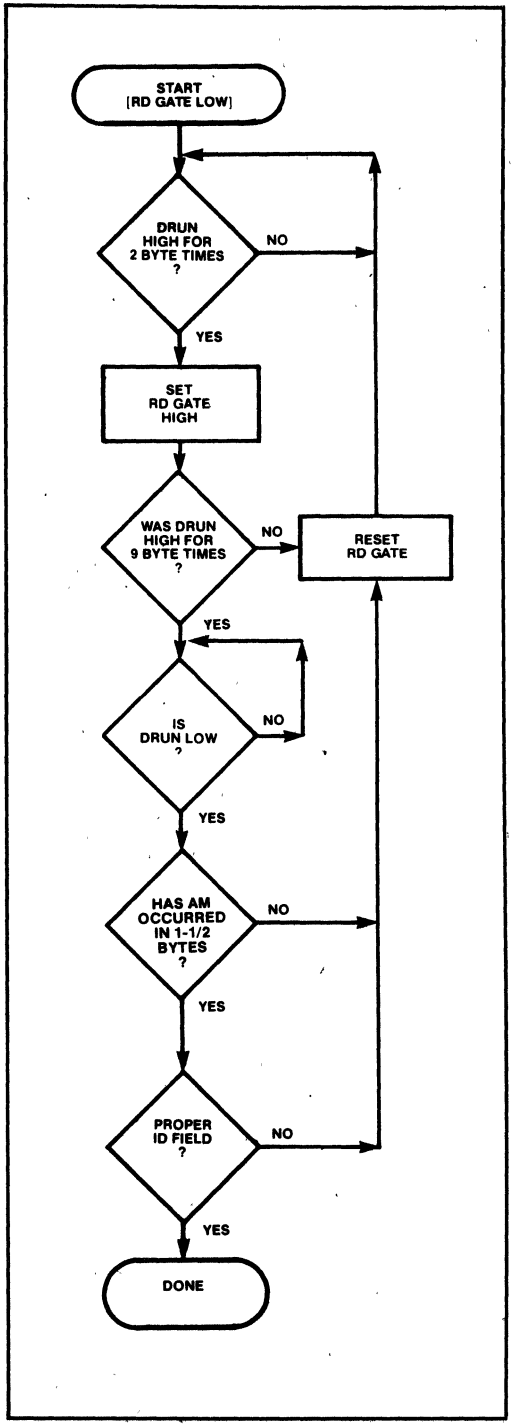


Figure 7. PLL Control Sequence

to set up parameters prior to issuing a command. It also allows the host to access the RAM buffer. A decoder is used to generate a chip select when A_{0-2} is '000', an unused address in the WDC. A binary counter is enabled whenever $\overline{RD}$ or $\overline{WR}$ go active and is incremented on the trailing edge of the chip select. This allows the host to access sequential bytes within the RAM. The decoder also generates another chip select when A_{0-2} does not equal '000', allowing access to the WDC's internal registers while keeping the RAM tri-stated.

During a WRITE SECTOR command, the host processor sets up data in the Task Register File and then issues the command. The 82062 WDC strobes the BUFFER COUNTER RESET ($\overline{BCR}$) signal to zero the counter. It then generates a status to inform the host that it may load the buffer with the data to be written. When the counter reaches its maximum count, the BUFFER READY (BRDY) signal is made active (by the "carry" out of the counter), informing the WDC that the buffer is full. (BRDY is a rising edge triggered signal which will be ignored if activated before the WDC issues $\overline{BCR}$). $\overline{BCS}$ is then made active, disconnecting the host through the transceivers, and the $\overline{RD}$ and $\overline{WR}$ lines become outputs from the WDC to allow it to access the buffer.

When the WDC is done using the buffer, it disables $\overline{BCS}$ which again allows the host to access the local bus. The READ SECTOR command operates in a similar manner, except the buffer is loaded by the WDC instead of the host processor.

Another control signal called BUFFER DATA REQUEST (BDRQ, not used in Figure 8) is a DMA signal that can inform a DMA controller when the 82062 WDC is requesting data. For further explanation, refer to the individual command descriptions and the A.C. Characteristics. In a READ SECTOR command, interrupts are generated at the termination of the command. An interrupt may be specified to occur either at the end of the command, or when BDRQ is activated. The INTERRUPT line (INTRQ) is cleared either by reading the STATUS register, or by writing a new command in the COMMAND register.

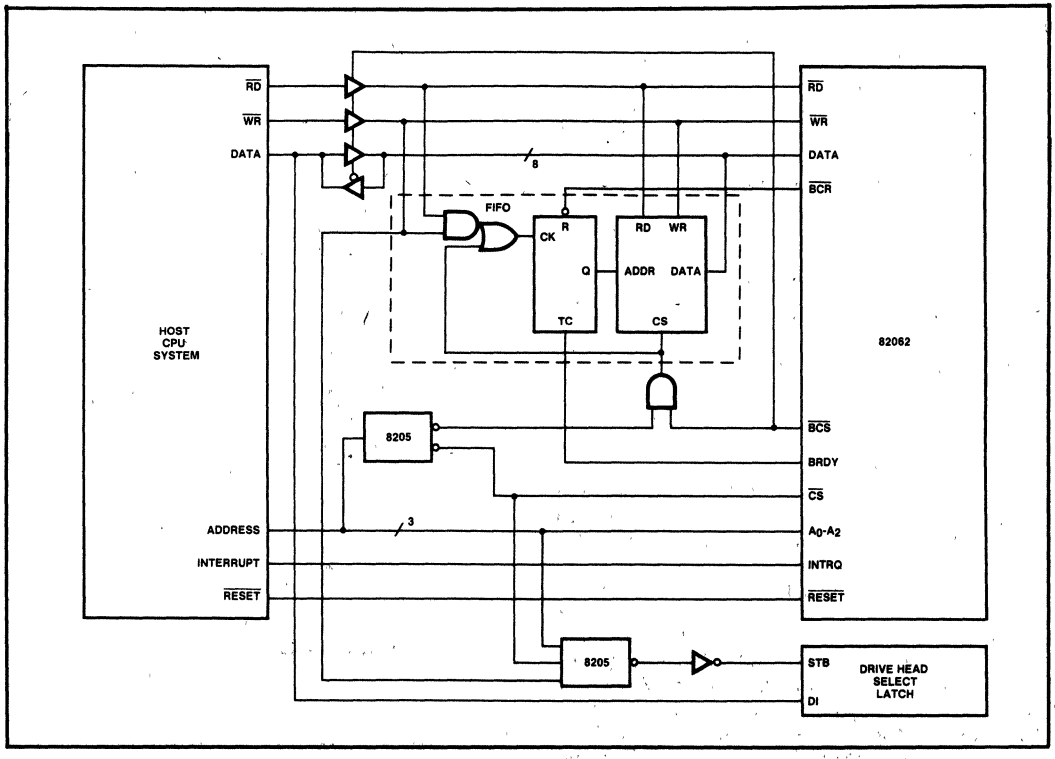


Figure 8. CPU Buffer Interface

TASK REGISTER FILE

The Task Register File is a bank of registers used to hold parameter information pertaining to each command. These registers and their addresses are:

A2	A1	A0	READ	WRITE
0	0	0	(Bus Tri-Stated)	(Bus Tri-Stated)
0	0	1	Error Flags	Reduce Write Current
0	1	0	Sector Count	Sector Count
0	1	1	Sector Number	Sector Number
1	0	0	Cylinder Low	Cylinder Low
1	0	1	Cylinder High	Cylinder High
1	1	0	SDH	SDH
1	1	1	Status Register	Command Register

NOTE: Registers are not cleared by $\overline{\text{RESET}}$.

ERROR REGISTER

This read-only register contains specific error status after the completion of a command. The bits are defined as follows:

7	6	5	4	3	2	1	0
BBD	CRC	—	ID	—	AC	TK000	DM

Bit 7 - Bad Block Detect

This bit is set when an ID field has been encountered that contains a bad block mark. It is used for bad sector mapping.

Bit 6 - CRC Data Field

This bit is set when a data field CRC error has ocurred or the Data Address Mark has not been found. The sector buffer may still be read but will contain errors.

Bit 5 - Reserved Not used.

Forced to zero.

Bit 4 - ID Not Found

This bit is set when the desired cylinder, head, sector, or size parameter cannot be found after 8 revolutions of the disk, or if an ID field CRC error has occured.

Bit 3 - Reserved Not used.

Forced to zero.

Bit 2 - Aborted Command

This bit is set if a command was issued while DRDY (Pin 28) or WR FAULT (Pin 30) is low. The Aborted Command bit will also be set if an undefined command is written into the COMMAND register, but an implied seek will be executed.

Bit 1 - TRACK 000

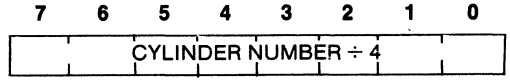
This bit is set only by the RESTORE command. It indicates that TRACK 000 (Pin 31) has not gone active after the issuance of 1024 stepping pulses.

Bit 0 - Data Address Mark

This bit is set during a READ SECTOR command if the Data Address Mark is not found after the proper Sector ID is read.

REDUCE WRITE CURRENT REGISTER

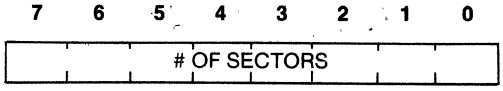
This register is used to define the cylinder number where RWC (Pin 33) is asserted:



The value (0-255) loaded into this register is internally multiplied by 4 to specify the actual cylinder where RWC is asserted. Thus a value of 01H will cause RWC to activate on cylinder 4, 02H on cylinder 8, and so on. RWC switching points are then 0,4,8, . . . 1020. RWC will be asserted when the present cylinder is greater than or equal to the cylinder indicated by this register. For example, the ST506 interface requires precomp on cylinder 128 (80H) and above. Therefore, the REDUCE WRITE CURRENT register should be loaded with 32 (20H). A value of FFH will make RWC stay low, regardless of the actual cylinder number.

SECTOR COUNT REGISTER

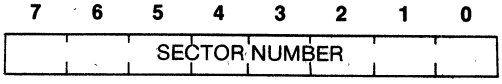
This register is used to define the number of sectors that need to be transfered to the buffer during a READ MULTIPLE SECTOR or WRITE MULTIPLE SECTOR command.:



The value contained in the register is decremented after each sector is transferred to/from the sector buffer. A zero represents a 256 sector transfer, a one a 0 sector transfer, etc. This register is a "don't care" when single sector commands are specified.

SECTOR NUMBER

This register holds the sector number of the desired sector:

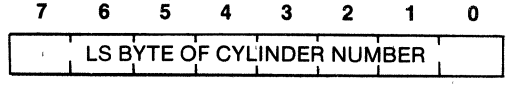


For a multiple sector command, it specifies the first sector to be transferred. It is decremented after each sector is transferred to/from the sector buffer. The SECTOR NUMBER register may contain any value from 0 to 255.

The SECTOR NUMBER register is also used to program the Gap 1 and Gap 3 lengths to be used when formatting a disk. See the WRITE FORMAT command description for further explanation.

CYLINDER NUMBER LOW REGISTER

This register holds the lower byte of the desired cylinder number:



It is used in conjunction with the CYLINDER NUMBER HIGH register to specify a range of 0 to 1023.

CYLINDER NUMBER HIGH REGISTER

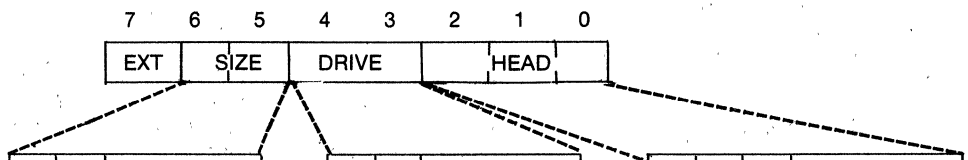
This register holds the two most significant bits of the desired cylinder number:

7	6	5	4	3	2	1	0
X	X	X	X	X	X	(9)	(8)

Internal to the 82062 WDC is another pair of registers that hold the actual position where the R/W heads are located. The CYLINDER NUMBER HIGH and LOW registers can be considered the cylinder destination for seeks and other commands. After these commands are executed, the internal cylinder position registers' contents are equal to the cylinder high/low registers. If a drive number change is detected on a new command, the WDC automatically reads an ID field to update its internal cylinder position registers. This affects all commands except a RESTORE.

SECTOR/DRIVE/HEAD REGISTER

The SDH register contains the desired sector size, drive number, and head number parameters. The format is diagramed below.



6	5	SECTOR SIZE
0	0	256
0	1	512
1	0	1024
1	1	128

4	3	DRIVE #
0	0	DSEL1
0	1	DSEL2
1	0	DSEL3
1	1	DSEL4

2	1	0	HEAD #
0	0	0	HSEL0
0	0	1	HSEL1
0	1	0	HSEL2
0	1	1	HSEL3
1	0	0	HSEL4
1	0	1	HSEL5
1	1	0	HSEL6
1	1	1	HSEL7

Both head number and sector size are compared against the disks' ID field. Head select and drive select lines are not available as outputs from the 82062 WDC and must be generated externally. Figure 9 shows a possible logic implementation of these select lines.

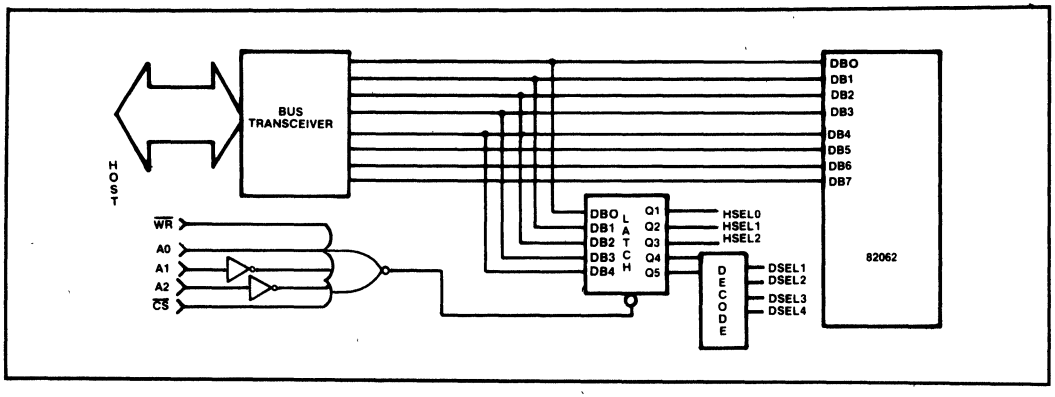


Figure 9. Drive/Head Select Logic

Bit 7, the extension bit (EXT), is used to extend the data field by seven bytes when using ECC codes. When EXT = 1, the CRC is not appended to the end of the data field, the data field becomes "sector size + 7" bytes long. The CRC is checked on the ID field regardless of the state of EXT. Note that the sector size bits (SIZE) are written to the ID field during a formatting command. The SDH byte written into the ID field is different than the SDH Register contents. The recorded SDH byte does not have the drive number (DRIVE) written but does have the BAD BLOCK mark written. The format is:

7	6	5	4	3	2	1	0
BAD BLOCK	SIZE		0	0		HEAD #	

Note that use of the extension bit requires the gap lengths to be modified as described in the WRITE FORMAT command description.

STATUS REGISTER

The status register is a read-only register which informs the host of certain events performed by the 82062 WDC as well as reporting status from the drive control lines. The format is:

7	6	5	4	3	2	1	0
BUSY	READY	WF	SC	DRQ	—	CIP	ERROR

Bit 7 - Busy

This bit is set whenever the 82062 WDC is accessing the disk. Commands should not be loaded into the COMMAND register while Busy is set. Busy is set when a command is written into the WDC and is cleared at the end of all commands except READ SECTOR. While executing a READ SECTOR command, Busy is cleared after the sector buffer has been filled. When the Busy bit is set, no other bits in either the STATUS or any other registers are valid.

Bit 6 - Ready

This bit normally reflects the state of the DRDY (Pin 28) line. When an interrupt is generated by an 'aborted command' error condition, the Ready bit is latched for later examination by the host. After a STATUS register read, the Ready bit will resume reflecting the state of DRDY.

Bit 5 - Write Fault

This bit reflects the state of the WR FAULT (Pin 30) line. Whenever WR FAULT goes high, an interrupt will be generated. The Write Fault bit is latched like the Ready bit (Bit 6).

Bit 4 - Seek Complete

This bit reflects the state of the SC (Pin 32) line. Certain commands will pause until Seek Complete is set. The Seek Complete bit is latched like the Ready bit.

Bit 3 - Data Request

The Data request bit (DRQ) reflects the state of the BDRQ (Pin 36) line. It is set when the sector buffer should be loaded with data or read by the host processor, depending upon the command. The DRQ bit and the BDRQ line remain high until BRDY is sensed, indicating the operation is completed. BDRQ can be used in DMA interfacing, while DRQ can be used for programmed I/O transfers.

Bit 2 - Reserved

Not Used. Forced to zero.

Bit 1 - Command in Progress

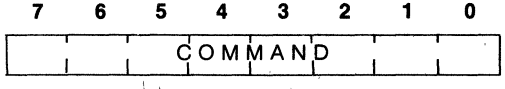
When this bit is set, a command is being executed and a new command should not be loaded until it is cleared. Although a command may be executing, the sector buffer is still available for access by the host processor. Only the STATUS register may be read. If other registers are read, the STATUS register contents will be returned.

Bit 0 - Error

This bit is set whenever any bits in the ERROR register are set. It is the logical 'or' of the bits in the error register and may be used by the host processor to quickly check for successful completion of a command. This bit is reset when a new command is written into the COMMAND register.

COMMAND REGISTER

This write-only register is loaded with the desired command:



The command begins to execute immediately upon loading. This register should not be loaded while the Busy or Command in Progress bits are set in the STATUS register. The INTRQ line (Pin 3), if set, will be cleared by a write to the COMMAND register.

INSTRUCTION SET

The 82062 WDC instruction set contains six commands. Prior to loading the command register, the host processor must first set up the Task Register File with the information needed for the command. Except for the COMMAND register, the registers may be loaded in any order. If a command is in progress, a subsequent write to the COMMAND register will be ignored until execution of the current command is completed as indicated by the command in progress bit in the STATUS register being cleared

COMMAND	7	6	5	4	3	2	1	0
RESTORE	0	0	0	1	R3	R2	R1	R0
SEEK	0	1	1	1	R3	R2	R1	R0
READ SECTOR	0	0	1	0	I	M	0	T
WRITE SECTOR	0	0	1	1	0	M	0	T
SCAN ID	0	1	0	0	0	0	0	0
WRITE FORMAT	0	1	0	1	0	0	0	0

R_{3-0} = Rate Field

For 5 MHz WR CLOCK:

R_{3-0} =	0000	- ≈35	us
	0001	- 0.5	ms
	0010	- 1.0	ms
	0011	- 1.5	ms
	0100	- 2.0	ms
	0101	- 2.5	ms
	0110	- 3.0	ms
	0111	- 3.5	ms
	1000	- 4.0	ms
	1001	- 4.5	ms
	1010	- 5.0	ms
	1011	- 5.5	ms
	1100	- 6.0	ms
	1101	- 6.5	ms
	1110	- 7.0	ms
	1111	- 7.5	ms

T = Retry Enable

T = 0	Enable Retries
T = 1	Disable Retries

M = Multiple Sector Flag

M = 0	Transfer 1 Sector
M = 1	Transfer Multiple Sectors

I = Interrupt Enable

I = 0	Interrupt at BDRQ time
I = 1	Interrupt at end of command

RESTORE COMMAND

The RESTORE command is usually used on a power-up comdition. The actual stepping rate used for the RESTORE is determined by the Seek Complete time. A step pulse is issued and the 82062 WDC waits for the Seek Complete (SC) line to go active before issuing the next pulse. If after 1,024 stepping pulses the TRACK 000 line does not go active, the WDC will set the TRACK 000 bit in the ERROR register and terminate with an INTRQ. An interrupt will also occur if WR FAULT goes active or DRDY goes inactive at any time during execution.

The rate field specified (R_{3-0}) is stored in an internal register for future use in commands with implied seeks.

A flowchart of the RESTORE command is shown in Figure 10.

SEEK COMMAND

Since all commands feature an implied seek, the SEEK command can be used for overlap seek operations on multiple drives. The actual stepping rate used is taken from the Rate Field of the command, and is stored in an internal register for future use. If DRDY goes inactive or WR FAULT goes active at any time during the seek, the command is terminated and an INTRQ is generated.

The direction and number of step pulses needed is calculated by comparing the contents of the CYLINDER NUMBER LOW/HIGH register pair to the internal cylinder position register. After all steps have been issued, the internal cylinder position register is updated and the command is terminated. The Seek Complete (SC) line is not checked at the beginning or end of the command.

If an implied seek was performed, the 82062 will search until a rising edge of SC is received.

A flowchart of the SEEK command is shown in Figure 11.

READ SECTOR

The READ SECTOR command is used to transfer one or more sectors of data from the disk to the sector buffer. Upon receipt of the READ SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. If an implied seek was performed, the WDC will search until a rising edge of SC is received. The WR FAULT and DRDY lines are monitored throughout the command.

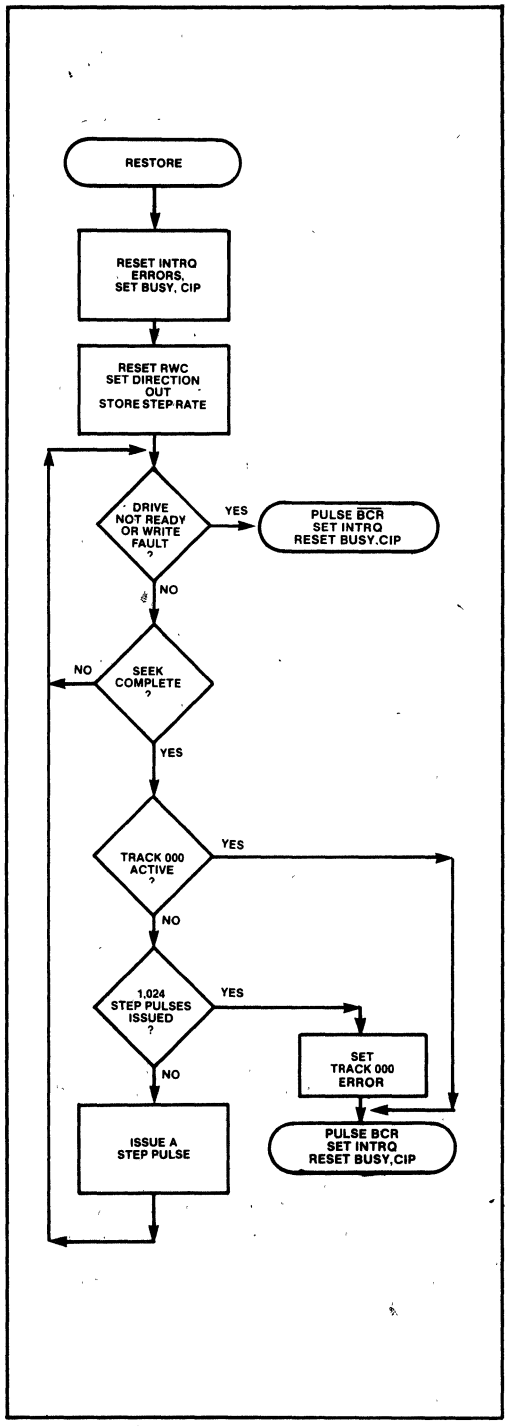


Figure 10. Restore Command Flow

When the Seek Complete (SC) line is high (with or without an implied seek having occured), the search for an ID field begins. If T = 0 (retries enabled), the 82062 WDC must find an ID with the correct cylinder number, head, sector size and CRC within 8 revolutions, or an automatic scan ID will be performed to obtain cylinder position information, and then a seek performed (if necessary). The search for the proper ID will be retried for up to 8 revolutions. If the correct sector is still not found, the appropriate error bits will be set and the command terminated. Data CRC errors will also be retried for up to 8 revolutions (if M = 0).

If T = 1 (retries disabled), the ID search must find the correct sector within 2 revolutions or the appropriate error bits will be set and the command terminated.

Both the READ SECTOR and WRITE SECTOR commands feature a "simulated completion" to ease programming. DRQ/BDRQ will be generated upon detecting an error condition. This allows the same program flow for successful or unsuccessful completion of a command.

When the data address mark is found, the WDC is ready to transfer data to the sector buffer. After the data has been transferred, the I bit is checked. If I = 0, INTRQ is made active coincident with BDRQ, indicating that a transfer of data from the buffer to the host processor is required. If I = 1, INTRQ will occur at the end of the command, i.e. after the buffer is unloaded by the host.

An optional M bit may be set for multiple sector transfers. When M = 0, one sector is transferred and the SECTOR COUNT register is ignored. When M = 1, multiple sectors are transferred. After each sector is transferred the 82062 decrements the SECTOR COUNT register and increments the SECTOR NUMBER register. The next logical sector will be transferred regardless of any interleave. Sectors are numbered at format time by a byte in the ID field.

For the 82062 to make multiple sector transfers to the buffer, the BRDY line must be toggled low to high for each sector. Transfers will continue until the SECTOR COUNT register equals zero, or the BRDY line goes active. If the SECTOR COUNT register is non-zero (indicating more sectors are to be transferred but the buffer is full), BDRQ will be made active and the host must unload the buffer. After this occurs, the buffer will again be free to accept the remaining sectors from the WDC. This scheme enables the user to transfer more sectors than the buffer memory has capacity for.

In summary then, READ SECTOR operation is as follows:

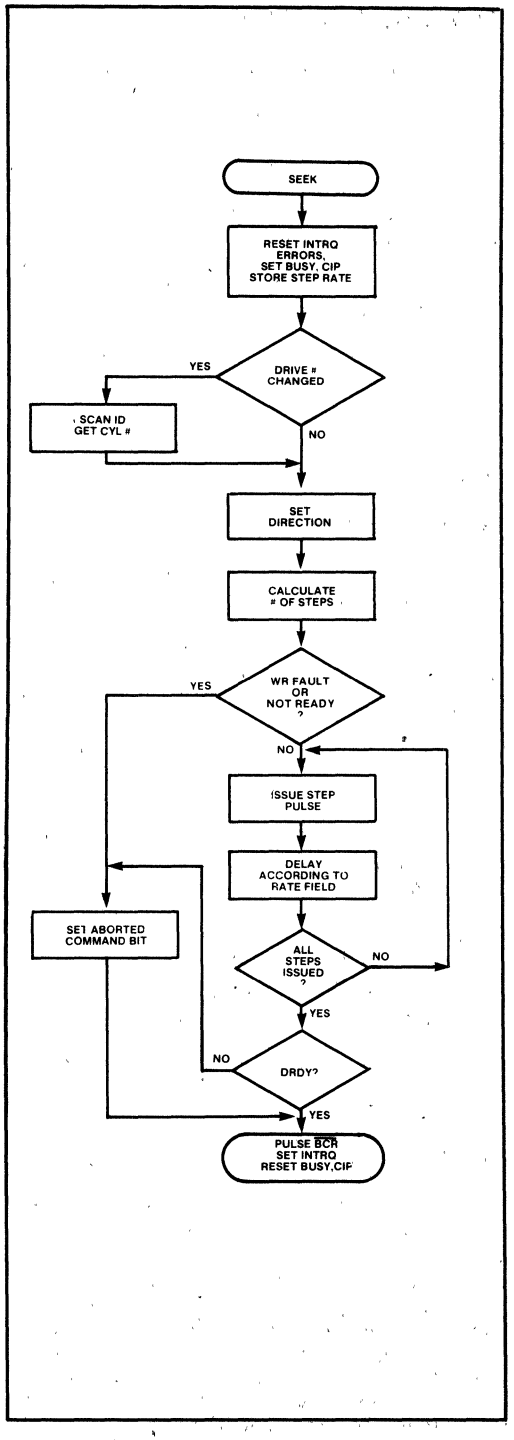


Figure 11. Seek Command Flow

When M = 0 (READ SECTOR)

(1)	**Host:**	Sets up parameters; issues READ SECTOR command.
(2)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	**82062:**	Finds sector specified; transfers data to buffer.
(4)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(5)	**82062:**	Sets BDRQ = 1, DRQ = 1.
(6)	**82062**	If I bit = 1 then go to (9).
(7)	**Host:**	Reads contents of sector buffer.
(8)	**82062:**	Waits for BRDY, then sets INTRQ = 1; END.
(9)	**82062:**	Sets INTRQ = 1.
(10)	**Host:**	Reads out contents of buffer; END.

When M = 1 (READ MULTIPLE SECTOR)

(1)	**Host:**	Sets up parameters; issues READ SECTOR command.
(2)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 0.
(3)	**82062:**	Finds sector specified; transfers data to buffer.
(4)	**82062:**	Decrements SECTOR COUNT register; increments SECTOR NUMBER register.
(5)	**82062:**	Strobes $\overline{BCR}$; sets $\overline{BCS}$ = 1.
(6)	**82062:**	Sets BDRQ = 1, DRQ = 1.
(7)	**Host:**	Reads out contents of buffer; END.
(8)	**Buffer:**	Indicates data has been transferred by activating BRDY.
(9)	**82062:**	When BRDY = 1, if Sector Count = 0, then go to (11).
(10)	**82062:**	Go to (2).
(11)	**82062:**	Set INTRQ = 1.

A flowchart of the READ SECTOR command is shown in Figure 12.

WRITE SECTOR

The WRITE SECTOR command is used to write one or more sectors of data to the disk from the sector buffer. Upon receipt of WRITE SECTOR command, the 82062 WDC checks the CYLINDER NUMBER LOW/HIGH register pair against the internal cylinder position register to see if they are equal. If not, the direction and number of steps calculation is performed and a seek takes place. The WR FAULT and DRDY lines are checked throughout the command.

When the Seek Complete (SC) line is found to be true (with or without an implied seek having occured), the BDRQ signal is made active and the host proceeds to unload the buffer. When the 82062 senses BRDY high, the ID field with the specified cylinder number, head, and sector size is searched for. Once found, WR GATE is made active and the data is written to the disk. If retries are enabled (T = 0), and if the ID field cannot be found within 8 revolutions, automatic scan ID and seek commands are performed. The ID Not Found error bit is set and the command is terminated if the correct ID field is not found within 8 additional revolutions. If retries are disabled, (T = 1), and if the ID field cannot be found within 2 revolutions, the ID Not Found error bit is set and the command is terminated.

During a WRITE MULTIPLE SECTOR command (M = 1), the SECTOR NUMBER register is decremented and the SECTOR COUNT register is incremented. If the BRDY line is low after the first sector is transferred from the buffer, the 82062 will transfer the next sector. If BRDY is high, the 82062 will set BDRQ and wait for the host processor to place more data in the buffer. In summary then, the WRITE SECTOR operation is as follows:

When M = 0,1 (WRITE SECTOR)

(1)	**Host:**	Sets up parameters; issues WRITE SECTOR command.
(2)	**82062:**	Strobes $\overline{BCR}$; sets BDRQ = 1, DRQ = 1.
(3)	**Host:**	Loads sector buffer with data.
(4)	**82062:**	Waits for BRDY = 1.
(5)	**82062:**	Finds specified ID field; writes sector to disk.
(6)	**82062:**	If M = 0, then set INTRQ = 1; END.
(7)	**82062**	Increment SECTOR NUMBER register; decrement SECTOR COUNT register.
(8)	**82062**	If SECTOR = 0, then set INTRQ = 1; END.
(9)	**82062**	If BRDY = 0, then go to (5).
(10)	**82062**	Go to (2).

A flowchart of the WRITE SECTOR command is shown in Figure 13.

SCAN ID

The SCAN ID command is used to update the SECTOR/DRIVE/HEAD, SECTOR NUMBER, and CYLINDER NUMBER LOW/HIGH registers.

After the command is loaded, the Seek Complete (SC) line is sampled until it is valid. The DRDY and WR FAULT lines are also monitored throughout execution of the command. When the first ID field is

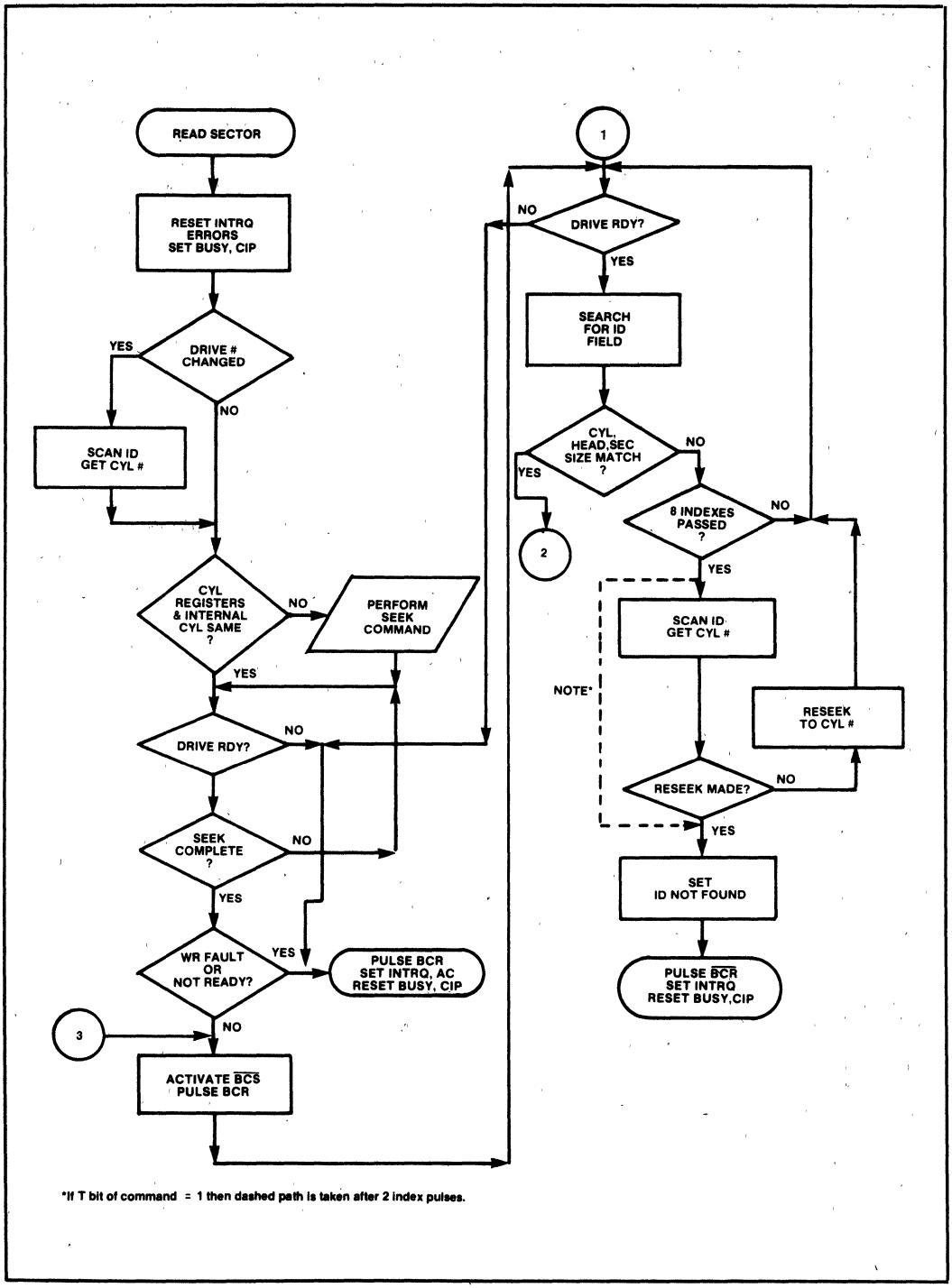


Figure 12A. Read Sector Command Flow

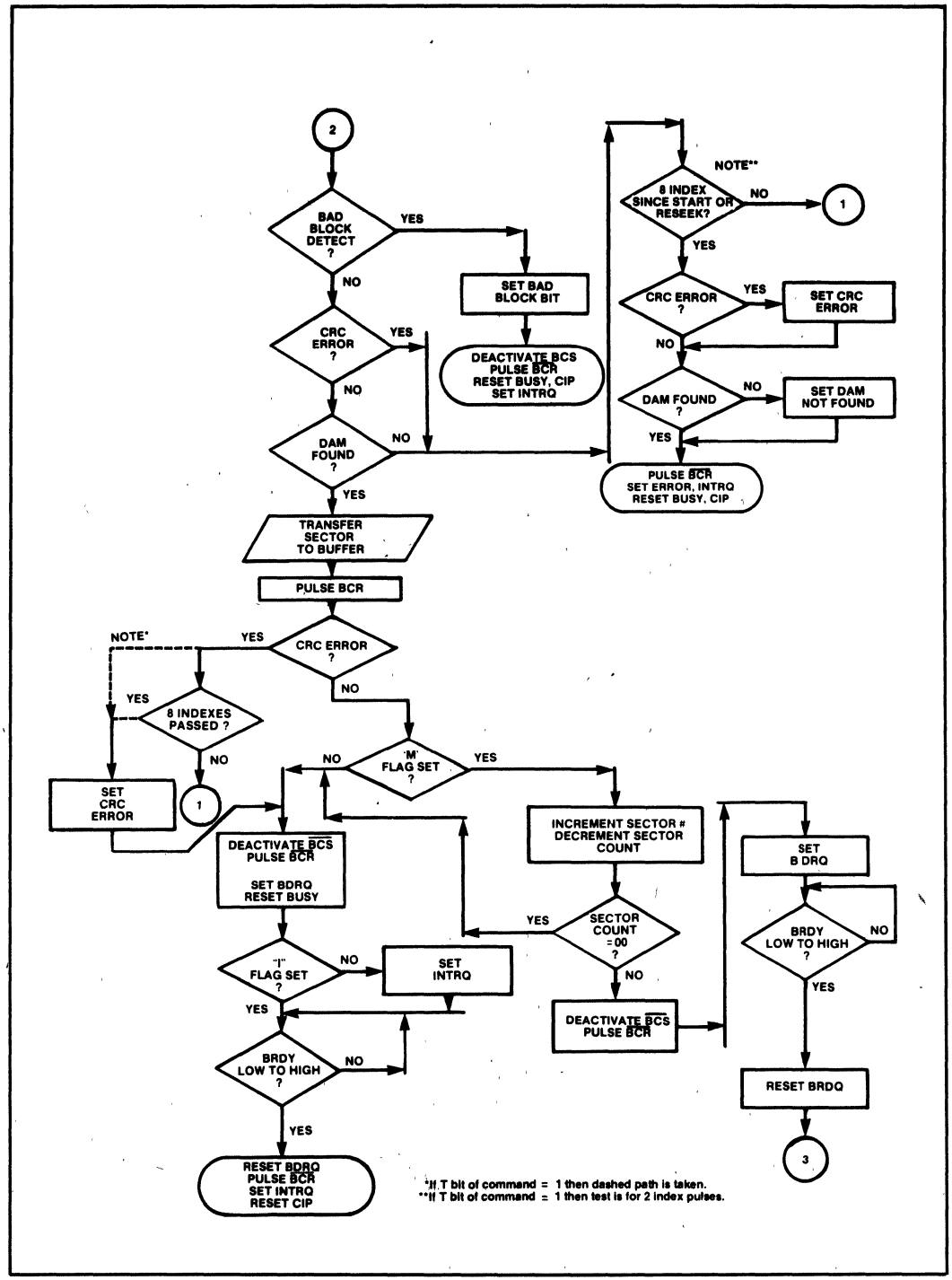


Figure 12B. Read Sector Command Flow

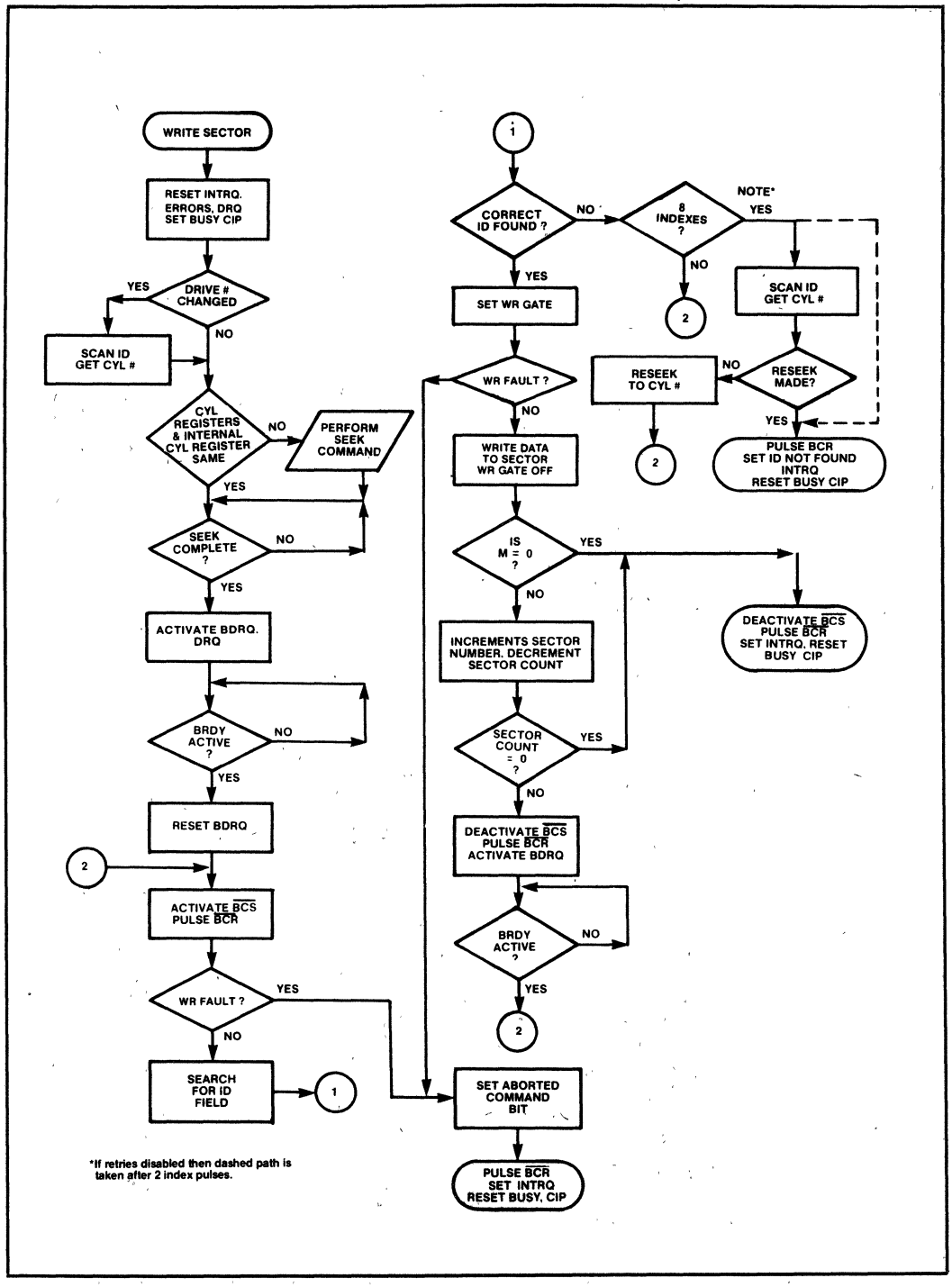


Figure 13. Write Sector Command Flow

found, the ID information is loaded into the SDH, SECTOR NUMBER, and CYLINDER NUMBER registers. The internal cylinder position register is also updated. If a bad block is detected, the BAD BLOCK bit will also be set. The CRC is checked and if an error is found, the 82062 will retry up to 8 revolutions to find an error-free ID field. There is no implied seek with this command and the sector buffer is not disturbed.

A flowchart of the SCAN ID command is shown in Figure 14.

WRITE FORMAT

The WRITE FORMAT command is used to format one track using the Task Register File and the sector buffer. During execution of this command, the sector buffer is used for additional parameter information instead of sector data. Shown in Figure 15 is the contents of the sector buffer for a 32 sector track format with an interleave factor of two. Each sector requires a two byte sequence. The first byte designates whether a bad block mark is to be recorded in the sector's ID field. An 00H is normal; an 80H indicates a bad block mark for that sector. In the example of Figure 15, sector 04 will get a bad block mark recorded.

The second byte indicates the logical sector number to be recorded. This allows sectors to be recorded with any interleave factor desired. The remaining memory in the sector buffer may be filled with any value; its only purpose is to generate a BRDY to tell the 82062 to begin formatting the track.

An implied seek is in effect on this command. As for other commands, if the drive number has been changed, an ID field will be scanned for cylinder position information before the implied seek is performed. If no ID field can be read (because the track had been erased or because an incomplete format had been been used), an ID Not Found error will result and the WRITE FORMAT command will be aborted. This can be avoided by issuing a RESTORE command before formatting.

The SECTOR COUNT register is used to hold the total number of sectors to be formatted (FFH = 255 sectors), while the SECTOR NUMBER register holds the number of bytes minus three to be used for Gap 1 and Gap 3; for instance, if the SECTOR COUNT register value is 02H and the SECTOR NUMBER register value is 00H, then 2 sectors are written and 3 bytes of 4EH are written for Gap 1 and Gap 3. The data fields are filled with FFH and the CRC is automatically generated and appended. The sector extension bit in the SDH register should not be set. After the last sector is written the track is filled with 4EH.

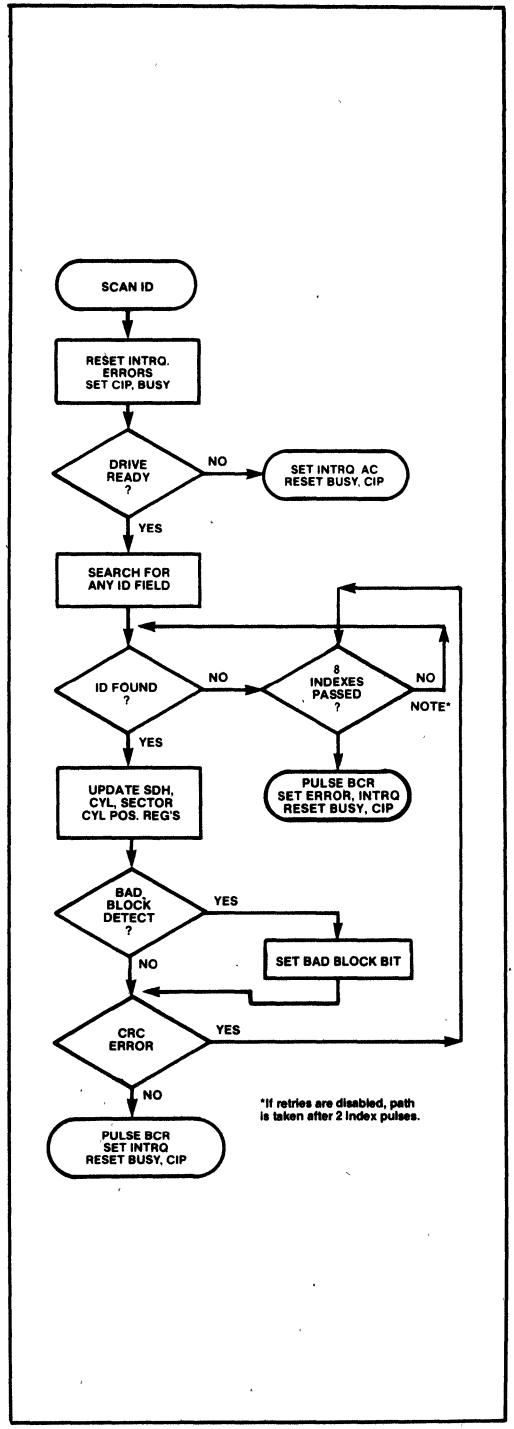


Figure 14. Scan ID Command Flow

FORMAT COMMAND SECTOR BUFFER CONTENTS

SECTOR BUFFER ADDRESS	BAD BLOCK?	LOCICAL SECTOR NUMBER
00	00	00
02	00	10
04	00	01
06	00	11
08	00	02
0A	00	12
0C	00	03
0E	00	13
10	80	04
12	00	14
14	00	05
16	00	15
18	00	06
1A	00	16
1C	00	07
1E	00	17
20	00	08
22	00	18
24	00	09
26	00	19
28	00	0A
2A	00	1A
2C	00	0B
2E	00	1B
30	00	0C
32	00	1C
34	00	0D
36	00	1D
38	00	0E
3A	00	1E
3C	00	0F
3E	00	1F
40	FF	FF
.	.	.
.	.	.
.	.	.
F0	FF	FF

Figure 15

The Gap 3 value is determined by the drive motor speed variation, data sector length, and the interleave factor. The interleave factor is only important when 1:1 interleave is used. The formula for determining the minimum Gap 3 length value is:

Gap 3 = (2 * M * S) + K + E

M = motor speed variation (e.g., 0.03 for ± 3%)

S = sector length in bytes

K = 25 for interleave factor of 1

K = 0 for any other interleave factor

E = 7 if the sector is to be extended

Like all commands, a WR FAULT or drive not ready condition will terminate execution of the WRITE FORMAT command. Figure 16 shows the format that the 82062 will write on the disk.

A flowchart of the WRITE FORMAT command is shown in Figure 17.

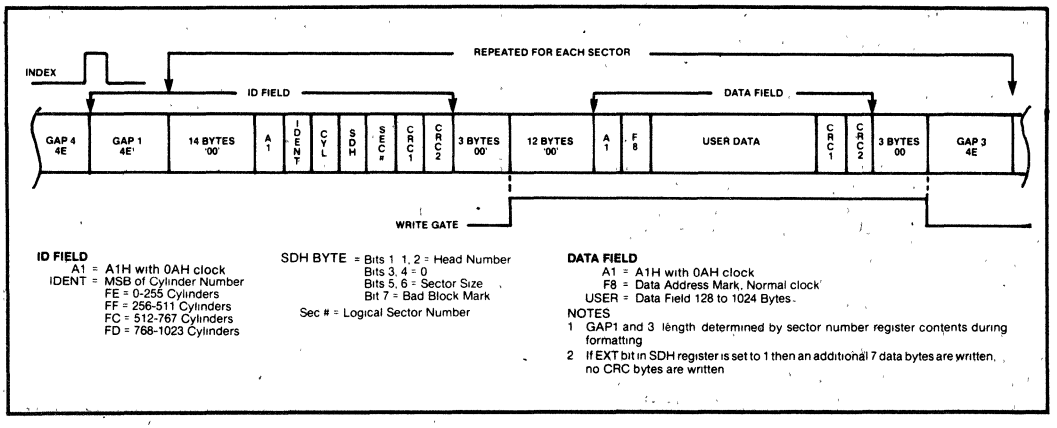


Figure 16. Track Format

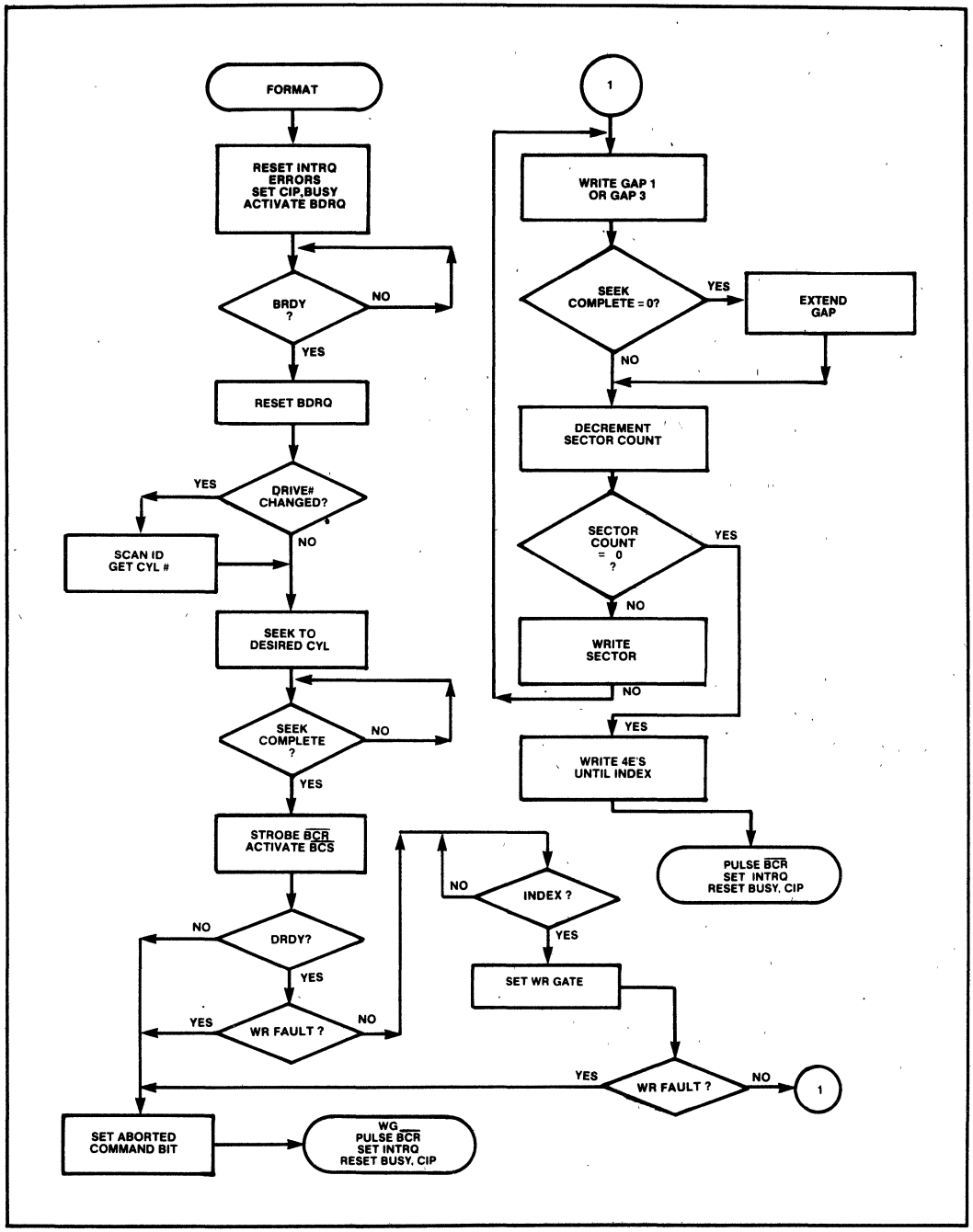


Figure 17. Write Format Command Flow

ELECTRICAL CHARACTERISTICS
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on any pin with
respect to GND ..... −0.5V to +7V
Power Dissipation .....1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

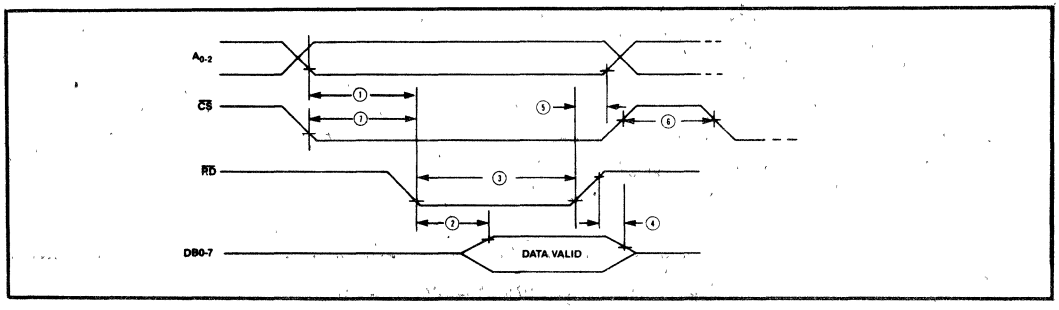
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
I_{IL}	Input Leakage Current		10	μA	$V_{IN} = V_{CC}$
I_{OFL}	Output Leakage Current		10	μA	$V_{OUT} = V_{CC}$
V_{IH}	Input High Voltage	2.0		V	
V_{IL}	Input Low Voltage		0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{OH} = 100uA
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 1.6mA 4.8mA P21,22,23
I_{CC}	Supply Current		250	mA	All Outputs Open
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND
	For Pins 25,34,37,39				
V_{IH}	Input High Voltage	4.6		V	
V_{IL}	Input Low Voltage		0.5	V	
TRS	Rise Time		30	ns	10% to 90% points

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ± 10%; GND = 0V)

HOST READ TIMING

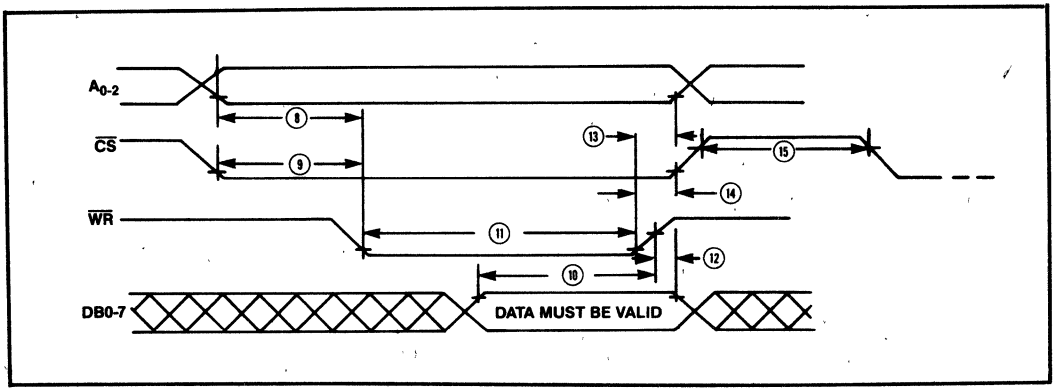
SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
1	Address Stable Before $\overline{RD}$↓	100		ns	
2	Data Delay From $\overline{RD}$↓		375	ns	
3	$\overline{RD}$ Pulse Width	0.4	10	ns	
4	$\overline{RD}$ to Data Floating	20	200	ns	
5	Address Hold Time after $\overline{RD}$↑	0		ns	
6	Read Recovery Time	300		ns	
7	$\overline{CS}$ Stable before $\overline{RD}$	0		ns	



210446-002

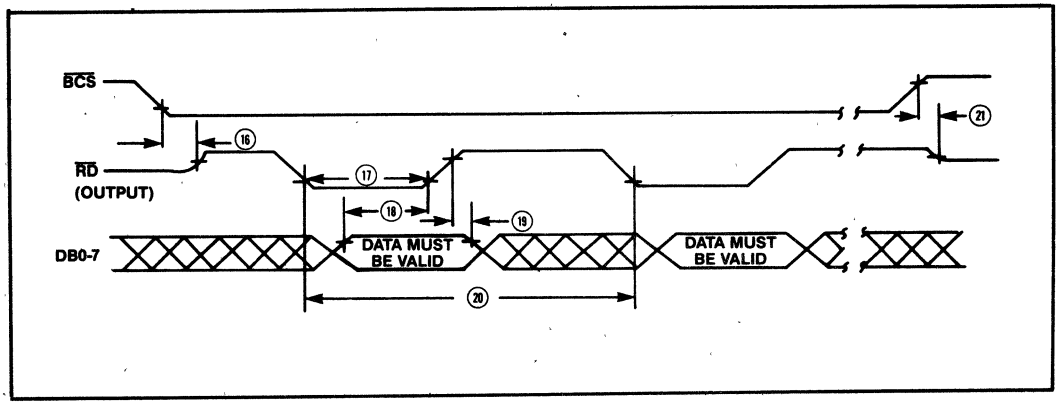
HOST WRITE TIMING

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITIONS
8	Address Stable Before $\overline{WR}$↓	0	10	μs	
9	$\overline{CS}$ Stable Before $\overline{WR}$↓	0	10	μs	
10	Data Setup Time Before $\overline{WR}$↑	0.2		μs	
11	$\overline{WR}$ Pulse Width	0.2	10	μs	
12	Data Hold Time After $\overline{WR}$↑	10		ns	
13	Address Hold Time After $\overline{WR}$↑	30		ns	
14	$\overline{CS}$ Hold Time After $\overline{WR}$↑	0		ns	
15	Write Recovery Time	1.0		μs	



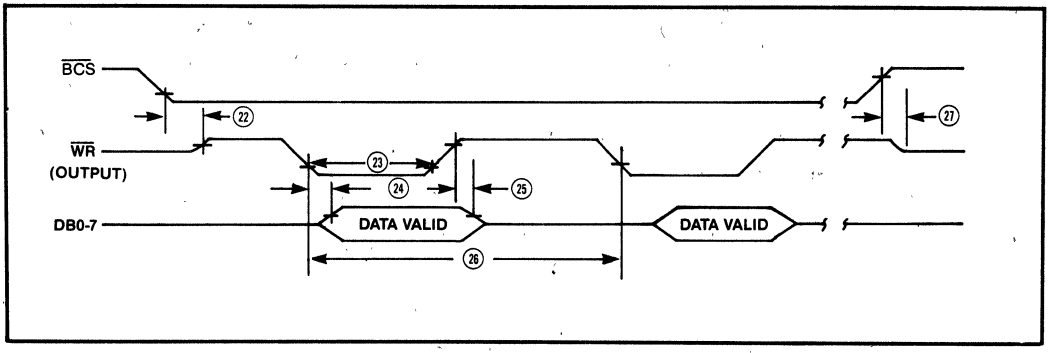
BUFFER READ TIMING (WRITE SECTOR COMMAND)

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
16	$\overline{RD}$ Float to $\overline{RD}$ Valid	15		100	ns	C_L = 50pF
17	$\overline{RD}$ Output Pulse Width	300	400	500	ns	See Note 3
18	Data Setup to $\overline{RD}$↑	140			ns	
19	Data Hold from $\overline{RD}$↑	0			ns	
20	$\overline{RD}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
21	$\overline{RD}$ Float from $\overline{BCS}$↑			100	ns	C_L = 50pF



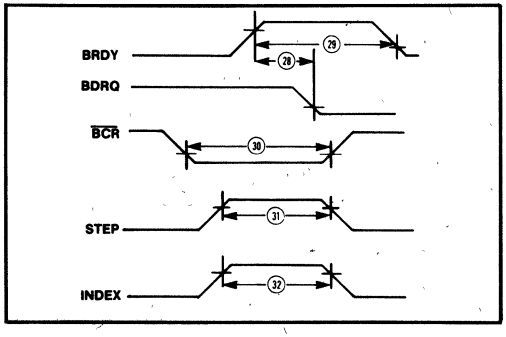
BUFFER WRITE TIMING (READ SECTOR COMMAND)

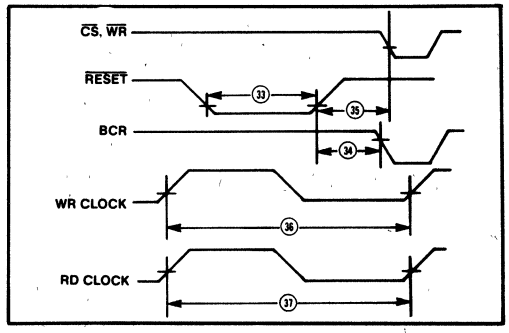
SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
22	$\overline{WR}$ Float to $\overline{WR}$ Valid	15		100	ns	C_L = 50pF
23	$\overline{WR}$ Output Pulse Width	300	400	500	ns	See Note 3
24	Data Valid from $\overline{WR}$↓			110	ns	
25	Data Hold from $\overline{WR}$↑	60			ns	
26	$\overline{WR}$ Repetition Rate	1.2	1.6	2.0	μs	See Note 1
27	$\overline{WR}$ Float from $\overline{BCS}$↑	15		100	ns	C_L = 50pF



MISCELLANEOUS TIMING

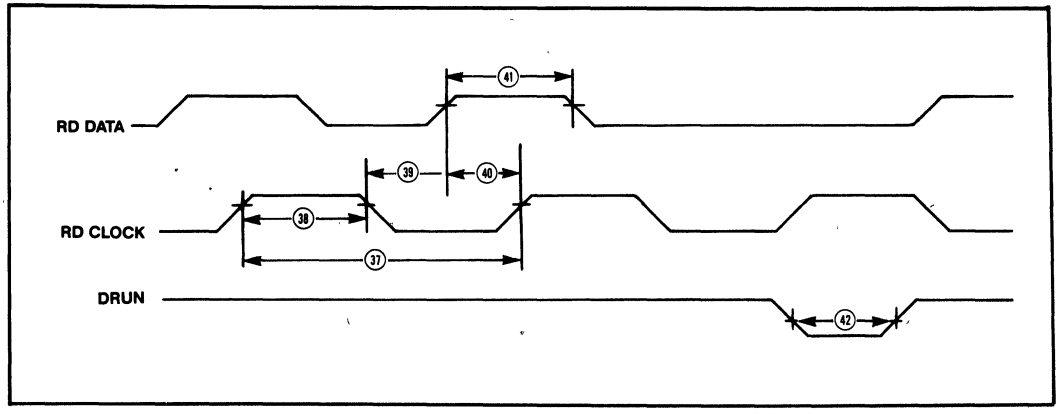
SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
28	BDRQ Reset from BRDY	40		200	ns	
29	BRDY Pulse Width	800			ns	See Note 4
30	$\overline{BCR}$ Pulse Width	1.4	1.6	1.8	μs	See Note 1
31	STEP Pulse Width	8.3	8.4	8.7	μs	See Note 1
32	INDEX Pulse Width	5000			ns	
33	$\overline{RESET}$ Pulse Width	24			WR CLK	See Note 2
34	$\overline{RESET}$↑ to $\overline{BCR}$	1.6	3.2	6.4	μs	See Note 1
35	$\overline{RESET}$↑ to $\overline{WR}$, $\overline{CS}$↓	6.4			μs	See Note 1
36	WR CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle
37	RD CLOCK Frequency	0.25	5.0	5.25	MHz	50% Duty Cycle See Note 5





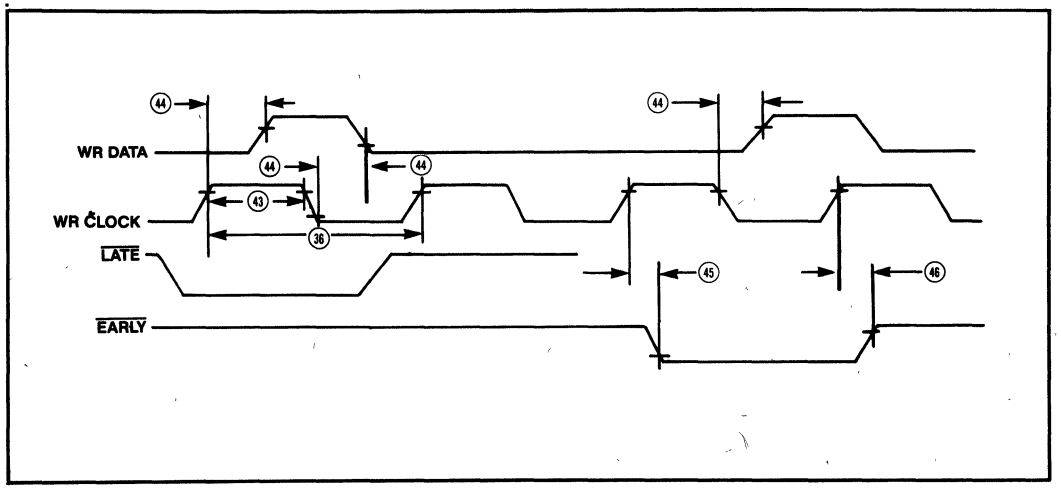
READ DATA TIMING

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
38	RD CLOCK Pulse Width	95		2000	ns	50% Duty Cycle
39	RD DATA after RD CLOCK↓	0		T38/2	ns	
40	RD DATA before RD CLOCK↑	20		T38/2	ns	
41	RD DATA Pulse Width	40		T38	ns	
42	DRUN PUlse Width	30			ns	

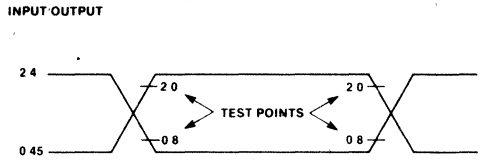


WRITE DATA TIMING

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT	TEST CONDITIONS
43	WR CLOCK Pulse Width	95		2000	ns	
44	Propogation Delay WR CLOCK to WR DATA	10		65	ns	
45	WR CLOCK to $\overline{\text{EARLY}}/\overline{\text{LATE}}$↓	10		65	ns	
46	WR CLOCK to $\overline{\text{EARLY}}/\overline{\text{LATE}}$↑	10		65	ns	

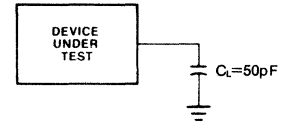


A.C. TESTING INPUT, OUTPUT WAVEFORM



AC TESTING: INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC .1, AND 0.45V FOR A LOGIC 0. TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC .1, AND0 8V FOR A LOGIC .0

A.C. TESTING LOAD CIRCUIT



C_L=50pF

C_L INCLUDES JIG CAPACITANCE

NOTES:
1. Based on WR CLOCK = 5.0 MHz.
2. 24 WR CLOCK periods = 4.8 μs at 5.0 MHz.
3. 2 WR CLOCK periods $\pm$ 100 ns.
4. BRDY must be 4 μs or a spurious BDRQ pulse may exist for up to 4 μs after the rising edge of BRDY.
5. WR CLOCK Frequency = RD CLOCK Frequency $\pm$ 15%.
6. 2 WR CLOCK periods $\pm$ 50 ns.

intel

MICROPROCESSOR PERIPHERALS
UPI™ USER'S MANUAL

APRIL 1982

CHAPTER 1
INTRODUCTION

Accompanying the introduction of microprocessors such as the 8080, 8085, 8088, and 8086 there has been a rapid proliferation of intelligent peripheral devices. These special purpose peripherals extend CPU performance and flexibility in a number of important ways.

Table 1-1. Intelligent Peripheral Devices

8255 (GPIO)	Programmable Peripheral Interface
8251A (USART)	Programmable Communication Interface
8253 (TIMER)	Programmable Interval Timer
8257 (DMA)	Programmable DMA Controller
8259	Programmable Interrupt Controller
8271 (SDFDC), 8272 (DDFDC)	Programmable Floppy Disk Controllers
8273 (SDLC)	Programmable Synchronous Data Link Controller
8274	Programmable Multiprotocol-Serial Communications Controller
8275/8276 (CRT)	Programmable CRT Controllers
8279 (PKD)	Programmable Keyboard/Display Controller
8291A, 8292, 8293	Programmable GPIB System Talker, Listener, Controller

Intelligent devices like the 8272 floppy disk controller and 8273 synchronous data link controller (see Table 1-1) can preprocess serial data and perform control tasks which off-load the main system processor. Higher overall system throughout is achieved and software complexity is greatly reduced. The intelligent peripheral chips simplify master processor control tasks by performing many functions externally in peripheral hardware rather than internally in main processor software.

Intelligent peripherals also provide system flexibility. They contain on-chip mode registers which are programmed by the master processor during system initialization. These control registers allow the peripheral to be configured into many different operation modes. The user-defined program for the peripheral is stored in main system memory and is transferred to the peripheral's registers whenever a mode change is required. Of course, this type of flexibility requires software overhead in the master system which tends to limit the benefit derived from the peripheral chip.

In the past, intelligent peripherals were designed to handle very specialized tasks. Separate chips were designed for communication disciplines, parallel I/O, keyboard encoding, interval timing, CRT control, etc. Yet, in spite of the large number of devices available and the increased flexibility built into these chips, there is still a large number of microcomputer peripheral control tasks which are not satisfied.

With the introduction of the Universal Peripheral Interface (UPI) microcomputer, Intel has taken the intelligent peripheral concept a step further by providing an intelligent controller that is fully user programmable. It is a complete single-chip microcomputer which can connect directly to a master processor data bus. It has the same advantages of intelligence and flexibility which previous peripheral chips offered. In addition, the UPI is user-programmable: it has 1K bytes of ROM or EPROM memory for program storage plus 64 bytes of RAM memory for data storage or initialization from the master processor. The UPI device allows a designer to fully specify his control algorithm in the peripheral chip without relying on the master processor. Devices like printer controllers and keyboard scanners can be completely self-contained, relying on the master processor only for data transfer.

The UPI family currently consists of five components:

- 8741A microcomputer with 1K EPROM memory
- 8041AH microcomputer with 1K ROM memory
- 8042 microcomputer with 2K ROM memory
- 8243 I/O expander device
- 8742 microcomputer with 2K EPROM memory

The 8741A, 8041AH, 8742 and 8042 single chip microcomputers are functionally equivalent except for the type and amount of program memory available with each. These devices have the following main features:

- 8-bit CPU
- 8-bit data bus interface registers
- 1K by 8 bit ROM or EPROM memory (2K for 8042/8742)
- 64 by 8 bit RAM memory (128 bytes for 8042/8742)
- Interval timer/event counter
- Two 8-bit TTL compatible I/O ports
- Resident clock oscillator
- 12 MHZ operation, 1.25 μsec instruction cycle for 8041AH, 8742, 8042

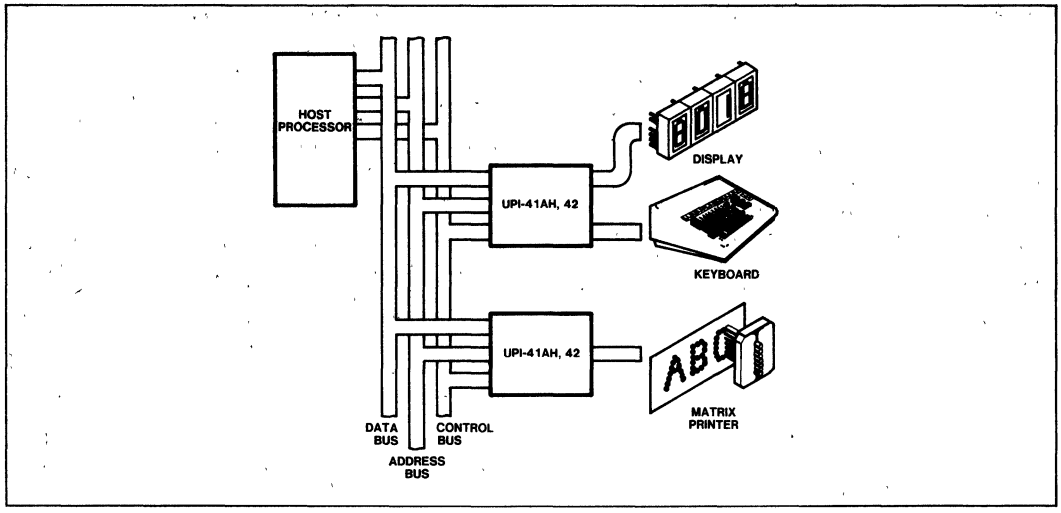


Figure 1-1. Interfacing Peripherals To Microcomputer Systems

HMOS processing has been applied to the UPI family to allow for additional performance and memory capability while reducing costs. The 8041AH, 8741A, 8042, 8742 are all pin and software compatible. This allows growth in present designs to incorporate new features and add additional performance. For new designs, the additional memory and performance of the 8042/8742 extends the UPI 'grow your own solution' concept to more complex motor control tasks, 80-column printers and process control applications as examples.

The 8243 device is an I/O multiplexer which allows expansion of I/O to over 100 lines (if seven devices are used). All three parts are fabricated with N-channel MOS technology and require a single, 5V supply for operation.

INTERFACE REGISTERS FOR MULTI-PROCESSOR CONFIGURATIONS

In the normal configuration, the 8041AH/8741A, 8042/8742 interfaces to the system bus, just like any intelligent peripheral device (see Figure 1-1). The host processor and the 8041AH/8741A, 8042/8742 form a loosely coupled multi-processor system, that is, communications between the two processors are direct. Common resources are three addressable registers located physically on the 8041AH/8741A, 8042/8742. These registers are the Data Bus Buffer Input (DBBIN), Data Bus Buffer Output (DBBOUT), and Status (STATUS) registers. The host processor may read data from DBBOUT or write commands and data into DBBIN. The status of DBBOUT and DBBIN plus user-defined status is supplied in STATUS. The host may read STATUS at any time. An interrupt to the UPI processor is automatically generated (if enabled) when DBBIN is loaded.

Because the UPI contains a complete microcomputer with program memory, data memory, and CPU it can function as a "Universal" controller. A designer can program the UPI to control printers, tape transports, or multiple serial communication channels. The UPI can also handle off-line arithmetic processing, or any number of other low speed control tasks.

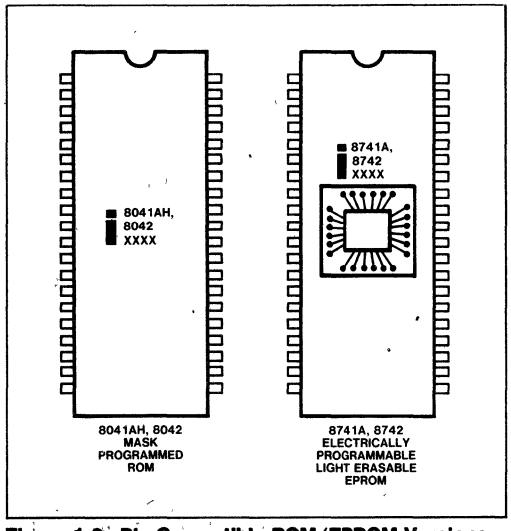


Figure 1-2. Pin Compatible ROM/EPROM Versions

POWERFUL 8-BIT PROCESSOR

The UPI contains a powerful, 8-bit CPU with as fast as 1.25 μsec cycle time and two single-level interrupts. Its instruction set includes over 90 instructions for easy software development. Most instructions are single byte and single cycle and none are more than two bytes long. The instruction set is optimized for bit manipulation and I/O operations. Special instructions are included to allow binary or BCD arithmetic operations, table lookup routines, loop counters, and N-way branch routines.

SPECIAL INSTRUCTION SET FEATURES

- For Loop Counters:
  Decrement Register and Jump if not zero.
- For Bit Manipulation:
  AND to A (immediate data or Register)
  OR to A (immediate data or Register)
  XOR to A (immediate data or Register)
  AND to Output Ports (Accumulator)
  OR to Output Ports (Accumulator)
  Jump Conditionally on any bit in A
- For BDC Arithmetic:
  Decimal Adjust A
  Swap 4-bit Nibbles of A
  Exchange lower nibbles of A and Register
  Rotate A left or right with or without Carry
- For Lookup Tables:
  Load A from Page of ROM (Address in A)
  Load A from Current Page of ROM (Address in A)

Features for Peripheral Control

The UPI 8-bit interval timer/event counter can be used to generate complex timing sequences for control applications or it can count external events such as switch closures and position encoder pulses. Software timing loops can be simplified or eliminated by the interval timer. If enabled, an interrupt to the CPU will occur when the timer overflows.

The UPI I/O complement contains two TTL-compatible 8-bit bidirectional I/O ports and two general-purpose test inputs. Each of the 16 port lines can individually function as either input or output under software control. Four of the port lines can also function as an interface for the 8243 I/O expander which provides four additional 4-bit ports that are directly addressable by UPI software. The 8243 expander allows low cost I/O expansion for large control applications while maintaining easy and efficient software port addressing.

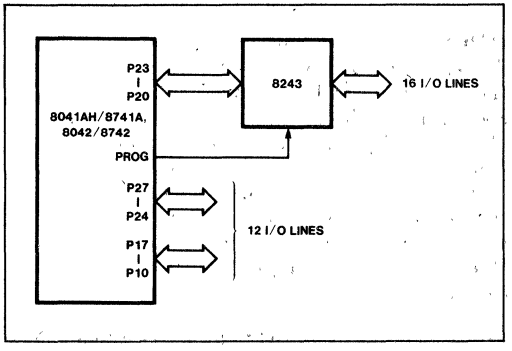


Figure 1-4. 8243 I/O Expander Interface

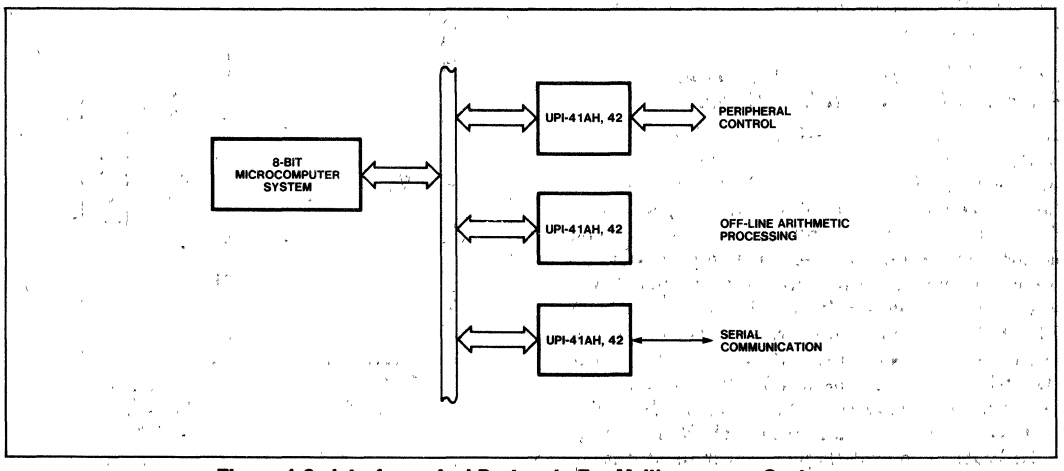


Figure 1-3. Interfaces And Protocols For Multiprocessor Systems

On-Chip Memory

The UPI's 64 (128) bytes of data memory include dual working register banks and an 8-level program counter stack. Switching between the register banks allows fast response to interrupts. The stack is used to store return addresses and processor status upon entering a subroutine.

The UPI program memory is available in two types to allow flexibility in moving from design to prototype to production with the same PC layout. The 8741A, 8742 device with EPROM memory is very economical for initial system design and development. Its program memory can be electrically programmed using the Intel Universal PROM Programmer. When changes are needed, the entire program can be erased using UV lamp and reprogrammed in about 20 minutes. This means the 8741A/8742 can be used as a single chip "breadboard" for very complex interface and control problems. After the 8741A/8742 is programmed it can be tested in the actual production level PC board and the actual functional environment. Changes required during system debugging can be made in the 8741A/8742 program much more easily than they could be made in a random logic design. The system configuration and PC layout can remain fixed during the development process and the turn around time between changes can be reduced to a minimum.

At any point during the development cycle, the 8741A/8742 EPROM part can be replaced with the low cost 8041AH, 8042 respectively with factory mask programmed memory. The transition from system development to mass production is made smoothly because the 8741A and 8041AH, 8742 and 8042 parts are completely pin compatible. 8742s or 8042s can be used in an 8041AH/8741 socket. This feature allows extensive testing with the EPROM part, even into initial shipments to customers. Yet, the transition to low-cost ROM is simplified to the point of being merely a package substitution.

PREPROGRAMMED UPI's

The 8292, 8294, and 8295 are 8041A's that are programmed by Intel and sold as standard peripherals. The 8292 is a GPIB controller, part of a three chip GPIB system. The 8294 is a Data Encryption Unit that implements the National Bureau of Standards data encryption algorithm. The 8295 is a dot matrix printer controller designed especially for the LRC 7040 series dot matrix impact printers. These parts illustrate the great flexibility offered by the UPI family.

DEVELOPMENT SUPPORT

The UPI microcomputer is fully supported by Intel with development tools like the UPP PROM programmer already mentioned. An ICE-41A in-circuit emulator is also available to allow UPI software and hardware to be developed easily and quickly. The combination of device features and Intel development support make the UPI an ideal component for low-speed peripheral control applications.

UPI DEVELOPMENT SUPPORT

- 8048/8041AH/8042 Assembler
- Universal PROM Programmer UPP Series
- ICE-41A Module
- MULTI-ICE
- Insite User's Library
- Application Engineers
- Training Courses

CHAPTER 2
FUNCTIONAL DESCRIPTION

The UPI-41AH, 42 microcomputer is an intelligent peripheral controller designed to operate in iAPX-86, 88, MCS-85, MCS-80, MCS-51 and MCS-48 systems. The UPI'S architecture, illustrated in Figure 2-1, is based on a low cost, single-chip microcomputer with program memory, data memory, CPU, I/O, event timer and clock oscillator in a single 40-pin package. Special interface registers are included which enable the UPI to function as a peripheral to an 8-bit master processor.

This chapter provides a basic description of the UPI microcomputer and its system interface registers. Unless otherwise noted the descriptions in this section apply to both the 8741A, 8742 (with UV erasable program memory) and the 8041AH, 8042 (with factory mask programmed memory). These two devices are so similar that they can be considered identical under most circumstances. All functions described in this chapter apply to the 8041AH, 8042, and 8741A, 8742.

PIN DESCRIPTION

The 8041AH/8741A, 8042/8742 are packaged in 40-pin Dual In-Line (DIP) packages. The pin configuration for both devices is shown in Figure 2-2. Figure 2-3 illustrates the UPI Logic Symbol.

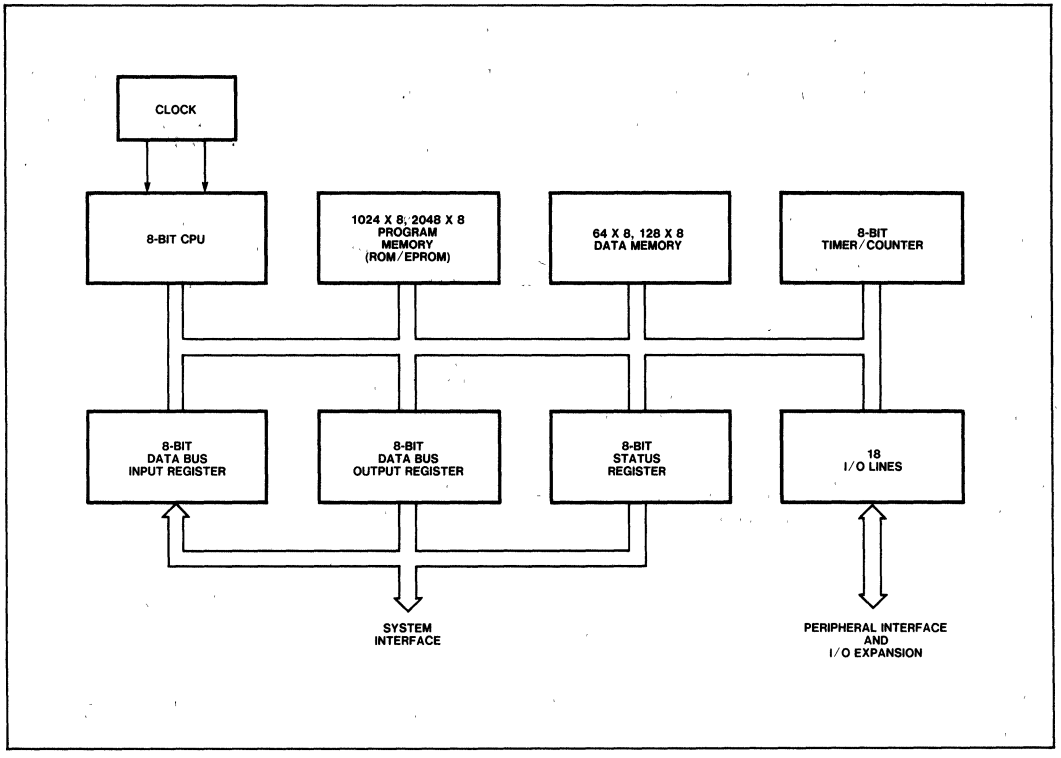


Figure 2-1. UPI-41AH, 42 Single Chip Microcomputer

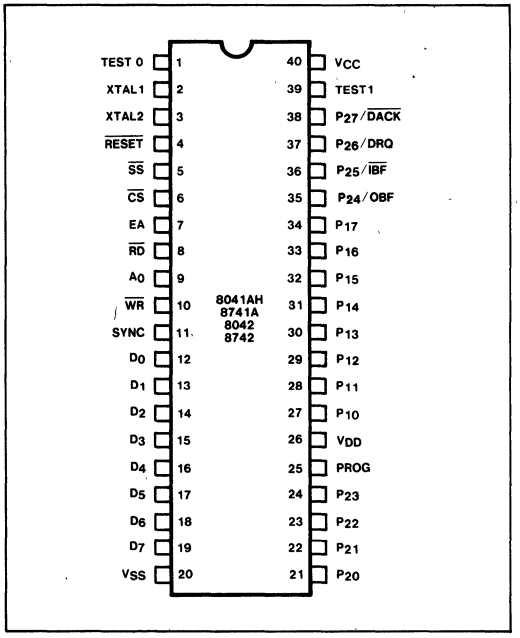


Figure 2-2. Pin Configuration

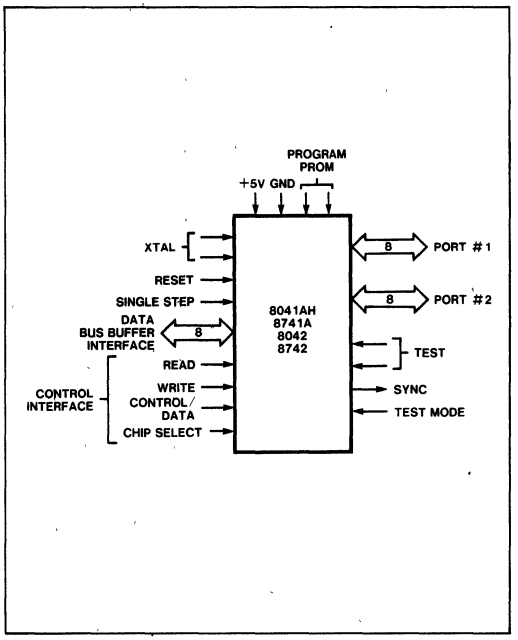


Figure 2-3. Logic Symbol

The following section summarizes the functions of each UPI-41A pin. NOTE that several pins have two or more functions which are described in separate paragraphs.

Table 2-1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_0-D_7 (BUS)	12-19	I/O	**Data Bus:** Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41AH, 42 microcomputer to an 8-bit master system data bus.
P_{10}-P_{17}	27-34	I/O	**Port 1:** 8-bit, PORT 1 quasi-bidirectional I/O lines.
P_{20}-P_{27}	21-24 35-38	I/O	**Port 2:** 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}-P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4-7 access. The upper 4 bits (P_{24}-P_{27}) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P_{24} as Output Buffer Full (OBF) interrupt, P_{25} as Input Buffer Full ($\overline{IBF}$) interrupt, P_{26} as DMA Request (DRQ), and P_{27} as DMA ACKnowledge ($\overline{DACK}$).
$\overline{WR}$	10	I	**Write:** I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.
$\overline{RD}$	8	I	**Read:** I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
$\overline{CS}$	6	I	**Chip Select:** Chip select input used to select one UPI-41AH, 42 microcomputer out of several connected to a common data bus.
A_0	9	I	**Command/Data Select:** Address input used by the master processor to indicate whether byte transfer is data (A_0=0) or command (A_0=1).
TEST 0, TEST 1	1 39	I	**Test Inputs:** Input pins which can be directly tested using conditional branch instructions. **Frequency Reference:** TEST 1 (T_1) also functions as the event timer input (under software control). TEST 0 (T_0) is used during PROM programming and verification in the 8741A, 8742.

Table 2-1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
XTAL 1, XTAL 2	2 3	I	**Inputs:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
SYNC	11	O	**Output Clock:** Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
EA	7	I	**External Access:** External access input which allows emulation, testing and PROM/ROM verification.
PROG	25	I/O	**Program:** Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.
$\overline{\text{RESET}}$	4	I	**Reset:** Input used to reset status flip-flops and to set the program counter to zero. $\overline{\text{RESET}}$ is also used during PROM programming and verification.
$\overline{\text{SS}}$	5	I	**Single Step:** Single step input used in conjunction with the SYNC output to step the program through each instruction.
V_{CC}	40		**Power:** +5V main power supply pin.
V_{DD}	26		**Power:** +5V during normal operation. +25V during programming operation, +21V for programming 8742. Low power standby pin in ROM version.
V_{SS}	20		**Ground:** Circuit ground potential.

The following sections provide a detailed functional description of the UPI microcomputer. Figure 2-4 illustrates the functional blocks within the UPI device.

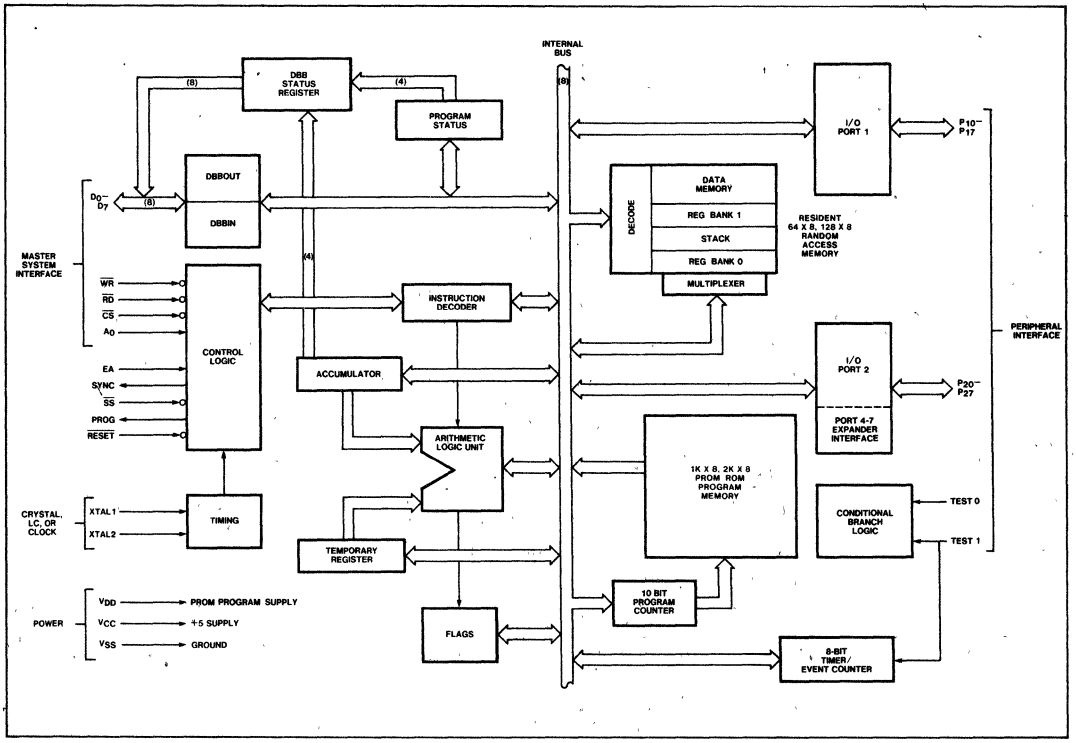


Figure 2-4. UPI-41AH, 42™ Block Diagram

CPU SECTION

The CPU section of the UPI-41AH, 42 microcomputer performs basic data manipulations and controls data flow throughout the single chip computer via the internal 8-bit data bus. The CPU section includes the following functional blocks shown in Figure 2-4:

- Arithmetic Logic Unit (ALU)
- Instruction Decoder
- Accumulator
- Flags

Arithmetic Logic Units (ALU)

The ALU is capable of performing the following operations:

- ADD with or without carry
- AND, OR, and EXCLUSIVE OR
- Increment, Decrement
- Bit complement
- Rotate left or right
- Swap
- BCD decimal adjust

In a typical operation data from the accumulator is combined in the ALU with data from some other source on the UPI-41AH, 42 internal bus (such as a register or an I/O port). The result of an ALU operation can be transferred to the internal bus or back to the accumulator.

If an operation such as an ADD or ROTATE requires more than 8 bits, the CARRY flag is used as an indicator. Likewise, during decimal adjust and other BCD operations the AUXILIARY CARRY flag can be set and acted upon. These flags are part of the Program Status Word (PSW).

Instruction Decoder

During an instruction fetch, the operation code (opcode) portion of each program instruction is stored and decoded by the instruction decoder. The decoder generates outputs used along with various timing signals to control the functions performed in the ALU. Also, the instruction decoder controls the source and destination of ALU data.

Accumulator

The accumulator is the single most important register in the processor. It is the primary source of data to the ALU and is often the destination for results as well. Data to and from the I/O ports and memory normally passes through the accumulator.

PROGRAM MEMORY

The UPI-41AH, 42 microcomputer has 1024, 2048 8-bit words of resident, read-only memory for program storage. Each of these memory locations is directly addressable by a 10-bit program counter. Depending on the type of application and the number of program changes anticipated, two types of program memory are available:

- 8041AH, 8042 with mask programmed ROM Memory
- 8741A, 8742 with electrically programmable EPROM Memory

The 8041AH and 8741A, 8042 and 8742 are functionally identical parts and are completely pin compatible. The 8742 and 8042 can be used in 8041AH, 8741A sockets. The 8041AH, 8042 has ROM memory which is mask programmed to user specification during fabrication. The 8741A/8742 are electrically programmed by the user using the Universal PROM Programmer (UPP series) with a UPP-848 or UPP-549 Personality Card. It can be erased using ultraviolet light and reprogrammed at any time.

A program memory map is illustrated in Figure 2-5. Memory is divided into 256 location 'pages' and three locations are reserved for special use:

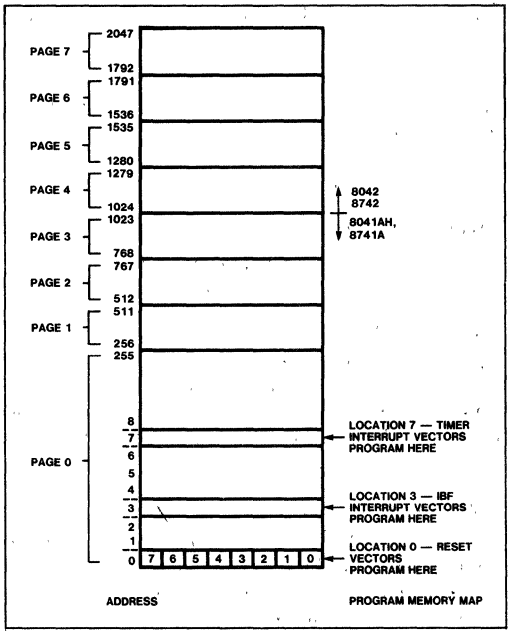


Figure 2-5. Program Memory Map

INTERRUPT VECTORS

1) **Location 0**
Following a $\overline{\text{RESET}}$ input to the processor, the next instruction is automatically fetched from location 0.

2) **Location 3**
An interrupt generated by an Input Buffer Full (IBF) condition (when the IBF interrupt is enabled) causes the next instruction to be fetched from location 3.

3) **Location 7**
A timer overflow interrupt (when enabled) will cause the next instruction to be fetched from location 7.

Following a system $\overline{\text{RESET}}$, program execution begins at location 0. Instructions in program memory are normally executed sequentially. Program control can be transferred out of the main line of code by an input buffer full (IBF) interrupt or a timer interrupt, or when a jump or call instruction is encountered. An IBF interrupt (if enabled) will automatically transfer control to location 3 while a timer interrupt will transfer control to location 7.

All conditional JUMP instructions and the indirect JUMP instruction are limited in range to the current 256-location page (that is, they alter PC bits 0–7 only). If a conditional JUMP or indirect JUMP begins in location 255 of a page, it must reference a destination on the following page.

Program memory can be used to store constants as well as program instructions. The UPI-41AH, 42 instruction set contains an instruction (MOVP3) designed specifically for efficient transfer of look-up table information from page 3 of memory.

DATA MEMORY

The UPI-41AH, 42 universal peripheral interface has 64, 128 8-bit words of random access data memory. This memory contains two working register banks, an 8-level program counter stack and a scratch pad memory, as shown in Figure 2-6. The amount of scratch pad memory available is variable depending on the number of addresses nested in the stack and the number of working registers being used.

Addressing Data Memory

The first eight locations in RAM are designated as working registers R_0–R_7. These locations (or registers) can be addressed directly by specifying a register number in the instruction. Since these locations are easily addressed, they are generally used to store frequently accessed intermediate results. Other locations in data memory are addressed indirectly by using R_0 or R_1 to specify the desired address. Since all RAM locations (including the eight working registers) can be addressed by 6 bits, the two most significant bits (6 and 7) of the addressing registers are ignored.

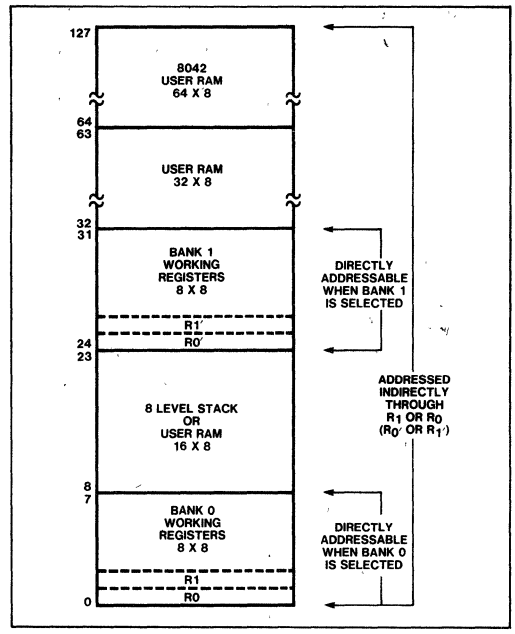


Figure 2-6. Data Memory Map

Working Registers

Dual banks of eight working registers are included in the UPI-41AH, 42 data memory. Locations 0–7 make up register bank 0 and locations 24–31 form register bank 1. A $\overline{\text{RESET}}$ signal automatically selects register bank 0. When bank 0 is selected, references to R_0–R_7 in UPI-41AH, 42 instructions operate on locations 0–7 in data memory. A "select register bank" instruction is used to select between the banks during program execution. If the instruction SEL RB1 (Select Register Bank 1) is executed, then program references to R_0–R_7 will operate on locations 24–31. As stated previously, registers 0 and 1 in the active register bank are used as indirect address registers for all locations in data memory.

Register bank 1 is normally reserved for handling interrupt service routines, thereby preserving the contents of the main program registers. The SEL RB1 instruction can be issued at the beginning of an interrupt service routine. Then, upon return to the main program, an RETR (return & restore status) instruction will automatically restore the previously selected bank. During interrupt processing, registers in bank 0 can be accessed indirectly using R_0' and R_1'.

If register bank 1 is not used, registers 24–31 can still serve as additional scratch pad memory.

Program Counter Stack

RAM locations 8–23 are used as an 8-level program counter stack. When program control is temporarily passed from the main program to a subroutine or interrupt service routine, the 10-bit program counter and bits 4–7 of the program status word (PSW) are stored in two stack locations. When control is returned to the main program via an RETR instruction, the program counter and PSW bits 4–7 are restored. Returning via an RET instruction does not restore the PSW bits, however. The program counter stack is addressed by three stack pointer bits in the PSW (bits 0–2). Operation of the program counter stack and the program status word is explained in detail in the following sections.

The stack allows up to eight levels of subroutine 'nesting'; that is, a subroutine may call a second subroutine, which may call a third, etc., up to eight levels. Unused stack locations can be used as scratch pad memory. Each unused level of subroutine nesting provides two additional RAM locations for general use.

The following sections provide a detailed description of the Program Counter Stack and the Program Status Word.

PROGRAM COUNTER

The UPI-41AH, 42 microcomputer has a 10-bit program counter (PC) which can directly address any of the 1024 locations in program memory. The program counter always contains the address of the next instruction to be executed and is normally incremented sequentially for each instruction to be executed when each instruction fetches occurs.

When control is temporarily passed from the main program to a subroutine or an interrupt routine, however, the PC contents must be altered to point to the address of the desired routine. The stack is used to save the current PC contents so that, at the end of the routine, main program execution can continue. The program counter is initialized to zero by a $\overline{\text{RESET}}$ signal.

PROGRAM COUNTER STACK

The Program Counter Stack is composed of 16 locations in Data Memory as illustrated in Figure 2-7. These RAM locations (8 through 23) are used to store the 10-bit program counter and 4 bits of the program status word.

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the program counter stack.

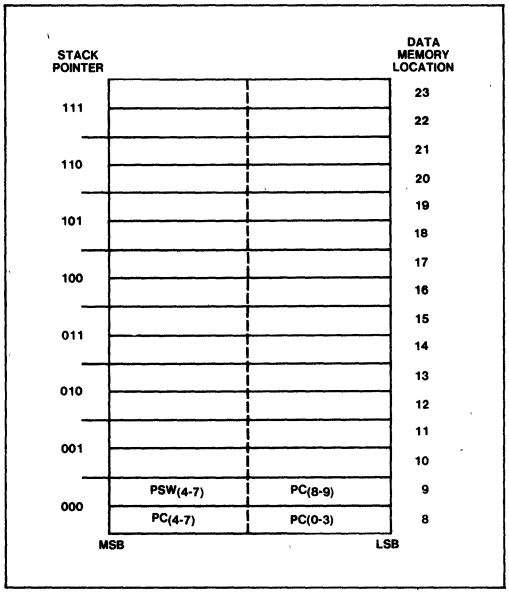


Figure 2-7. Program Counter Stack

A 3-bit Stack Pointer which is part of the Program Status Word (PSW) determines the stack pair to be used at a given time. The stack pointer is initialized by a $\overline{\text{RESET}}$ signal to 00H which corresponds to RAM locations 8 and 9.

The first call or interrupt results in the program counter and PSW contents being transferred to RAM locations 8 and 9 in the format shown in Figure 2-7. The stack pointer is automatically incremented by 1 to point to locations 10 and 11 in anticipation of another CALL.

Nesting of subroutines within subroutines can continue up to 8 levels without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 07H to 00H. Likewise, the stack pointer will underflow from 00H to 07H.

The end of a subroutine is signaled by a return instruction, either RET or RETR. Each instruction will automatically decrement the Stack Pointer and transfer the contents of the proper RAM register pair to the Program Counter.

PROGRAM STATUS WORD

The 8-bit program status word illustrated in Figure 2-8 is used to store general information about program execution. In addition to the 3-bit Stack

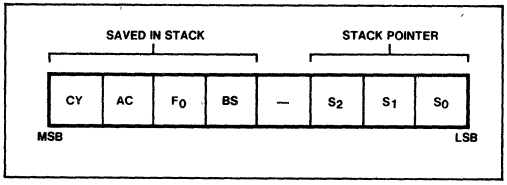


Figure 2-8. Program Status Word

Pointer discussed previously, the PSW includes the following flags:

- CY — Carry
- AC — Auxiliary Carry
- F_0 — Flag 0
- BS — Register Bank Select

The Program Status Word (PSW) is actually a collection of flip-flops located throughout the machine which are read or written as a whole. The PSW can be loaded to or from the accumulator by the MOV A, PSW or MOV PSW,A instructions. The ability to write directly to the PSW allows easy restoration of machine status after a power-down sequence.

The upper 4 bits of the PSW (bits 4, 5, 6, and 7) are stored in the PC Stack with every subroutine CALL or interrupt vector. Restoring the bits on a return is optional. The bits are restored if an RETR instruction is executed, but not if an RET is executed.

PSW bit definitions are as follows:

- Bits 0–2 Stack Pointer Bits S_0, S_1, S_2
- Bit 3 Not Used
- Bit 4 Working Register Bank
  0 = Bank 0
  1 = Bank 1
- Bit 5 Flag 0 bit (F_0)
  This is a general purpose flag which can be cleared or complemented and tested with conditional jump instructions. It may be used during data transfer to an external processor.
- Bit 6 Auxiliary Carry (AC)
  The flag status is determined by an ADD instruction and is used by the Decimal Adjustment instruction DAA.
- Bit 7 Carry (CY)
  The flag indicates that a previous operation resulted in overflow of the accumulator.

CONDITIONAL BRANCH LOGIC

Conditional Branch Logic in the UPI-41AH, 42 allows the status of various processor flags, inputs, and other hardware functions to directly affect program execution. The status is sampled in state 3 of the first cycle.

Table 2-2 lists the internal conditions which are testable and indicates the condition which will cause a jump. In all cases, the destination address must be within the page of program memory (256 locations) in which the jump instruction occurs.

OSCILLATOR AND TIMING CIRCUITS

The 8041A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 2-9. Figure 2-10 shows instruction cycle timing.

Table 2-2. Conditional Branch Instructions

Device	Instruction Mnemonic		Jump Condition Jump if:
Accumulator	JZ	addr	All bits zero
	JNZ	addr	Any bit not zero
Accumulator bit	JBb	addr	Bit "b" = 1
Carry flag	JC	addr	Carry flag = 1
	JNC	addr	Carry flag = 0
User flag	JF0	addr	F_0 flag = 1
	JF1	addr	F_1 flag = 1
Timer flag	JTF	addr	Timer flag = 1
Test Input 0	JT0	addr	T_0 = 1
	JNT0	addr	T_0 = 0
Test Input 1	JT1	addr	T_1 = 1
	JNT1	addr	T_1 = 0
Input Buffer flag	JNIBF	addr	IBF flag = 0
Output Buffer flag	JOBF	addr	OBF flag = 1

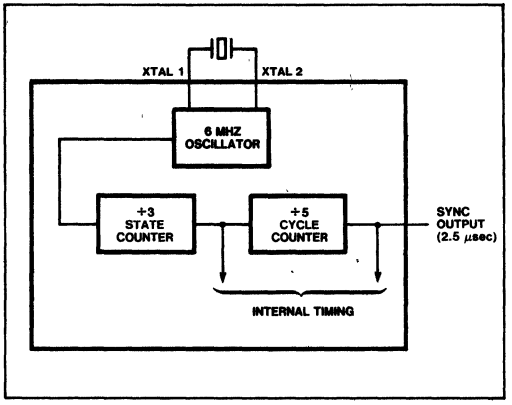


Figure 2-9. Oscillator Configuration

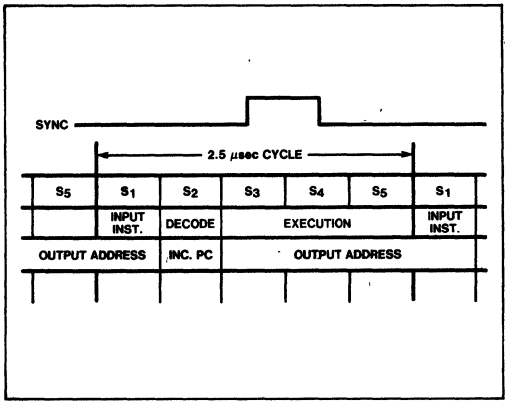


Figure 2-10. Instruction Cycle Timing

Oscillator

The on-board oscillator is a series resonant circuit with a frequency range of 1 to 12 (8041AH-2/8042/8742) MHz. Pins XTAL 1 and XTAL 2 are input and output (respectively) of a high gain amplifier stage. A crystal or inductor and capacitor connected between XTAL 1 and XTAL 2 provide the feedback and proper phase shift for oscillation. Recommended connections for crystal or L-C are shown in Figure 2-11.

State Counter

The output of the oscillator is divided by 3 in the state counter to generate a signal which defines the state times of the machine.

Each instruction cycle consists of five states as illustrated in Figure 2-10 and Table 2-3. The overlap of address and execution operations illustrated in Figure 2-10 allows fast instruction execution.

Table 2-3. Instruction Timing Diagram

INSTRUCTION	CYCLE 1					CYCLE 2				
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,P_P	Fetch Instruction	Increment Program Counter	—	Increment Timer	—	—	Read Port	—	—	—
OUTL P_P,A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Output To Port	—	—	—	—	—
ANL P_P, DATA	Fetch Instruction	Increment Program Counter	—	Increment Timer	Read Port	Fetch Immediate Data	—	Increment Program Counter	Output To Port	—
ORL P_P, DATA	Fetch Instruction	Increment Program Counter	—	Increment Timer	Read Port	Fetch Immediate Data	—	Increment Program Counter	Output To Port	—
MOVD A,P_P	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	—	—	Read P2 Lower	—	—	—
MOVD P_P,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data To P2 Lower	—	—	—	—	—
ANLD P_P,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data	—	—	—	—	—
ORLD P_P,A	Fetch Instruction	Increment Program Counter	Output Opcode/Address	Increment Timer	Output Data	—	—	—	—	—
J (Conditional)	Fetch Instruction	Increment Program Counter	Sample Condition	Increment Timer	—	Fetch Immediate Data	—	Update Program Counter	—	—
MOV STS, A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Update Status Register					
IN A,DBB	Fetch Instruction	Increment Program Counter	—	Increment Timer	—					
OUT DBB,A	Fetch Instruction	Increment Program Counter	—	Increment Timer	Output To Port					
STRT T STRT CNT	Fetch Instruction	Increment Program Counter	—	—	Start Counter					
STOP TCNT	Fetch Instruction	Increment Program Counter	—	—	Stop Counter					
EN I	Fetch Instruction	Increment Program Counter	—	Enable Interrupt	—					
DIS I	Fetch Instruction	Increment Program Counter	—	Disable Interrupt	—					
EN DMA	Fetch Instruction	Increment Program Counter	—	DMA Enabled DRQ Cleared	—					
EN FLAGS	Fetch Instruction	Increment Program Counter	—	OBF, $\overline{IBF}$ Output Enabled	—					

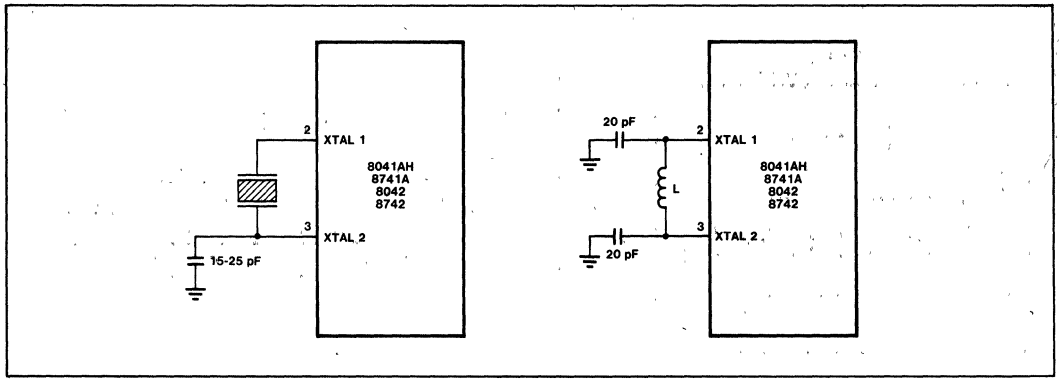


Figure 2-11. Recommended Crystal and L-C Connections

Cycle Counter

The output of the state counter is divided by 5 in the cycle counter to generate a signal which defines a machine cycle. This signal is call SYNC and is available continously on the SYNC output pin. It can be used to synchronize external circuitry or as a general purpose clock output. It is also used for synchronizing single-step.

Frequency Reference

The external crystal provides high speed and accurate timing generation. A crystal frequency of 5.9904 MHz is useful for generation of standard communication frequencies by the 8041AH/8741, 8042/8742. However, if an accurate frequency reference and maximum processor speed are not required, an inductor and capacitor may be used in place of the crystal as shown in Figure 2-11.

A recommended range of inductance and capacitance combinations is given below:

- L = 130 μH corresponds to 3 MHz
- L = 45 μH corresponds to 5 MHz

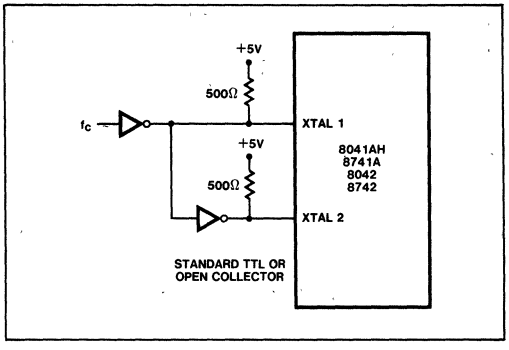


Figure 2-12. Recommended Connection For External Clock Signal

An external clock signal can also be used as a frequency reference to the 8741AH, 8741A, 8742 or 8042; however, the levels are *not* TTL compatible. The signal must be in the 1–12 MHz frequency range and must be connected to pins XTAL 1 and XTAL 2 by buffers with a suitable pull-up resistor to guarantee that a logic "1" is above 3.8 volts. The recommended connection is shown in Figure 2-12.

INTERVAL TIMER/EVENT COUNTER

The 8041AH, 8042 has a resident 8-bit timer/counter which has several software selectable modes of operation. As an interval timer, it can generate accurate delays from 80 microseconds to 20.48 milliseconds without placing undue burden on the processor. In the counter mode, external events such as switch closures or tachometer pulses can be counted and used to direct program flow.

Timer Configuration

Figure 2-13 illustrates the basic timer/counter configuration. An 8-bit register is used to count pulses from either the internal clock and prescaler or from an external source. The counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice-versa (i.e. MOV T, A and MOV A, T). The counter is stopped by a $\overline{\text{RESET}}$ or STOP TCNT instruction and remains stopped until restarted either as a timer (START T instruction) or as a counter (START CNT instruction). Once started, the counter will increment to its maximum count (FFH) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or $\overline{\text{RESET}}$.

The increment from maximum count to zero (overflow) results in setting the Timer Flag (TF) and generating an interrupt request. The state of the overflow flag is testable with the conditional jump

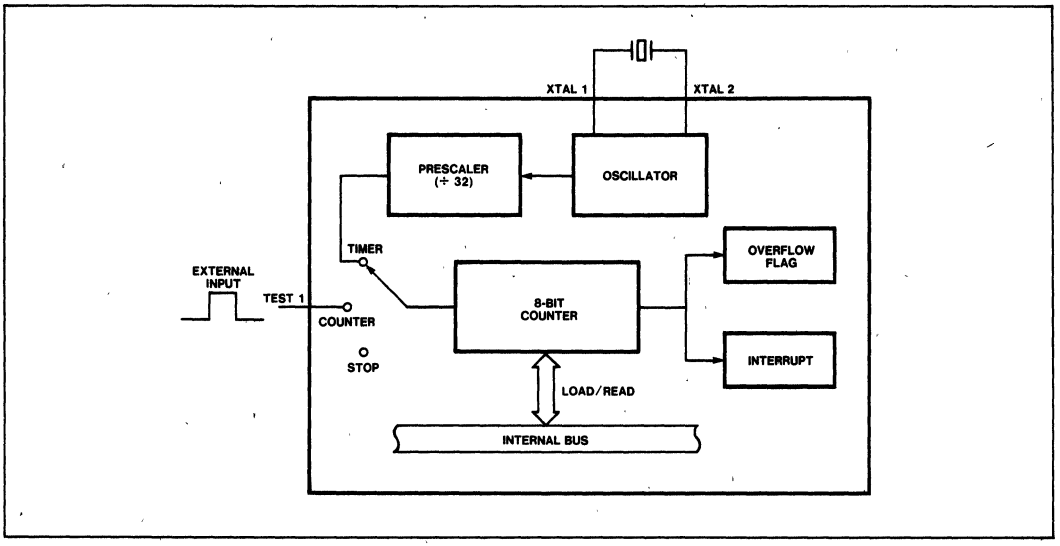


Figure 2-13. Timer Counter

instruction, JTF. The flag is reset by executing a JTF or by a RESET signal.

The timer interrupt request is stored in a latch and ORed with the input buffer full interrupt request. The timer interrupt can be enabled or disabled independent of the IBF interrupt by the EN TCNTI and DIS TCTNI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer service routine is stored. If the timer and Input Buffer Full interrupts occur simultaneously, the IBF source will be recognized and the call will be to location 3. Since the timer interrupt is latched, it will remain pending until the DBBIN register has been serviced and will immediately be recognized upon return from the service routine. A pending timer interrupt is reset by the initiation of a timer interrupt service routine.

Event Counter Mode

The STRT CNT instruction connects the TEST 1 input pin to the counter input and enables the counter. Note this instruction does not clear the counter. The counter is incremented on high to low transitions of the TEST 1 input. The TEST 1 input must remain high for a minimum of one state in order to be registered (250 ns at 12 MHz). The maximum count frequency is one count per three instruction cycles (267 kHz at 12 MHz). There is no minimum frequency limit.

Timer Mode

The STRT T instruction connects the internal clock to the counter input and enables the counter. The input clock is derived from the SYNC signal of the internal oscillator and the divide-by-32 prescaler. The configuration is illustrated in Figure 2-13. Note this instruction does not clear the timer register. Various delays and timing sequences between 40 μsec and 10.24 msec can easily be generated with a minimum of software timing loops (at 12 MHz).

Times longer than 10.24 msec can be accurately measured by accumulating multiple overflows in a register under software control. For time resolution less than 40 μsec, an external clock can be applied to the TEST 1 counter input (see Event Counter Mode). The minimum time resolution with an external clock is 3.75 μsec (267 kHz at 12 MHz).

TEST 1 Event Counter Input

The TEST 1 pin is multifunctional. It is automatically initialized as a test input by a RESET signal and can be tested using UPI-41A conditional branch instructions.

In the second mode of operation, illustrated in Figure 2-13, the TEST 1 pin is used as an input to the internal 8-bit event counter. The Start Counter (STRT CNT) instruction controls an internal switch which connects TEST 1 through an edge detector to the 8-bit internal counter. Note that this instruction does not inhibit the testing of TEST 1 via conditional Jump instructions.

In the counter mode the TEST 1 input is sampled once per instruction cycle. After a high level is detected, the next occurence of a low level at TEST 1

will cause the counter to increment by one.

The event counter functions can be stopped by the Stop Timer/Counter (STOP TCNT) instruction. When this instruction is executed the TEST 1 pin becomes a test input and functions as previously described.

TEST INPUTS

There are two multifunction pins designated as Test Inputs, TEST 0 and TEST 1. In the normal mode of operation, status of each of these lines can be directly tested using the following conditional Jump instructions:

- JT0 Jump if TEST 0 = 1
- JNT0 Jump if TEST 0 = 0
- JT1 Jump if TEST 1 = 1
- JNT1 Jump if TEST 1 = 0

The test inputs are TTL compatible. An external logic signal connected to one of the test inputs will be sampled at the time the appropriate conditional jump instruction is executed. The path of program execution will be altered depending on the state of the external signal when sampled.

INTERRUPTS

The 8041AH/8741A, 8042/8742 has the following internal interrupts:

- Input Buffer Full (IBF) interrupt
- Timer Overflow interrupt

The IBF interrupt forces a CALL to location 3 in program memory; a timer-overflow interrupt forces a CALL to location 7. The IBF interrupt is enabled by the EN I instruction and disabled by the DIS I instruction. The timer-overflow interrupt is enabled and disabled by the EN TNCTI and DIS TCNTI instructions, respectively.

Figure 2-14 illustrates the internal interrupt logic. An IBF interrupt request is generated whenever $\overline{WR}$ and $\overline{CS}$ are both low, regardless of whether interrupts are enabled. The interrupt request is cleared upon entering the IBF service routine only. That is, the DIS I instruction does not clear a pending IBF interrupt.

Interrupt Timing Latency

When the IBF interrupt is enabled and an IBF interrupt request occurs, an interrupt sequence is initiated as soon as the currently executing instruction is completed. The following sequence occurs:

- A CALL to location 3 is forced.
- The program counter and bits 4–7 of the Program Status Word are stored in the stack.
- The stack pointer is incremented.

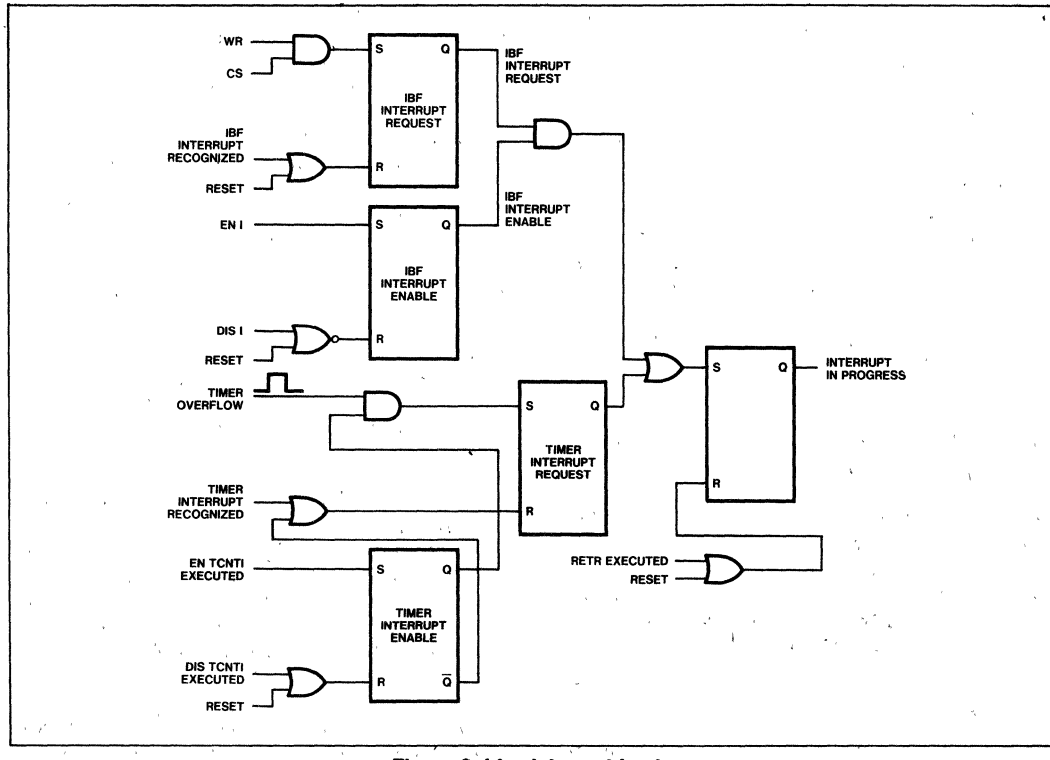


Figure 2-14. Interrupt Logic

Location 3 in program memory should contain an unconditional jump to the beginning of the IBF interrupt service routine elsewhere in program memory. At the end of the service routine, an RETR (Return and Restore Status) instruction is used to return control to the main program. This instruction will restore the program counter and PSW bits 4–7, providing automatic restoration of the previously active register bank as well. RETR also re-enables interrupts.

A timer-overflow interrupt is enabled by the EN TCNTI instruction and disabled by the DIS TCNTI instruction. If enabled, this interrupt occurs when the timer/counter register overflows. A CALL to location 7 is forced and the interrupt routine proceeds as described above.

The interrupt service latency is the sum of current instruction time, interrupt recognition time, and the internal call to the interrupt vector address. The worst case latency time for servicing an interrupt is 7 clock cycles. Best case latency is 4 clock cycles.

Interrupt Timing

Interrupt inputs may be enabled or disabled under program control using EN I, DIS I, EN TCNTI and DIS TCNTI instructions. Also, a $\overline{\text{RESET}}$ input will disable interrupts. An interrupt request must be removed before the RETR instruction is executed to return from the service routine, otherwise the processor will re-enter the service routine immediately. Thus, the $\overline{\text{WR}}$ and $\overline{\text{CS}}$ inputs should not be held low longer than the duration of the interrupt service routine.

The interrupt system is single level. Once an interrupt is detected, all further interrupt requests are latched but are not acted upon until execution of an RETR instruction re-enables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. If an IBF interrupt and a timer-overflow interrupt occur simultaneously, the IBF interrupt will be recognized first and the timer-overflow interrupt will remain pending until the end of the interrupt service routine.

External Interrupts

An external interrupt can be created using the UPI-41AH, 42 timer/counter in the event counter mode The counter is first preset to FFH and the EN TCNTI instruction is executed. A timer-overflow interrupt is generated by the first high to low transition of the TEST 1 input pin. Also, if an IBF interrupt occurs during servicing of the timer/counter interrupt, it will remain pending until the end of the service routine.

Host Interrupts And DMA

If needed, two external interrupts to the host system can be created using the EN FLAGS instruction. This instruction allocates two I/O lines on PORT 2 (P_{24} and P_{25}). P_{24} is the Output Buffer Full interrupt request line to the host system; P_{25} is the Input Buffer empty interrupt request line. These interrupt outputs reflect the internal status of the OBF flag and the IBF inverted flag. Note, these outputs may be inhibited by writing a "0" to these pins. Reenabling interrupts is done by writing a "1" to these port pins. Interrupts are typically enabled after power on since the I/O ports are set in a "1" condition. The EN FLAG's effect is only cancelled by a device RESET.

DMA handshaking controls are available from two pins on PORT 2 of the UPI-41A microcomputer. These lines (P_{26} and P_{27}) are enabled by the EN DMA instruction. P_{26} becomes DMA request ($\overline{\text{DRQ}}$) and P_{27} becomes DMA acknowledge ($\overline{\text{DACK}}$). The UPI program initiates a DMA request by writing a "1" to P_{26}. The DMA controller transfers the data into the DBBIN data register using DACK which acts as a chip select. The EN DMA instruction can only be cancelled by a chip RESET.

RESET

The $\overline{\text{RESET}}$ input provides a means for internal initialization of the processor. An automatic initialization pulse can be generated at power-on by simply connecting a 1 μfd capacitor between the $\overline{\text{RESET}}$ input and ground as shown in Figure 2-15. It has an internal pull-up resistor to charge the capacitor and a Schmitt-trigger circuit to generate a clean transition. A 2-stage sychronizer has been added to support reliable operation up to 12 MHz.

If automatic initialization is used, $\overline{\text{RESET}}$ should be held low for at least 10 milliseconds to allow the power supply to stabilize. If an external $\overline{\text{RESET}}$ signal is used, $\overline{\text{RESET}}$ may be held low for a minimum of 8 instruction cycles. Figure 2-15 illustrates a configuration using an external TTL gate to generate the $\overline{\text{RESET}}$ input. This configuration can be used to derive the $\overline{\text{RESET}}$ signal from the 8224 clock generator in an 8080 system.

The $\overline{\text{RESET}}$ input performs the following functions:

- Disables Interrupts
- Clears Program Counter to Zero
- Clears Stack Pointer
- Clears Status Register and Flags
- Clears Timer and Timer Flag
- Stops Timer
- Selects Register Bank 0
- Sets PORTS 1 and 2 to Input Mode

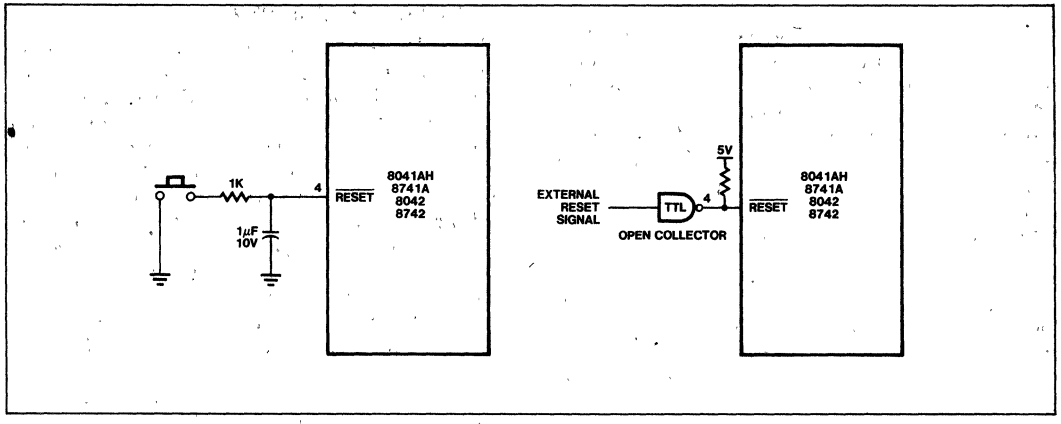


Figure 2-15. External Reset Configuration

DATA BUS BUFFER

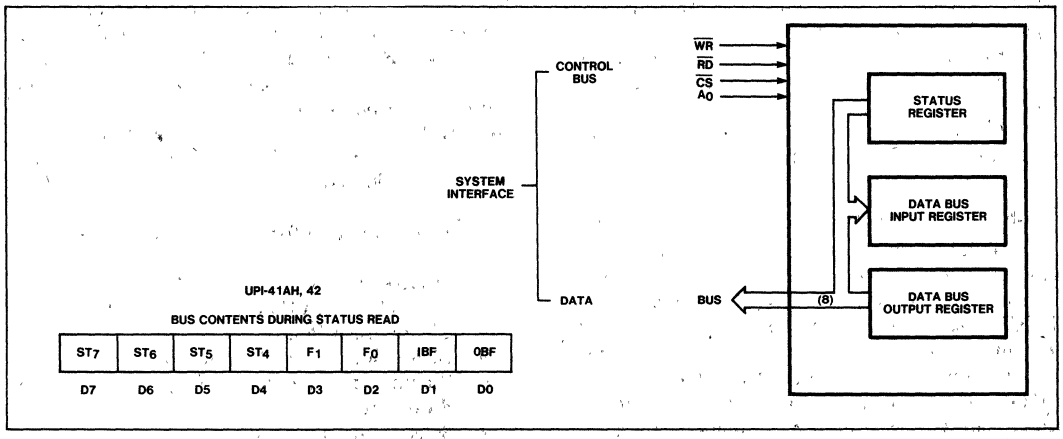
Two 8-bit data bus buffer registers, DBBIN and DBBOUT, serve as temporary buffers for commands and data flowing between it and the master processor. Externally, data is transmitted or received by the DBB registers upon execution of an INput or OUTput instruction by the master processor. Four control signals are used:

- A_0 Address input signifying control or data
- $\overline{CS}$ Chip Select
- $\overline{RD}$ Read strobe
- $\overline{WR}$ Write strobe

Transfer can be implemented with or without UPI program interference by enabling or disabling an internal UPI interrupt. Internally, data transfer between the DBB and the UPI accumulator is under software control and is completely asynchronous to the external processor timing. This allows the UPI software to handle peripheral control tasks independent of the main processor while still maintaining a data interface with the master system.

Configuration

Figure 2-16 illustrates the internal configuration of the DBB registers. Data is stored in two 8-bit buffer registers, DBBIN and DBBOUT. DBBIN and DBBOUT may be accessed by the external processor using the $\overline{WR}$ line and the $\overline{RD}$ line, respectively. The data bus is a bidirectional, three-state bus which can be connected directly to an 8-bit microprocessor system. Four control lines ($\overline{WR}$, $\overline{RD}$, $\overline{CS}$, A_0) are used by the external processor to transfer data to and from the DBBIN and DBBOUT registers.



2-16. Data Bus Buffer Configuration

An 8-bit register containing status flags is used to indicate the status of the DBB registers. The eight status flags are defined as follows:

- **OBF Output Buffer Full** This flag is automatically set when the UPI-Microcomputer loads the DBBOUT register and is cleared when the master processor reads the data register.
- **IBF Input Buffer Full** This flag is set when the master processor writes a character to the DBBIN register and is cleared when the UPI INputs the data register contents to its accumulator.
- **F_0** This is a general purpose flag which can be cleared or toggled under UPI software control. The flag is used to transfer UPI status information to the master processor.
- **F_1 Command/Data** This flag is set to the condition of the A_0 input line when the master processor writes a character to the data register. The F_1 flag can also be cleared or toggled under UPI-Microcomputer program control.
- **ST4 Through ST7** These bits are user defined status bits. They are defined by the MOV STS,A instruction.

All flags in the status register are automatically cleared by a $\overline{\text{RESET}}$ input.

SYSTEM INTERFACE

Figure 2-17 illustrates how an UPI-Microcomputer can be connected to a standard 8080-type bus system. Data lines D_0–D_7 form a three-state, bidirectional port which can be connected directly to the system data bus. The UPI bus interface has sufficient drive capability (400 μA) for small systems, however, a larger system may require buffers.

Four control signals are required to handle the data and status information transfer:

- $\overline{\text{WR}}$ I/O WRITE signal used to transfer data from the system bus to the UPI DBBIN register and set the F_1 flag in the status register.
- $\overline{\text{RD}}$ I/O READ signal used to transfer data from the DBBOUT register or status register to the system data bus.
- $\overline{\text{CS}}$ CHIP SELECT signal used to enable one 8041A out of several connected to a common bus.
- A_0 Address input used to select either the 8-bit status register or DBBOUT register during an I/O READ.
  Also, the signal is used to set the F_1 flag in the status register during an I/O WRITE.

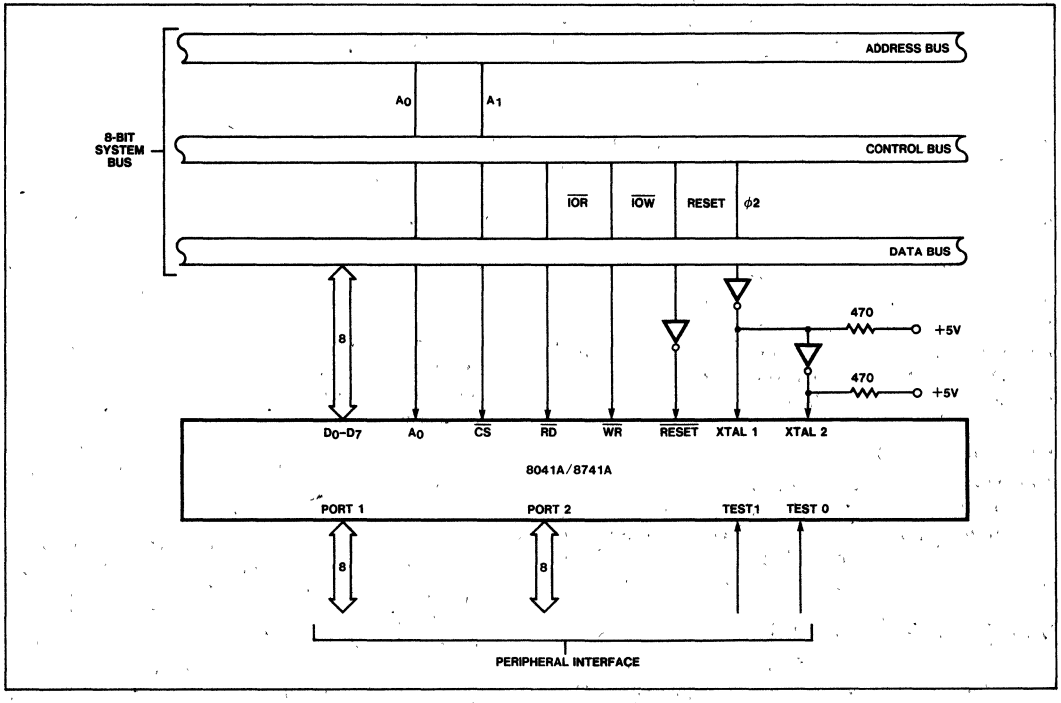


Figure 2-17. Interface to 8080 System Bus

The $\overline{WR}$ and $\overline{RD}$ signals are active low and are standard MCS-80 peripheral control signals used to synchronize data transfer between the system bus and peripheral devices.

The $\overline{CS}$ and A_0 signals are decoded from the address bus of the master system. In a system with few I/O devices a linear addressing configuration can be used where A_0 and A_1 lines are connected directly to A_0 and $\overline{CS}$ inputs (see Figure 2-17).

Data Read

Table 2-4 illustrates the relative timing of a DBBOUT Read. When $\overline{CS}$, A_0, and $\overline{RD}$ are low, the contents of the DBBOUT register is placed on the three-state Data lines D_0–D_7 and the OBF flag is cleared.

The master processor uses $\overline{CS}$, A_0, $\overline{WR}$, and $\overline{RD}$ to control data transfer between the DBBOUT register and the master system. The following operations are under master processor control:

Table 2-4. Data Transfer Controls

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_0	
0	0	1	0	Read DBBOUT register
0	0	1	1	Read STATUS register
0	1	0	0	Write DBBIN data register
0	1	0	1	Write DBBIN command register
1	x	x	x	Disable DBB

Status Read

Table 2-4 shows the logic sequence required for a STATUS register read. When $\overline{CS}$ and $\overline{RD}$ are low with A_0 high, the contents of the 8-bit status register appears on Data lines D_0–D_7.

Data Write

Table 2-4 shows the sequence for writing information to the DBBIN register. When $\overline{CS}$ and $\overline{WR}$ are low, the contents of the system data bus is latched into DBBIN. Also, the IBF flag is set and an interrupt is generated, if enabled.

Command Write

During any write (Table 2-4), the state of the A_0 input is latched into the status register in the F_1 (command/data) flag location. This additional bit is used to signal whether DBBIN contents are command (A_0 = 1) or data (A_0 = 0) information.

INPUT/OUTPUT INTERFACE

The UPI-41A has 16 lines for input and output functions. These I/O lines are grouped as two 8-bit TTL compatible ports: PORTS 1 and 2. The port lines can individually function as either inputs or outputs under software control. In addition, the lower 4 lines of PORT 2 can be used to interface to an 8243 I/O expander device to increase I/O capacity to 28 or more lines. The additional lines are grouped as 4-bit ports: PORTS 4, 5, 6, and 7.

PORTS 1 and 2

PORTS 1 and 2 are each 8 bits wide and have the same I/O characteristics. Data written to these ports by an OUTL Pp,A instruction is latched and remains unchanged until it is rewritten. Input data is sampled at the time the IN, A,Pp instruction is executed. Therefore, input data must be present at the PORT until read by an INput instruction. PORT 1 and 2 inputs are fully TTL compatible and outputs will drive one standard TTL load.

Circuit Configuration

The PORT 1 and 2 lines have a special output structure (shown in Figure 2-18) that allows each line to serve as an input, an output, or both, even though outputs are statically latched.

Each line has a permanent high impedance pull-up (50KΩ) which is sufficient to provide source current for a TTL high level, yet can be pulled low by a standard TTL gate drive. Whenever a "1" is written to a line, a low impedance pull-up (5K) is switched in momentarily (500 ns) to provide a fast transition from 0 to 1. When a "0" is written to the line, a low impedance pull-down (300Ω) is active to provide TTL current sinking capability.

To use a particular PORT pin as an input, a logic "1" must first be written to that pin.

NOTE: A $\overline{RESET}$ intializes all PORT pins to the high impedance logic "1" state.

An external TTL device connected to the pin has sufficient current sinking capability to pull-down the pin to the low state. An IN A,Pp instruction will sample the status of PORT pin and will input the proper logic level. With no external input connected, the IN A,Pp instruction inputs the previous output status.

This structure allows input and output information on the same pin and also allows any mix of input and output lines on the same port. However, when inputs and outputs are mixed on one PORT, a PORT write will cause the strong internal pull-ups to turn on at all inputs. If a switch or other low impedance device is connected to an input, a PORT write ("1" to an input) could cause current limits on internal lines to

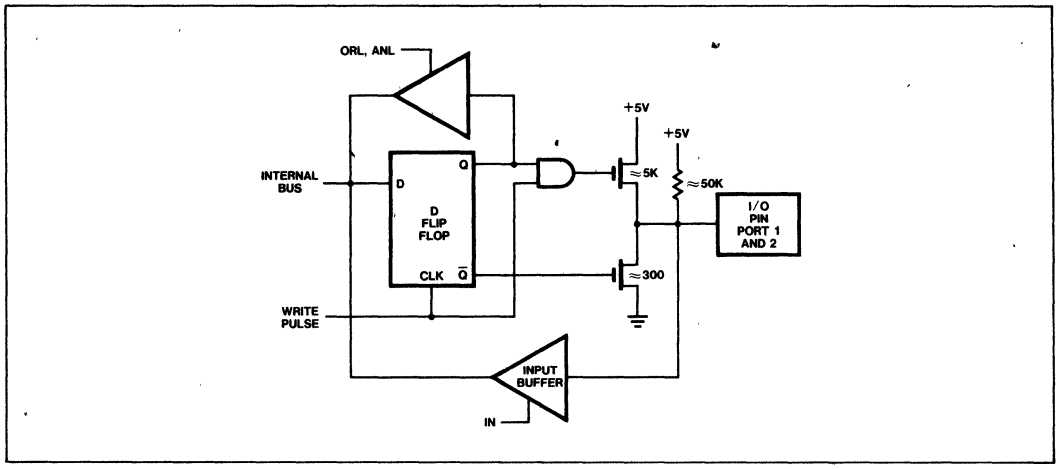


Figure 2-18. Quasi-Bidirectional Port Structure

be exceeded. Figure 2-19 illustrates the recommended connection when inputs and outputs are mixed on one PORT.

The bidirectional port structure in combination with the UPI-41AH, 42 logical AND and OR instructions provides an efficient means for handling single line inputs and outputs within an 8-bit processor.

PORTS 4, 5, 6, and 7

By using an 8243 I/O expander, 16 additional I/O lines can be connected to the UPI-41AH, 42 and directly addressed as 4-bit I/O ports using UPI-41AH, 42 instructions. This feature saves program space and design time, and improves the bit handling capability of the UPI-41AH, 42.

The lower half of PORT 2 provides an interface to the 8243 as illustrated in Figure 2-20. The PROG pin is used as a strobe to clock address and data information via the PORT 2 interface. The extra 16 I/O lines are referred to in UPI software as PORTS 4, 5, 6, and 7. Each PORT can be directly addressed and can be ANDed and ORed with an immediate data mask. Data can be moved directly to the accumulator from the expander PORTS (or vice-versa).

The 8243 I/O ports, PORTS 4, 5, 6, and 7, provide more drive capability than the UPI-41AH, 42 bidirectional ports. The 8243 output is capable of driving about 5 standard TTL loads.

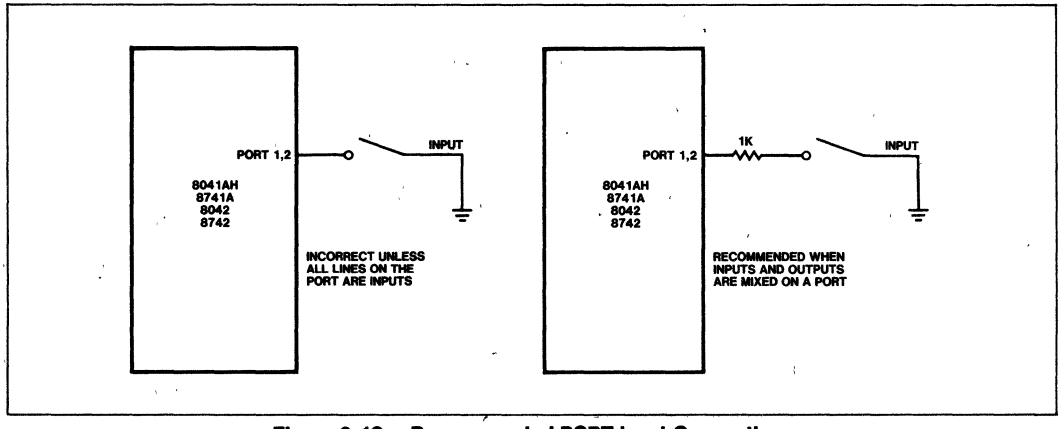


Figure 2-19. Recommended PORT Input Connections

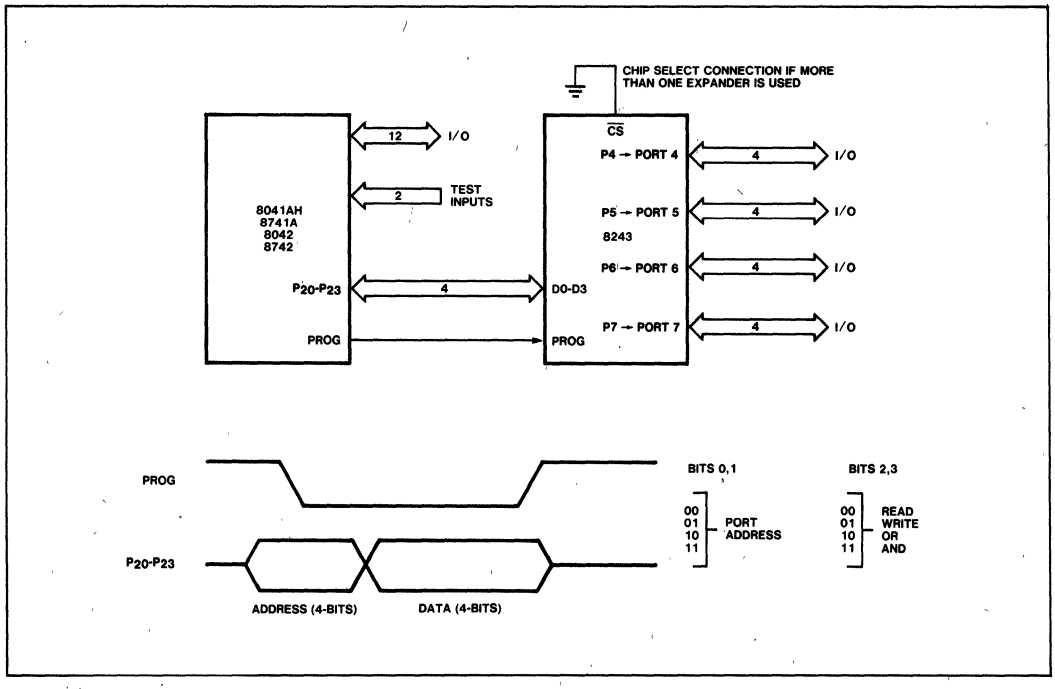


Figure 2-20. 8243 Expander Interface

Multiple 8243's can be connected to the PORT 2 interface. In normal operation, only one of the 8243's would be active at the time an Input or Output command is executed. The upper half of PORT 2 is used to provide chip select signals to the 8243's. Figure 2-21 shows how four 8243's could be connected. Software is needed to select and set the proper PORT 2 pin before an INPUT or OUTPUT command to PORTS 4–7 is executed. In general, the software overhead required is very minor compared to the added flexibility of having a large number of I/O pins available.

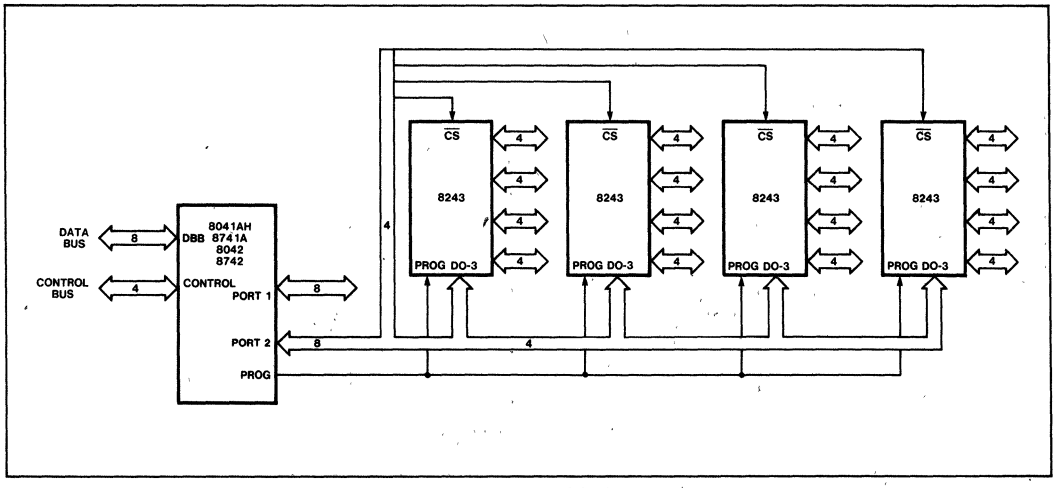


Figure 2-21. Multiple 8243 Expansion

CHAPTER 3
INSTRUCTION SET

The UPI-41AH, 42 Instruction Set is opcode-compatible with the MCS-48 set except for the elimination of external program and data memory instructions and the addition of the data bus buffer instructions. It is very straightforward and efficient in its use of program memory. All instructions are either 1 or 2 bytes in length (over 70% are only 1 byte long) and over half of the instructions execute in one machine cycle. The remainder require only two cycles and include Branch, Immediate, and I/O operations.

The UPI-41AH, 42 Instruction Set efficiently handles the single-bit operations required in control applications. Special instructions allow port bits to be set or cleared individually. Also, any accumulator bit can be directly tested via conditional branch instructions. Additional instructions are included to simplify loop counters, table look-up routines and N-way branch routines.

The UPI-41AH, 42 Microcomputer handles arithmetic operations in both binary and BCD for efficient interface to peripherals such as keyboards and displays.

The instruction set can be divided into the following groups:

- Data Moves
- Accumulator Operations
- Flags
- Register Operations
- Branch Instructions
- Control
- Timer Operations
- Subroutines
- Input/Output Instructions

Data Moves (See Instruction Summary)

The 8-bit accumulator is the control point for all data transfers within the UPI-41AH, 42. Data can be transferred between the 8 registers of each working register bank and the accumulator directly (i.e., with a source or destination register specified by 3 bits in the instruction). The remaining locations in the RAM array are addressed either by R_0 or R_1 of the active register bank. Transfers to and from RAM require one cycle.

Constants stored in Program Memory can be loaded directly into the accumulator or the eight working registers. Data can also be transferred directly between the accumulator and the on-board timer/counter, the Status Register (STS), or the Program Status Word (PSW). Transfers to the STS register alter bits 4-7 only. Transfers to the PSW alter machine status accordingly and provide a means of restoring status after an interrupt or of altering the stack pointer if necessary.

Accumulator Operations

Immediate data, data memory, or the working registers can be added (with or without carry) to the accumulator. These sources can also be ANDed, ORed, or exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

The lower 4 bits of the accumulator can be exchanged with the lower 4 bits of any of the internal RAM locations. This operation, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides easy handling of BCD numbers and other 4-bit quantities. To facilitate BCD arithmetic a Decimal Adjust instruction is also included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the desired BCD result.

The accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

A subtract operation can be easily implemented in UPI-41AH, 42 software using three single-byte, single-cycle instructions. A value can be subtracted from the accumulator by using the following instructions:

- Complement the accumulator
- Add the value to the accumulator
- Complement the accumulator

Flags

There are four user accessible flags:

- Carry
- Auxiliary Carry
- F_0
- F_1

The Carry flag indicates overflow of the accumulator, while the Auxiliary Carry flag indicates overflow between BCD digits and is used during decimal adjust operations. Both Carry and Auxiliary Carry are part of the Program Status Word (PSW) and are stored in the stack during subroutine calls. The F_0 and F_1 flags are general-purpose flags which can be cleared or complemented by UPI instructions. F_0 is accessible via the Program Status Word and is stored in the stack with the Carry flags. F_1 reflects the condition of the A_0 line, and caution must be used when setting or clearing it.

Register Operations

The working registers can be accessed via the accumulator as explained above, or they can be loaded with immediate data constants from program memory. In addition, they can be incremented or decremented directly, or they can be used as loop counters as explained in the section on branch instructions.

Additional Data Memory locations can be accessed with indirect instructions via R_0 and R_1.

Branch Instructions

The UPI-41AH, 42 Instruction Set includes 17 jump instructions. The unconditional jump instruction allows jumps anywhere in the 1K words of program memory. All other jump instructions are limited to the current page (256 words) of program memory.

Conditional jump instructions can test the following inputs and machine flags:

- TEST 0 input pin
- TEST 1 input pin
- Input Buffer Full flag
- Output Buffer Full flag
- Timer flag
- Accumulator zero
- Accumulator bit
- Carry flag
- F_0 flag
- F_1 flag

The conditions tested by these instructions are the instantaneous values at the time the conditional jump instruction is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate flag.

The decrement register and jump if not zero (DJNZ) instruction combines decrement and branch operations in a single instruction which is useful in implementing a loop counter. This instruction can designate any of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A special indirect jump instruction (JMPP @A) allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator point to a location in program memory which contains the jump address. As an example, this instruction could be used to vector to any one of several routines based on an ASCII character which has been loaded into the accumulator. In this way, ASCII inputs can be used to initiate various routines.

Control

The UPI-41AH, 42 Instruction Set has six instructions for control of the DMA, interrupts, and selection of working register banks.

The UPI-41AH, 42 provides two instructions for control of the external microcomputer system. IBF and OBF flags can be routed to PORT 2 allowing interrupts of the external processor. DMA handshaking signals can also be enabled using lines from PORT 2.

The IBF interrupt can be enabled and disabled using two instructions. Also, the interrupt is automatically disabled following a RESET input or during an interrupt service routine.

The working register bank switch instructions allow the programmer to immediately substitute a second 8 register bank for the one in use. This effectively provides either 16 working registers or the means for quickly saving the contents of the first 8 registers in response to an interrupt. The user has the option of switching register banks when an interrupt occurs. However, if the banks are switched, the original bank will automatically be restored upon execution of a return and restore status (RETR) instruction at the end of the interrupt service routine.

Timer

The 8-bit on-board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting.

The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the TEST 1 pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

Subroutines

Subroutines are entered by executing a call instruction. Calls can be made to any address in the 1K word program memory. Two separate return instructions determine whether or not status (i.e., the upper 4 bits of the PSW) is restored upon return from a subroutine.

Input/Output Instructions

Two 8-bit data bus buffer registers (DBBIN and DBBOUT) and an 8-bit status register (STS) enable the UPI-41A universal peripheral interface to communicate with the external microcomputer system. Data can be INputted from the DBBIN register to

the accumulator. Data can be OUTputted from the accumulator to the DBBOUT register.

The STS register contains four user-definable bits (ST_4–ST_7) plus four reserved status bits (IBF, OBF, F_0, and F_1). The user-definable bits are set from the accumulator.

The UPI-41AH, 42 peripheral interface has two 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs to the ports are sampled at the time an IN instruction is executed. In addition, immediate data from program memory can be ANDed and ORed directly to PORTS 1 and 2 with the result remaining on the port. This allows "masks" stored in program memory to be used to set or reset individual bits on the I/O ports. PORTS 1 and 2 are configured to allow input on a given pin by first writing a "1" to the pin.

Four additional 4-bit ports are available through the 8243 I/O expander device. The 8243 interfaces to the UPI-41AH, 42 peripheral interface via four PORT 2 lines which form an expander bus. The 8243 ports have their own AND and OR instructions like the on-board ports, as well as move instructions to transfer data in or out. The expander AND or OR instructions, however, combine the contents of the accumulator with the selected port rather than with immediate data as is done with the on-board ports.

INSTRUCTION SET DESCRIPTION

The following section provides a detailed description of each UPI instruction and illustrates how the instructions are used.

For further information about programming the UPI, consult the *8048/8041A Assembly Language Manual.*

Table 3-1. Symbols and Abbreviations Used

Symbol	Definition
A	Accumulator
C	Carry
DBBIN	Data Bus Buffer Input
DBBOUT	Data Bus Buffer Output
F_0, F_1	FLAG 0, FLAG 1 (C/D flag)
I	Interrupt
P	Mnemonic for "in-page" operation
PC	Program Counter
Pp	Port designator (p = 1, 2, or 4–7)
PSW	Program Status Word
Rr	Register designator (r = 0–7)
SP	Stack Pointer
STS	Status register
T	Timer
TF	Timer Flag
T_0, T_1	TEST 0, TEST 1
#	Immediate data prefix
@	Indirect address prefix
(( ))	Double parentheses show the effect of @, that is, @RO is shown as ((RO)).
( )	Contents of

Table 3-2. Instruction Set Summary

Mnemonic		Operation Description	Bytes	Cycles
Accumulator				
ADD	A,Rr	Add register to A	1	1
ADD	A,@Rr	Add data memory to A	1	1
ADD	A,#data	Add immediate to A	2	2
ADDC	A,Rr	Add register to A with carry	1	1
ADDC	A,@Rr	Add data memory to A with carry	1	1
ADDC	A,#data	Add immediate to A with carry	2	2
ANL	A,Rr	And register to A	1	1
ANL	A,@Rr	And data memory to A	1	1
ANL	A,#data	And immediate to A	2	2
ORL	A,Rr	Or register to A	1	1
ORL	A,@Rr	Or data memory to A	1	1
ORL	A,#data	Or immediate to A	2	2
XRL	A,Rr	Exclusive Or register to A	1	1
XRL	A,@Rr	Exclusive Or data memory to A	1	1
XRL	A,#data	Exclusive Or immediate to A	2	2
INC	A	Increment A	1	1
DEC	A	Decrement A	1	1
CLR	A	Clear A	1	1
CPL	A	Complement A	1	1
DA	A	Decimal Adjust A	1	1
SWAP	A	Swap nibbles of A	1	1
RL	A	Rotate A left	1	1
RLC	A	Rotate A left through carry	1	1
RR	A	Rotate A right	1	1
RRC	A	Rotate A right through carry	1	1

Table 3-2. Instruction Set Summary (Con't.)

Mnemonic		Operation Description	Bytes	Cycles
INPUT/OUTPUT				
IN	A,Pp	Input port to A	1	2
OUTL	Pp,A	Output A to port	1	2
ANL	Pp,#data	And immediate to port	2	2
ORL	Pp,#data	Or immediate to port	2	2
IN	A,DBB	Input DBB to A, clear IBF	1	1
OUT	DBB,A	Output A to DBB, Set OBF	1	1
MOV	STS,A	A_4–A_7 to bits 4–7 of status	1	1
MOVD	A,Pp	Input Expander port to A	1	2
MOVD	Pp,A	Output A to Expander port	1	2
ANLD	Pp,A	And A to Expander port	1	2
ORLD	Pp,A	Or A to Expander port	1	2
DATA MOVES				
MOV	A,Rr	Move register to A	1	1
MOV	A,@Rr	Move data memory to A	1	1
MOV	A,#data	Move immediate to A	2	2
MOV	Rr,A	Move A to register	1	1
MOV	@Rr,A	Move A to data memory	1	1
MOV	Rr,#data	Move immediate to register	2	2
MOV	@Rr,#data	Move immediate to data memory	2	2
MOV	A,PSW	Move PSW to A	1	1
MOV	PSW,A	Move A to PSW	1	1
XCH	A,Rr	Exchange A and registers	1	1
XCH	A,@Rr	Exchange A and data memory	1	1
XCHD	A,@Rr	Exchange digit of A and register	1	1
MOVP	A,@A	Move to A from current page	1	2
MOVP3	A,@A	Move to A from Page 3	1	2
TIMER/COUNTER				
MOV	A,T	Read Timer/Counter	1	1
MOV	T,A	Load Timer/Counter	1	1
STRT	T	Start Timer	1	1
STRT	CNT	Start Counter	1	1
STOP	TCNT	Stop Timer/Counter	1	1
EN	TCNTI	Enable Timer/Counter Interrupt	1	1
DIS	TCNTI	Disable Timer/Counter Interrupt	1	1
CONTROL				
EN	DMA	Enable DMA Handshake Lines	1	1
EN	I	Enable IBF interrupt	1	1
DIS	I	Disable IBF interrupt	1	1
EN	FLAGS	Enable Master Interrupts	1	1
SEL	RB0	Select register bank 0	1	1
SEL	RB1	Select register bank 1	1	1
NOP		No Operation	1	1
REGISTERS				
INC	Rr	Increment register	1	1
INC	@Rr	Increment data memory	1	1
DEC	Rr	Decrement register	1	1
SUBROUTINE				
CALL	addr	Jump to subroutine	2	2
RET		Return	1	2
RETR		Return and restore status	1	2
FLAGS				
CLR C		Clear Carry	1	1
CPL C		Complement Carry	1	1
CLR F0		Clear Flag 0	1	1
CPL F0		Complement Flag 0	1	1
CLR F1		Clear F_1 Flag	1	1
CPL F1		Complement F_1 Flag	1	1

Table 3-2. Instruction Set Summary (Con't.)

Mnemonic		Operation Description	Bytes	Cycles
BRANCH				
JMP	addr	Jump unconditional	2	2
JMPP	@A	Jump indirect	1	2
DJNZ	Rr,addr	Decrement register and jump on non-zero	2	2
JC	addr	Jump on Carry=1	2	2
JNC	addr	Jump on Carry=0	2	2
JZ	addr	Jump on A Zero	2	2
JNZ	addr	Jump on A not Zero	2	2
JT0	addr	Jump on T_0=1	2	2
JNT0	addr	Jump on T_0=0	2	2
JT1	addr	Jump on T_1=1	2	2
JNT1	addr	Jump on T_1=0	2	2
JF0	addr	Jump on F_0 Flag=1	2	2
JF1	addr	Jump on F_1 Flag=1	2	2
JTF	addr	Jump on Timer Flag=1	2	2
JNIBF	addr	Jump on IBF Flag=0	2	2
JOBF	addr	Jump on OBF Flag=1	2	2
JBb	addr	Jump on Accumulator Bit	2	2

ALPHABETIC LISTING

ADD A,Rr Add Register Contents to Accumulator

Opcode:

0	1	1	0	1	r_2	r_1	r_0

The contents of register 'r' are added to the accumulator. Carry is affected.

(A) ← (A) + (Rr) r=0–7

```
Example: ADDREG: ADD A,R6      ;ADD REG 6 CONTENTS
                               ;TO ACC
```

ADD A,@Rr Add Data Memory Contents to Accumulator

Opcode:

0	1	1	0	0	0	0	r

The contents of the standard data memory location addressed by register 'r' bits 0–5 are added to the accumulator. Carry is affected.

(A) ← (A) + ((Rr)) r=0–1

```
Example: ADDM: MOV R0,#47      ;MOVE 47 DECIMAL TO REG 0
         ADD A,@R0             ;ADD VALUE OF LOCATION
                               ;47 TO ACC
```

ADD A,#data Add Immediate Data to Accumulator

Opcode:

0	0	0	0	0	0	1	1	•	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

(A) ← (A) + data

```
Example: ADDID: ADD A,#ADDER   ;ADD VALUE OF SYMBOL
                               ;'ADDER' TO ACC
```

ADDC A,Rr Add Carry and Register Contents to Accumulator

Opcode:

0	1	1	1	1	r_2	r_1	r_0

The content of the carry bit is added to accumulator location 0. The contents of register 'r' are then added to the accumulator. Carry is affected.

(A) ← (A) + (Rr) + (C) r=0–7

```
Example: ADDRGC: ADDC A,R4      ;ADD CARRY AND REG 4
                                ;CONTENTS TO ACC
```

ADDC A,@Rr Add Carry and Data Memory Contents to Accumulator

Opcode:

0	1	1	1	0	0	0	r

The content of the carry bit is added to accumulator location 0. Then the contents of the standard data memory location addressed by register 'r' bits 0–5 are added to the accumulator. Carry is affected.

(A) ← (A) + ((Rr)) + (C) r=0–1

```
Example: ADDMC: MOV R1,#40      ;MOV '40' DEC TO REG 1
                ADDC A,@R1      ;ADD CARRY AND LOCATION 40
                                ;CONTENTS TO ACC
```

ADDC A,#data Add Carry and Immediate Data to Accumulator

Opcode:

0	0	0	1	0	0	1	1	•	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0. Then the specified data is added to the accumulator. Carry is affected.

(A) ← (A) + data + (C)

```
Example: ADDC A,#255            ;ADD CARRY AND '225' DEC
                                ;TO ACC
```

ANL A,Rr Logical AND Accumulator With Register Mask

Opcode:

0	1	0	1	1	r_2	r_1	r_0

Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

(A) ← (A) AND (Rr) r=0–7

```
Example: ANDREG: ANL A,R3       ;'AND' ACC CONTENTS WITH MASK
                                ;MASK IN REG 3
```

ANL A,@Rr Logical AND Accumulator With Memory Mask

Opcode:

0	1	0	1	0	0	0	r

Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'r', bits 0–5.

(A) ← (A) AND ((Rr)) r=0–1

```
Example: ANDDM: MOV R0,#0FFH    ;MOVE 'FF' HEX TO REG 0
                ANL A,#0AFH     ;'AND' ACC CONTENTS WITH
                                ;MASK IN LOCATION 63
```

ANL A,#data Logical AND Accumulator With Immediate Mask

Opcode: | 0 1 0 1 | 0 0 1 1 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.
(A) ← (A) AND data

Example:
```
ANDID: ANL A,#0AFH      ;'AND' ACC CONTENTS
                        ;WITH MASK 10101111
       ANL A,#3+X/Y     ;'AND' ACC CONTENTS
                        ;WITH VALUE OF EXP
                        ;'3+X/Y'
```

ANL Pp,#data Logical AND Port 1–2 With Immediate Mask

Opcode: | 1 0 0 1 | 1 0 p_1 p_0 | • | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 |

This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.
(Pp) ← (Pp) AND data p=1–2

Note: Bits 0-1 of the opcode are used to represent PORT 1 and PORT 2. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits	p1	p0	Port
	0	0	X
	0	1	1
	1	0	2
	1	1	X

Example:
```
ANDP2: ANL P2,#0F0H     ;'AND' PORT 2 CONTENTS
                        ;WITH MASK 'F0' HEX
                        ;(CLEAR P20–23)
```

ANLD Pp,A Logical AND Port 4–7 With Accumulator Mask

Opcode: | 1 0 0 1 | 1 1 p_1 p_0 |

This is a 2-cycle instruction. Data on port 'p' on the 8243 expander is logically ANDed with the digit mask contained in accumulator bits 0–3.
(Pp) ← (Pp) AND (A0–3) p=4–7

Note: The mapping of Port 'p' to opcode bits p_1,p_0 is as follows:

p1	p0	Port
0	0	4
0	1	5
1	0	6
1	1	7

Example:
```
ANDP4: ANLD P4,A        ;'AND' PORT 4 CONTENTS
                        ;WITH ACC BITS 0–3
```

CALL address Subroutine Call

Opcode:

0	a_9	a_8	1	0	1	0	0	•	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

This is a 2-cycle instruction. The program counter and PSW bits 4–7 are saved in the stack. The stack pointer (PSW bits 0–2) is updated. Program control is then passed to the location specified by 'address'.

Execution continues at the instruction following the CALL upon return from the subroutine.
((SP)) ← (PC), (PSW_{4-7})
(SP) ← (SP) + 1
(PC_{8-9}) ← ($addr_{8-9}$)
(PC_{0-7}) ← ($addr_{0-7}$)

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
        MOV R0,#50      ;MOVE '50' DEC TO ADDRESS
                        ;REG 0
BEGADD: MOV A,R1        ;MOVE CONTENTS OF REG 1
                        ;TO ACC
        ADD A,R2        ;ADD REG 2 TO ACC
        CALL SUBTOT     ;CALL SUBROUTINE 'SUBTOT'
        ADD A,R3        ;ADD REG 3 TO ACC
        ADD A,R4        ;ADD REG 4 TO ACC
        CALL SUBTOT     ;CALL SUBROUTINE 'SUBTOT'
        ADD A,R5        ;ADD REG 5 TO ACC
        ADD A,R6        ;ADD REG 6 TO ACC
        CALL SUBTOT     ;CALL SUBROUTINE 'SUBTOT'
                .
                .
                .
SUBTOT: MOV @R0,A       ;MOVE CONTENTS OF ACC TO
                        ;LOCATION ADDRESSED BY
                        ;REG 0
        INC R0          ;INCREMENT REG 0
        RET             ;RETURN TO MAIN PROGRAM
```

CLR A Clear Accumulator

Opcode:

0	0	1	0	0	1	1	1

The contents of the accumulator are cleared to zero.
(A) ← 00H

CLR C Clear Carry Bit

Opcode:

1	0	0	1	0	1	1	1

During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPLC, RRC, and DAA instructions. This instruction resets the carry bit to zero.
(C) ← 0

CLR F1 Clear Flag 1

Opcode:

1	0	1	0	0	1	0	1

The F_1 flag is cleared to zero.
(F_1) ← 0

CLR F0 Clear Flag 0

Opcode:	1	0	0	0	0	1	0	1

Flag 0 is cleared to zero.
$(F_0) \leftarrow 0$

CPL A Complement Accumulator

Opcode:	0	0	1	1	0	1	1	1

The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.
(A) ← NOT (A)

Example: Assume accumulator contains 01101010.

```
CPLA: CPL A                ;ACC CONTENTS ARE COMPLE-
                           ;MENTED TO 10010101
```

CPL C Complement Carry Bit

Opcode:	1	0	1	0	0	1	1	1

The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.
(C) ← NOT (C)

Example: Set C to one; current setting is unknown.

```
CT01: CLR C                ;C IS CLEARED TO ZERO
      CPL C                ;C IS SET TO ONE
```

CPL F0 Complement Flag 0

Opcode:	1	0	0	1	0	1	0	1

The setting of Flag 0 is complemented; one is changed to zero, and zero is changed to one.
$F_0 \leftarrow$ NOT (F0)

CPL F1 Complement Flag 1

Opcode:	1	0	1	1	0	1	0	1

The setting of the F_1 Flag is complemented; one is changed to zero, and zero is changed to one.
$(F_1) \leftarrow$ NOT (F_1)

DA A Decimal Adjust Accumulator

Opcode:

0	1	0	1	0	1	1	1

The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0–3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4–7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one; otherwise, it is cleared to zero.

Example: Assume accumulator contains 9AH.

```
DA   A    ;ACC ADJUSTED TO 01H with C set

C    AC   ACC
0    0    9AH    INITIAL CONTENTS
          06H    ADD SIX TO LOW DIGIT
0    0    A1H
          60H    ADD SIX TO HIGH DIGIT
1    0    01H    RESULT
```

DEC A Decrement Accumulator

Opcode:

0	0	0	0	0	1	1	1

The contents of the accumulator are decremented by one.

(A) ← (A) − 1

Example: Decrement contents of data memory location 63.

```
MOV R0,#3FH      ;MOVE '3F' HEX TO REG 0
MOV A,@R0        ;MOVE CONTENTS OF LOCATION 63
                 ;TO ACC
DEC A            ;DECREMENT ACC
MOV @R0,A        ;MOVE CONTENTS OF ACC TO
                 ;LOCATION 63
```

DEC Rr Decrement Register

Opcode:

1	1	0	0	1	r_2	r_1	r_0

The contents of working register 'r' are decremented by one.

(Rr) ← (Rr) − 1 r=0–7

Example:
```
DECR1: DEC R1    ;DECREMENT ADDRESS REG 1
```

DIS I Disable IBF Interrupt

Opcode:

0	0	0	1	0	1	0	1

The input Buffer Full interrupt is disabled. The interrupt sequence is not initiated by $\overline{WR}$ and $\overline{CS}$, however, an IBF interrupt request is latched and remains pending until an EN I (enable IBF interrupt) instruction is executed.

Note: The IBF flag is set and cleared independent of the IBF interrupt request so that handshaking protocol can continue normally.

DIS TCNTI Disable Timer/Counter Interrupt

Opcode:

0	0	1	1	0	1	0	1

The timer/counter interrupt is disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ Rr, address Decrement Register and Test

Opcode:

1	1	1	0	1	r_2	r_1	r_0	•	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

This is a 2-cycle instruction. Register 'r' is decremented and tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified address within the current page.

(Rr) ← (Rr) − 1

If R ≠ 0, then;

(PC_{0-7}) ← addr

Note: A 10-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it will jump to a target address on the following page. Otherwise, it is limited to a jump within the current page.

Example: Increment values in data memory locations 50–54.

```
        MOV R0,#50          ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
        MOV R3,#05          ;MOVE '5' DEC TO COUNTER
                            ;REG 3
INCRT:  INC @R0             ;INCREMENT CONTENTS OF
                            ;LOCATION ADDRESSED BY
                            ;REG 0
        INC R0              ;INCREMENT ADDRESS IN REG 0
        DJNZ R3,INCRT       ;DECREMENT REG 3——JUMP TO
                            ;'INCRT' IF REG 3 NONZERO
        NEXT——              ;'NEXT' ROUTINE EXECUTED
                            ;IF R3 IS ZERO
```

EN DMA Enable DMA Handshake Lines

Opcode:

1	1	1	0	0	1	0	1

DMA handshaking is enabled using P_{26} as DMA request (DRQ) and P_{27} as DMA acknowledge ($\overline{DACK}$). The $\overline{DACK}$ line forces $\overline{CS}$ and A_0 low internally and clears DRQ.

EN FLAGS Enable Master Interrupts

Opcode:

1	1	1	1	0	1	0	1

The Output Buffer Full (OBF) and the Input Buffer Full (IBF) flags (IBF is inverted) are routed to P_{24} and P_{25}. For proper operation, a "1" should be written to P_{25} and P_{24} before the EN FLAGS instruction. A "0" written to P_{24} or P_{25} disables the pin.

EN I Enable IBF Interrupt

Opcode:

0	0	0	0	0	1	0	1

The Input Buffer Full interrupt is enabled. A low signal on $\overline{WR}$ and $\overline{CS}$ initiates the interrupt sequence.

EN TCNTI Enable Timer/Counter Interrupt

Opcode:

0	0	1	0	0	1	0	1

The timer/counter interrupt is enabled. An overflow of this register initiates the interrupt sequence.

IN A,DBB Input Data Bus Buffer Contents to Accumulator

Opcode:

0	0	1	0	0	0	1	0

Data in the DBBIN register is transferred to the accumulator and the Input Buffer Full (IBF) flag is set to zero.
(A) ← (DBB)
(IBF) ← 0

Example:
```
INDBB: IN A,DBB          ;INPUT DBBIN CONTENTS TO
                         ;ACCUMULATOR
```

IN A,Pp Input Port 1–2 Data to Accumulator

Opcode:

0	0	0	0	1	0	p_1	p_0

This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.
(A) ← (Pp) p=1–2 (see ANL instruction)

Example:
```
INP12: IN A,P1           ;INPUT PORT 1 CONTENTS
                         ;TO ACC
       MOV R6,A          ;MOVE ACC CONTENTS TO
                         ;REG 6
       IN A,P2           ;INPUT PORT 2 CONTENTS
                         ;TO ACC
       MOV R7,A          ;MOVE ACC CONTENTS TO REG 7
```

INC A Increment Accumulator

Opcode:

0	0	0	1	0	1	1	1

The contents of the accumulator are incremented by one.
(A) ← (A) + 1

Example: Increment contents of location 10 in data memory.
```
INCA: MOV R0,#10         ;MOV '10' DEC TO ADDRESS
                         ;REG 0
      MOV A,@R0          ;MOVE CONTENTS OF LOCATION
                         ;10 TO ACC
      INC A              ;INCREMENT ACC
      MOV @R0,A          ;MOVE ACC CONTENTS TO
                         ;LOCATION 10
```

INC Rr Increment Register

Opcode: | 0 0 0 1 | 1 r_2 r_1 r_0 |

The contents of working register 'r' are incremented by one.

(Rr) ← (Rr) + 1 r=0–7

Example:
```
INCR0: INC R0          ;INCREMENT ADDRESS REG 0
```

INC @Rr Increment Data Memory Location

Opcode: | 0 0 0 1 | 0 0 0 r |

The contents of the resident data memory location addressed by register 'r' bits 0–5 are incremented by one.

((Rr)) ← ((Rr)) + 1 r=0–1

Example:
```
INCDM: MOV R1,#OFFH    ;MOVE ONES TO REG 1
       INC @R1         ;INCREMENT LOCATION 63
```

JBb address Jump If Accumulator Bit Is Set

Opcode: | b_2 b_1 b_0 1 | 0 0 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

(PC_{0-7}) ← addr if b=1

(PC) ← (PC) + 2 if b=0

Example:
```
JB4IS1: JB4 NEXT       ;JUMP TO 'NEXT' ROUTINE
                       ;IF ACC BIT 4=1
```

JC address Jump If Carry Is Set

Opcode: | 1 1 1 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

(PC_{0-7}) ← addr if C=1

(PC) ← (PC) + 2 if C=0

Example:
```
JC1: JC OVERFLOW       ;JUMP TO 'OVFLOW' ROUTINE
                       ;IF C=1
```

JF0 address Jump If Flag 0 Is Set

Opcode: | 1 0 1 1 | 0 1 1 0 | • | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 |

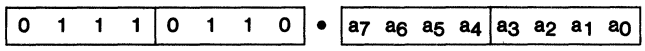
This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

(PC_{0-7}) ← addr if F_0=1

Example:
```
JF0IS1: JF0 TOTAL      ;JUMP TO 'TOTAL' ROUTINE
                       ;IF F0=1
```

JF1 address Jump If C/D Flag (F1) Is Set

Opcode:



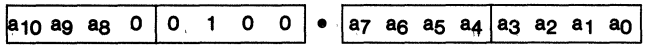
This is a 2-cycle instruction. Control passes to the specified address if the C/D flag (F_1) is set to one.

$(PC_{0-7}) \leftarrow$ addr if $F_1=1$

Example:
```
JF 1IS1: JF1 FILBUF     ;JUMP TO 'FILBUF'
                        ;ROUTINE IF F1=1
```

JMP address Direct Jump Within 1K Block

Opcode:



This is a 2-cycle instruction. Bits 0–9 of the program counter are replaced with the directly-specified address.

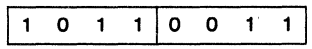
$(PC_{8-9}) \leftarrow$ addr 8–9

$(PC_{0-7}) \leftarrow$ addr 0–7

Example:
```
JMP SUBTOT              ;JUMP TO SUBROUTINE 'SUBTOT'
JMP $-6                 ;JUMP TO INSTRUCTION SIX LOCATIONS
                        ;BEFORE CURRENT LOCATION
JMP 2FH                 ;JUMP TO ADDRESS '2F' HEX
```

JMPP @A Indirect Jump Within Page

Opcode:



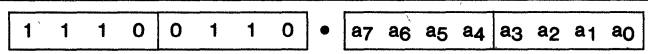
This is a 2-cycle instruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC 0–7).

$(PC_{0-7}) \leftarrow ((A))$

Example: Assume accumulator contains OFH
```
JMPPAG: JMPP @A         ;JMP TO ADDRESS STORED IN
                        ;LOCATION 15 IN CURRENT PAGE
```

JNC address Jump If Carry Is Not Set

Opcode:



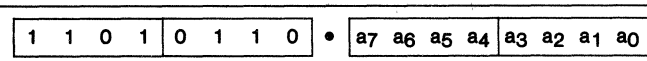
This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

$(PC_{0-7}) \leftarrow$ addr if C=0

Example:
```
JC0: JNC NOVFLO         ;JUMP TO 'NOVFLO' ROUTINE
                        ;IF C=0
```

JNIBF address Jump If Input Buffer Full Flag Is Low

Opcode:



This is a 2-cycle instruction. Control passes to the specified address if the Input Buffer Full flag is low (IBF=0).

$(PC_{0-7}) \leftarrow$ addr if IBF=0

Example:
```
LOC 3:JNIBF LOC 3       ;JUMP TO SELF IF IBF=0
                        ;OTHERWISE CONTINUE
```

JNT0 address Jump If TEST 0 Is Low

Opcode: 0 0 1 0 | 0 1 1 0 • a7 a6 a5 a4 | a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address, if the TEST 0 signal is low. Pin is sampled during SYNC.

	(PC_{0-7}) ← addr	if T_0=0
Example:	JT0LOW: JNT0 60	;JUMP TO LOCATION 60 DEC
		;IF T_0=0

JNT1 address Jump If TEST 1 Is Low

Opcode: 0 1 0 0 | 0 1 1 0 • a7 a6 a5 a4 | a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is low. Pin is sampled during SYNC.

	(PC_{0-7}) ← addr	if T_1=0
Example:	JT1LOW: JNT1 OBBH	;JUMP TO LOCATION 'BB' HEX
		;IF T_1=0

JNZ address Jump If Accumulator Is Not Zero

Opcode: 1 0 0 1 | 0 1 1 0 • a7 a6 a5 a4 | a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

	(PC_{0-7}) ← addr	if A≠0
Example:	JACCNO: JNZ OABH	;JUMP TO LOCATION 'AB' HEX
		;IF ACC VALUE IS NONZERO

JOBF Address Jump If Output Buffer Full Flag Is Set

Opcode: 1 0 0 0 | 0 1 1 0 • a7 a6 a5 a4 | a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the Output Buffer Full (OBF) flag is set (=1) at the time this instruction is executed.

	(PC_{0-7}) ← addr	if OBF=1
Example:	JOBFHI: JOBF OAAH	;JUMP TO LOCATION 'AA' HEX
		;IF OBF=1

JTF address Jump If Timer Flag Is Set

Opcode: 0 0 0 1 | 0 1 1 0 • a7 a6 a5 a4 | a3 a2 a1 a0

This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register overflows to zero. The timer flag is cleared upon execution of this instruction. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

	(PC_{0-7}) ← addr	if TF=1
Example:	JTF1: JTF TIMER	;JUMP TO 'TIMER' ROUTINE
		;IF TF=1

JTO address Jump If TEST 0 Is High

Opcode:	0 0 1 1	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 0 signal is high (= 1). Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_0 = 1$

```
Example: JT0HI: JT0 53          ;JUMP TO LOCATION 53 DEC
                                ;IF T0=1
```

JT1 address Jump If TEST 1 Is High

Opcode:	0 1 0 1	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the TEST 1 signal is high (= 1). Pin is sampled during SYNC.

$(PC_{0-7}) \leftarrow$ addr if $T_1 = 1$

```
Example: JT1HI: JT1 COUNT       ;JUMP TO 'COUNT' ROUTINE
                                ;IF T1=1
```

JZ address Jump If Accumulator Is Zero

Opcode:	1 1 0 0	0 1 1 0	•	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

$(PC_{0-7}) \leftarrow$ addr if A=0

```
Example: JACCO: JZ OA3H         ;JUMP TO LOCATION 'A3' HEX
                                ;IF ACC VALUE IS ZERO
```

MOV A,#data Move Immediate Data to Accumulator

Opcode:	0 0 1 0	0 0 1 1	•	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0

This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

(A) ← data

```
Example: MOV A,#OA3H            ;MOV 'A3' HEX TO ACC
```

MOV A,PSW Move PSW Contents to Accumulator

Opcode:	1 1 0 0	0 1 1 1

The contents of the program status word are moved to the accumulator.

(A) ← (PSW)

Example: Jump to 'RB1SET' routine if bank switch, PSW bit 4, is set.

```
BSCHK: MOV A,PSW        ;MOV PSW CONTENTS TO ACC
       JB4 RB1 SET      ;JUMP TO 'RB1SET' IF ACC
                        ;BIT 4=1
```

MOV A, Rr Move Register Contents to Accumulator

Opcode:

1	1	1	1	1	r_2	r_1	r_0

Eight bits of data are moved from working register 'r' into the accumulator.
(A) ← (Rr) r=0–7

Example:
```
MAR: MOV A,R3          ;MOVE CONTENTS OF REG 3
                       ;TO ACC
```

MOV A,@Rr Move Data Memory Contents to Accumulator

Opcode:

1	1	1	1	0	0	0	r

The contents of the data memory location addressed by bits 0–5 of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.
(A) ← ((Rr)) r=0–1

Example: Assume R1 contains 00110110.
```
MADM: MOV A,@R1        ;MOVE CONTENTS OF DATA MEM
                       ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

Opcode:

0	1	0	0	0	0	1	0

The contents of the timer/event-counter register are moved to the accumulator. The timer/event-counter is not stopped.
(A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 is set—assuming initialization to zero.
```
TIMCHK: MOV A,T        ;MOVE TIMER CONTENTS TO
                       ;ACC
        JB6 EXIT       ;JUMP TO 'EXIT' IF ACC BIT
                       ;6=1
```

MOV PSW,A Move Accumulator Contents to PSW

Opcode:

1	1	0	1	0	1	1	1

The contents of the accumulator are moved into the program status word. All condition bits and the stack pointer are affected by this move.
(PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.
```
INCPTR: MOV A,PSW      ;MOVE PSW CONTENTS TO ACC
        INC A          ;INCREMENT ACC BY ONE
        MOV PSW,A      ;MOVE ACC CONTENTS TO PSW
```

MOV Rr,A Move Accumulator Contents to Register

Opcode: | 1 | 0 | 1 | 0 | 1 | r_2 | r_1 | r_0 |

The contents of the accumulator are moved to register 'r'.

(Rr) ← (A) r=0–7

Example:
```
MRA MOV R0,A          ;MOVE CONTENTS OF ACC TO
                      ;REG 0
```

MOV Rr,#data Move Immediate Data to Register

Opcode: | 1 | 0 | 1 | 1 | 1 | r_2 | r_1 | r_0 | • | d_7 | d_6 | d_5 | d_4 | d_3 | d_2 | d_1 | d_0 |

This a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

(Rr) ← data r=0–7

Example:
```
MIR4: MOV R4,#HEXTEN  ;THE VALUE OF THE SYMBOL
                      ;'HEXTEN' IS MOVED INTO
                      ;REG 4
MIR5: MOV R5;#PI*(R*R) ;THE VALUE OF THE
                      ;EXPRESSION 'PI*(R*R)'
                      ;IS MOVED INTO REG 5
MIR6: MOV R6,#OADH    ;'AD' HEX IS MOVED INTO
                      ;REG 6
```

MOV @Rr,A Move Accumulator Contents to Data Memory

Opcode: | 1 | 0 | 1 | 0 | 0 | 0 | 0 | r |

The contents of the accumulator are moved to the data memory location whose address is specified by bits 0–5 of register 'r'. Register 'r' contents are unaffected.

((Rr)) ← (A) r=0–1

Example: Assume R0 contains 11000111.
```
MDMA: MOV @R,A        ;MOVE CONTENTS OF ACC TO
                      ;LOCATION 7 (REG)
```

MOV @Rr,#data Move Immediate Data to Data Memory

Opcode: | 1 | 0 | 1 | 1 | 0 | 0 | 0 | r | • | d_7 | d_6 | d_5 | d_4 | d_3 | d_2 | d_1 | d_0 |

This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the standard data memory location addressed by register 'r', bit 0–5.

((Rr)) ← data r=0–1

Example: Move the hexadecimal value AC3F to locations 62–63.
```
MIDM: MOV R0,#62      ;MOVE '62' DEC TO ADDR REG0
      MOV @R0,#OACH   ;MOVE 'AC' HEX TO LOCATION 62
      INC R0          ;INCREMENT REG 0 TO '63'
      MOV @R0,#3FH    ;MOVE '3F' HEX TO LOCATION 63
```

MOV STS,A Move Accumulator Contents to STS Register

Opcode:

1	0	0	1	0	0	0	0

The contents of the accumulator are moved into the status register. Only bits 4–7 are affected.
$(STS_{4-7}) \leftarrow (A_{4-7})$

Example: Set ST_4–ST_7 to "1".

```
MSTS: MOV A,#0F0H       ;SET ACC
      MOV STS,A         ;MOVE TO STS
```

MOV T,A Move Accumulator Contents to Timer/Counter

Opcode:

0	1	1	0	0	0	1	0

The contents of the accumulator are moved to the timer/event-counter register.
$(T) \leftarrow (A)$

Example: Initialize and start event counter.

```
INITEC: CLR A           ;CLEAR ACC TO ZEROS
        MOV T,A         ;MOVE ZEROS TO EVENT COUNTER
        STRT CNT        ;START COUNTER
```

MOVD A,Pp Move Port 4–7 Data to Accumulator

Opcode:

0	0	0	0	1	1	p_1	p_0

This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0–3. Accumulator bits 4–7 are zeroed.
$(A_{0-3}) \leftarrow Pp$ p=4–7
$(A_{4-7}) \leftarrow 0$

Note: Bits 0–1 of the opcode are used to represent PORTS 4–7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits p_1	p_0	Port
0	0	4
0	1	5
1	0	6
1	1	7

Example:

```
INPPT5: MOVD A,P5       ;MOVE PORT 5 DATA TO ACC
                        ;BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A Move Accumulator Data to Port 4, 5, 6 and 7

Opcode:

0	0	1	1	1	1	p_1	p_0

This is a 2-cycle instruction. Data in accumulator bits 0–3 is moved (written) to 8243 port 'p'. Accumulator bits 4–7 are unaffected. (See NOTE above regarding port mapping.)
$(Pp) \leftarrow (A_{0-3})$ p=4–7

Example: Move data in accumulator to ports 4 and 5.

```
OUTP45: MOVD P4,A       ;MOVE ACC BITS 0-3 TO PORT 4
        SWAP A          ;EXCHANGE ACC BITS 0-3 AND 4-7
        MOVD P5,A       ;MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A Move Current Page Data to Accumulator

Opcode:

1	0	1	0	0	0	1	1

This is a 2-cycle instruction. The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0–7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored following this operation.
(A) ← ((A))

Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the following page.

Example:
```
MOV128: MOV   A,#128      ;MOVE '128' DEC TO ACC
        MOVP A,@A         ;CONTENTS OF 129TH LOCATION
                          ;IN CURRENT PAGE ARE MOVED TO
                          ;ACC
```

MOVP3 A,@A Move Page 3 Data to Accumulator

Opcode:

1	1	1	0	0	0	1	1

This is a 2-cycle instruction. The contents of the program memory location within page 3, addressed by the accumulator, are moved to the accumulator. The program counter is restored following this operation.
(A) ← ((A)) within page 3

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.
```
TABSCH: MOV A,#OB8H       ;MOVE 'B8' HEX TO ACC (10111000)
        ANL A,#7FH        ;LOGICAL AND ACC TO MASK BIT
                          ;7 (00111000)
        MOVP3, A,@A       ;MOVE CONTENTS OF LOCATION
                          ;'38' HEX IN PAGE 3 TO ACC
                          ;(ASCII '8')
```
Access contents of location in page 3 labelled TAB1. Assume current program location is not in page 3.
```
TABSCH: MOV A,#TAB1       ;ISOLATE BITS 0–7
                          ;OF LABEL
                          ;ADDRESS VALUE
        MOVP3 A,@A        ;MOVE CONTENT OF PAGE 3
                          ;LOCATION LABELED 'TAB1'
                          ;TO ACC
```

NOP The NOP Instruction

Opcode:

0	0	0	0	0	0	0	0

No operation is performed. Execution continues with the following instruction.

ORL A,Rr Logical OR Accumulator With Register Mask

Opcode:

0	1	0	0	1	r_2	r_1	r_0

Data in the accumulator is logically ORed with the mask contained in working register 'r'.
(A) ← (A) OR (Rr) r=0–7

Example:
```
ORREG: ORL A,R4           ;'OR' ACC CONTENTS WITH
                          ;MASK IN REG 4
```

ORL A,@Rr Logical OR Accumulator With Memory Mask

Opcode:

0	1	0	0	0	0	0	r

Data in the accumulator is logically ORed with the mask contained in the data memory location referenced by register 'r', bits 0–5.
(A) ← (A) OR ((Rr)) r=0–1

```
Example: ORDM: MOVE R0,#3FH    ;MOVE '3F' HEX TO REG 0
         ORL A,@R0             ;'OR' ACC CONTENTS WITH MASK
                               ;IN LOCATION 63
```

ORL A,#data Logical OR Accumulator With Immediate Mask

Opcode:

0	1	0	0	0	0	1	1

•

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.
(A) ← (A) OR data

```
Example: ORID: ORL A,#'X'      ;'OR' ACC CONTENTS WITH MASK
                               ;01011000 (ASCII VALUE OF 'X')
```

ORL Pp,#data Logical OR Port 1–2 With Immediate Mask

Opcode:

1	0	0	0	1	0	p_1	p_0

•

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.
(Pp) ← (Pp) OR data p=1–2 (see OUTL instruction)

```
Example: ORP1: ORL P1,#OFFH    ;'OR' PORT 1 CONTENTS WITH
                               ;MASK 'FF' HEX (SET PORT 1
                               'TO ALL ONES)
```

ORLD Pp,A Logical OR Port 4–7 With Accumulator Mask

Opcode:

1	0	0	0	1	1	p_1	p_0

This is a 2-cycle instruction. Data on 8243 port 'p' is logically ORed with the digit mask contained in accumulator bits 0–3,
(Pp) (Pp) OR (A_{0-3}) p=4–7 (See MOVD instruction)

```
Example: ORP7: ORLD P7,A       ;'OR' PORT 7 CONTENTS
                               ;WITH ACC BITS 0–3
```

OUT DBB,A Output Accumulator Contents to Data Bus Buffer

Opcode:

0	0	0	0	0	0	1	0

Contents of the accumulator are transferred to the Data Bus Buffer Output register and the Output Buffer Full (OBF) flag is set to one.
(DBB) ← (A)
OBF ← 1

```
Example: OUTDBB: OUT DBB,A     ;OUTPUT THE CONTENTS OF
                               ;THE ACC TO DBBOUT
```

OUTL Pp,A Output Accumulator Data to Port 1 and 2

Opcode:

0	0	1	1	1	0	p_1	p_0

This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.
(Pp) ← (A) P=1–2

Note: Bits 0–1 of the opcode are used to represent PORT 1 and PORT 2. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits	p1	p0	Port
	0	0	X
	0	1	1
	1	0	2
	1	1	X

Example:
```
OUTLP: MOV A,R7      ;MOVE REG 7 CONTENTS TO ACC
       OUTL P2,A     ;OUTPUT ACC CONTENTS TO PORT2
       MOV A,R6      ;MOVE REG 6 CONTENTS TO ACC
       OUTL P1,A     ;OUTPUT ACC CONTENTS TO PORT 1
```

RET Return Without PSW Restore

Opcode:

1	0	0	0	0	0	1	1

This is a 2-cycle instruction. The stack pointer (PSW bits 0–2) is decremented. The program counter is then restored from the stack. PSW bits 4–7 are not restored.
(SP) ← (SP) – 1
(PC) ← ((SP))

RETR Return With PSW Restore

Opcode:

1	0	0	1	0	0	1	1

This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4–7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine.
(SP) ← (SP) – 1
(PC) ← ((SP))
(PSW_{4-7}) ← ((SP))

RL A Rotate Left Without Carry

Opcode:

1	1	1	0	0	1	1	1

The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.
(A_{n+1}) ← (A_n) n=0–6
(A_0) ← (A_7)

Example:
```
Assume accumulator contains 10110001.
RLNC: RL A           ;NEW ACC CONTENTS ARE 01100011
```

RLC A Rotate Left Through Carry

Opcode:

1	1	1	1	0	1	1	1

The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotated into the bit 0 position.

$(A_{n+1}) \leftarrow (A_n)$ n=0–6
$(A_0) \leftarrow (C)$
$(C) \leftarrow (A7)$

Example: Assume accumulator contains a 'signed' number; isolate sign without changing value.

```
RLTC: CLR C          ;CLEAR CARRY TO ZERO
      RLC A          ;ROTATE ACC LEFT, SIGN
                     ;BIT (7) IS PLACED IN CARRY
      RR A           ;ROTATE ACC RIGHT — VALUE
                     ;(BITS 0–6) IS RESTORED,
                     ;CARRY UNCHANGED, BIT 7
                     ;IS ZERO
```

RR A Rotate Right Without Carry

Opcode:

0	1	1	1	0	1	1	1

The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

$(A_n) \leftarrow (A_{n+1})$ n=0–6
$(A_7) \leftarrow (A_0)$

Example: Assume accumulator contains 10110001.

```
RRNC: RRA            ;NEW ACC CONTENTS ARE 11011000
```

RRC A Rotate Right Through Carry

Opcode:

0	1	1	0	0	1	1	1

The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

$(A_n) \leftarrow (A_{n+1})$ n=0–6
$(A_7) \leftarrow (C)$
$(C) \leftarrow (A_0)$

Example: Assume carry is not set and accumulator contains 10110001.

```
RRTC: RRCA           ;CARRY IS SET AND ACC
                     ;CONTAINS 01011000
```

SEL RB0 Select Register Bank 0

Opcode:

1	1	0	0	0	1	0	1

PSW BIT 4 is set to zero. References to working registers 0–7 address data memory locations 0–7. This is the recommended setting for normal program execution.

(BS) ← 0

SEL RB1 Select Register Bank 1

Opcode:

1	1	0	1	0	1	0	1

PSW bit 4 is set to one. References to working registers 0–7 address data memory locations 24–31. This is the recommended setting for interrupt service routines, since locations 0–7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

(BS) ← 1

Example: Assume an IBF interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

```
LOC3: JMP INIT              ;JUMP TO ROUTINE 'INIT'
                .
                .
                .
INIT: MOV R7,A              ;MOV ACC CONTENTS TO
                            ;LOCATION 7
      SEL RB1               ;SELECT REG BANK 1
      MOV R7,#OFAH          ;MOVE 'FA' HEX TO LOCATION 31
                .
                .
                .
      SEL RB0               ;SELECT REG BANK 0
      MOV A,R7              ;RESTORE ACC FROM LOCATION 7
      RETR                  ;RETURN---RESTORE PC AND PSW
```

STOP TCNT Stop Timer/Event Counter

Opcode:

0	1	1	0	0	1	0	1

This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
START: DIS TCNTI            ;DISABLE TIMER INTERRUPT
       CLR A                ;CLEAR ACC TO ZERO
       MOV T,A              :MOV ZERO TO TIMER
       MOV R7,A             :MOVE ZERO TO REG 7
       STRT T               ;START TIMER
MAIN: JTF COUNT             ;JUMP TO ROUTINE 'COUNT'
                            ;IF TF=1 AND CLEAR TIMER FLAG
      JMP MAIN              ;CLOSE LOOP
COUNT: INC R7               ;INCREMENT REG 7
       MOV A,R7             ;MOVE REG 7 CONTENTS TO ACC
       JB3 INT              ;JUMP TO ROUTINE 'INT' IF ACC
                            ;BIT 3 IS SET (REG 7=8)
       JMP MAIN             ;OTHERWISE RETURN TO ROUTINE
                            ;MAIN
                .
                .
                .
INT: STOP TCNT              ;STOP TIMER
     JMP 7H                 ;JUMP TO LOCATION 7 (TIMER
                            ;INTERRUPT ROUTINE)
```

STRT CNT Start Event Counter

Opcode:

0	1	0	0	0	1	0	1

The TEST 1 (T_1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high to low transition on the T_1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T_1 input.

```
STARTC: EN TCNTI          ;ENABLE COUNTER INTERRUPT
        MOV A,#OFFH       ;MOVE 'FF' HEX (ONES) TO
                          ;ACC
        MOV T,A           ;MOVE ONES TO COUNTER.
        STRT CNT          ;INPUT AND START
```

STRT T Start Timer

Opcode:

0	1	0	1	0	1	0	1

Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
STARTT: EN TCNTI          ;ENABLE TIMER INTERRUPT
        CLR A             ;CLEAR ACC TO ZEROS
        MOV T,A           ;MOVE ZEROS TO TIMER
        STRT T            ;START TIMER
```

SWAP A Swap Nibbles Within Accumulator

Opcode:

0	1	0	0	0	1	1	1

Bits 0–3 of the accumulator are swapped with bits 4–7 of the accumulator.

$(A_{4-7}) \longleftrightarrow (A_{0-3})$

Example: Pack bits 0–3 of locations 50–51 into location 50.

```
PCKDIG: MOV R0,#50        ;MOVE '50' DEC TO REG 0
        MOV R1,#51        ;MOVE '51' DEC TO REG 1
        XCHD A,@R0        ;EXCHANGE BIT 0-3 OF ACC
                          ;AND LOCATION 50
        SWAP A            ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1        ;EXCHANGE BITS 0-3 OF ACC AND
                          ;LOCATION 51
        MOV @R0,A         ;MOVE CONTENTS OF ACC TO
                          ;LOCATION 51
```

XCH A,Rr Exchange Accumulator-Register Contents

Opcode:

0	0	1	0	1	r_2	r_1	r_0

The contents of the accumulator and the contents of working register 'r' are exchanged.

$(A) \longleftrightarrow (Rr)$ r=0–7

Example: Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7          ;EXCHANGE CONTENTS OF REG 7
                          ;AND ACC
        MOV A,PSW         ;MOVE PSW CONTENTS TO ACC
        XCH A,R7          ;EXCHANGE CONTENTS OF REG 7
                          ;AND ACC AGAIN
```

XCH A,@Rr Exchange Accumulator and Data Memory Contents

Opcode:

0	0	1	0	0	0	0	r

The contents of the accumulator and the contents of the data memory location addressed by bits 0–5 of register 'r' are exchanged. Register 'r' contents are unaffected.

(A) ⟷ ((Rr)) r=0–1

Example: Decrement contents of location 52.

```
DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS
                       ;REG 0
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52
       DEC A           ;DECREMENT ACC CONTENTS
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52 AGAIN
```

XCHD A,@Rr Exchange Accumulator and Data Memory 4-bit Data

Opcode:

0	0	1	1	0	0	0	r

This instruction exchanges bits 0–3 of the accumulator with bits 0–3 of the data memory location addressed by bits 0–5 of register 'r'. Bits 4–7 of the accumulator, bits 4–7 of the data memory location, and the contents of register 'r' are unaffected.

(A_{0-3}) ⟷ $((Rr_{0-3}))$ r=0–1

Example: Assume program counter contents have been stacked in locations 22–23.

```
XCHNIB: MOV R0,#23     ;MOVE '23' DEC TO REG 0
        CLR A          ;CLEAR ACC TO ZEROS
        XCHD A,@R0     ;EXCHANGE BITS 0-3 OF ACC
                       ;AND LOCATION 23 (BITS 8-11
                       ;OF PC ARE ZEROED, ADDRESS
                       ;REFERS TO PAGE 0)
```

XRL A,Rr Logical XOR Accumulator With Register Mask

Opcode:

1	1	0	1	1	r_2	r_1	r_0

Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

(A) ← (A) XOR (Rr) r=0–7

Example:

```
XORREG: XRL A,R5       ;'XOR' ACC CONTENTS WITH
                       ;MASK IN REG 5
```

XRL A,@Rr Logical XOR Accumulator With Memory Mask

Opcode:

1	1	0	1	0	0	0	r

Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'r', bits 0–5.

(A) ← (A) XOR ((Rr)) r=0–1

Example:

```
XORDM: MOV R1,#20H     ;MOVE '20' HEX TO REG 1
       XRL A,@R1       ;'XOR' ACC CONTENTS WITH MASK
                       ;IN LOCATION 32
```

XRL A,#data Logical XOR Accumulator With Immediate Mask

Opcode:

1	1	0	1	0	0	1	1	•	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

(A) ← (A) XOR data

Example:

```
XORID: XOR A,#HEXTEN      ;XOR CONTENTS OF ACC WITH
                          ;MASK EQUAL VALUE OF SYMBOL
                          ;'HEXTEN'
```

CHAPTER 4
SINGLE-STEP, PROGRAMMING, AND POWER-DOWN MODES

SINGLE-STEP

The UPI family has a single-step mode which allows the user to manually step through his program one instruction at a time. While stopped, the address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. The single-step feature simplifies program debugging by allowing the user to easily follow program execution.

Figure 4-1 illustrates a recommended circuit for single-step operation, while Figure 4-2 shows the timing relationship between the SYNC output and the $\overline{SS}$ input. During single-step operation, PORT 1 and part of PORT 2 are used to output address information. In order to retain the normal I/O functions of PORTS 1 and 2, a separate latch can be used as shown in Figure 4-3.

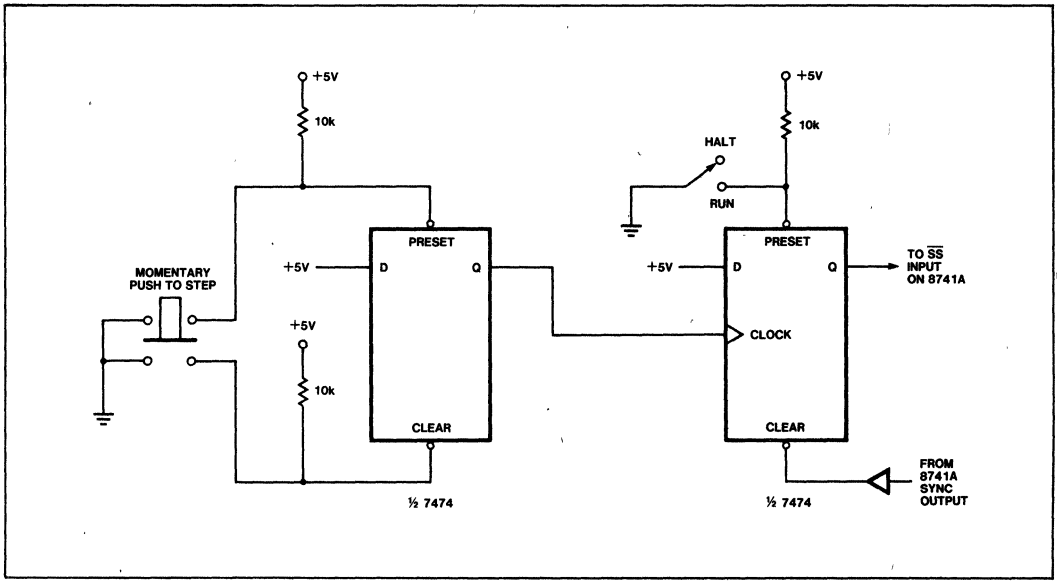


Figure 4-1. Single-Step Circuit

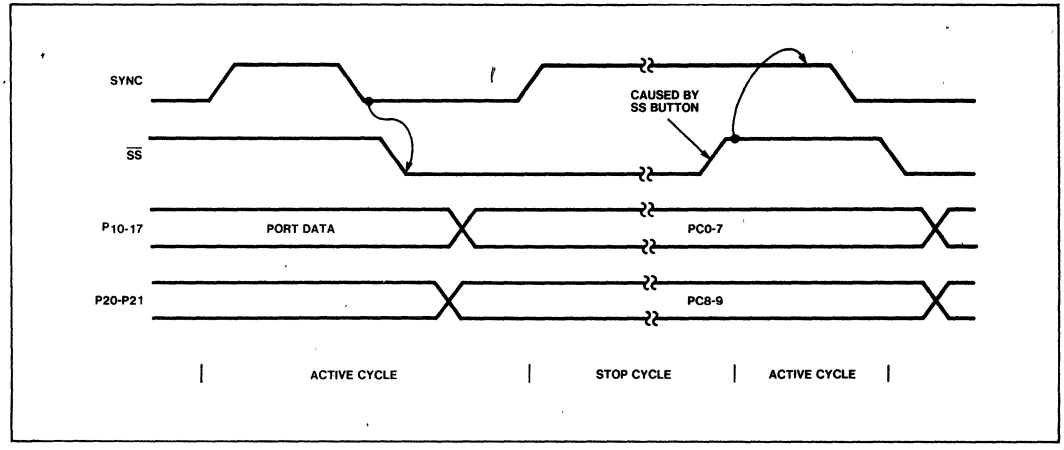


Figure 4-2. Single-Step Timing

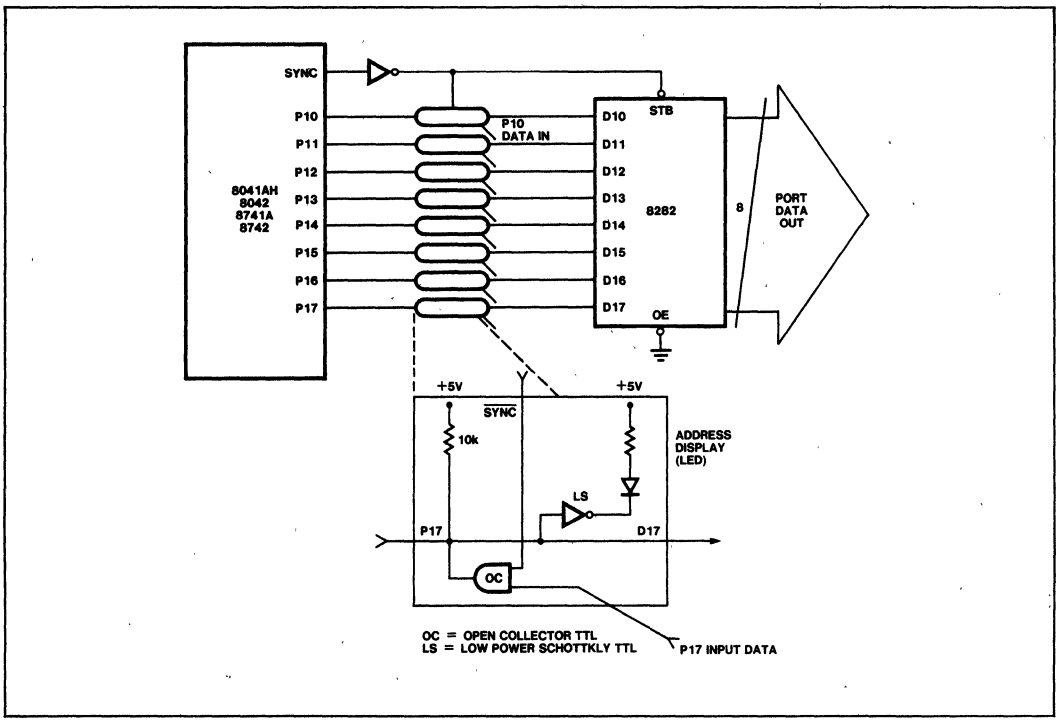


Figure 4-3. Latching Port Data

Timing

The sequence of single-step operation is as follows:

1) The processor is requested to stop by applying a low level on $\overline{SS}$. The $\overline{SS}$ input should not be brought low while SYNC is high. (The UPI samples the $\overline{SS}$ pin in the middle of the SYNC pulse).

2) The processor responds to the request by stopping during the instruction fetch portion of the next instruction. If a double cycle instruction is in progress when the single-step command is received, both cycles will be completed before stopping.

3) The processor acknowledges it has entered the stopped state by raising SYNC high. In this state, which can be maintained indefinitely, the 10-bit address of the next instruction to be fetched is present on PORT 1 and the lower 2 bits of PORT 2.

4) $\overline{SS}$ is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing SYNC low.

5) To stop the processor at the next instruction $\overline{SS}$ must be brought low again before the next SYNC pulse—the circuit in Figure 4-1 uses the trailing edge of the previous pulse. If $\overline{SS}$ is left high, the processor remains in the "RUN" mode.

Figure 4-1 shows a schematic for implementing single-step. A single D-type flip-flop with preset and clear is used to generate $\overline{SS}$. In the RUN mode $\overline{SS}$ is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single-step, preset is removed allowing SYNC to bring $\overline{SS}$ low via the clear input. Note that SYNC must be buffered since the SN7474 is equivalent to 3 TTL loads.

The processor is now in the stopped state. The next instruction is initiated by clocking "1" into the flip-flop. This "1" will not appear on $\overline{SS}$ unless SYNC is high (i.e., clear must be removed from the flip-flop). In response to $\overline{SS}$ going high, the processor begins an instruction fetch which brings SYNC low. $\overline{SS}$ is then reset through the clear input and the processor again enters the stopped state.

PROGRAMMING, VERIFYING AND ERASING EPROM (8741A, 8742 EPROM ONLY)

The internal Program Memory of the 8741A and 8742 may be erased and reprogrammed by the user as explained in the following sections. See the data sheet for more detail.

Programming

The programming procedure consists of the following: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. Figure 4-4 illustrates the programming and verifying sequence. The following is a list of the pins used for programming and a description of their functions:

- XTAL 1, XTAL 2 — Clock Input
- $\overline{RESET}$ — Initialization and Address Latching
- TEST 0 — Selection of Program or Verify Mode
- EA — Activation of Program/Verify Modes
- D_0–D_7 — Address and Data Input Data Output During Verify
- P_{20}, P_{21} — Address Input
- V_{DD} — Programming Power Supply
- PROG — Program Pulse Input

NOTE: All set-up and hold times are 4 cycles.

The detailed Programming sequence (for one byte) is as follows:

1) Initial Conditions: $V_{CC} = V_{DD} = 5V$; Clock Running; $A_0 = 0V$, $\overline{CS} = 5V$; EA = 5V; D_0–D_7 and PROG Floating.
2) $\overline{RESET} = 0V$, TEST 0 = 0V (Select Programming Mode).
3) EA = 23V for 8741A
   EA = 18V for 8742
4) Address applied to D_0–D_7 and PORTS 20–22.
5) $\overline{RESET} = 5V$ (Latch Address).
6) Data applied to D_0–D_7.
7) $V_{DD} = 25V$ for 8741A
   $V_{DD} = 21V$ for 8742 (Programming Power).

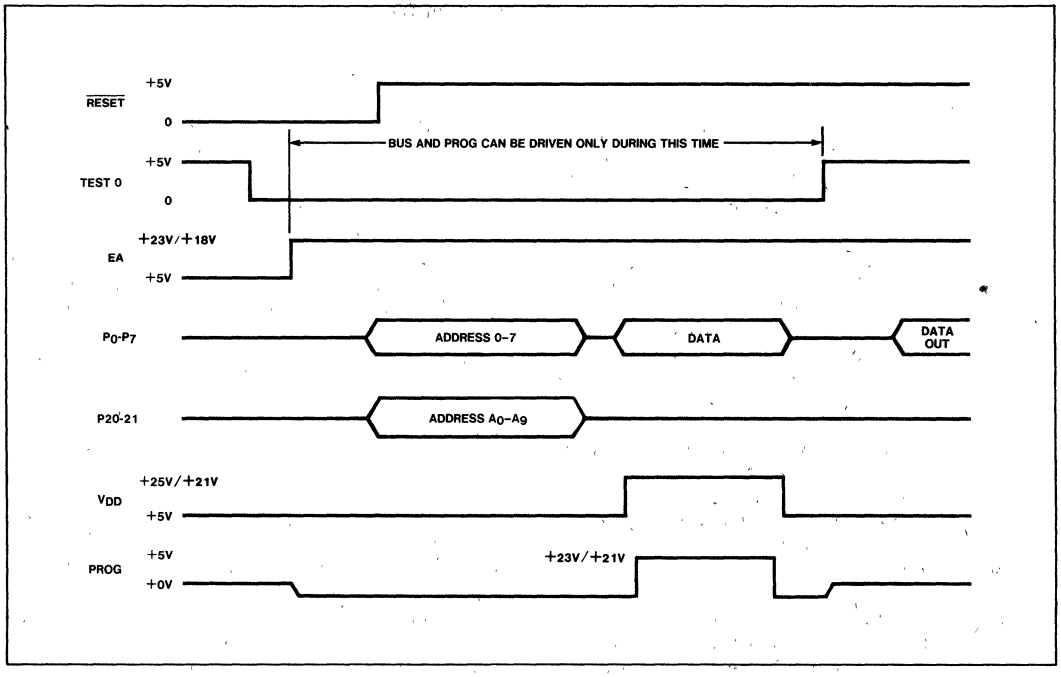


Figure 4-4. Programming Sequence

8) PROG = 0V followed by one 50 msec pulse of 23V for 8741A
PROG = 0V followed by one 50 msec pulse of 18V for 8742.

9) V_{DD} = 5V.

10) TEST 0 = 5V (Select Verify Mode).

11) Read data on D_0–D_7 and verify EPROM cell contents.

WARNING

An attempt to program a mis-socketed 8741A or 8742 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

Verification

Verification is accomplished by latching in an address as in the Programming Mode and then applying "1" to the TEST 0 input. The word stored at the selected address then appears on the D_0–D_7 lines. Note that verification can be applied to both ROM's and EPROM's independently of the programming procedure. See the data sheet.

The detailed Verifying sequence (for one byte) is as follows:

1) Initial Conditions: $V_{CC} = V_{DD}$ = 5V; Clock Running; A_0 = 0V, $\overline{CS}$ = 5V; EA = 5V; D_0–D_7 and PROG Floating.

2) $\overline{RESET}$ = 0V, TEST 0 = 5V (Verify Mode).

3) EA = 23V for 8741A
EA = 18V for 8742

4) Address applied to D_0–D_7 and PORTS 20–22.

5) $\overline{RESET}$ = 5V (Latch Address)

6) Read data on D_0–D_7 and verify EPROM cell contents.

Erasing

The program memory of the 8741A or 8742 may be erased to zeros by exposing its translucent lid to shortwave ultraviolet light.

EPROM Light Sensitivity

The erasure characteristics of the 8741A or 8742 EPROM are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms. It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Angstrom range. Data shows that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8741A or 8742 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels (available from Intel) should be placed over the 8741A or 8742 window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A or 8742 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms. The integrated dose (i.e., UV intensity × exposure time) for erasure should be a minimum of 15W-sec/cm^2 power rating. The erasure time with this dosage is approximately 15 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8741A or 8742 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

EXTERNAL ACCESS

The UPI family has an External Access mode (EA) which puts the processor into a test mode. This mode allows the user to disable the internal program memory and execute from external memory. External Access mode is useful in testing because it allows the user to test the processor's functions directly. It is only useful for testing since this mode uses D_0–D_7, PORTS 10–17 and PORTS 20–22.

This mode is invoked by connecting the EA pin to 5V. The 11-bit current program counter contents then come out on PORTS 10–17 and PORTS 20–22 after the SYNC output goes high. (PORT 10 is the least significant bit.) The desired instruction opcode is placed on D_0–D_7 before the start of state S_1. During state S_1, the opcode is sampled from D_0–D_7 and subsequently executed in place of the internal program memory contents.

The program counter contents are multiplexed with the I/O port data on PORTS 10–17 and PORTS 20–22. The I/O port data may be demultiplexed using an external latch on the rising edge of SYNC. The program counter contents may be demultiplexed similarly using the trailing edge of SYNC.

Reading and/or writing the Data Bus Buffer registers is still allowed although only when D_0–D_7 are not being sampled for opcode data. In practice, since this sampling time is not known externally, reads or

writes on the system bus are done during SYNC high time. Approximately 600ns are available for each read or write cycle.

POWER DOWN MODE (8041AH/8042 ROM ONLY)

Extra circuitry is included in the ROM version to allow low-power, standby operation. Power is removed from all system elements except the internal data RAM in the low-power mode. Thus the contents of RAM can be maintained and the device draws only 10 to 15% of its normal power.

The V_{CC} pin serves as the 5V power supply pin for all of the ROM version's circuitry except the data RAM array. The V_{DD} pin supplies only the RAM array. In normal operation, both V_{CC} and V_{DD} are connected to the same 5V power supply.

To enter the Power-Down mode, the $\overline{RESET}$ signal to the UPI is asserted. This ensures the memory will not be inadvertently altered by the UPI during power-down. The V_{CC} pin is then grounded while V_{DD} is maintained at 5V. Figure 4-5 illustrates a recommended Power-Down sequence. The sequence typically occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. The signal must occur early enough to guarantee the 8041AH or 8042 can save all necessary data before V_{CC} falls outside normal operating tolerance.

2) A "Power Failure" signal is used to interrupt the processor (via a timer overflow interrupt, for instance) and call a Power Failure service routine.

3) The Power Failure routine saves all important data and machine status in the RAM array. The routine may also initiate transfer of a backup supply to the V_{DD} pin and indicate to external circuitry that the Power Failure routine is complete.

4) A $\overline{RESET}$ signal is applied by external hardware to guarantee data will not be altered as the power supply falls out of limits. $\overline{RESET}$ must be low until V_{CC} reaches ground potential.

Recovery from the Power-Down mode can occur as any other power-on sequence. An external 1 μfd capacitor on the $\overline{RESET}$ input will provide the necessary initialization pulse.

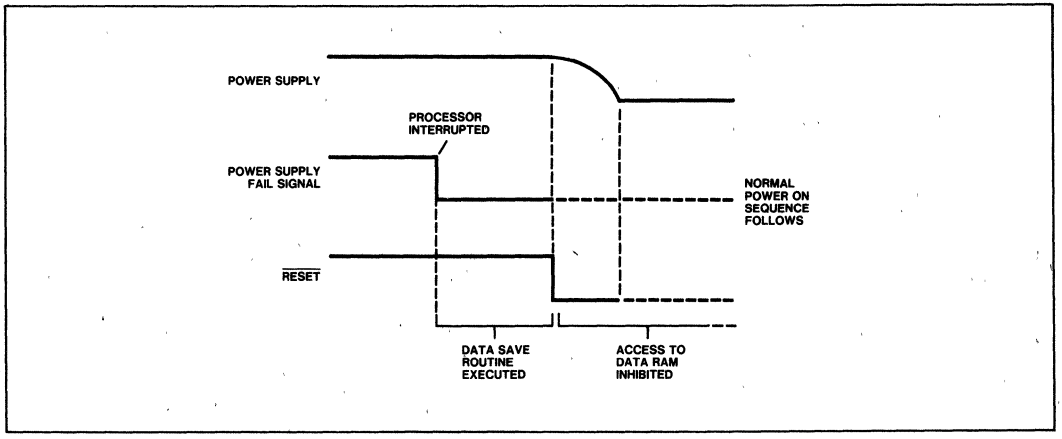


Figure 4-5. Power-Down Sequence

CHAPTER 5
SYSTEM OPERATION

BUS INTERFACE

The UPI-41AH, 42 Microcomputer functions as a peripheral to a master processor by using the data bus buffer registers to handle data transfers. The DBB configuration is illustrated in Figure 5-1. The UPI-41AH, 42 Microcomputer's 8 three-state data lines (D_7–D_0) connect directly to the master processor's data bus. Data transfer to the master is controlled by 4 external inputs to the UPI:

- A_0 Address Input signifying command or data
- $\overline{CS}$ Chip Select
- $\overline{RD}$ Read strobe
- $\overline{WR}$ Write strobe

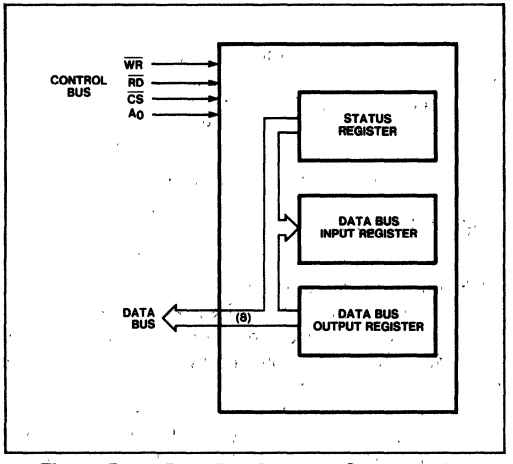


Figure 5-1. Data Bus Register Configuration

The master processor addresses the UPI-41AH, 42 Microcomputer as a standard peripheral device. Table 5-1 shows the conditions for data transfer:

Table 5-1. Data Transfer Controls

$\overline{CS}$	A_0	$\overline{RD}$	$\overline{WR}$	Condition
0	0	0	1	Read DBBOUT
0	1	0	1	Read STATUS
0	0	1	0	Write DBBIN data, set $F_1 = 0$
0	1	1	0	Write DBBIN command set $F_1 = 1$
1	x	x	x	Disable DBB

Reading the DBBOUT Register

The sequence for reading the DBBOUT register is shown in Figure 5-2. This operation causes the 8-bit contents of the DBBOUT register to be placed on the system Data Bus. The OBF flag is cleared automatically.

Reading STATUS

The sequence for reading the UPI-41AH, 42 Microcomputer's 8 STATUS bits is shown in Figure 5-3. This operation causes the 8-bit STATUS register contents to be placed on the system Data Bus as shown.

Write Data to DBBIN

The sequence for writing data to the DBBIN register is shown in Figure 5-4. This operation causes the system Data Bus contents to be transferred to the DBBIN register and the IBF flag is set. Also, the F_1 flag is cleared ($F_1 = 0$) and an interrupt request is generated. When the IBF interrupt is enabled, a jump to location 3 will occur. The interrupt request is cleared upon entering the IBF service routine or by a system RESET input.

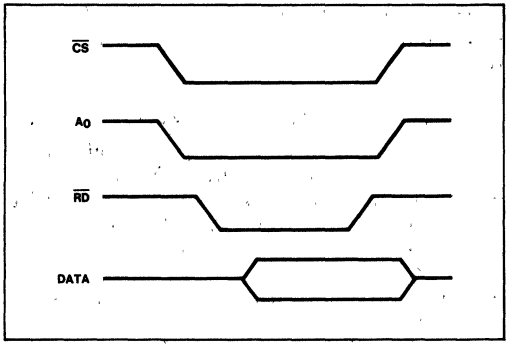


Figure 5-2. DBBOUT Read

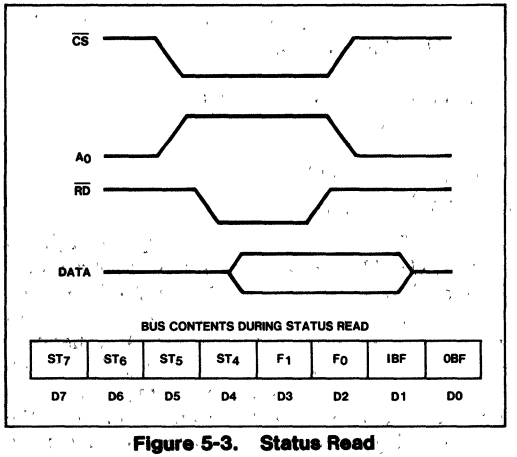


Figure 5-3. Status Read

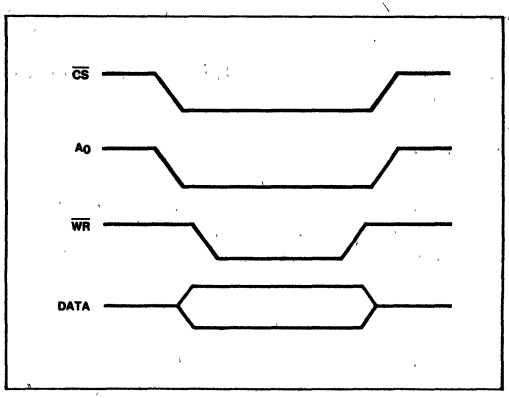


Figure 5-4. Writing Data to DBBIN

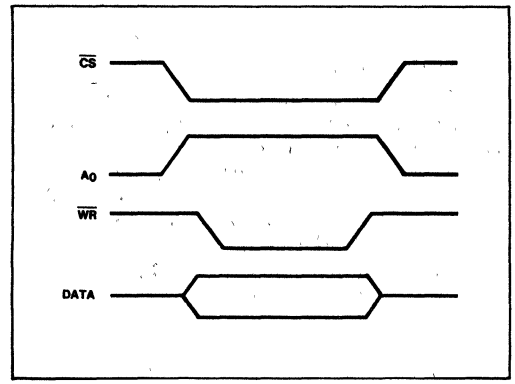


Figure 5-5. Writing Commands to DBBIN

Writing Commands to DBBIN

The sequence for writing commands to the DBBIN register is shown in Figure 5-5. This sequence is identical to a data write except that the A_0 input is latched in the F_1 flag ($F_1 = 1$). The IBF flag is set and an interrupt request is generated when the master writes a command to DBB.

Operations of Data Bus Registers

The UPI-41AH, 42 Microcomputer controls the transfer of DBB data to its accumulator by executing INput and OUTput instructions. An IN A,DBB instruction causes the contents to be transferred to the UPI accumulator and the IBF flag is cleared.

The OUT DBB,A instruction causes the contents of the accumulator to be transferred to the DBBOUT register. The OBF flag is set.

The UPI's data bus buffer interface is applicable to a variety of microprocessors including the 8086, 8088, 8085AH, 8080, and 8048.

A description of the interface to each of these processors follows.

DESIGN EXAMPLES

8085AH Interface

Figure 5-6 illustrates an 8085AH system using a UPI-41AH, 42. The 8085AH system uses a multiplexed address and data bus. During I/O the 8 upper address lines (A_8–A_{15}) contain the same I/O address as the lower 8 address/data lines (A_0–A_7); therefore I/O address decoding is done using only the upper 8 lines to eliminate latching of the address. An 8205 decoder provides address decoding for both the UPI-41AH, 42 and the 8237. Data is transferred using the two DMA handshaking lines of PORT 2. The 8237 performs the actual bus transfer operation. Using the UPI-41AH, 42's OBF master interrupt, the UPI-41A notifies the 8085AH upon transfer completion using the RST 5.5 interrupt input. The $\overline{IBF}$ master interrupt is not used in this example.

8088 Interface

Figure 5-7 illustrates a UPI-41AH, 42 interface to an 8088 minimum mode system. Two 8-bit latches are used to demultiplex the address and data bus. The address bus is 20-lines wide. For I/O only, the lower 16 address lines are used, providing an addressing range of 64K. UPI address selection is accomplished using an 8205 decoder. The A_0 address line of the bus is connected to the corresponding UPI input for register selection. Since the UPI-41A is polled by the 8088, neither DMA nor master interrupt capabilities of the UPI-41AH, 42 are used in the figure.

8086 Interface

The UPI-41AH, 42 can be used on an 8086 maximum mode system as shown in figure 5-8. The address and data bus is demultiplexed using three 8282 latches providing separate address and data buses. The address bus is 20-lines wide and the data bus is 16-lines wide. Multiplexed control lines are decoded by the 8288. The UPI's $\overline{CS}$ input is provided by linear selection. Note that the UPI-41AH, 42 is both I/O mapped and memory mapped as a result of the linear addressing technique. An address decoder may be used to limit the UPI-41AH, 42 to a specific I/O mapped address. Address line A_1 is connected to the UPI's A_0 input. This insures that the registers of the UPI will have even I/O addresses. Data will be transferred on D_0–D_7 lines only. This allows the I/O registers to be accessed using byte manipulation instructions.

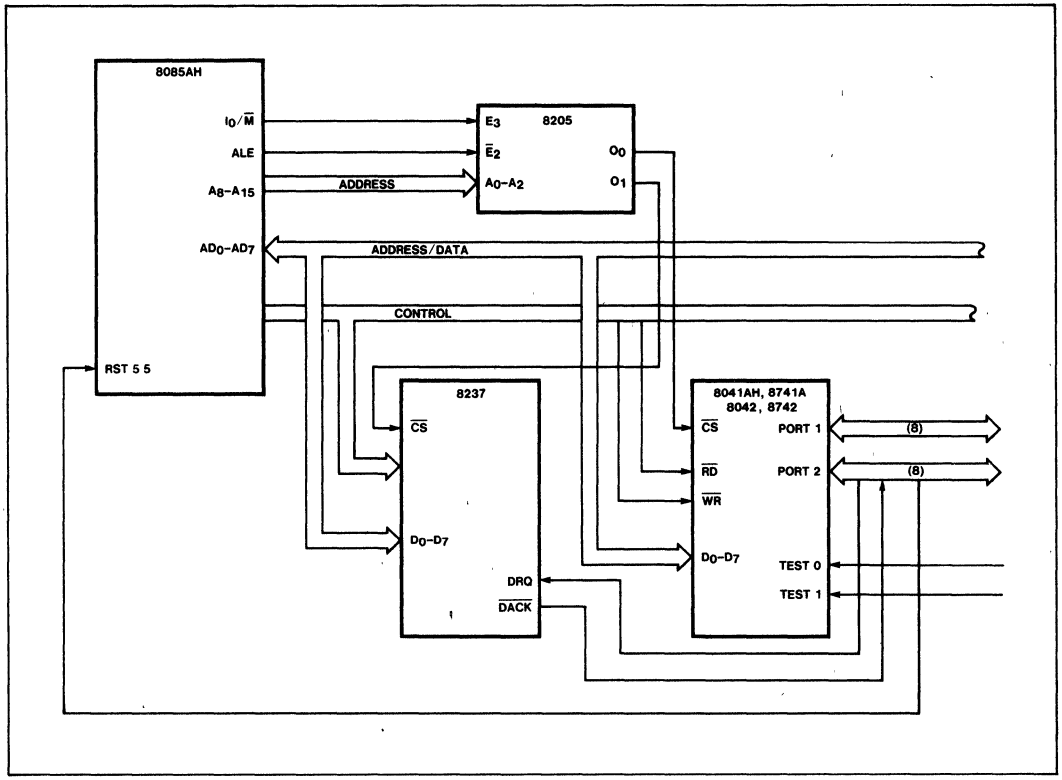


Figure 5-6. 8085AH-UPI System

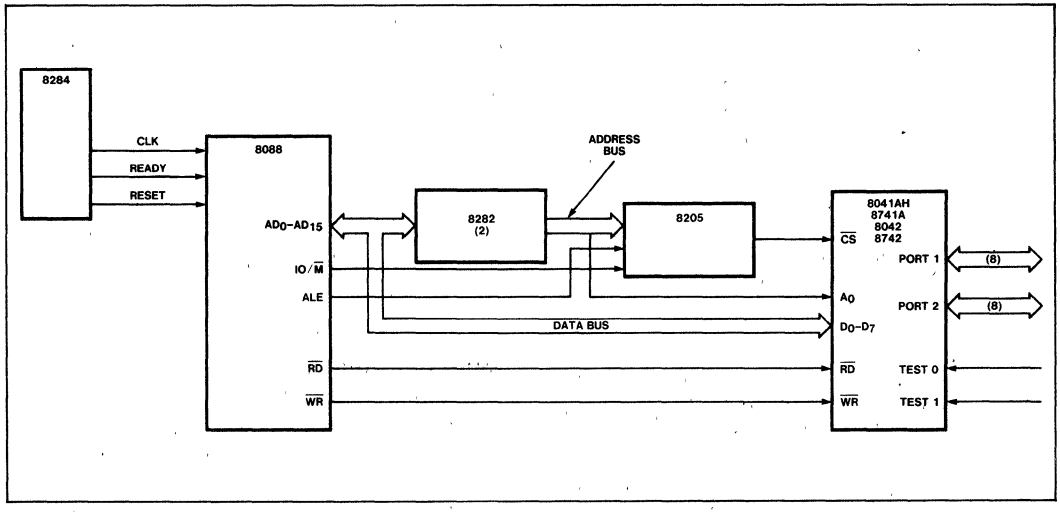


Figure 5-7. 8088-UPI Minimum Mode System

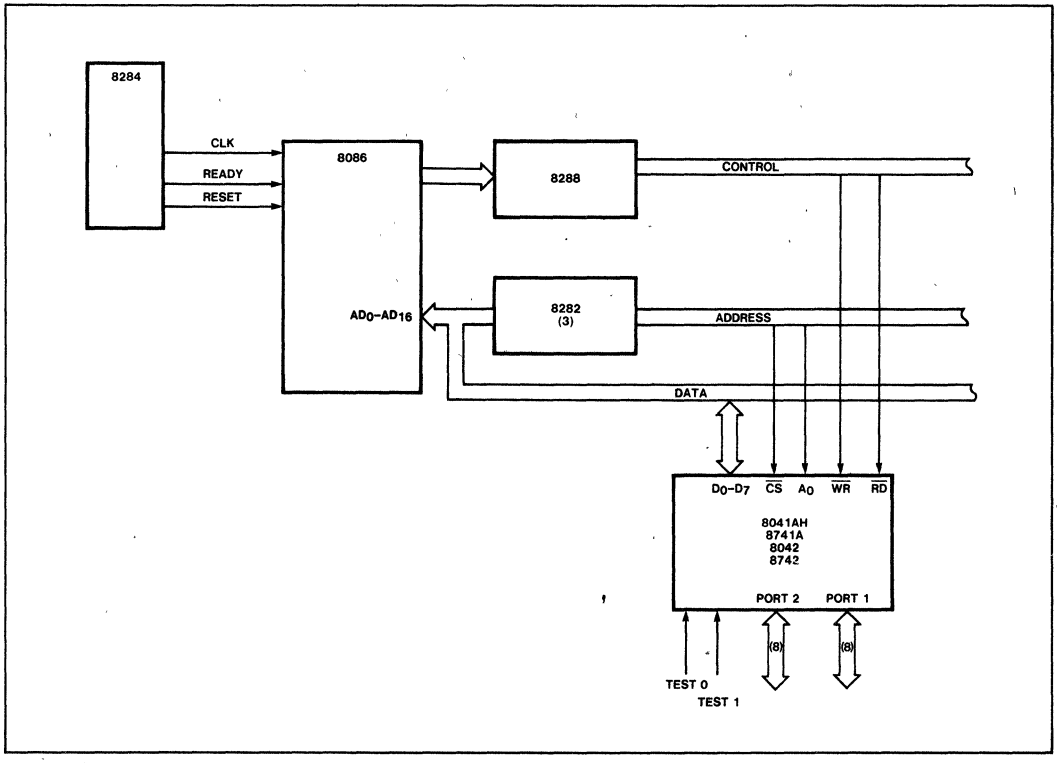


Figure 5-8. 8086-UPI Maximum Mode Systems

8080 Interface

Figure 5-9 illustrates the interface to an 8080A system. In this example, a crystal and capacitor are used for UPI-41AH, 42 timing reference and power-on RESET. If the 2-MHz 8080A 2-phase clock were used instead of the crystal, the UPI-41AH, UPI-42 would run at only 16% full speed.

The A_0 and $\overline{CS}$ inputs are direct connections to the 8080 address bus. In larger systems, however, either of these inputs may be decoded from the 16 address lines.

The $\overline{RD}$ and $\overline{WR}$ inputs to the UPI can be either the $\overline{IOR}$ and $\overline{IOW}$ or the $\overline{MEMR}$ and $\overline{MEMR}$ signals depending on the I/O mapping technique to be used.

The UPI can be addressed as an I/O device using INput and OUTput instructions in 8080 software.

8048 Interface

Figure 5-10 shows the UPI interface to an 8048 master processor.

The 8048 $\overline{RD}$ and $\overline{WR}$ outputs are directly compatible with the UPI. Figure 5-11 shows a distributed processing system with up to seven UPI's connected to a single 8048 master processor.

In this configuration the 8048 uses PORT 0 as a data bus. I/O PORT 2 is used to select one of the seven UPI's when data transfer occurs. The UPI's are programmed to handle isolated tasks and, since they operate in parallel, system throughput is increased.

GENERAL HANDSHAKING PROTOCOL

1) Master reads STATUS register ($\overline{RD}$, $\overline{CS}$, A_0 = (0, 0, 1)) in polling or in response to either an $\overline{IBF}$ or an OBF interrupt.

2) If the UPI DBBIN register is empty (IBF flag = 0), Master writes a word to the DBBIN register ($\overline{WR}$, $\overline{CS}$, A_0 = (0, 0, 1) or (0, 0, 0)). If A_0 = 1, write command word, set F_1. If A_0 = 0, write data word, F_1 = 0.

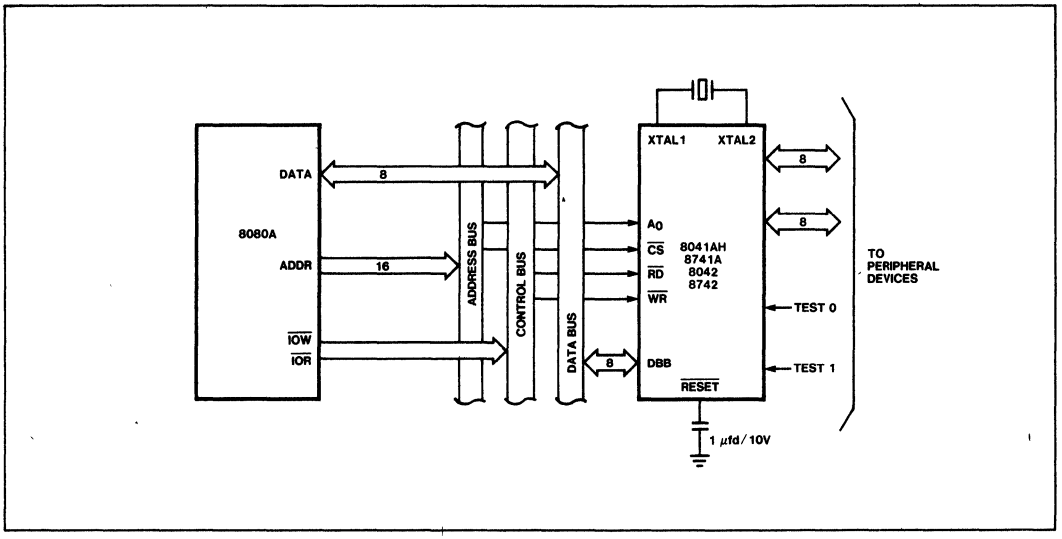


Figure 5-9. 8080A-UPI Interface

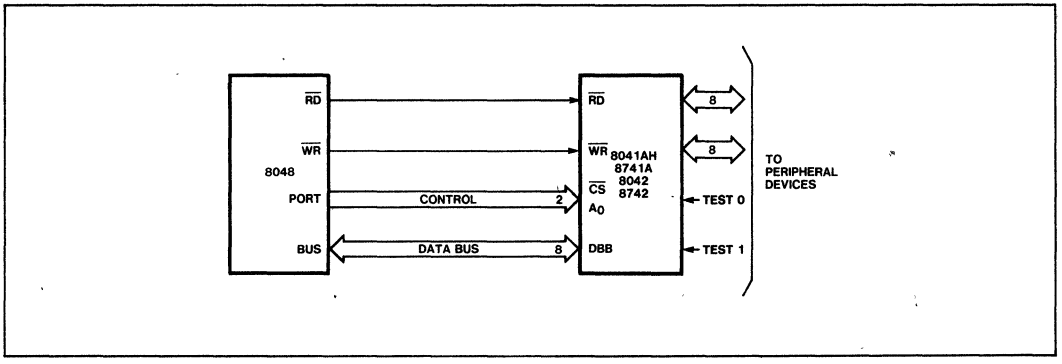


Figure 5-10. 8048-UPI Interface

3) If the UPI DBBOUT register is full (OBF flag = 1), Master reads a word from the DBBOUT register ($\overline{RD}$, $\overline{CS}$, A_0 = (0, 0, 0)).

4) UPI recognizes IBF (via IBF interrupt or JNIBF). Input data or command word is processed, depending on F_1; IBF is reset. Repeat step 1 above.

5) UPI-41AH, 42 recognizes OBF flag = 0 (via JOBF). Next word is output to DBBOUT register, OBF is set. Repeat step 1 above.

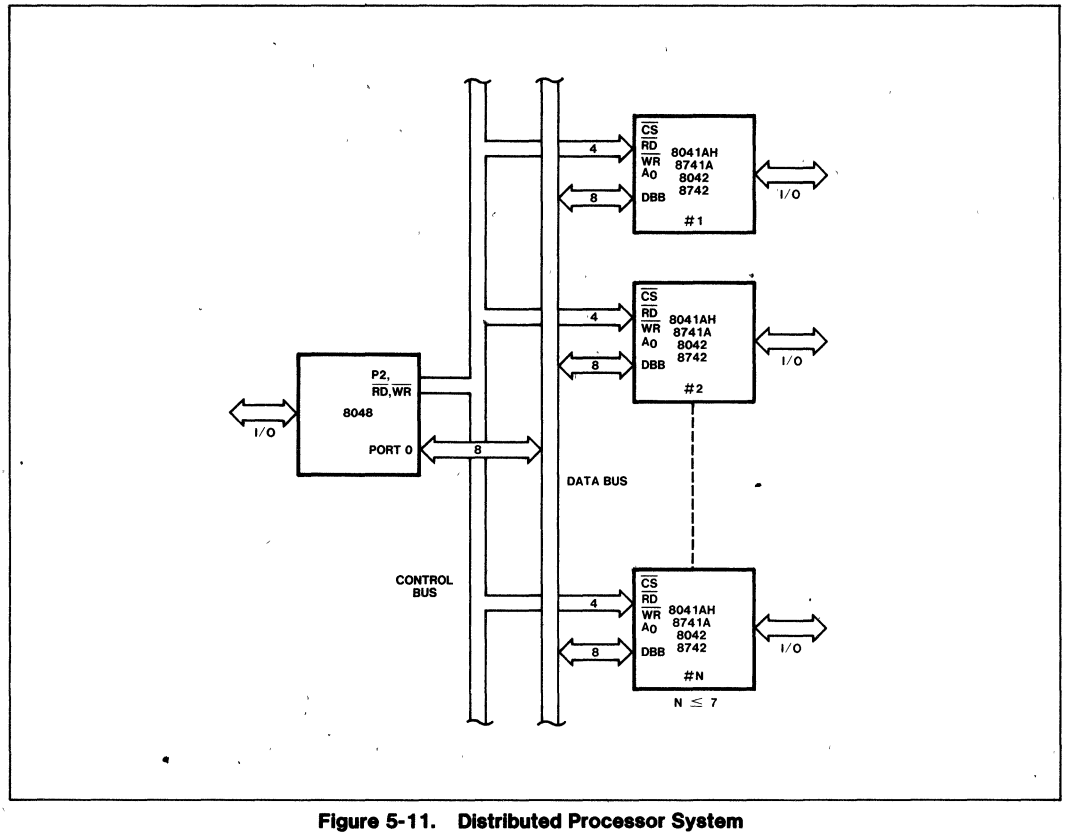


Figure 5-11. Distributed Processor System

Chapter 6
APPLICATIONS

ABSTRACTS

The UPI-41A is designed to fill a wide variety of low to medium speed peripheral interface applications where flexibility and easy implementation are important considerations. The following examples illustrate some typical applications.

Keyboard Encoder

Figure 6-1 illustrates a keyboard encoder configuration using the UPI and the 8243 I/O expander to scan a 128-key matrix. The encoder has switch matrix scanning logic, N-key rollover logic, ROM look-up table, FIFO character buffer, and additional outputs for display functions, control keys or other special functions.

PORT 1 and PORTs 4–7 provide the interface to the keyboard. PORT 1 lines are set one at a time to select the various key matrix rows.

When a row is energized, all 16 columns (i.e., PORTs 4–7 inputs) are sampled to determine if any switch in the row is closed. The scanning software is code efficient because the UPI instruction set includes individual bit set/clear operations and expander PORTs 4–7 can be directly addressed with single, 2-byte instructions. Also, accumulator bits can be tested in a single operation. Scan time for 128 keys is about 10 ms. Each matrix point has a unique binary code which is used to address ROM when a key closure is detected. Page 3 of ROM contains a look-up table with useable codes (i.e., ASCII, EBCDIC, etc.) which correspond to each key. When a valid key closure is detected the ROM code corresponding to that key is stored in a FIFO buffer in data memory for transfer to the master processor. To avoid stray noise and switch bounce, a key closure must be detected on two consecutive scans before it is considered valid and loaded into the FIFO buffer. The FIFO buffer allows multiple keys to be processed as they are depressed without regard to when they are released, a condition known as N-key rollover.

The basic features of this encoder are fairly standard and require only about 500 bytes of memory. Since the UPI is programmable and has additional memory capacity it can handle a number of other functions. For example, special keys can be programmed to give an entry on closing as well as opening. Also, I/O lines are available to control a 16-digit, 7-segment display. The UPI can also be programmed to recognize special combinations of characters such as commands, then transfer only the decoded information to the master processor.

A complete keyboard application has been developed for the UPI-41A. A description is included in this section. The code for the application is available in the Intel Insite Library (program AB 147).

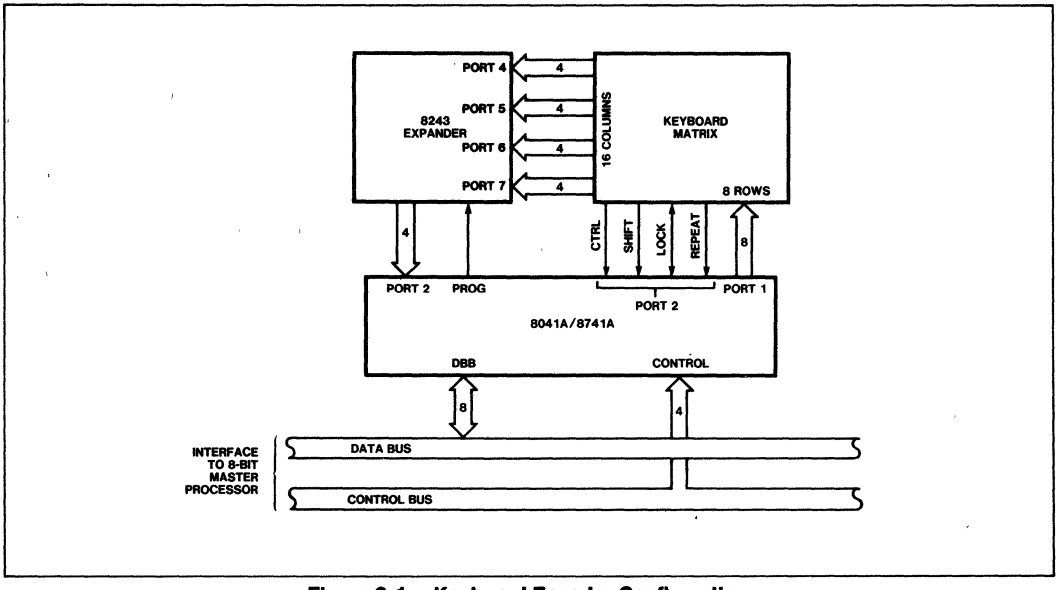


Figure 6-1. Keyboard Encoder Configuration

Matrix Printer Interface

The matrix printer interface illustrated in Figure 6-2 is a typical application for the UPI-41A. The actual printer mechanism could be any of the numerous dot-matrix types and similar configurations can be shown for drum, spherical head, daisy wheel or chain type printers.

The bus structure shown represents a generalized, 8-bit system bus configuration. The UPI's three-state interface port and asynchronous data buffer registers allow it to connect directly to this type of system for efficient, two-way data transfer.

The UPI's two on-board I/O ports provide up to 16 input and output signals to control the printer mechanism. The timer/event counter is used for generating a timing sequence to control print head position, line feed, carriage return, and other sequences. The on-board program memory provides character generation for 5 × 7, 7 × 9, or other dot matrix formats. As an added feature a portion of the 64 × 8-bit data memory can be used as a FIFO buffer so that the master processor can send a block of data at a high rate. The UPI can then output characters from the buffer at a rate the printer can accept while the master processor returns to other tasks.

The 8295 Printer Controller is an example of an 8041A preprogrammed as a dot matrix printer interface.

Tape Cassette Controller

Figure 6-3 illustrates a digital cassette interface which can be implemented with the UPI-41A. Two sections of the tape transport are controlled by the UPI: digital data/command logic, and motor servo control.

The motor servo requires a speed reference in the form of a monostable pulse whose width is proportional to the desired speed. The UPI monitors a prerecorded clock from the tape and uses its on-board interval timer to generate the required speed reference pulses at each clock transition.

Recorded data from the tape is supplied serially by the data/command logic and is converted to 8-bit words by the UPI, then transferred to the master processor. At 10 ips tape speed the UPI can easily handle the 8000 bps data rate. To record data, the UPI uses the two input lines to the data/command logic which control the flux direction in the recording head. The UPI also monitors 4 status lines from the tape transport including: end of tape, cassette

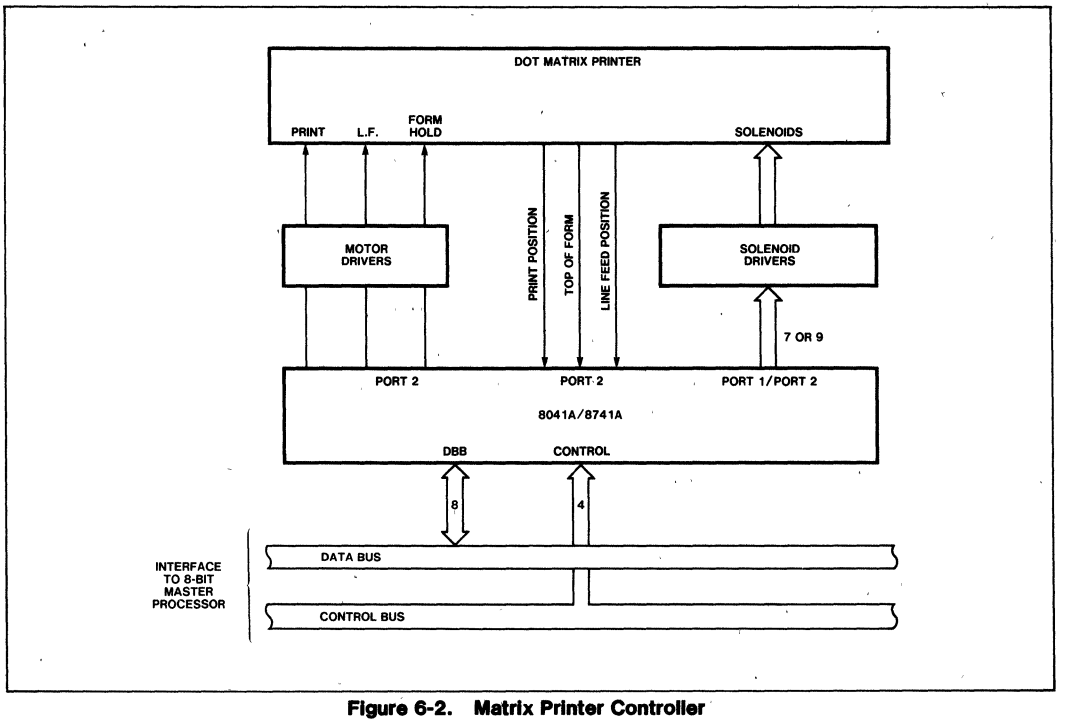


Figure 6-2. Matrix Printer Controller

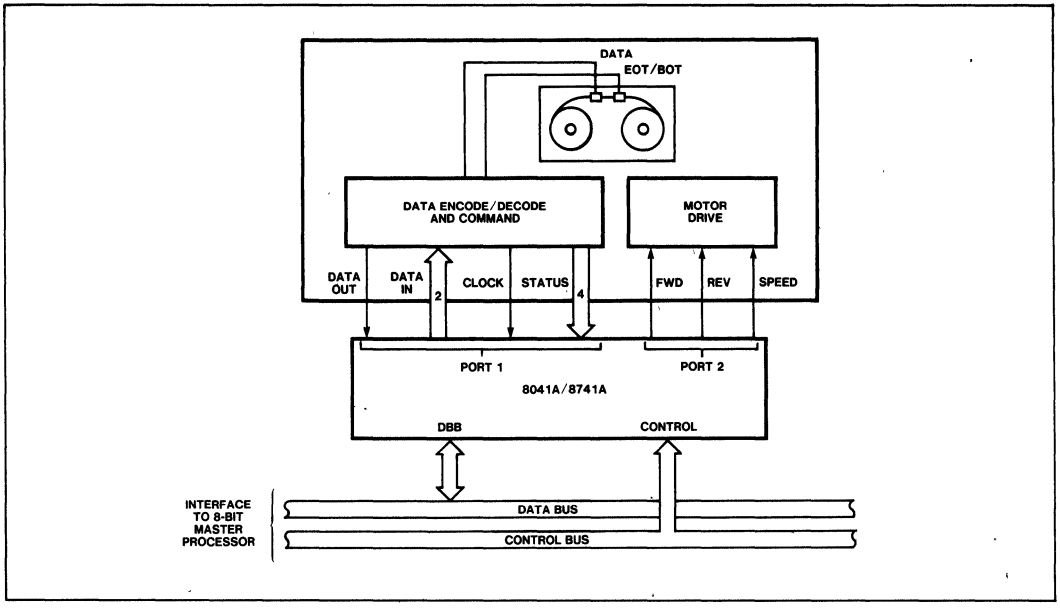


Figure 6-3. Tape Transport Controller

inserted, busy, and write permit. All control signals can be handled by the UPI's two I/O ports.

Universal I/O Interface

Figure 6-4 shows an I/O interface design based on the UPI. This configuration includes 12 parallel I/O lines and a serial (RS232C) interface for full duplex data transfer up to 1200 baud. This type of design can be used to interface a master processor to a broad spectrum of peripheral devices as well as to a serial communication channel.

PORT 1 is used strictly for I/O in this example while PORT 2 lines provide five functions:

- P_{23}-P_{20} I/O lines (bidirectional)
- P_{24} Request to send (RTS)
- P_{25} Clear to Send (CTS)
- P_{26} Interrupt to master
- P_{27} Serial data out

The parallel I/O lines make use of the bidirectional port structure of the UPI. Any line can function as an input or output. All port lines are automatically initialized to 1 by a system $\overline{\text{RESET}}$ pulse and remain

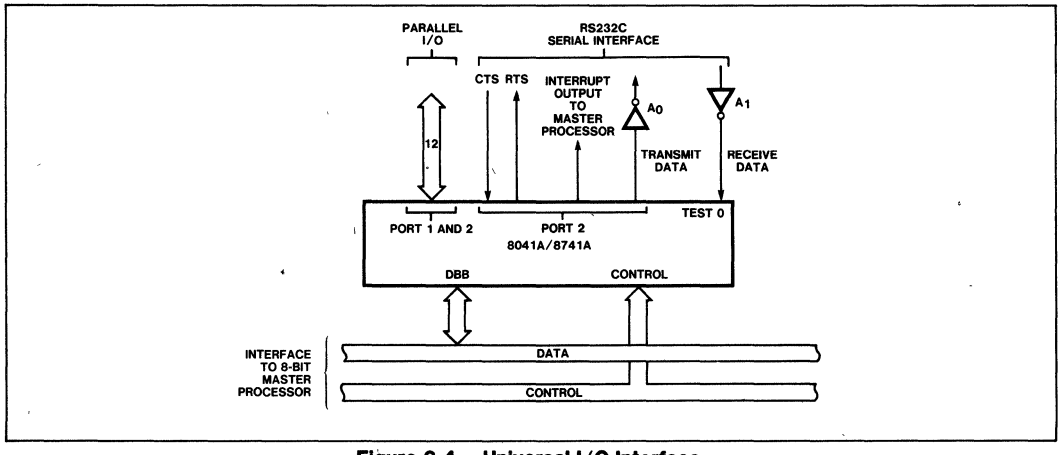


Figure 6-4. Universal I/O Interface

latched. An external TTL signal connected to a port line will override the UPI's 50K-ohm internal pull-up so that an INPUT instruction will correctly sample the TTL signal.

Four PORT 2 lines function as general I/O similar to PORT 1. Also, the RTS signal is generated on PORT 2 under software control when the UPI has serial data to send. The CTS signal is monitored via PORT 2 as an enable to the UPI to send serial data. A PORT 2 line is also used as a software generated interrupt to the master processor. The interrupt functions as a service request when the UPI has a byte of data to transfer or when it is ready to receive. Alternatively, the EN FLAGS instruction could be used to create the OBF and $\overline{IBF}$ interrupts on P_{24} and P_{25}.

The RS232C interface is implemented using the TEST 0 pin as a receive input and a PORT 2 pin as a transmit output. External packages (A_0, A_1) are used to provide RS232C drive requirements. The serial receive software is interrupt driven and uses the on-chip timer to perform time critical serial control. After a start bit is detected the interval timer can be preset to generate an interrupt at the proper time for sampling the serial bit stream. This eliminates the need for software timing loops and allows the processor to proceed to other tasks (i.e., parallel I/O operations) between serial bit samples. Software flags are used so the main program can determine when the interrupt driven receive program has a character assembled for it.

This type of configuration allows system designers flexibility in designing custom I/O interfaces for specific serial and parallel I/O applications. For instance, a second or third serial channel could be substituted in place of the parallel I/O if required. The UPI's data memory can buffer data and commands for up to 4 low-speed channels (110 baud teletypewriter, etc.)

Application Notes

The following application notes illustrate the various applications of the UPI family. Other related publications including the *8048 Family Application Handbook* are available through the Intel Literature Department.

INTRODUCTION TO THE UPI-41A™

Introduction

Since the introduction in 1974 of the second generation of microprocessors, such as the 8080, a wide range of peripheral interface devices have appeared. At first, these devices solved application problems of a general nature; i.e., parallel interface (8255), serial interface (8251), timing (8253), interrupt control (8259). However, as the speed and density of LSI technology increased, more and more intelligence was incorporated into the peripheral devices. This allowed more specific application problems to be solved, such as floppy disk control (8271), CRT control (8275), and data link control (8273). The advantage to the system designer of this increased peripheral device intelligence is that many of the peripheral control tasks are now handled externally to the main processor in the peripheral hardware rather than internally in the main processor software. This reduced main processor overhead results in increased system throughput and reduced software complexity.

In spite of the number of peripheral devices available, the pervasiveness of the microprocessor has been such that there is still a large number of peripheral control applications not yet satisfied by dedicated LSI. Complicating this problem is the fact that new applications are emerging faster than the manufacturers can react in developing new, dedicated peripheral controllers. To address this problem, a new microcomputer-based Universal Peripheral Interface (UPI-41A) device was developed.

In essence, the UPI-41A acts as a slave processor to the main system CPU. The UPI contains its own processor, memory, and I/O, and is completely user programmable; that is, the entire peripheral control algorithm can be programmed locally in the UPI, instead of taxing the master processor's main memory. This distributed processing concept allows the UPI to handle the real-time tasks such as encoding keyboards, controlling printers, or multiplexing displays, while the main processor is handling non-real-time dependent tasks such as buffer management or arithmetic. The UPI relies on the master only for initialization, elementary commands, and data transfers. This technique results in an overall increase in system efficiency since both processors—the master CPU and the slave UPI—are working in parallel.

This application note presents three UPI-41A applications which are roughly divided into two groups: applications whose complexity and UPI code space requirements allow them to either stand alone or be incorporated as just one task in a "multi-tasking" UPI, and applications which are complete UPI applications in themselves. Applications in the first group are a simple LED display and sensor matrix controllers. A combination serial/parallel/ I/O device is an application in the second group. Each application illustrates different UPI configurations and features. However, before the application details are presented, a section on the UPI/master protocol requirements is included. These protocol requirements are key to UPI software development. For convenience, the UPI block diagram is reproduced in Figure 1 and the instruction set summary in Table 1.

UPI-41 vs. UPI-41A

The UPI-41A is an enhanced version of the UPI-41. It incorporates several architectural features not found on the "non-A" device:

- Separate Data In and Data Out data bus buffer registers
- User-definable STATUS register bits
- Programmable master interrupts for the OBF and $\overline{\text{IBF}}$ flags
- Programmable DMA interface to external DMA controller.

The separate Data In (DBBIN) and Data Out (DBBOUT) registers greatly simplify the master/UPI protocol compated to the UPI-41. The master need only check IBF before writing to DBBIN and OBF before reading DBBOUT. No data bus buffer lock-out is required.

The most significant nibble of the STATUS register, undefined in the UPI-41, is user-definable in UPI-41A. It may be loaded directly from the most significant nibble of the Accumulator (MOV STS,A). These extra four STATUS bits are useful for transferring additional status information to the master. This application note uses this feature extensively.

A new instruction, EN FLAGS, allows OBF and $\overline{\text{IBF}}$ to be reflected on PORT 2 BIT 4 and PORT 2 BIT 5 respectively. This feature enables interrupt-driven data transfers when these pins are interrupt sources to the master.

By executing an EN DMA instruction PORT 2 BIT 6 becomes a DRQ (DMA Request) output and PORT 2 BIT 7 becomes $\overline{\text{DACK}}$ (DMA Acknowledge). Setting DRQ requests a DMA cycle to an external DMA controller. When the cycle is granted, the DMA controller returns DACK plus either $\overline{\text{RD}}$ (Read) or $\overline{\text{WR}}$ (Write). DACK automatically forces

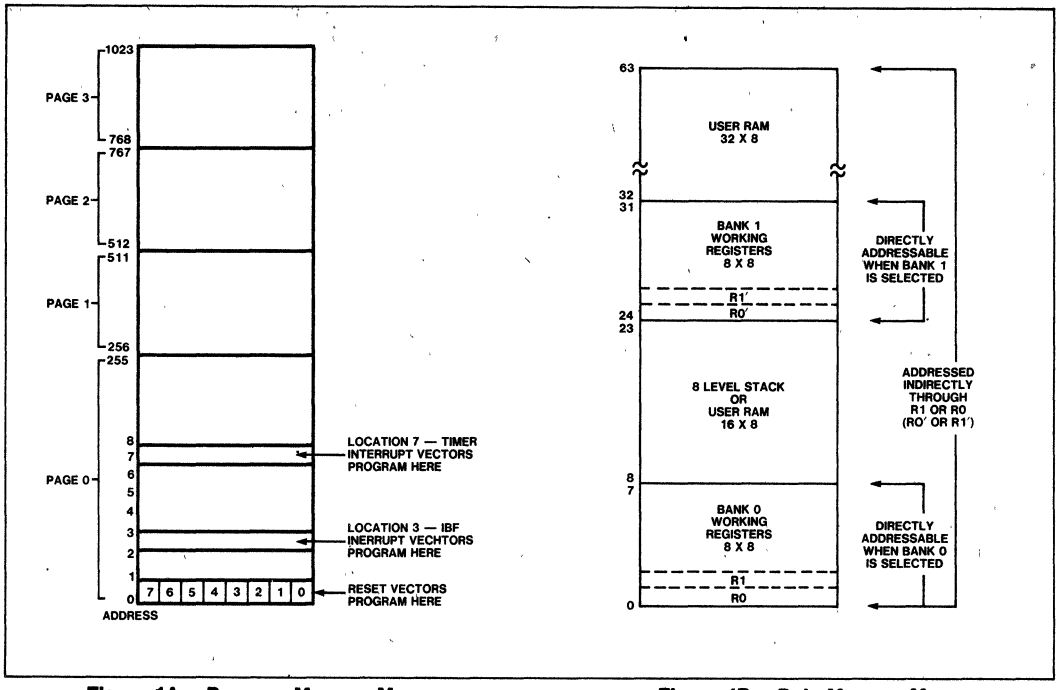


Figure 1A. Program Memory Map

Figure 1B. Data Memory Map

$\overline{CS}$ and A_0 low internally and clears DRQ. This selects the appropriate data buffer register (DBBOUT for DACK and $\overline{RD}$, DBBIN for DACK and $\overline{WR}$) for the DMA transfer.

Like the "non-A", the UPI-41A is available in both ROM (8041A) and EPROM (8741A) Program Memory versions. This application note deals exclusively with the UPI-41A since the applications use the "A"s enhanced features.

UPI/MASTER PROTOCOL

As in most closely coupled multiprocessor systems, the various processors communicate via a shared resource. This shared resource is typically specific locations in RAM or in registers through which status and data are passed. In the case of a master processor and a UPI-41A, the shared resource is 3 separate, master-addressable, registers internal to the UPI. These registers are the status register (STATUS), the Data Bus Buffer Input register (DBBIN), and the Data Bus Output register (DBBOUT). [Data Bus Buffer direction is relative to the UPI]. To illustrate this register interface, consider the 8085A/UPI system in Figure 2.

Looking into the UPI from the 8085A, the 8085A sees only the three registers mentioned above. If the 8085A wishes to issue a command to the UPI, it does so by writing the command to the DBBIN register according to the decoding of Table 2. Data for the UPI is also passed via the DBBIN register. (The UPI differentiates commands and data by examining the A_0 pin. Just how this is done is covered shortly.) Data from the UPI for the 8085A is passed in the DBBOUT register. The 8085A may interrogate the UPI's status by reading the UPI's STATUS register. Four bits of the STATUS register act as flags and are used to handshake data and commands into and out of the UPI. The STATUS register format is shown in Figure 3.

BIT 0 is OBF (Output Buffer Full). This flag indicates to the master when the UPI has placed data in the DBBOUT register. OBF is set when the UPI writes to DBBOUT and is reset when the master reads DBBOUT. The master finds meaningful data in the DBBOUT register only when OBF is set.

The Input Buffer Full (IBF) flag is BIT 1. The UPI uses this flag as an indicator that the master has written to the DBBIN register. The master uses IBF

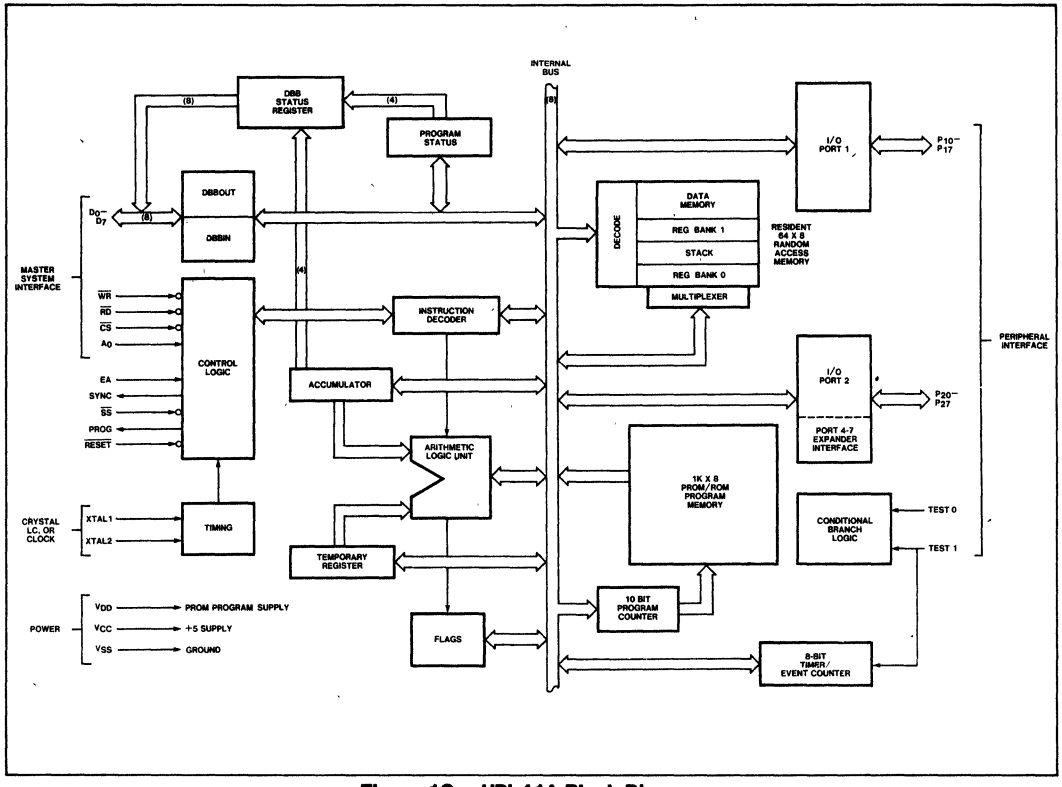


Figure 1C. UPI-41A Block Diagram

to indicate when the UPI has accepted a particular command or data byte. The master should examine IBF before outputting anything to the UPI. IBF is set when the master writes to DBBIN and is reset when the UPI reads DBBIN. The master must wait until IBF=0 before writing new data or commands to DBBIN. Conversely, the UPI must ensure IBF=1 before reading DBBIN.

The third STATUS register bit is F_0 (FLAG 0). This is a general purpose flag that the UPI can set, reset, and test. It is typically used to indicate a UPI error or busy condition to the master.

FLAG 1 (F_1) is the final dedicated STATUS bit. Like F_0 the UPI can set, reset, and test this flag. However, in addition, F_1 reflects the state of the A_0 pin whenever the master writes to the DBBIN register. The UPI uses this flag to delineate between master command and data writes to DBBIN.

The remaining four STATUS register bits are user definable. Typical uses of these bits are as status indicators for individual tasks in a multitasking UPI or as UPI generated interrupt status. These bits find a wide variety of uses in the upcoming applications.

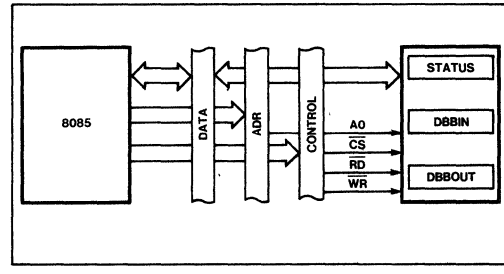


Figure 2. Register Interface

Looking into the 8085A from the UPI, the UPI sees the two DBB registers plus the IBF, OBF, and F_1 flags. The UPI can write from its accumulator to DBBOUT or read DBBIN into the accumulator. The UPI cannot read OBF, IBF, or F_1 directly, but these flags may be tested using conditional jump

AFN-01536A

Table 1. Instruction Set Summary

Mnemonic	Description	Bytes	Cycles
Accumulator			
ADD A,R_r	Add register to A	1	1
ADD A,@R_r	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,R_r	Add register to A with carry	1	1
ADDC A @R_r	Add data memory to A with carry	1	1
ADDC A,#data	Add immed. to A with carry	2	2
ANL a,R_r	AND register to A	1	1
ANL A,@R_r	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,R_r	OR register to A	1	1
ORL A@R_r	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,R_r	Exclusive OR register to A	1	1
XRL A,@R_r	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap digits of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A,P_p	Input port to A	1	2
OUTL P_p,A	Output A to port	1	2
ANL P_p,#data	AND immediate to port	2	2
ORL P_p,#data	OR immediate to port	2	2
IN A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB,A	Output A to DBB, set OBF	1	1
MOV STS,A	A_4-A_7 to Bits 4-7 of Status	1	1
MOVD A,P_p	Input Expander port to A	1	2
MOVD P_p,A	Output A to Expander port	1	2
ANLD P_p,A	AND A to Expander port	1	2
ORLD P_p,A	OR A to Expander port	1	2
Data Moves			
MOV A,R_r	Move register to A	1	1
MOV A,@R_r	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV R_r,A	Move A to register	1	1
MOV @R_r,A	Move A to data memory	1	1
MOV R_r,#data	Move immediate to register	2	2
MOV @R_r,#data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,R_r	Exchange A and register	1	1
XCH A,@R_r	Exchange A and data memory	1	1
XCHD A@R_r	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3, A,@A	Move to A from page 3	1	2

Mnemonic	Description	Bytes	Cycles
Timer/Counter			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
Registers			
INC R_r	Increment register	1	1
INC @R_r	Increment data memory	1	1
DEC R_r	Decrement register	1	1
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
Branch			
JMP ADDR	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R,addr	Decrement register and skip	2	2
JC addr	Jump on Carry=1	2	2
JNC addr	Jump on Carry=0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0=1	2	2
JNT0 addr	Jump on T0=0	2	2
JT1 addr	Jump on T1=1	2	2
JNT1 addr	Jump on T1=0	2	2
JF0 addr	Jump on F0 Flag=1	2	2
JF1 addr	Jump on F1 Flag=1	2	2
JTF addr	Jump on Timer Flag=1,Clear Flag	2	2
JNIBF addr	Jump on IBF Flag=0	2	2
JOBF addr	Jump on OBF Flag=1	2	2
JBb addr	Jump on Accumulator Bit	2	2

Table 2. Register Decoding

$\overline{CS}$	A0	$\overline{RD}$	$\overline{WR}$	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

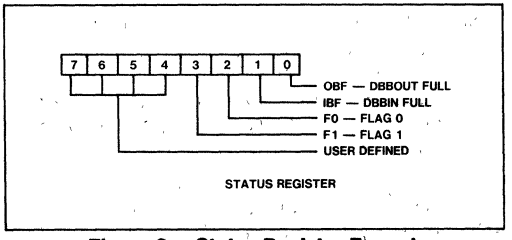


Figure 3. Status Register Format

AFN-01536A

instructions. The UPI should make sure that OBF is reset before writing new data into DBBOUT to ensure that the master has read previous DBBOUT data. IBF should also be tested before reading DBBIN since DBBIN data is valid only when IBF is set. As was mentioned earlier, the UPI uses F_1 to differentiate between command and data contents in DBBIN when IBF is set. The UPI may also write the upper 4-bits of its accumulator to the upper 4-bits of the STATUS register. These bits are thus user definable.

The UPI can test the flags at any time during its internal program execution. It essentially "polls" the STATUS register for changes. If faster response is needed to master commands and data, the UPI's internal interrupt structure can be used. If IBF interrupts are enabled, a master write to DBBIN (either command or data) sets IBF which generates an internal CALL to location 03H in program memory. At this point, working register contents can be saved using bank switching, the accumulator saved in a spare working register, and the DBBIN register read and serviced. The interrupt logic for the IBF interrupt is shown in Figure 4. A few observations concerning this logic are appropriate. Note that if the master writes to DBBIN while the UPI is still servicing the last IBF interrupt (a RETR instruction has not been executed), the IBF Interrupt Pending line is made high which causes a new CALL to 03H as soon as the first RETR is executed. No EN I (Enable Interrupt) instruction is needed to rearm the interrupt logic as is needed in an 8080 or 8085A system; the RETR performs this function. Also note that executing a DIS I to disable further IBF interrupts does not clear a pending interrupt. Only a CALL to location 03H or RESET clears a pending IBF interrupt.

Keeping in mind that the actual master/UPI protocol is dependent on the application, probably the best way to illustrate correct protocol is by example. Let's consider using the UPI as a simple parallel I/O device. (This is a trivial application but it embodies all of the important protocol considerations.) Since the UPI may be either interrupt or non-interrupt driven internally, both cases are considered.

Let's take the easiest configuration first; using the UPI PORT 1 as an 8-bit output port. From the UPI's point-of-view, this is an input-only application since all that is required is that the UPI input data from the master. Once the master writes data to the UPI, the UPI reads the DBBIN register and transfers the data to PORT 1. No testing for commands versus data is needed since the UPI "knows" it only performs one task—no commands are needed.

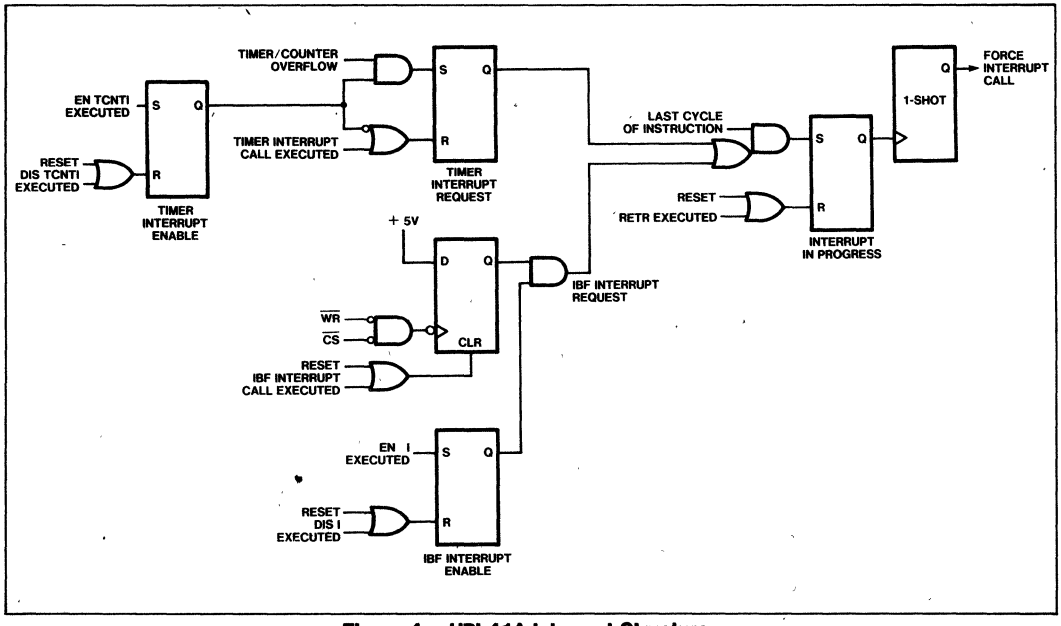


Figure 4. UPI-41A Interrupt Structure

AFN-01536A

Non-interrupt driven UPI software is shown in Figure 5A while Figure 5B shows interrupt based software. For Figure 5A, the UPI simply waits until it sees IBF go high indicating the master has written a data byte to DBBIN. The UPI then reads DBBIN, transfers it to PORT 1, and returns to waiting for the next data. For the interrupt-driven UPI, Figure 5B, once the EN I instruction is executed, the UPI simply waits for the IBF interrupt before handling the data. The UPI could handle other tasks during this waiting time. When the master writes the data to DBBIN, an IBF interrupt is generated which performs a CALL to location 03H. At this point the UPI reads DBBIN (no testing of IBF is needed since an IBF interrupt implies that IBF is set), transfers the data to PORT 1, and executes an RETR which returns program flow to the main program.

Software for the master 8085A is included in Figure 5C. The only requirement for the master to output data to the UPI is that it check the UPI to be sure the previous data had been taken before writing new data. To accomplish this the master simply reads the STATUS register looking for IBF=0 before writing the next data.

```
;
; UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       UPI POLLS IBF FOR DATA
;
RESET:  JNIBF   RESET       ; WAIT ON IBF FOR INPUT
        IN      A,DBB       ; INPUT THERE, SO READ IT
        OUTL    P1,A        ; TRANSFER DATA TO PORT 1
        JMP     RESET       ; GO WAIT FOR NEXT DATA
```

Figure 5A. Single Output Port Example—Polling

```
;
; UPI INPUT ONLY EXAMPLE—PORT 1 USED AS OUTPUT PORT
;       DATA INPUT IS INTERRUPT-DRIVEN ON IBF
;
RESET:  EN      I           ; ENABLE IBF INTERRUPTS
        JMP     RESET+1     ; LOOP WAITING FOR INPUT
IBFINT: IN      A,DBB       ; READ DATA FROM DBBIN
        OUTL    P1,A        ; TRANSFER DATA TO PORT 1
        RETR                ; RETURN WITH RESTORE
```

Figure 5B. Single Output Port Example—Interrupt

```
;
; 8085 SOFTWARE FOR UPI INPUT-ONLY EXAMPLE
;       DATA FOR OUTPUT IS PASSED IN REG. C
;
UPIOUT: IN      STATUS      ; READ UPI STATUS
        ANI     IBF         ; LOOK AT IBF
        JNZ     UPIOUT      ; WAIT FOR IBF=0
        MOV     A,C         ; GET DATA FROM C
        OUT     DBBIN       ; OUTPUT DATA TO DBBIN
        RET                 ; DONE, RETURN
```

Figure 5C. 8085A Code for Single Output Port Example

Figure 6A illustrates the case where UPI PORT 2 is used as an 8-bit input port. This configuration is termed UPI output-only as the master does not write (input) to the UPI but simply reads either the STATUS or the DBBOUT registers. In this example only the OBF flag is used. OBF signals the master that the UPI has placed new port data in DBBOUT. The UPI loops testing OBF. When OBF is clear, the master has read the previous data and UPI then reads its input port (PORT 2) and places this data in DBBOUT. It then waits on OBF until the master reads DBBOUT before reading the input port again. When the master wishes to read the input port data, Figure 6B, it simply checks for OBF being set in the STATUS register before reading DBBOUT. While this technique illustrates proper protocol, it should be noted that it is not meant to be a good method of using the UPI as an input port since the master would never get the newest status of the port.

```
;
; UPI OUTPUT ONLY EXAMPLE—PORT 2 USED AS INPUT PORT
;       PORT DATA IS AVAILABLE IN DBBOUT
;
RESET:  JOBF    RESET       ; LOOP IF OBF=1 (DATA NOT READ)
        IN      A,P2        ; DBBOUT CLEAR, READ PORT
        OUT     DBB,A       ; TRANSFER PORT DATA TO DBBOUT
        JMP     RESET       ; WAIT FOR MASTER TO READ DATA
```

Figure 6A. Single Input Port Example

```
;
; 8085 SOFTWARE FOR UPI OUTPUT—ONLY EXAMPLE
;       INPUT DATA RETURNED IN REG. A
;
UPIIN:  IN      STATUS      ; READ UPI STATUS
        ANI     OBF         ; LOOK AT OBF
        JZ      UPIIN       ; WAIT UNTIL OBF=1
        IN      DBBOUT      ; READ DBBOUT
        RET                 ; RETURN WITH DATA IN A
```

Figure 6B. 8085A Single Input Port Code

The above examples can easily be combined. Figure 7 shows UPI software to use PORT 1 as an output port simultaneously with PORT 2 as an input port. The program starts with the UPI checking IBF to see if the master has written data destined for the output port into DBBIN. If IBF is set, the UPI reads DBBIN and transfers the data to the output port (PORT 1). If IBF is not set or once the data is transferred to the output port if it was, OBF is tested. If OBF is reset (indicating the master has read DBBOUT), the input port (PORT 2) is read and transferred to DBBOUT. If OBF is set, the master has yet to read DBBOUT so the program just loops back to test IBF.

The master software is identical to the separate input/output examples; the master must test IBF

```
;
; UPI INPUT/OUTPUT EXAMPLE—PORT 1OUTPUT, PORT 2 INPUT
;
RESET:  JNIBF   OUT1        ; IF IBF=0, DO OUTPUT
        IN      A, DBB      ; IF IBF=1, READ DBBIN
        OUTL    P1, A       ; TRANSFER DATA TO PORT 1
OUT1:   JOBF    RESET       ; IF OBF=1, GO TEST IBF
        IN      A, P2       ; IF OBF=0, READ PORT 2
        OUT     DBB, A      ; TRANSFER PORT DATA TO DBBOUT
        JMP     RESET       ; GO CHECK FOR INPUT
```

Figure 7. Combination Output/Input Port Example

and OBF before writing output port data into DBBIN or before reading input port from DBBOUT respectively.

In all of the three examples above, the UPI treats information from the master solely as data. There has been no need to check if DBBIN information is a command rather than data since the applications do not require commands. But what if both PORTs 1 and 2 were used as output ports? The UPI needs to know into which port to put the data. Let's use a command to select which port.

Recall that both commands and data pass through DBBIN. The state of the A_0 pin at the time of the write to DBBIN is used to distinguish commands from data. By convention, DBBIN writes with A_0=0 are for data, and those with A_0=1 are commands. When DBBIN is written into, F_1 (FLAG 1) is set to the state of A_0. The UPI tests F_1 to determine if the information in the DBBIN register is data or command.

For the case of two output ports, let's assume that the master selects the desired port with a command prior to writing the data. (We could just use F_1 as a port select but that would not illustrate the subtle differences between commands and data). Let's define the port select commands such that BIT 1=1 if the next data is for PORT 1 (Write PORT 1=0000 0010) and BIT 2=1 if the next data is for PORT 2 (Write PORT 2=0000 0100). (The number of the set bit selects the port.) Any other bits are ignored. This assignment is completely arbitrary; we could use any command structure, but this one has the advantage of being simple.

Note that the UPI must "remember" from DBBIN write to write which port has been selected. Let's use F_0 (FLAG 0) for this purpose. If a Write PORT 1 command is received, F_0 is reset. If the command is Write PORT 2, F_0 is set. When the UPI finds data in DBBIN, F_0 is interrogated and the data is loaded into the previously selected port. The UPI software is shown in Figure 8A.

```
;
; UPI DUAL OUTPUT PORT EXAMPLE—BOTH PORT 1 AND 2 OUTPUTS
;       COMMAND SELECTS DESIRED PORT
;       WRITE PORT 1—0000 0010 (02H)
;       WRITE PORT 2—0000 0100 (04H)
;
;       FLAG 0 USED TO REMEMBER WHICH PORT WAS SELECTED
;       BY LAST COMMAND.
;
RESET:  JNIBF   RESET       ; WAIT FOR MASTER INPUT
        IN      A, DBB      ; READ INPUT
        JF1     CMD         ; IF F1=1, COMMAND INPUT
        JF0     PORT2       ; INPUT IS DATA, TEST F0
        OUTL    P1,A        ; F0=0, SO OUTPUT TO PORT 1
        JMP     RESET       ; WAIT FOR NEXT INPUT
PORT2:  OUTL    P2,A        ; F0=1, SO OUTPUT TO PORT 2
        JMP     RESET       ; WAIT FOR NEXT INPUT
CMD:    JB1     PT1         ; TEST COMMAND BITS (BIT 1)
        JB2     PT2         ; TEST BIT 2
        JMP     RESET       ; NEITHER BIT SET, WAIT FOR INPUT
PT1:    CLR     F0          ; PORT 1 SELECTED, CLEAR F0
        JMP     RESET       ; WAIT FOR INPUT
PT2:    CLR     F0          ; PORT 2 SELECTED, SET F0
        CPL     F0
        JMP     RESET       ; WAIT FOR INPUT
```

Figure 8A. Dual Output Port Example

Initially, the UPI simply waits until IBF is set indicating the master has written into DBBIN. Once IBF is set, DBBIN is read and F_1 is tested for a command. If F_1=1, the DBBIN byte is a command. Assuming a command, BIT 1 is tested to see if the command selected PORT 1. If so, F_0 is cleared and the program returns to wait for the data. If BIT 1=0, BIT 2 is tested. If BIT 2 is set, PORT 2 is selected so F_0 is set. The program then loops back waiting for the next master input. This input is the desired port data. If BIT 2 was not set, F_0 is not changed and no action is taken.

When IBF=1 is again detected, the input is again tested for command or data. Since it is necessarily data, DBBIN is read and F_0 is tested to determine which port was previously selected. The data is then output to that port, following which the program waits for the next input. Note that since F_0 still selects the previous port, the next input could be more data for that port. The port selection command could be thought of as a port select flip-flop control; once a selection is made, data may be repeatedly written to that port until the other port is selected. Master software, Figure 8B, simply must check IBF before writing either a command or data to DBBIN. Otherwise, the master software is straightforward.

For the sake of completeness, UPI software for implementing two input ports is given in Figure 9. This case is simpler than the dual output case since the UPI can assume that all writes to DBBIN are port selection commands so no command/data testing is required. Once the Port Read command is input, the selected port is read and the port data is placed in DBBOUT. Note that in this case F_0 is used as a UPI

AFN-01536A

error indicator. If the master happened to issue an invalid command (a command without either BIT 1 or 2 set), F_0 is set to notify the master that the UPI did not know how to interpret the command. F_0 is also set if the master commanded a port read before it had read DBBOUT from the previous command. The UPI simply tests OBF just prior to loading DBBOUT and if OBF=1, F_0 is set to indicate the error.

All of the above examples are, in themselves, rather trivial applications of the UPI although they could easily be incorporated as one of several tasks in a UPI handling multiple small tasks. We have covered them primarily to introduce the UPI concept and to illustrate some master/UPI protocol. Before moving on to more realistic UPI applications, let's discuss two UPI features that do not directly relate to the master/UPI protocol but greatly enhance the UPI's transfer capability.

In addition to the OBF and IBF bits in the STATUS register, these flags can also be made available directly on two port pins. These port pins can then be used as interrupt sources to the master. By executing an EN FLAGS instruction, PORT 2 pin 4 reflects the condition of OBF and PORT 2 pin 5 reflects the inverted condition of IBF ($\overline{IBF}$). These dedicated outputs can then be enabled or disabled via their respective port bit values; i.e., P_{24} reflects OBF as long as an instruction is executed which sets P_{24} (i.e. ORL P2,#10H). The same action applies to the $\overline{IBF}$ output except P_{25} is used. Thus P_{24} may serve as a DATA AVAILABLE interrupt output. Likewise for P_{25} as a READY-TO-ACCEPT-DATA interrupt. This greatly simplifies interrupt-driven master-slave data transfers.

```
;
; 8085 SOFTWARE FOR DUAL OUTPUT PORT EXAMPLE
;       THIS ROUTINE WRITES DATA IN REG. C TO PORT 1
;       (SAME ROUTINE FOR PORT 2—JUST CHANGE COMMAND)
;
PORT1:  IN      STATUS      ; READ UPI STATUS
        ANI     IBF         ; LOOK AT IBF
        JNZ     PORT1       ; WAIT UNTIL IBF=0
        MVI     A, 00000010B ; LOAD WRITE PORT1 CMD
        OUT     UPICMD      ; OUTPUT TO UPI COMMAND PORT
P1:     IN      STATUS      ; READ UPI STATUS AGAIN
        ANI     IBF         ; LOOK AT IBF
        JNZ     P1          ; WAIT UNTIL COMMAND ACCEPTED
        MOV     A, C        ; GET DATA FROM C
        OUT     DBBIN       ; OUTPUT TO DBBIN
        RET                 ; DONE, RETURN
```

Figure 8B. 8085A Dual Output Port Example Code

The UPI also supports a DMA transfer interface. If an EN DMA instruction is executed, PORT 2 pin 6 becomes a DMA Request (DRQ) output and P_{27} becomes a high impedance DMA Acknowledge

```
;
; UPI DUAL INPUT PORT EXAMPLE—BOTH PORT 1 AND 2 INPUTS
;       COMMAND SELECTS WHICH PORT IS TO BE READ
;       FLAG 0 USED AS ERROR FLAG
;
RESET:  JNIBF   RESET       ; WAIT FOR INPUT
        CLR     F0          ; CLEAR ERROR FLAG
        IN      A, DBB      ; READ INPUT (COMMAND)
        JB1     PT1         ; TEST BIT 1 (PORT 1)
        JB2     PT2         ; TEST BIT 2 (PORT 2)
ERROR:  CPL     F0          ; ERROR—COMPLEMENT F0
        JMP     RESET       ; WAIT FOR INPUT
PT1.    IN      A, P1       ; READ PORT 1
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT 1 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
PT2:    IN      A, P2       ; READ PORT 2
        JOBF    ERROR       ; TEST OBF BEFORE LOADING DBBOUT
        OUT     DBB, A      ; LOAD PORT 2 DATA INTO DBBOUT
        JMP     RESET       ; WAIT FOR INPUT
```

Figure 9. Dual Input Port Example

($\overline{DACK}$) input. Any instruction which would normally set P_{26} now sets DRQ. DRQ is cleared when $\overline{DACK}$ is low and either $\overline{RD}$ or $\overline{WR}$ is low. When $\overline{DACK}$ is low, $\overline{CS}$ and A0 are forced low internally which allows data bus transfers between DBBOUT or DBBIN to occur, depending upon whether $\overline{WR}$ or $\overline{RD}$ is true. Of course, the function requires the use of an external DMA controller.

Now that we have discussed the aspects of the UPI protocol and data transfer interfaces, let's move on to the actual applications.

EXAMPLE APPLICATIONS

Each of the following three sections presents the hardware and software details of a UPI application. Each application utilizes one of the protocols mentioned in the last section. The first example is a simple 8-digit LED display controller. This application requires only that the UPI perform input operations from the DBBIN; DBBOUT is not used. The reverse is true for the second application: a sensor matrix controller. The final application involves both DBBOUT and DBBIN operations: a combination serial/parallel I/O device.

The core master processor system with which these applications were developed is the iSBC 80/30 single board computer. This board provides an especially convenient UPI environment since it contains a dedicated socket specifically interfaced for the UPI-41A. The 80/30 uses the 8085A as the master processor. The I/O and peripheral complement on the 80/30 include 12 vectored priority interrupts (8 on an 8259 Programmable Interrupt Controller and 4 on the 8085A itself), an 8253 Programmable Interval Timer supplying three 16-bit programmable timers (one is dedicated as a programmable baud rate generator), a high speed serial channel provided by a 8251 Programmable USART, and 24 parallel I/O

AFN-01536A

lines implemented with an 8255A Programmable Parallel Interface. The memory complement contains 16K bytes of RAM using 2117 16K bit Dynamic RAMs and the 8202 Dynamic RAM Controller, and up to 8K bytes of ROM/EPROM with sockets compatible with 2716, 2758, or 2332 devices. The 80/30's RAM uses a dual port architecture. That is, the memory can be considered a global system resource, accessible from the on-board 8085A as well as from remote CPUs and other devices via the MULTIBUS. The 80/30 contains MULTIBUS control logic which allows up to 16 80/30s or other bus masters to share the same system bus. (More detailed information on the iSBC 80/30 and other iSBC products may be found in the latest Intel *Systems Data Catalog.*)

A block diagram of the iSBC 80/30 is shown in Figure 10. Details of the UPI interface are shown in Figure 11. This interface decodes the UPI registers in the following format:

Register	Operations
Read STATUS	IN E5H
Write DBBIN (command)	OUT E5H
Read DBBOUT (data)	IN E4H
Write DBBIN (data)	OUT E4H

8-Digit Multiplexed LED Display

The traditional method of interfacing an LED display with a microprocessor is to use a data latch along with a BDC-to-7-segment decoder for each digit of the display. Thus two ICs, seven current limiting resistors, and about 45 connections are required for each digit. These requirements are, of course, multiplied by the total number of digits desired. The obvious disadvantages of this method are high parts count and high power dissipation since each digit is "ON" continuously. Instead, a scheme of time multiplexing the display can be used to decrease both parts count and power dissipation.

Display multiplexing basically involves connecting the same segment (a, b, c, d, e, f, or g) of each digit in parallel and driving the common digit element (anode or cathode) of each digit separately. This is shown schematically in Figure 12. The various digits of the display are not all on at once; rather, only one digit at a time is energized. As each digit is energized, the appropriate segments for that digit are turned on. Each digit is enabled in this way, in sequence, at a rate fast enough to ensure that each digit appears to be "ON" continuously. This implies that the display must be "refreshed" at periodic intervals to keep the digits flicker-free. If the CPU had to handle this task, it would have to suspend normal processing, go update the display, and then return to its normal flow. This extra burden is ideally handled by a UPI. The master CPU could simply give characters to the UPI and let the UPI do the actual segment decoding, display multiplexing, and refreshing.

As an example of this technique, Figure 13 shows the UPI controlling an 8-digit LED display. All digit segments are connected in parallel and are driven through segment drivers by the UPI PORT 1. The lower 3 bits of PORT 2 are inputs to a 3-to-8 decoder which selects an individual digit through a digit driver. A fourth PORT 2 line is used as a decoder enable input. The remaining PORT 2 lines plus the TEST 0 and TEST 1 inputs are available for other tasks.

Internally, the UPI uses the counter/timer in the interval timer mode to define the interval between display refreshes. Once the timer is loaded with the desired interval and started, the UPI is free to handle other tasks. It is only when a timer overflow interrupt occurs that the UPI handles the short display multiplexing routine. The display multiplexing can be considered a background task which is entirely interrupt-driven. The amount of time spent multiplexing is such that there is ample time to handle a non-timer task in the UPI foreground. (We'll discuss this timing shortly.)

When a timer interrupt occurs, the UPI turns off all digits via the decoder enable. The next digit's segment contents are retrieved from the internal data memory and output via PORT 1 to the segment drivers. Finally, the next digit's location is placed on PORT 2 (P_{20}–P_{22}) and the decoder enabled. This displays the digit's segment information until the next interrupt. The timer is then restarted for the next interval. This process continues repeatedly for each digit in sequence.

As a prelude to discussing the UPI software, let's examine the internal data memory structure used in this application, Figure 14. This application requires only 14 of the 64 total data memory locations. The top eight locations are dedicated to the Display Map; one location for each digit. These locations contain the segment and decimal point information for each character. Just how characters are loaded into this section of memory is covered shortly. Register R_7 of Register Bank 1 is used as the temporary Accumulator store during the interrupt service routines. Register R_3 stores the digit number of the next digit to be displayed. R_2 is a temporary storage register for characters during input routine. R_0 is

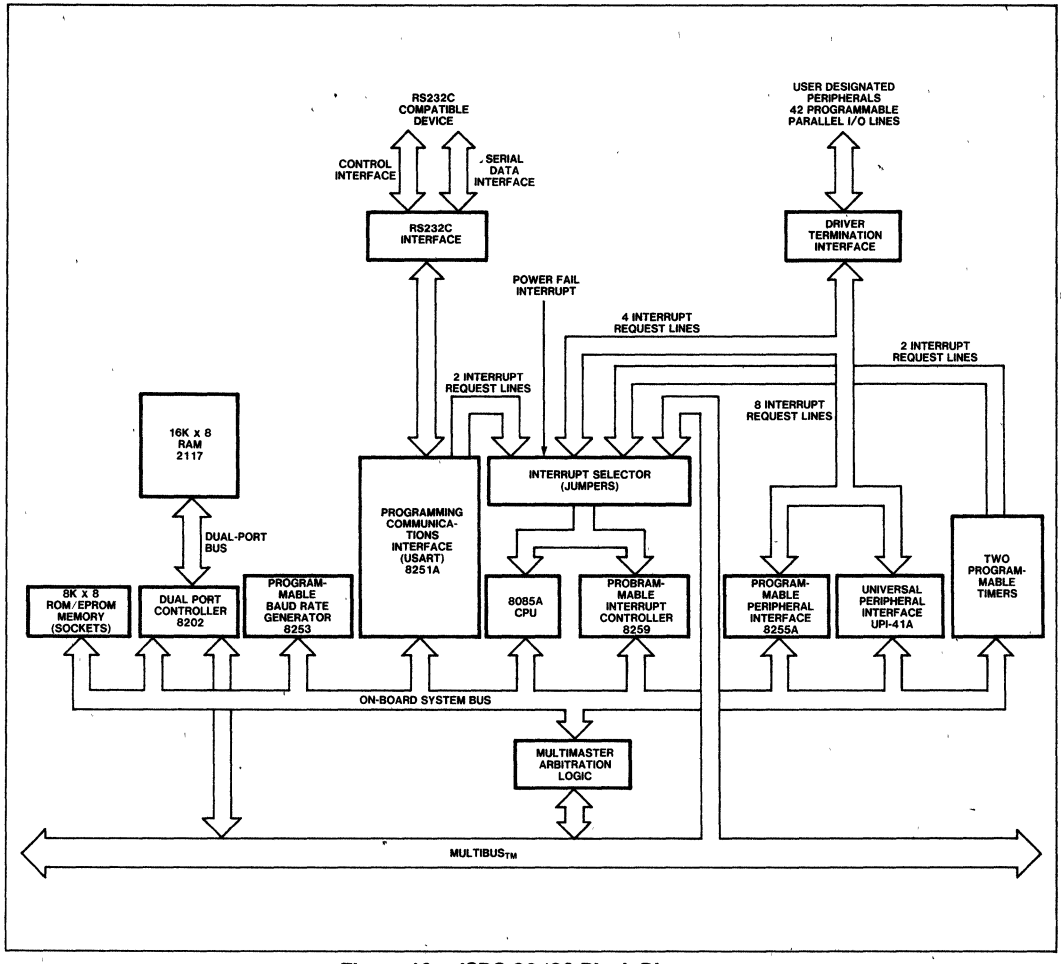


Figure 10. iSBC 80/30 Block Diagram

the offset pointer pointing to the Display Map location of the next digit. That makes 12 locations so far. The remaining two locations are the two stack locations required to store the return address plus status during the timer and input interrupt service routines. The remaining unused locations, all of Register Bank 0, 14 bytes of stack, 4 in Register Bank 1, and 24 general purpose RAM locations, are all available for use by any foreground task.

The UPI software consists of only three short routines. One, INIT, is used strictly during initialization. DISPLA is the multiplexing routine called at a timer interrupt. INPUT is the character input handler called at an IBF interrupt. The flow charts for these routines are shown in Figures 14A through 14C.

INIT initializes the UPI by simply turning off all segment and digit drivers, filling the Display Map with blank characters, loading and starting the timer, and enabling both timer and IBF interrupts. Although the flow chart shows the program looping at this point, it is here that the code for any foreground task is inserted. The only restrictions on this foreground task are that it not use I/O lines dedicated to the display and that it not require dedicated use of the timer. It could share the timer if precautions are taken to ensure that the display will still be refreshed at the required interval.

AFN-01536A

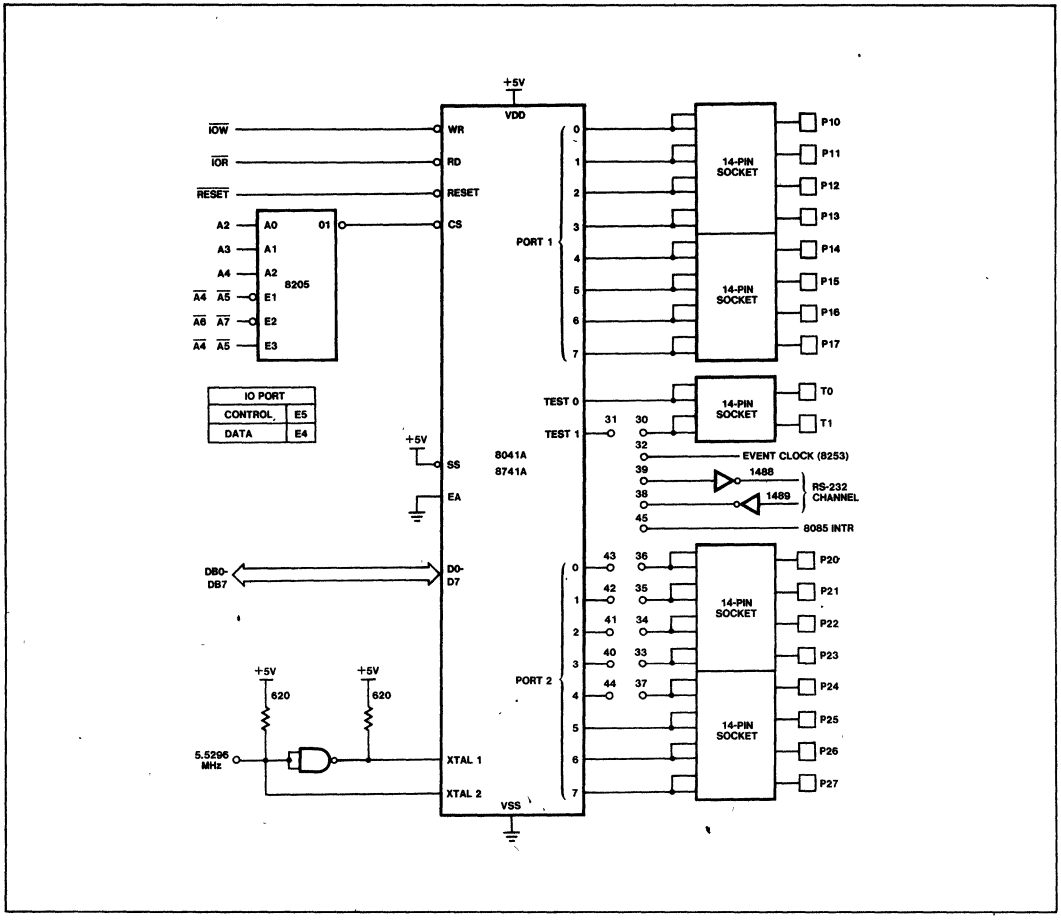


Figure 11. UPI Interface on iSBC 80/30

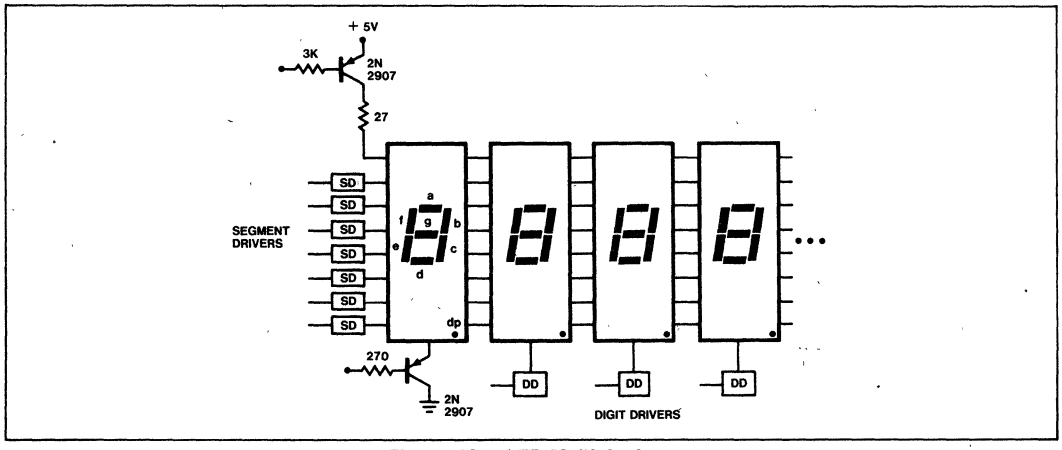


Figure 12. LED Multiplexing

AFN-01536A

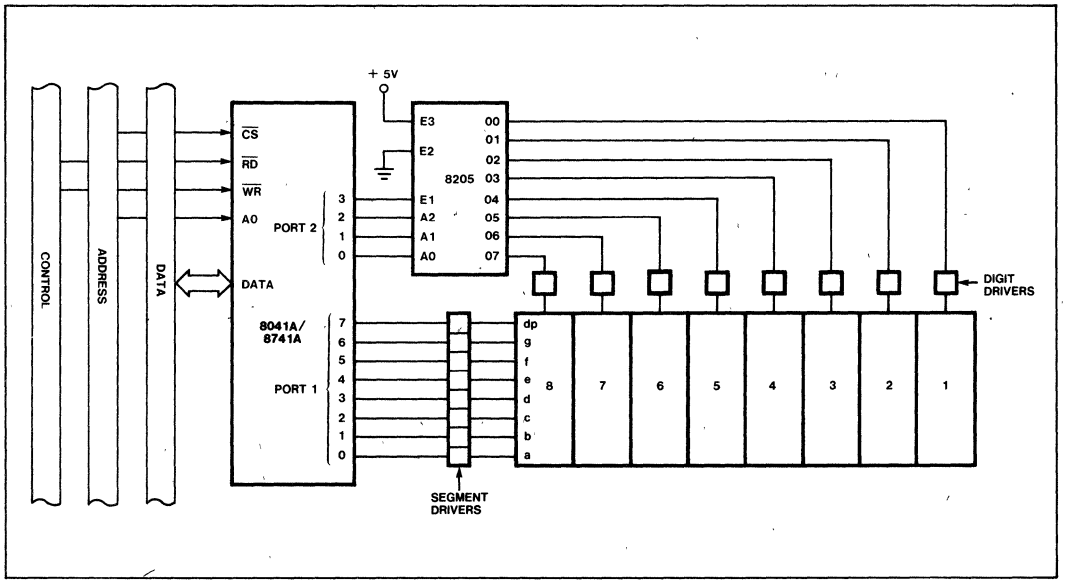


Figure 13. UPI Controlled 8-Digit LED Display

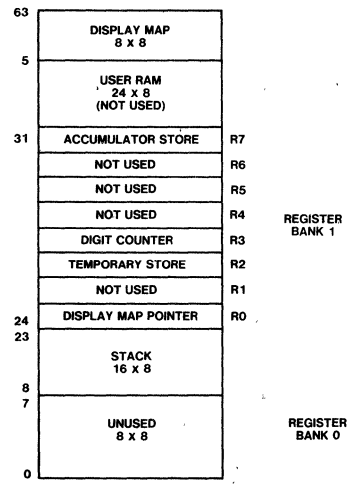


Figure 14. LED Display Controller Data Memory Allocation

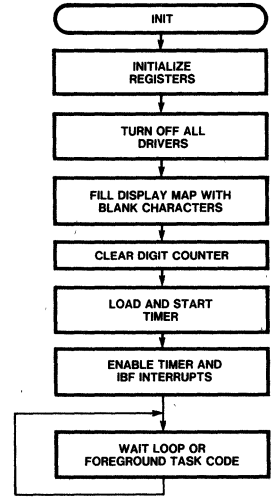


Figure 14A. INIT Routine Flow

AFN-01536A

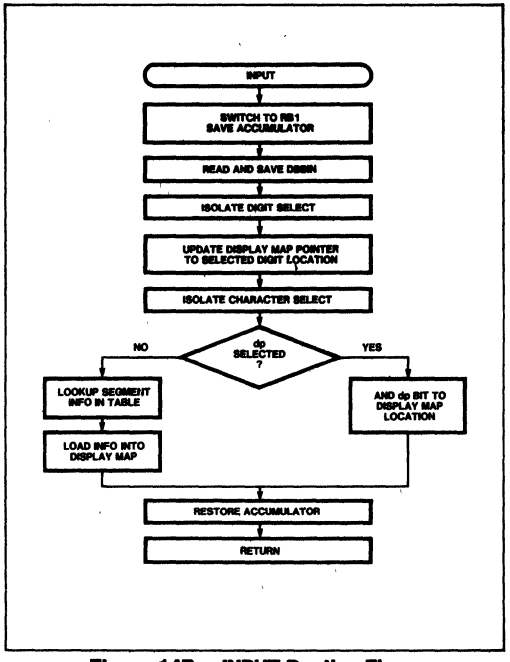


Figure 14B. INPUT Routine Flow

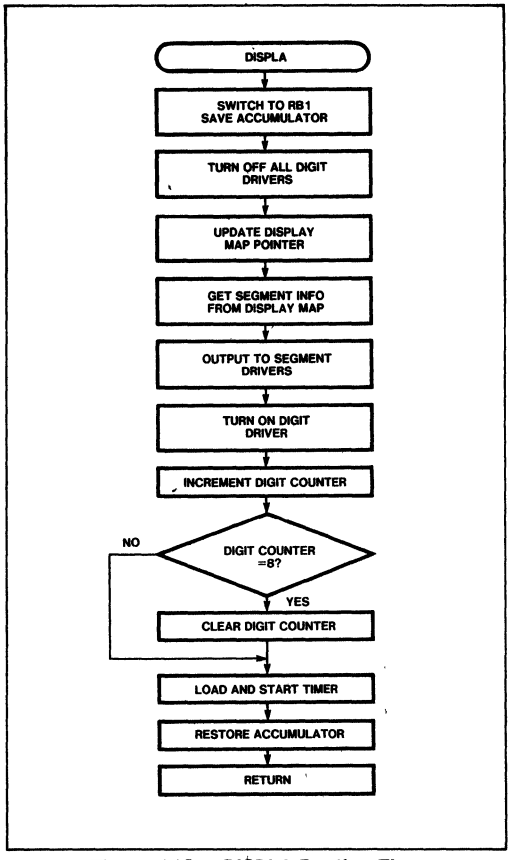


Figure 14C. DISPLA Routine Flow

The INPUT routine handles the character input. It is called when an IBF interrupt occurs. After the usual swapping of register banks and saving of the accumulator, DBBIN is read and stored in register R_2. DBBIN contains the Display Data Word. The format for this word, Figure 15, has two fields: Digit Select and Character Select. The Digit Select field selects the digit number into which the character from the Character Select field is placed. Notice that the character set is not limited strictly to numerics, some alphanumeric capability is provided. Once DBBIN is read, the offset for the selected digit is computed and placed in the Display Map Pointer R_0. Next the segment information for the selected character is found through a look-up table starting in page 3 of the program memory. This segment information is then stored at the location pointed at by the Display Map Pointer. If the Character Select field specified a decimal point, the segment corresponding to the decimal point is ANDed into the present segment information for that digit. After the accumulator is restored, execution is returned to the main program.

The DISPLA routine simply implements the multiplexing actions described earlier. It is called whenever a timer interrupt occurs. After saving pre-interrupt status by switching register banks and storing the Accumulator, all digit drivers are turned off. The Display Map Pointer is then updated using the Current Digit Register to point at that digit's segment information in the Display Map. This information is output to PORT 1; the segment drivers. The number of the current digit, R_3, is then sent to the digit select decoder and the decoder is enabled. This turns on the current digit. The digit counter is incremented and tested to see if all eight digits have been refreshed. If so, the digit counter is reset to zero. If not, nothing is done. Finally, the timer is loaded and restarted, the Accumulator is restored, and the routine returns execution to the main program. Thus DISPLA refreshes one digit each time it is CALLed by the timer interrupt. The digit remains on until the next time DISPLA is executed.

The UPI software listing is included as Appendix A1. Appendix A2 shows the 8085A test routine used

AFN-01536A

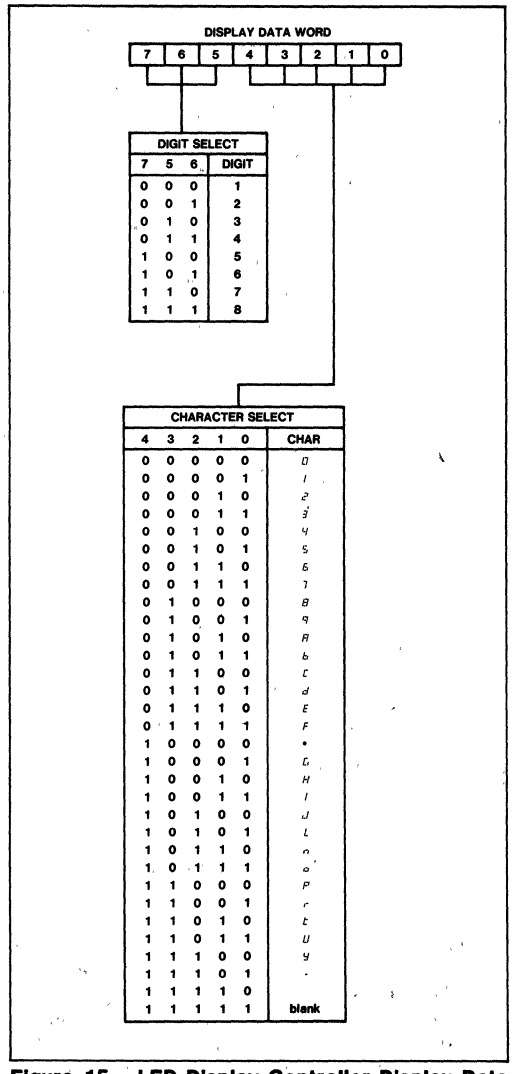


Figure 15. LED Display Controller Display Data Word Format

to display the contents of a display buffer on the display. The 8085A software takes care of the display digit numbering. Since the application is input-only for the UPI, the only protocol required is that the master must test IBF before writing a Display Data Word into DBBIN.

On the iSBC 80/30, the UPI frequency is at 5.5296 MHz. To obtain a flicker-free display, the whole display must be refreshed at a rate of 50 Hz or greater. If we assume a 50 Hz refresh rate and an 8-digit display, this means the DISPLA routine must be CALLed 50×8 or 400 times/sec. This transfers, using the timer interval of 87 μs at 5.5296 MHz, to a timer count of 227. (Recall from the UPI-41A *User's Manual* that the timer is an "8-bit up-counter".) Hence the TIME equate of 227D in the UPI listing. Obviously, different frequency sources or display lengths would require that this equate be modified.

With the UPI running at 5.5296 MHz, the instruction cycle time is 2.713 μs. The DISPLA routine requires 28 instruction cycles, therefore, the routine executes in 76 μs. Since DISPLA is CALLed 400 times/sec, the total time spent refreshing the display during one second is then 30 ms or 3% of the total UPI time. This leaves 97.0% for any foreground tasks that could be added.

While the basic UPI software is useful just as it stands, there are several enhancements that could be incorporated depending on the application. Auto-incrementing of the digit location could be added to the input routine to alleviate the need for the master to keep track of digit numbers. This could be (optionally) either right-handed or left-handed entry a la TI or HP calculators. The character set could be easily modified by simply changing the lookup table. The display could be expanded to 16 digits at the expense of one additional PORT 2 digit select line, the replacement of the 3-to-8 decoder with a 4-to-16 decoder, and 8 more Display Map locations.

Now let's move on to a slightly more complex application that is UPI output-only—a sensor matrix controller.

Sensor Matrix Controller

Quite often a microprocessor system is called upon to read the status of a large number of simple SPST switches or sensors. This is especially true in a process or industrial control environment. Alarm systems are also good examples of systems with a large sensor population. If the number of sensors is small, it might be reasonable to dedicate a single input port pin for each sensor. However, as the number of sensors increase, this technique becomes very wasteful. A better arrangement is to configure the sensors in a matrix organization like that shown in Figure 16. This arrangement of 16 sensors requires only 4 input and 4 output lines; half the number needed if dedicated inputs were used. The line saving becomes even more substantial as the number of sensors increases.

AFN-01536A

In Figure 16, the basic operation of the matrix involves scanning individual row select lines in sequence while reading the column return lines. The state of any particular sensor can then be determined by decoding the row and column information. The typical configuration pulls up the column return lines and the selected row is held low. Deselected rows are held high. Thus a return line remains high for an open sensor on the selected row and is pulled low for a closed sensor. Diode isolation is used to prevent a phantom closure which would occur when a sensor is closed on a selected row and there are two or more closures on a deselected row. Germanium diodes are used to provide greater noise margin at the return line input.

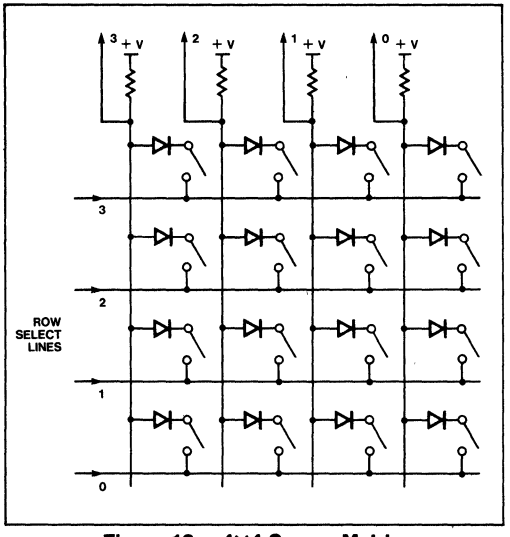


Figure 16. 4X4 Sensor Matrix

If the main processor was required to control such a matrix it would periodically have to output at the row port and then read the column return port. The processor would need to maintain in memory a map of the previous state of the matrix. A comparison of the new return information to the old information would then be made to determine whether a sensor change had occurred. Any changes would be processed as needed. A row counter and matrix map pointer also require maintenance each scan. Since in most applications sensors change very slowly compared to most processing actions, the processor probably would scan the rows only periodically with other tasks being processed between scans.

Rather than require the processor to handle the rather mundane tasks of scanning, comparing, and decoding the matrix, why not use a dedicated processor? The UPI is perfect.

Figure 17 shows a UPI configuration for controlling up to 128 sensors arranged in a 16X8 matrix. The 4-to-16 line decoder is used as the row selector to save port pins and provides the expansion to 128 sensors over the maximum of 64 sensors if the port had been used directly. It also helps increase the port drive capability. The column return lines go directly into PORT 1. Features of this design include complete matrix management. As the UPI scans the matrix it compares its present status to the previous scan. If any change is detected, the location of the change is decoded and loaded, along with the sensor's present state, into DBBOUT. This byte is called a Change Word. The Master processor has only to read one byte to determine the status and coordinate of a changed sensor. If the master had not read a previous Change Word in DBBOUT (OBF=1) before a new sensor change is detected, the new Change

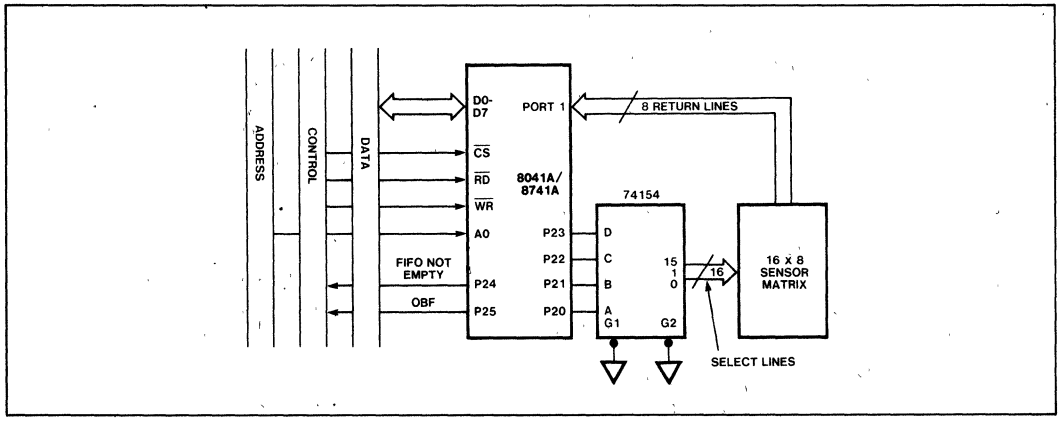


Figure 17. 128 Sensor Matrix Controller

AFN-01536A

Word is loaded into an internal FIFO. This FIFO buffers up to 40 changes before it fills. The status of the FIFO and OBF is made available to the master either by polling the UPI STATUS register, Figure 18A, or as interrupt sources on port pins P_{24} and P_{25} respectively, Figure 17. The FIFO NOT EMPTY pin and bit are true as long as there are changes not yet read in the FIFO. As long as the FIFO is not empty, the UPI monitors OBF and loads new Change Words from the FIFO into DBBOUT. Thus, the UPI provides complete FIFO management.

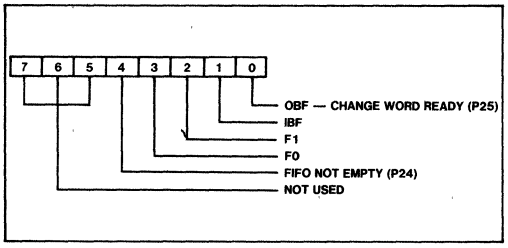


Figure 18A. Sensor Matrix Status Register Format

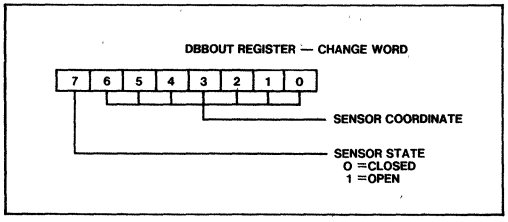


Figure 18B. Sensor Matrix Change Word Format

Internally, the matrix scanning software is programmed to run as a foreground task. This allows the timer/counter to be used by any background task although the hardware configuration leaves only 2 inputs (TEST 0 and TEST 1) plus 2 I/O port pins available. Also, to add a background task, the FIFO would have to be made smaller to accommodate the needed register and data memory space. (It would be possible however to turn the table here and make the scanning software timer/counter interrupt-driven where the timer times the scan interval.)

The data memory organization for this application is shown in Figure 19. The upper 16 bytes form the Matrix Map and store the sensor states from the previous scan; one bit for each sensor. The Change Word FIFO occupies the next 40 locations. (The top and bottom addresses of this FIFO are treated as equate variables in the program so that the FIFO size may easily be changed to accommodate the register needs of other tasks.) Register R_0 serves as a pointer into the matrix map area for comparisons and updates of the sensor status. R_1 is a general FIFO pointer. The FIFO is implemented as a circular buffer with In and Out pointer registers which are stored in R_4 and R_5 respectively. These registers are moved into FIFO pointer R_1 for actual transfers into or out of the FIFO. R_2 is the Row Select Counter. It stores the number of the row being scanned.

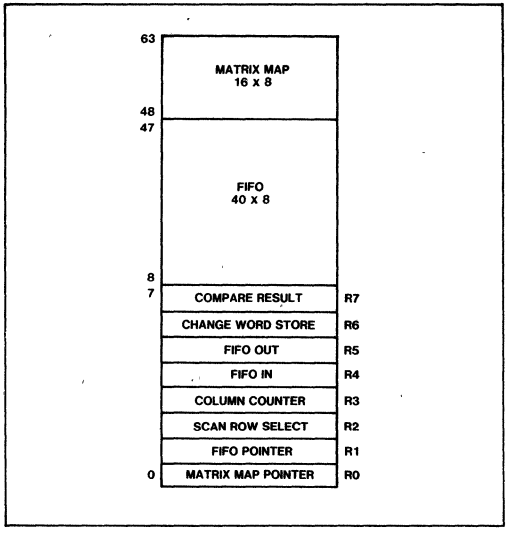


Figure 19. Sensor Matrix Data Memory Map

Register R_3 is the Column Counter. This counter is normally set to 00H; however, when a change is detected somewhere in a particular row, it is used to inspect each sensor status bit individually for a change. When a changed counter sensor bit is found, the Row Select Counter and Column Counter are combined to give the sensor's matrix coordinate. This coordinate is temporarily stored in the Change Word Store, register R_6. Register R_7 is the Compare Result. As each row is scanned, the return information is Exclusive-OR'd with the return information from the previous scan of that row. The result of this operation is stored in R_7. If R_7 is zero, there have been no changes on that row. A non-zero result indicates at least one changed sensor.

The basic program operation is shown in the flow chart of Figure 20. At RESET, the software initializes the working registers, the ports, and clears the STATUS register. To get a starting point from which to perform the sensor comparisons, the current status of the matrix is read and stored in the Matrix Map. At this point, the UPI begins looking for changed sensors starting with the first row.

AFN-01536A

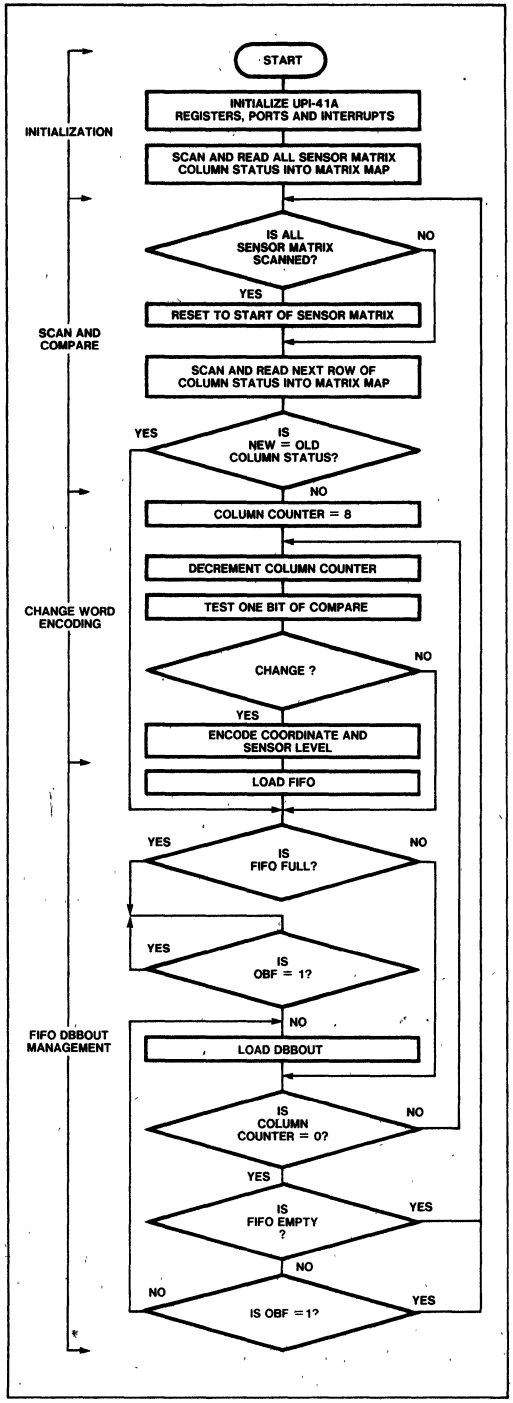


Figure 20. Sensor Matrix Controller Flow Chart

Before delving further into the flow, let's pause to describe the general format of the operation. The UPI scans the matrix one row at a time. If no changes are detected on a particular row, the UPI simply moves to the next row after checking the status of DBBOUT and the FIFO. If a change is detected, the UPI must check each bit (sensor) within the row to determine the actual sensor location. (More than one sensor on the scanned row could have changed.) Rather than test all 8 bits of the row before checking the DBBOUT and FIFO status again, the UPI performs the status check in between each of the bit tests. This ensures the fastest response to the master reading previous Change Words from DBBOUT and the FIFO.

With this general overview in mind, let's go first thru the flow chart assuming we are scanning a row where no changes have occurred. Starting at the Scan-and-Compare section, the UPI first checks if the entire matrix has been scanned. If it has, the various pointers are reset. If not, the address of the next row is placed on PORTs 20 thru 23. This selects the desired row. The state of the row is then read on PORT 1; the column return lines. This present state is compared to the previous state by retrieving the previous state from the matrix map and performing an Exclusive-OR with the present state. Since we are assuming that no change has occurred, the result is zero. No coordinate decoding is needed and the flow branches to the FIFO-DBBOUT Management section.

The FIFO-DBBOUT Management section simply maintains the FIFO and loads DBBOUT whenever Change Words are present in the FIFO and DBBOUT is clear (OBF=0). The section first tests if the FIFO is full. (If we assume our "no-change" row is the first row scanned, the FIFO obviously would not be full.) If it is, the UPI waits until OBF=0, at which point the next Change Word is retrieved from the FIFO and placed in DBBOUT. This "unfills" the FIFO making room for more Change Words. At this point, the Column Counter, R_3, is checked. For rows with no changes, the Column Counter is always zero so the test simply falls through. (We cover the case for changes shortly.) Now the FIFO is tested for being empty. If it is, there is no sense in any further tests so the flow simply goes back up to scan the next row. If the FIFO is not empty, DBBOUT is tested again through OBF. If a Change Word is in DBBOUT waiting for the master to read it, nothing can be done and the flow likewise branches up for the next row. However, if the DBBOUT is free and remembering that the previous test showed that the FIFO was not empty, DBBOUT is loaded with the next Change Word and the last two conditional tests repeat.

AFN-01536A

Now let's assume the next row contains several changed sensors. Like before, the row is selected, the return lines read, and the sensor status compared to the previous scan. Since changes have occurred, the Exclusive-OR result is now non-zero. Any 1's in the result reflect the positions of the changed sensors. This non-zero result is stored in the Compare Result register, R_7. At this point, the Column Counter is preset to 8. To determine the changed sensors' locations, the Compare Result register is shifted bit-by-bit to the left while decrementing the Column Counter. After each shift, BIT 7 of the result is tested. If it is a one, a changed sensor has been found. The Column Counter then reflected the sensor's matrix column position while the Scan Row Select register holds it row position. These registers are then combined in R_6, the Change Word Store, to form the sensor's matrix coordinate section of the Change Word. The 8th bit of the Change Word Store is coded with the sensor's present state (Figure 18). This byte forms the complete Change Word. It is loaded into the next available FIFO position. If BIT 7 of the Compare Result had been zero, that particular sensor had not changed and the coordinate decoding is not performed.

In between each shift, test, and coordinate encode (if necessary), the FIFO-DBBOUT Management is performed. It is the Column Counter test within this section that routes the flow back up to the Change Word Encoding section if the entire Compare Result (row) has not been shifted and tested.

The FIFO is implemented as a circular buffer with IN and OUT pointers (R_4 and R_5 respectively). The operations of the FIFO is best understood using an example, Figure 21. This series of figures show how the FIFO, DBBOUT, and OBF interact as changes are detected and Change Words are read by the master. The letters correspond to sequential Change Words being loaded into the FIFO. Note that the figures show only a 4X8 FIFO however, the principles are the same in the 40X8 FIFO.

Figure 21A shows the condition where no Change Words have been loaded into the FIFO or DBBOUT. In Figure 21B a change, "A", has been detected, decoded, and loaded into the FIFO at the location equal to the value of the FIFO-IN pointer. The FIFO-OUT pointer is reset to the bottom of the FIFO since it had reached the FIFO top. Now that a Change Word is in the FIFO, OBF is checked to see if DBBOUT is empty. Because OBF=0, DBBOUT is empty and the Change Word is loaded from the FIFO location pointed at by the FIFO-OUT pointer. This is shown in Figure 21C. Loading DBBOUT automatically sets OBF. OBF remains set until the master reads DBBOUT. Figures 21D and 21E show two more Change Words loaded into the FIFO. In Figure 21F the first Change Word is finally read by the master resetting OBF. This allows the next Change Word to be loaded into DBBOUT. Note that each time the FIFO is loaded, the FIFO-IN pointer increments. Each time DBBOUT is read the FIFO-OUT pointer increments unless there are no more Change Words in the FIFO. Both pointers wrap-around to the bottom once they reach the FIFO top. The remaining figures show more Change Words being loaded into the FIFO. When the entire FIFO fills and DBBOUT can not be loaded (OBF=1), scanning stops until the master reads DBBOUT making room for more Change Words.

As was mentioned earlier, two interrupt outputs to the master are available: Change Word Ready (P_{25}, OBF) and FIFO NOT EMPTY (P_{24}). The Change Word Ready interrupt simply reflects OBF and is handled automatically by the UPI since an EN FLAGS instruction is executed during initialization. The FIFO NOT EMPTY interrupt is generated and cleared as appropriate, each pass through the FIFO management code.

No debouncing is provided although it could be added. Rather, the scan time is left as an equate variable so that it could be varied to account for both debounce time and expected sensor change rates. The minimum scan time for this application is 2msec when using a 6MHz clock. Since the matrix controller is coded as a foreground task, scan time simply uses a software delay loop.

The UPI software is included as Appendix B1. Appendix B2 is 8085A test software which builds a Change Word buffer starting at BUFSRT. This software simply polls the STATUS register looking for Change Word Ready to go true. DBBOUT is then read and loaded into the buffer. Now let's move on to an application which combines both the foreground and background concepts.

Combination I/O Device

The final UPI application was designed especially to add additional serial and parallel I/O ports to the iSBC 80/30. This UPI simulates a full-duplex UART (Universal Asynchronous Receiver/Transmitter) combined with an 8-bit parallel I/O port. Features of the UART include: software selectable baud rates (110, 300, 600, or 1200 baud), double buffering for both the transmitter and receiver, and receiver testing for false start bit, framing, and overrun errors. For parallel I/O, one 8-bit port is programmable for either input or output. The output port is statically latched and the input port is sampled.

AFN-01538A

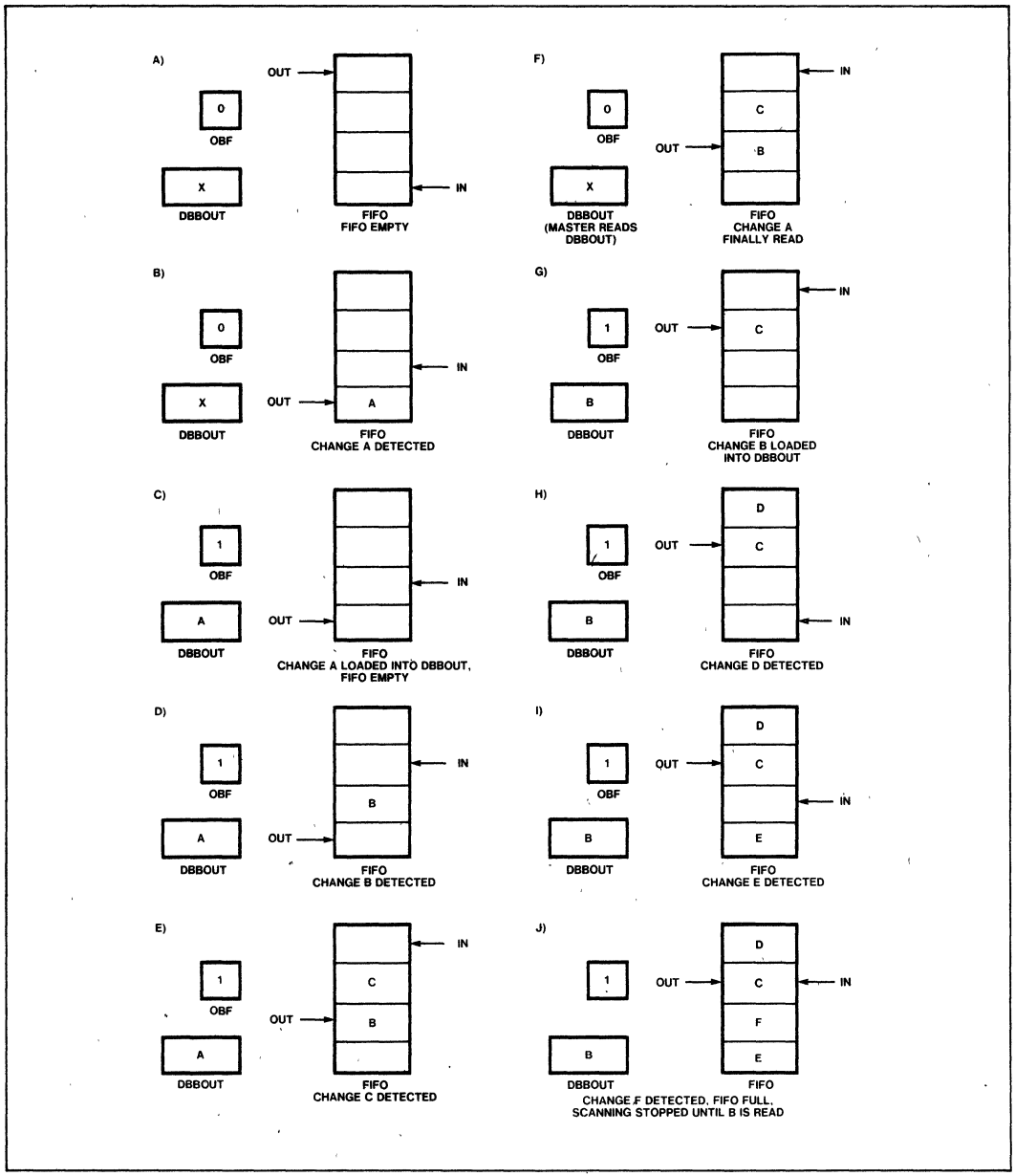


Figure 21A-J. FIFO Operation Example

AFN-01536A

Figure 22 shows the interface of this combination I/O device to the dedicated UPI socket on the iSBC 80/30. The only external requirement is a 76.8 kHz source which serves as the baud rate standard. The internal baud rates are generated as multiples of this external clock. This clock is obtained from one of the 8253 counters. Otherwise, an RS-232 driver and receiver already available for UPI use in serial I/O applications. Sockets are also provided for termination of the parallel port.

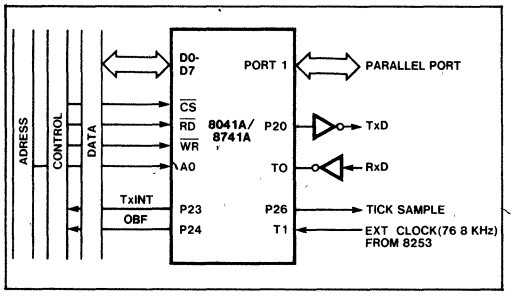


Figure 22. Combination I/O Device

There are three commands for this application. Their format is shown in Figure 23. The CONFIGURE command specifies the serial baud rate and the parallel I/O direction. Normally this command is issued once during system initialization. The I/O command causes a parallel I/O operation to be performed. If the parallel port direction is out, the UPI expects the data byte immediately following an I/O command to be data for the output port. If the port is in the input direction, an I/O command causes the port to be read and the data placed in DBBOUT. The RESET ERROR command resets the serial receiver error bits in the STATUS register.

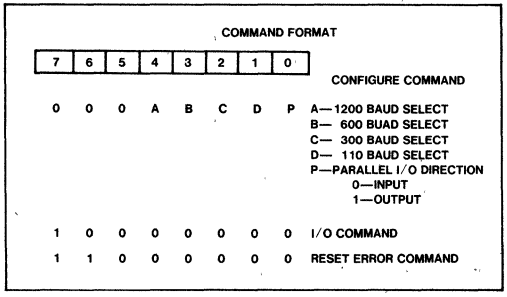


Figure 23. Combination I/O Command Format

The STATUS register format is shown in Figure 24. Looking at each bit, BIT 0 (OBF) is the DATA AVAILABLE flag. It is set whenever the UPI places data into DBBOUT. Since the data may come from either the receiver or the parallel input port, the F_0 and F_1 flags (BITs 2 and 3) code the source. Thus, when the master finds OBF set, it must decode F_0 and F_1 to determine the source.

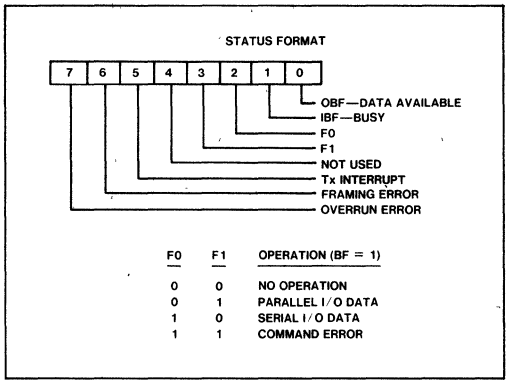


Figure 24. STATUS Register Format

BIT 1 (IBF) functions as a busy bit. When IBF is set, no writes to DBBIN are allowed. BIT 5 is the TxINT (Transmitter Interrupt) bit. It is asserted whenever the transmitter buffer register is empty. The master uses this bit to determine when the transmitter is ready to accept a data character.

BITS 6 and 7 are receiver error flags. The framing error flag, BIT 6, is set whenever a character is received with an invalid stop bit. BIT 7, overrun error, is set if a character is received before the master has read a previous character. If an overrun occurs, the previous character is overwritten and lost. Once an error occurs, the error flag remains set until reset by a RESET ERROR command. A set error flag does not inhibit receiver operation however.

Figure 25 shows the port pin definition for this application. PORT 1 is the parallel I/O port. The UART uses PORT 2 and the Test inputs. P_{20} is the transmitter data out pin. It is set for a mark and reset for a space. P_{23} is a transmitter interrupt output. This pin has the same timing as the TxINT bit in the STATUS register. It is normally used in interrupt-driven systems to interrupt the master processor when the transmitter is ready to accept a new data character.

The OBF flag is brought out on P_{24} as a master interrupt when data is available in DBBOUT. P_{26} is a diagnostic pin which pulses at four times the selected baud rate. (More about this pin later.) The receiver data input uses the TEST 0 input. One of the PORT 2 pins could have been used, however, the

AFN-01536A

PORT PIN DEFINITION

PORT	BIT	FUNCTION
1	0-7	PARALLEL I/O
2	0	Tx Data
	1	NOT USED
	2	NOT USED
	3	Tx INTERRUPT
	4	OBF INTERRUPT
	5	NOT USED
	6	NOT USED (TICK SAMPLE)
	7	NOT USED
T0		Rx DATA
T1		EXTERNAL CLOCK (76.8 kHz)

Figure 25. Combination I/O Port Definition

software can test the TEST 0 in one instruction without first reading a port.

The TEST 1 input is the baud rate external source. The UART divides this input to determine the timing needed for the selected baud rate. The input is a non-synchronous 76.8 kHz source.

Internally, when the CONFIGURE command is received and the selected baud rate is determined, the internal timer/counter is loaded with a baud rate constant and started in the event counter mode. Timer/counter interrupts are then enabled. The baud rate constant is selected to provide a counter interrupt at four times the desired baud rate. At each interrupt, both the transmitter and receiver are handled. Between interrupts, any new commands and data are recognized and executed.

As a prelude to discussing the flow charts, Figure 26 shows the register definition. Register Bank 0 serves the UART receiver and parallel I/O while Register Bank 1 handles the UART transmitter and commands. Looking at RB0 first, R_3 is the receiver status register, RxSTS. Reflected in the bits of this register is the current receiver status in sequential order. Figure 27 shows this bit definition. BIT 0 is the Rx flag. It is set whenever a possible start bit is received. BIT 1 signifies that the start bit is good and character construction should begin with the next received bit. BIT 1 is the Good Start flag. BIT 2 is the Byte Finished flag. When all data bits of a character are received, this flag is set. When all the bits, data and stop bits are received, the assembled character is loaded into the holding register (R_4 in Figure 27) BIT 3, the Data Ready flag, is set. The foreground routine which looks for commands and data continuously, looks at this bit to determine when the receiver has received a character. BITS 4 and 5 signify any error conditions for a particular character.

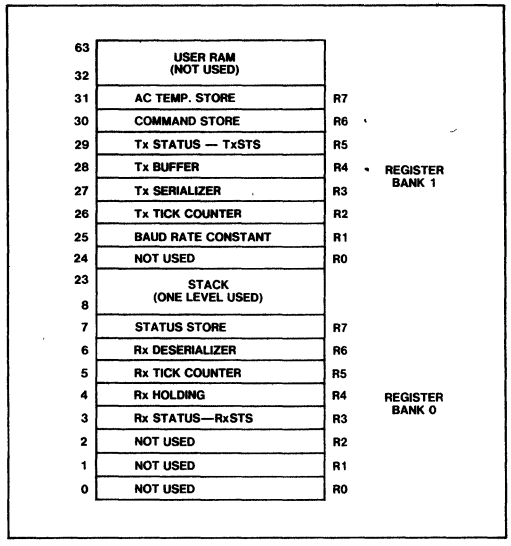


Figure 26. Combination I/O Register Map

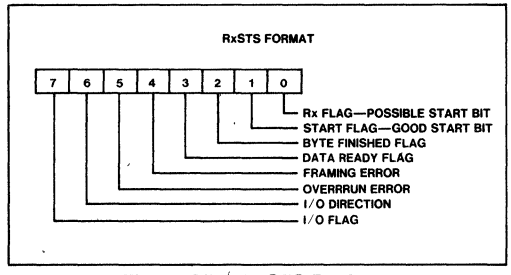


Figure 27. RxSTS Register

The parallel I/O port software uses BITS 6 and 7. BIT 6 codes the I/O direction specified by the last CONFIGURE command. BIT 7 is set whenever an I/O command is received. The foreground routine tests this bit to determine when an I/O operation has been requested by the master.

As was mentioned, R_4 is the receiver holding register. Assembled characters are held in this register until the foreground routine finds DBBOUT free, at which time the data is transferred from R_4 to DBBOUT. R_5 is the receiver tick counter. Recall that counter interrupts occur at four times the baud rate. Therefore, once a start bit is found, the receiver only needs to look at the data every four interrrupts or tick counts. R_5 holds the current tick count.

R_6 is the receiver de-serializing register. Data characters are assembled in this register. R_6 is preset to 80H when a good start bit is received. As each bit is

AFN-01536A

sampled every four timer ticks, they are rotated into the leftmost bit of R_6. The software knows the character assembly is complete when the original preset bit rotates into the carry.

An image of the upper 4 bits of the STATUS register is stored in R_7. These bits are the TxINT, Framing and Overrun bits. This image is needed since the UPI may load the upper 4 STATUS register bits from its accumulator; however, it cannot read STATUS directly.

In Register Bank 1 (Figure 26), R1 holds the baud rate constant which is found from decoding the baud rate select bits of the CONFIGURE command. The counter is reloaded with this constant every timer tick. Like the receiver, the transmitter only needs to update the transmitter output every four ticks. R_2 holds the transmitter tick count. The value of R_2 determines which portion of the data is being transmitted; start bit, data bits, or stop bit. The transmit serializer is R_3. R_3 holds the data character as each character bit is transmitted.

R_4 is the transmitter holding register. It provides the double buffering for the transmitter. While transmitting one character, it is possible to load the next character into R_4 via DBBIN. The TxINT bit in STATUS and pin on PORT 2 reflect the "fullness" of R_4. If the holding register is empty, the interrupt bit and pin are set. They are reset when the master writes a new data byte for the transmitter into DBBIN. The transmitter status register (TxSTS) is R_5. Like RxSTS,TxSTS contains flag bits which indicate the current state of the transmitter. This flag bit format is shown in Figure 28.

TxSTS BIT 0 is the Tx flag. It is set whenever the transmitter is transmitting a character. It is set from the beginning of the start bit until the end of the stop bit. BIT 1 is the Tx request flag. This bit is set by the foreground routine when it transfers a new character from DBBIN to the Tx holding register, R_4. The transmitter software uses this flag to tell if new data is available. It is reset when the transmitter transfers the character from the holding register to the serializer.

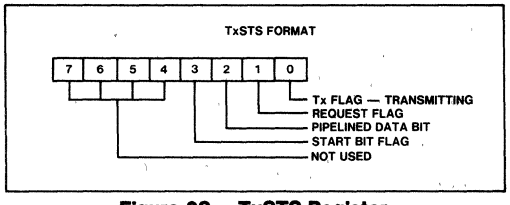


Figure 28. TxSTS Register

BIT 2 is the pipelined Tx data bit. The transmitter uses a pipelining technique which sets up the next output level in BIT 2 after processing the current timer tick. The output level is always changed at the same point after a timer tick interrupt. This technique ensures that no bit timing distortion results from different length processing paths through the receiver and transmitter routines.

BIT 3 of TxSTS is the Start Bit flag. It is set by the transmitter when the start bit space is set up in the pipelined data bit. This allows the transmitter to differentiate between the start bit and the data bits on following timer ticks.

The flow charts for this application are shown in Figures 29A–F. At reset, the INIT routine is executed which initializes the registers and port pins. After initialization, IBF and OBF are tested in MNLOOP. These flags are tested continually in this loop. If IBF is set, F_1 is tested for command or data and execution is transferred to the appropriate routine (CMD or DATA). If IBF=0, OBF is checked. If OBF=0 (DBBOUT is free), the Rx data ready and I/O flags in RxSTS are tested. If Rx data ready is set, the received data is retrieved from the Rx holding register and transferred to DBBOUT. Any error flags associated with that data are also transferred to STATUS. If the I/O flag is set and the I/O direction is input, PORT 1 is read and the data transferred to DBBOUT. In either case, F_0 and F_1 are set to indicate the data source.

If IBF is set by a command write to DBBIN, CMD reads the command and decodes the desired operation. If an I/O operation is specified, the I/O flag is set to indicate to the MNLOOP and DATA routines that an I/O operation is to be performed. If the command is a CONFIGURE command, the constant for the selected baud rate is loaded into both Baud Rate Constant register and the timer/counter. The timer/counter is started in the event counter mode and timer/counter interrupts are enabled. In addition, the I/O port is initialized to all 1's if the I/O direction bit specifies an input port. If the command is a RESET ERROR command, the two error flags in STATUS are cleared.

If the IBF flag is set by a data write, the DATA routine reads DBBIN and places the data in the appropriate place. If the I/O flag is set, the data is for the output port so the port is loaded. If the I/O flag is reset, the data is for the UART transmitter. Data for the transmitter resets the TxINT bit and pin plus sets the Tx request flag in TxSTS. The data is transferred to the Tx holding register, R_4.

AFN-01536A

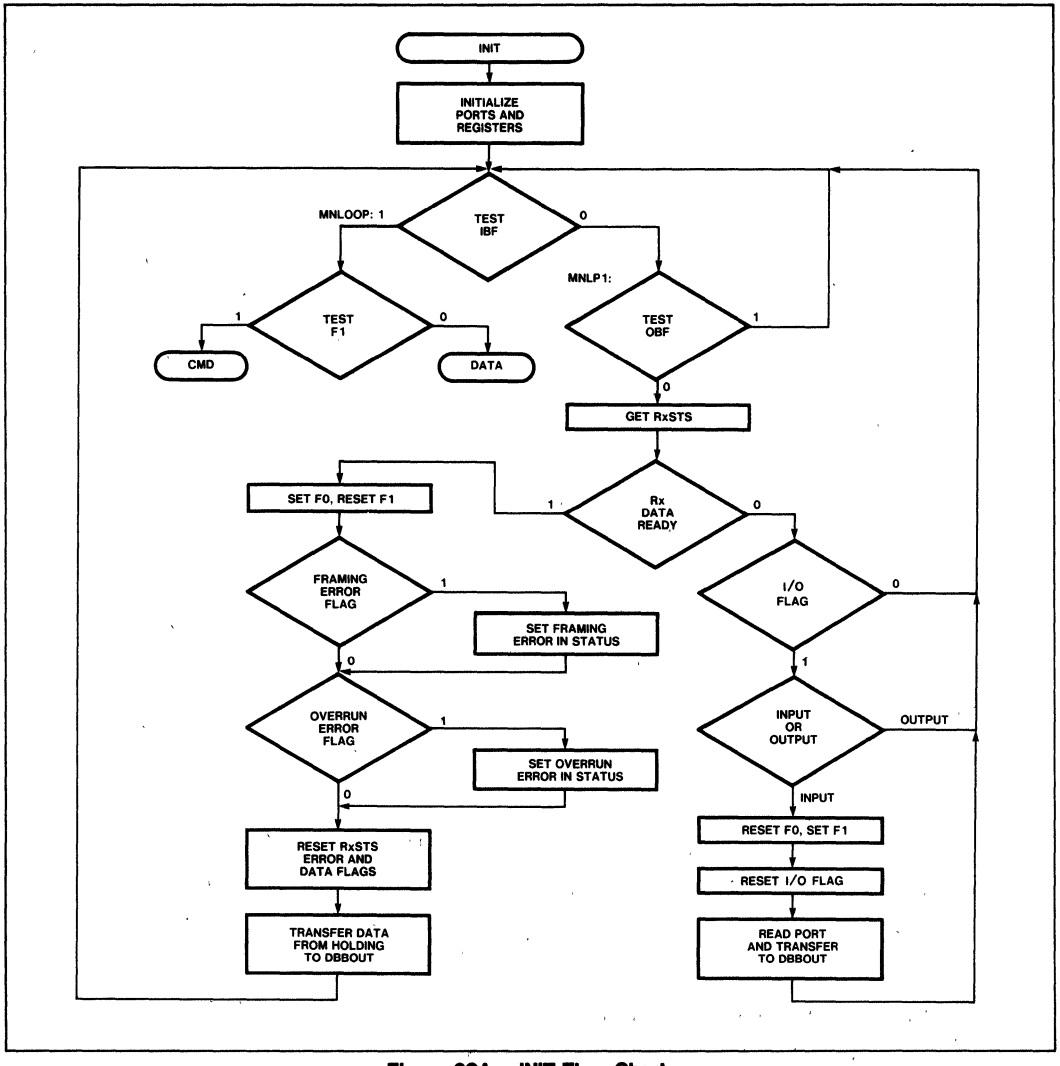


Figure 29A. INIT Flow Chart

Once a CONFIGURE command is received and the counter started, timer/counter interrupts start occurring at four times the selected baud rate. These interrupts cause a vector to the TIMINT routine, Figure 29D. A 76.8 kHz counter input provides a 13.02 μs counter resolution. Since it requires several UPI instruction cycles to reload the counter, the counter is set to two counts less than the desired baud rate and the counter is reloaded in TIMINT synchronous with the second low-going transition after the interrupt. Once the counter is reloaded, an output port (P_{26}) is toggled to give an external indication of internal counter interval. This is a helpful diagnostic feature. After the tick sample output, the pipelined transmitter data in TxSTS is output to the TxD pin. Although this occurs every timer tick, the pipelined data is changed only every fourth tick.

The receiver is now handled, Figure 29E. The Rx flag in RxSTS is examined to see if the receiver is currently in the process of receiving a character. If it is not, the RxD input is tested for a space condition which might indicate a possible start bit. If the input is a mark, no start bit is possible and execution

AFN-01536A

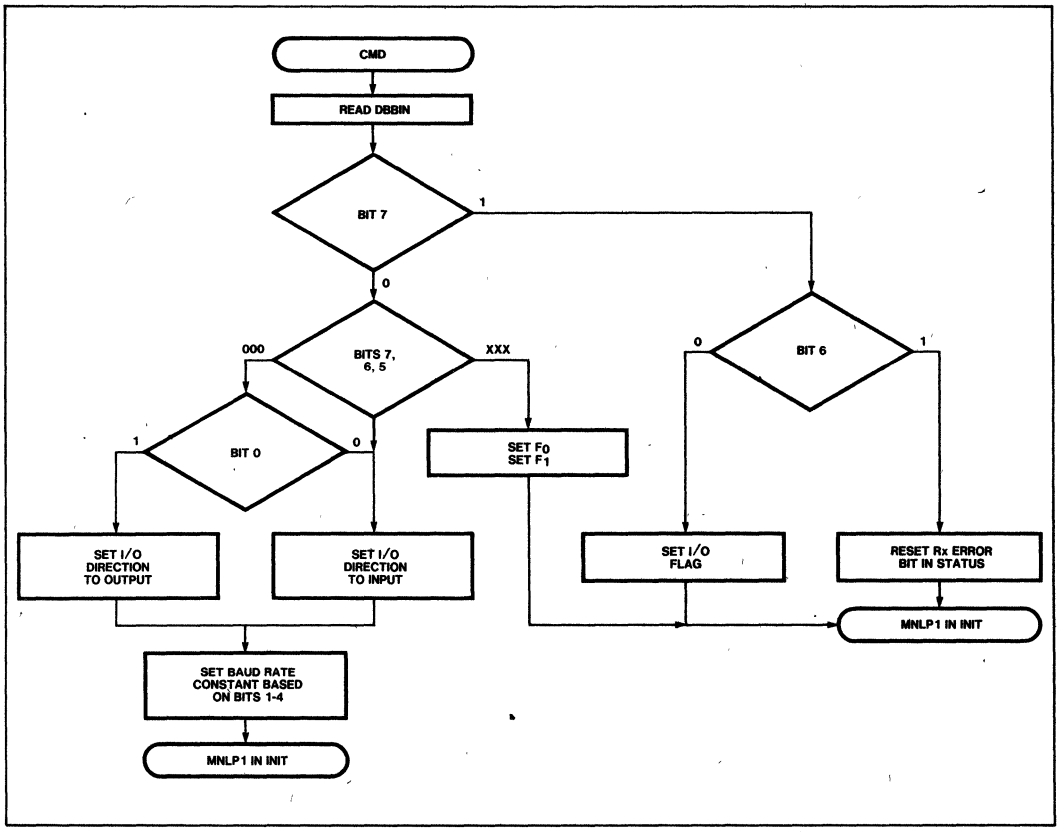


Figure 29B. CMD Flow Chart

branches to the transmitter flow, XMIT. If the input is a space, the Rx flag is set before proceeding with XMIT.

If the Rx flag is found set when entering RCV, the receiver is in the process of receiving a character. If so, the start bit flag is then tested to determine if a good start bit was received. The Rx tick counter is initialized to 4 and the Rx deserializer is set to 80H. A mark indicates a bad start bit; the Rx flag is reset to abort the reception.

If the start bit flag is set, the program is somewhere in the middle of the received character. Since the data should be sampled every fourth timer tick, the tick counter is decremented and tested for zero. If non-zero no sample is needed and execution continues with XMIT. If zero, the tick counter is reset to four. Now the byte finished flag is tested to determine if the data sample is a data or stop bit. If reset, the sample is a data bit. The sample is done and the new bit rotated into the Rx deserializer. If this rotate

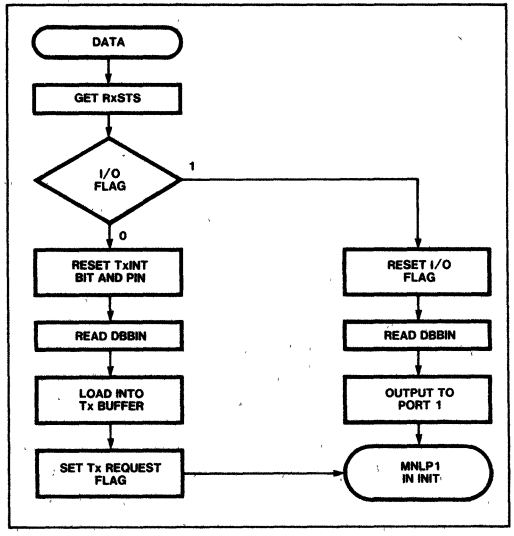


Figure 29C. Data Flow Chart

AFN-01536A

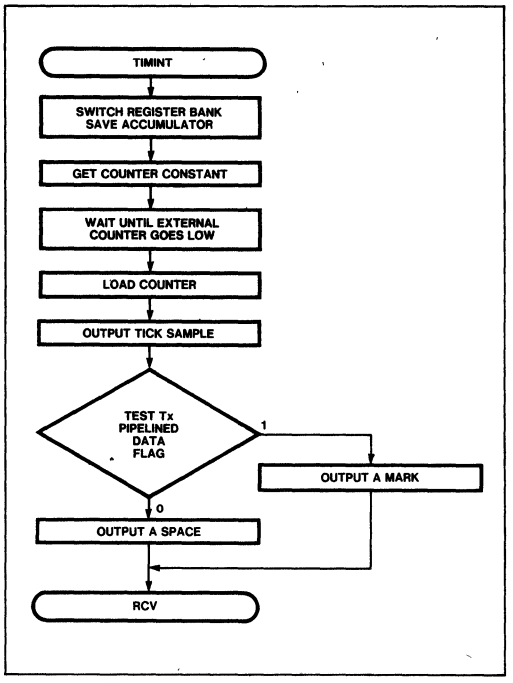


Figure 29D. TIMINT Flow Chart

sets the carry, that data bit was the last so the byte finished flag is set. If the carry is reset, the data bit is not the last so execution simply continues with XMIT.

Had the byte finished flag been set, this sample is for the stop bit. The RxD input is tested and if a space, the framing error flag is set. Otherwise, it is reset. Next, the Rx data ready flag is tested. If it is set, the master has not read the previous character so the overrun error flag is set. Then the Rx data ready flag is set and the received data character is transferred into the Rx holding register. The Rx, start bit, and byte finished flags are reset to get ready for the next character.

Execution of the transmitter routine, XMIT, follows the receiver, Figure 29F. The transmitter starts by checking the start bit flag in TxSTS. Recall that the actual transmit data is output at the beginning of the timer routine. The start bit flag indicates whether the current timer tick interrupt started the start bit. If it is set, the pipelined data output earlier in the routine was the start of the start bit so the flag is reset and the Tx tick counter is initialized. Nothing else is done this timer tick so the routine returns to the foreground.

If the start bit flag is reset, the Tx tick counter is incremented and tested. The test is performed modulo 4. If the counter mod 4 is not zero, it has not been four ticks since the transmitter was handled last so the routine simply returns. If the counter mod 4 is zero, it is time to handle the transmitter and the Tx flag is tested.

The Tx flag indicates whether the transmitter is active. If the transmitter is inactive, no character is currently being transmitted so the Tx request flag is tested to see if a new character is waiting in the Tx buffer. If no character is waiting (Tx request flag=0), the Tx interrupt pin and bit are set before returning to the foreground. If there is a character waiting, it is retrieved from the buffer and placed in the Tx serializer. The Tx request flag is reset while the Tx and start bit flags are set. A space is placed in the Tx pipelined data bit so a start bit will be output on the next tick. Since the Tx buffer is now empty, the Tx interrupt bit and pin are set to indicate the availability of the buffer to the master. The routine then returns to the foreground.

If the tick counter mod 4 is zero and the Tx flag indicates the transmitter is in the middle of a character, the tick counter is checked to see what transmitter operation is needed. If the counter is 28H (40D), all data bits plus the stop bits are complete. The character is therefore done and the Tx flag is reset. If the counter is 24H (36D), the data bits are complete and the next output should be a mark for the stop bit so a mark is loaded into the Tx pipelined data bit.

If neither of the above conditions are met for the counter, the transmitter is some place in the data field, so the next data bit is rotated out of the Tx serializer into the pipelined data bit. The next tick outputs this bit.

At this point the program execution is returned to the foreground.

That completes the discussion of the combination I/O device flow charts. The UPI software listing is shown in Appendix C1. Appendix C2 is example 8085A driver software.

Several observations concerning the drivers are appropriate. Notice that since the receiver and input port of the UPI use the OBF flag and interrupt output, the interrupt and flag are cleared when the master reads DBBOUT. This is not true for the transmitter. There is always some time after a master write of new transmitter data before the transmitter bit and pin are cleared. Thus in an interrupt-driven system, edge-sensitive interrupts should be

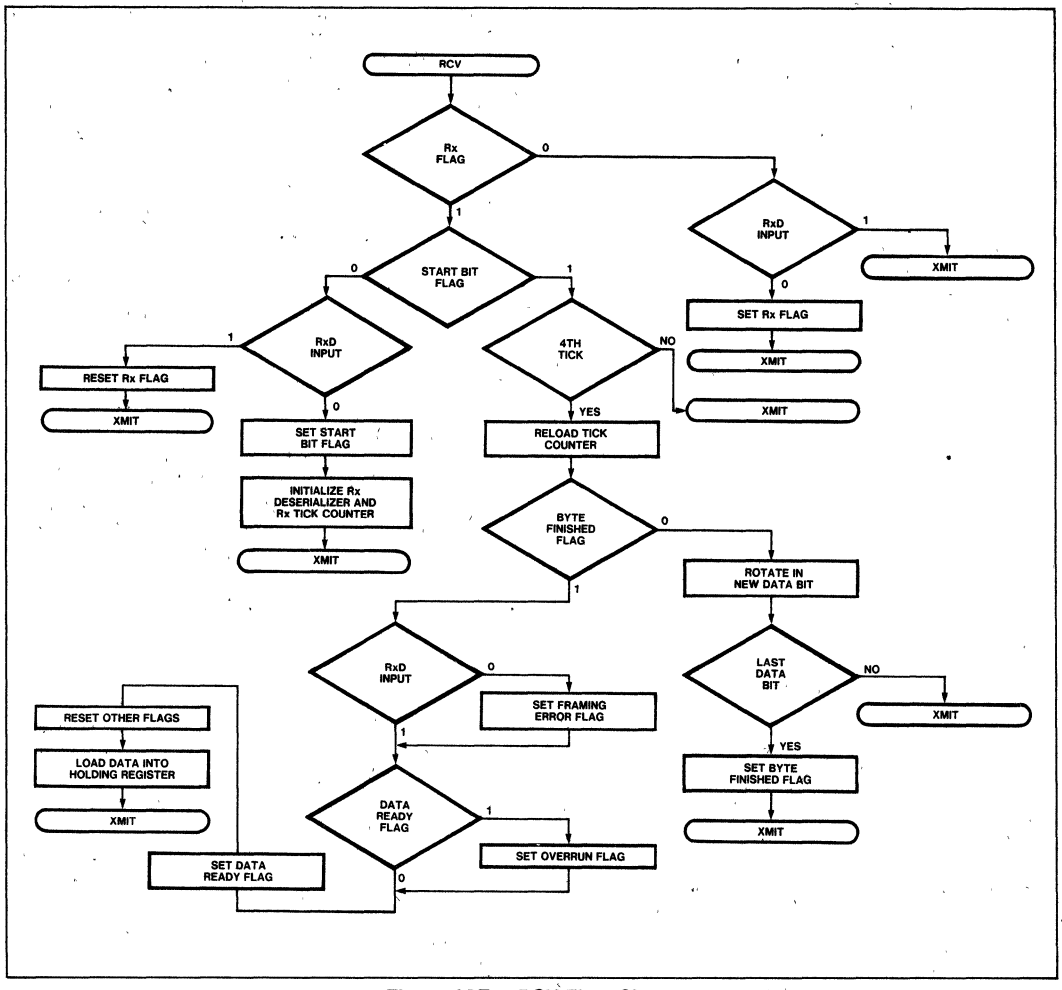


Figure 29E. RCV Flow Chart

used. For polled-systems, the software must wait after writing new data for IBF=0 before re-examining the Tx interrupt flag in STATUS.

Notice that this application uses none of the user data memory above Register Bank 1 and only 361 bytes of program memory. This leaves the door open for many improvements. Improvements that come to mind are increased buffering of the transmit or received data, modem control pins, and parallel port handshaking inputs.

This completes our discussion of specific UPI applications. Before concluding, let's look briefly at two debug techniques used during the development of these applications that you might find useful in your own designs.

DEBUG TECHNIQUES

Since the UPI is essentially a single-chip microcomputer, the classical data, address, and control buses are not available to the outside world during normal operation. This fact normally makes debugging a UPI design difficult; however, certain "tricks" can be included in the UPI software to ease this task.

If a UPI is handling multiple tasks, it is usually easier to code and debug each task individually. This is fairly standard procedure. Since each task usually utilizes only a subset of the total number of I/O pins,

AFN-01536A

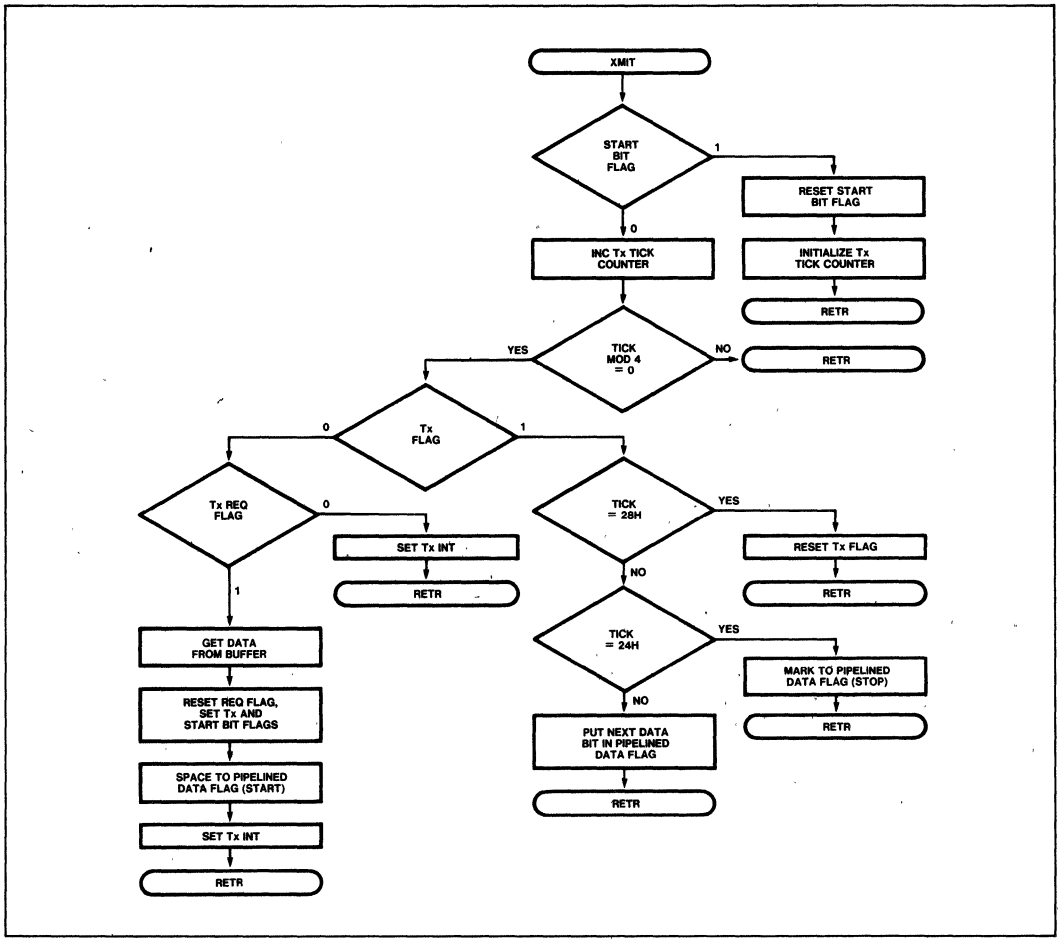


Figure 29F. XMIT Flow Chart

coding only one task leaves some I/O pins free. Port output instructions can then be added in the task code being debugged which toggle these unused pins to determine which section of task code is being executed at any particular time. The task can also be made to "wait" at various points by using an extra pin as an input and adding code to loop until a particular input condition is met.

One example of using an extra pin as an output is included in the combination serial/parallel device code. During initial development the receiver was not receiving characters correctly. Since this could be caused by incorrect sampling, three lines of code were added to toggle BIT 6 of PORT 2 at each tick of the sample clock. This code is at lines 184 and 185 of the listing. Thus by looking at the location of the tick sample pulse with respect to the received bit, the UPI sampling interval can be observed. The tick sample time was incorrect and the code was modified accordingly. Similar techniques could be applied at other locations in the program.

The EPROM version of the UPI (8741A) also contains another feature to aid in debug: the capability to single step thru a program. The user may step thru the program instruction-by-instruction. The address of the next instruction to be fetched is available on PORT 1 and the lower 2 bits of PORT 2. Figure 30 shows the timing used in the discussion below. When the single step input, $\overline{SS}$, is brought low, the internal processor responds by stopping during the fetch portion of the next instruction. This action is acknowledged by the processor raising the SYNC

AFN-01536A

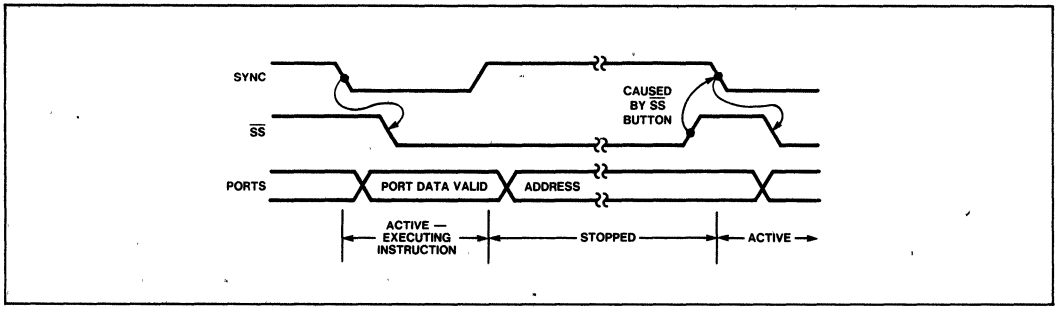


Figure 30. Single Step Timing

output. The address of the instruction to be fetched is then placed on the port pins. This state may be held indefinitely. To step to the next instruction, $\overline{SS}$ is raised high, which causes SYNC to go low, which is then used to return $\overline{SS}$ low. This allows the processor to advance to the next instruction. If $\overline{SS}$ is left high, the processor continues to execute at normal speed until $\overline{SS}$ goes low.

To preserve port functionality, port data is valid while SYNC is low. Figure 31 shows the external circuitry required to implement single step while preserving port functionality. S_1 is the RUN/STOP switch. When in the RUN position, the 7474 is held preset so $\overline{SS}$ is high and the UPI executes normally. When switched to STOP, the preset is removed and the next low-going transition of SYNC causes the 7474 to clear, lowering $\overline{SS}$. While sync is low, the port data is valid and the current instruction is executing. Low SYNC is also used to enable the tri-state buffers when the ports are used as inputs. When execution is complete, SYNC goes high. This transition latches the valid port data in the 74LS374s. SYNC going high also signifies that the address of the next instruction will appear on the port pins. This state can be held indefinitely with the address data displayed on the LEDs.

When the S_2 is depressed, the 7474 is set which causes $\overline{SS}$ to go high. This allows the processor to fetch and execute the instruction whose address was displayed. SYNC going low during execution, clears

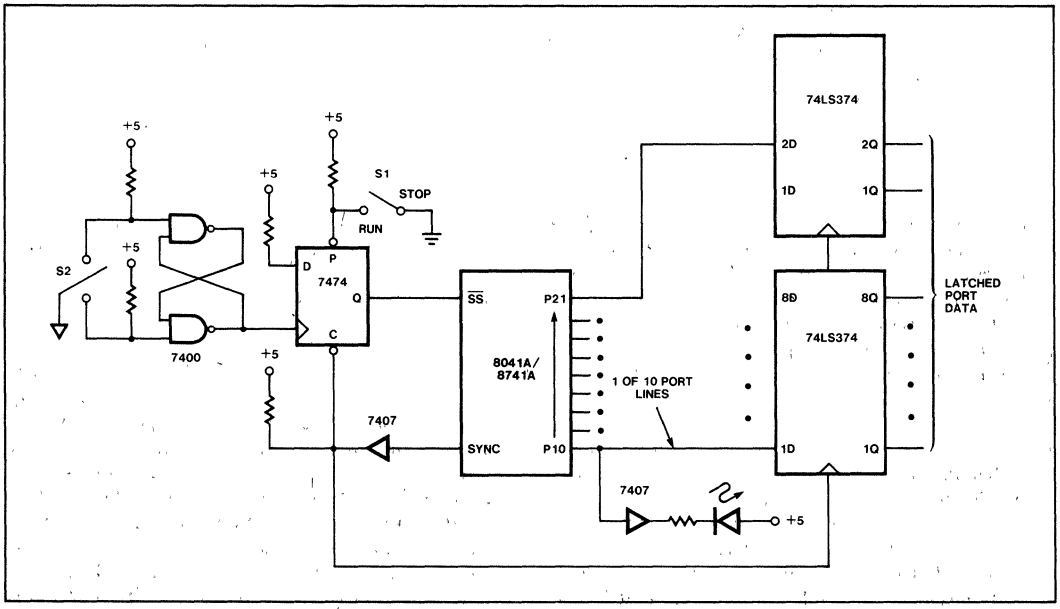


Figure 31. Single Step External Circuitry

AFN-01536A

the 7474 lowering $\overline{SS}$. Thus the processor again stops when execution is complete and the next fetch is started.

All UPI functions continue to operate while single stepping (the processor is actually executing NOPs internally while stopped). Both IBF and timer/counter interrupts can be serviced. The only change is that the interval timer is prescaled on single stepped instructions and, of course, will not indicate the correct intervals in real time. The total number of instruction which would have been executed during a given interval is the same however.

The single step circuitry can be used to step through a complete program; however, this might be a time-consuming job if the program is long or if only a portion is to be examined. The circuitry could easily be modified to incorporate the output toggling technique to determine when to run and stop. If you would like to step thru a particular section of code, an extra port pin could replace switch S_1. Extra instructions would then be added to lower the port when entering the code section and raise the port when exiting the section. The program would then stop when that section of code is reached allowing it to be stepped through. At the end of the section, the program would execute at normal speed.

CONCLUSION

Well, that's it. Machine readable (floppy disk or paper tape) source listings of UPI software for these applications are available in Insite, the Intel library of user-donated programs. Also available in Insite are the source listings for some of Intel's pre-programmed UPI products.

For information about Insite, write to:

Insite
Intel Corp.
3065 Bowers Ave.
Santa Clara, Ca 95051

APPENDIX A

```
F1 ASM48  F3 LED PRINT( LP ) NOOBJECT


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0                PAGE    1


 LOC  OBJ          LINE        SOURCE STATEMENT
                                 *
                     1 $MOD41A
                     2 ;        ***********************************************
                     3 ;        *    UPI-41A 8-DIGIT LED DISPLAY CONTROLLER    *
                     4 ;        ***********************************************
                     5 ;
                     6 ;
                     7 ;
                     8 ;THIS PROGRAM USES THE UPI-41A AS A LED DISPLAY CONTROLLER
                     9 ;WHICH SCANS AND REFRESHES EIGHT SEVEN-SEGMENT LED DISPLAYS.
                    10 ;THE CHARACTERS ARE DEFINED BY INPUT FROM A MASTER CPU IN THE
                    11 ;FORM OF ONE EIGHT BIT WORD PER DIGIT-CHARACTER SELECTION.
                    12 ;
                    13 ;
                    14 ;
                    15 ;*********************************************************************
                    16 ;
                    17 ;REGISTER DEFINITIONS.
                    18 ;        REGISTER                RB1                     RB0
                    19 ;        --------                ---                     ---
                    20 ;        R0              DISPLAY MAP POINTER             NOT USED
                    21 ;        R1              NOT USED                        NOT USED
                    22 ;        R2              DATA WORD AND CHARACTER STORAGE NOT USED
                    23 ;        R3              DIGIT COUNTER                   NOT USED
                    24 ;        R4              NOT USED                        NOT USED
                    25 ;        R5              NOT USED                        NOT USED
                    26 ;        R6              NOT USED                        NOT USED
                    27 ;        R7              ACCUMULATOR STORAGE             NOT USED
                    28 ;*********************************************************************
                    29 ;
                    30 ;PORT PIN DEFINITIONS
                    31 ;        PIN             PORT 1 FUNCTION         PORT 2 FUNCTION
                    32 ;        ---             ---------------         ---------------
                    33 ;        P0-7            SEGMENT DRIVER CONTROL  DIGIT DRIVER CONTROL
                    34 ;
                    35 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0                PAGE    2

 LOC  OBJ          LINE        SOURCE STATEMENT

                     36 ;*************************************************************************
                     37 ;DISPLAY DATA WORD BIT DEFINITION:
                     38 ;        BIT                 FUNCTION
                     39 ;        ---                 --------
                     40 ;        0-4                 CHARACTER SELECT
                     41 ;        5-7                 DIGIT SELECT
                     42 ;
                     43 ;CHARACTER SELECT:
                     44 ;                    D4  D3  D2  D1  D0  CHARACTER
                     45 ;                    0   0   0   0   0       0
                     46 ;                    0   0   0   0   1       1
                     47 ;                    0   0   0   1   0       2
                     48 ;                    0   0   0   1   1       3
                     49 ;                    0   0   1   0   0       4
                     50 ;                    0   0   1   0   1       5
                     51 ;                    0   0   1   1   0       6
                     52 ;                    0   0   1   1   1       7
                     53 ;                    0   1   0   0   0       8
                     54 ;                    0   1   0   0   1       9
                     55 ;                    0   1   0   1   0       A
                     56 ;                    0   1   0   1   1       B
                     57 ;                    0   1   1   0   0       C
                     58 ;                    0   1   1   0   1       D
                     59 ;                    0   1   1   1   0       E
                     60 ;                    0   1   1   1   1       F
                     61 ;                    1   0   0   0   0       .
                     62 ;                    1   0   0   0   1       G
                     63 ;                    1   0   0   1   0       H
                     64 ;                    1   0   0   1   1       I
                     65 ;                    1   0   1   0   0       J
                     66 ;                    1   0   1   0   1       L
                     67 ;                    1   0   1   1   0       N
                     68 ;                    1   0   1   1   1       O
                     69 ;                    1   1   0   0   0       P
                     70 ;                    1   1   0   0   1       R
                     71 ;                    1   1   0   1   0       T
                     72 ;                    1   1   0   1   1       U
                     73 ;                    1   1   1   0   0       Y
                     74 ;                    1   1   1   0   1       -
                     75 ;                    1   1   1   1   0       /
                     76 ;                    1   1   1   1   1       "BLANK"
                     77 ;
                     78 ;DIGIT SELECT:
                     79 ;                    D7  D6  D5    DIGIT NUMBER
                     80 ;                    0   0   0           1
                     81 ;                    0   0   1           2
                     82 ;                    0   1   0           3
                     83 ;                    0   1   1           4
                     84 ;                    1   0   0           5
                     85 ;                    1   0   1           6
                     86 ;                    1   1   0           7
                     87 ;                    1   1   1           8
                     88 ;*************************************************************************
                     89 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    3

 LOC  OBJ          LINE        SOURCE STATEMENT

                     90 ,*************************************************************************
                     91 ;                              EQUATES
                     92 ;THE FOLLOWING CODE DESIGNATES "TIME" AS A VARIABLE. THIS
                     93 ;ADJUSTS THE AMOUNT OF CYCLES THE TIMER COUNTS BEFORE
                     94 ;A TIMER INTERRUPT OCCURS AND REFRESHES THE DISPLAY. APPROXIMATELY
                     95 ;50 TIMES PER SECOND.
 FFF1                96 TIME     EQU     -0FH    ;TIMER VALUE 2.5MSEC
                     97 ;*************************************************************************
                     98 ;                          INTERRUPT BRANCHING
                     99 ;THIS PORTION OF MEMORY IS DEDICATED FOR USE OF RESET AND
                    100 ;INTERRUPT BRANCHING. WHEN THE INTERRUPTS ARE ENABLED THE
                    101 ;CODE AT THE FOLLOWING DESIGNATED SPOTS ARE EXECUTED WHEN A
                    102 ;RESET OR A INTERRUPT OCCURS
 0000               103         ORG     0               ;
 0000 0409          104         JMP     START           ;RESET
 0002 00            105         NOP                     ;
 0003 0436          106         JMP     INPUT           ;IBF INTERRUPT
 0005 00            107         NOP                     ;
 0006 00            108         NOP                     ;
 0007 041D          109         JMP     DISPLA          ;TIMER INTERRUPT
                    110 ;*************************************************************************
                    111 ;                          INITIALIZATION
                    112 ;THE FOLLOWING CODE SETS UP THE UPI-41 AND DISPLAY HARDWARE
                    113 ;INTO OPERATIONAL FORMAT. THE DISPLAY IS TURNED OFF, THE DISPLAY
                    114 ;MAP IS FILLED WITH "BLANK" CHARACTERS, THE TIMER SET AND THE
                    115 ;INTERRUPTS ARE ENABLED
                    116 ;
 0009 D5            117 START:  SEL     RB1             ;
 000A 8A08          118         ORL     P2,#08H         ;TURN DIGIT DRIVERS OFF
 000C B838          119         MOV     R0,#38H         ;DISPLAY MAP POINTER,BOTTOM OF DISPLAY MAP
 000E 23FF          120 BLKMAP  MOV     A,#0FFH         ;FF="BLANK"
 0010 A0            121         MOV     @R0,A           ;BLANK TO DISPLAY MAP
 0011 18            122         INC     R0              ;INCREMENT DISPLAY MAP POINTER
 0012 F8            123         MOV     A,R0            ;DISPLAY MAP POINTER TO ACCUMULATOR
 0013 B20E          124         JB5     BLKMAP          ;BLANK DISPLAY MAP TILL FILLED
 0015 BB00          125         MOV     R3,#00H         ;SET DIGIT COUNTER TO 0
 0017 23F1          126         MOV     A,#TIME         ;TIMER VALUE
 0019 62            127         MOV     T,A             ;LOAD TIMER
 001A 55            128         STRT    T               ;START TIMER
 001B 25            129         EN      TCNTI           ;ENABLE TIMER INTERRUPT
 001C 05            130         EN      I               ;ENABLE IBF INTERRUPT
                    131 ;*************************************************************************
                    132 ;                          USER PROGRAM
                    133 ;A USERS PROGRAM WOULD INITIALIZE AT THIS POINT. THE FOLLOWING
                    134 ;CODE IS UND CONCLUDED WITH
                    135 ;SYNC CHARACTERS (0AAH).  A CHECKSUM BYTE IMMEDIATELY PRECEEDS THE
                    136 ;FINAL SYNC   WHEN READING, THE CONTROLLE**************************************
                    137 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    4

 LOC  OBJ          LINE        SOURCE STATEMENT

                    138 ;*************************************************************************
                    139 ,                          DISPLAY ROUTINE
                    140 ;  THIS PORTION OF THIS PROGRAM IS AN INTERRUPT ROUTINE WHICH IS
                    141 ,ACTED UPON WHEN THE TIMER COUNT IS COMPLETED.  THE ROUTINE UPDATES
                    142 ;ONE DISPLAY DIGIT FROM THE DISPLAY MAP PER INTERRUPT SEQUENTIALLY,
                    143 ;THUS EIGHT TIMER INTERRUPTS WILL HAVE REFRESHED THE ENTIRE DISPLAY.
                    144 ;REGISTER BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON
                    145 ;ENTERING THE ROUTINE  ONCE THE DISPLAY HAS BEEN REFRESHED THE TIMER
                    146 ;IS RESET AND THE ACCUMULATOR AND PRE-INTERRUPT REGISTER BANK IS RESTORED.
                    147 ;
 001D D5            148 DISPLA  SEL     RB1             ;REGISTER BANK 1
 001E AF            149         MOV     R7,A            ;SAVE ACCUMULATOR
 001F 8A08          150         ORL     P2,#08H         ;TURN DIGIT DRIVERS OFF
 0021 FB            151         MOV     A,R3            ;DIGIT COUNTER TO ACCUMULATOR
 0022 4338          152         ORL     A,#38H          ;"OR" TO GET DISPLAY MAP ADDRESS
 0024 A8            153         MOV     R0,A            ;DISPLAY MAP POINTER
 0025 F0            154         MOV     A,@R0           ;GET CHARACTER FROM DISPLAY MAP
 0026 39            155         OUTL    P1,A            ;OUTPUT CHARACTER TO SEGMENT DRIVERS
 0027 FB            156         MOV     A,R3            ;DIGIT COUNTER VALUE TO ACCUMULATOR
 0028 3A            157         OUTL    P2,A            ;OUTPUT TO DIGIT DRIVERS
 0029 1B            158         INC     R3              ,INCREMENT DIGIT COUNTER
 002A D307          159         XRL     A,#07H          ;CHECK IF AT LAST DIGIT
 002C 9630          160         JNZ     SETIME          ;RESET TIMER IN NOT LAST DIGIT
 002E BB00          161         MOV     R3,#00H         ;RESET DIGIT COUNTER
 0030 23F1          162 SETIME: MOV     A,#TIME         ;TIMER VALUE
 0032 62            163         MOV     T,A             ;LOAD TIMER
 0033 55            164         STRT    T               ;START TIMER
 0034 FF            165         MOV     A,R7            ;RESTORE ACCUMULATOR
 0035 93            166         RETR                    ;RETURN
                    167 ;*************************************************************************
                    168 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0              PAGE    5


 LOC  OBJ          LINE        SOURCE STATEMENT

                    169 ;
                    170 ;*************************************************************************
                    171 ;                   INPUT CHARACTER AND DIGIT ROUTINE
                    172 ;  THIS PORTION OF THE PROGRAM IS AN INTERRUPT ROUTINE WHICH
                    173 ;IS ACTED UPON WHEN THE IBF BIT IS SET.  THE ROUTINE GETS THE
                    174 ;DISPLAY DATA WORD FROM THE DBB AND DEFINES BOTH THE DIGIT AND
                    175 ;THE CHARACTER TO BE DISPLAYED.  THIS IS DONE BY MEANS OF A
                    176 ;CHARACTER LOOP-UP TABLE AND A DISPLAY MAP FOR DIGIT AND CHARACTER
                    177 ;LOCATION.  SPECIAL CONSIDERATION IS TAKEN FOR A DECIMAL POINT WHICH IS
                    178 ;SIMPLY ADDED TO THE EXISTING CHARACTER IN THE DISPLAY MAP.  REGISTER
                    179 ;BANK 1 IS SELECTED AND THE ACCUMULATOR IS SAVED UPON ENTERING
                    180 ;THE ROUTINE.  ONCE THE DATA WORD HAS BEEN FULLY DEFINED THE ACCUMULATOR
                    181 ;AND THE PRE-INTERRUPT REGISTER BANK IS RESTORED.
                    182 ;
 0036 D5            183 INPUT:  SEL     RB1             ;REGISTER BANK 1
 0037 AF            184         MOV     R7,A            ;SAVE ACCUMULATOR
 0038 22            185         IN      A,DBB           ;GET DATA
 0039 AA            186         MOV     R2,A            ;SAVE DATA WORD
 003A 47            187         SWAP    A               ;DEFINE DIGIT LOCATION
 003B 77            188         RR      A               ;
 003C 5307          189         ANL     A,#07H          ;
 003E 4338          190         ORL     A,#38H          ;
 0040 A8            191         MOV     R0,A            ;DIGIT LOCATION IN DIGIT POINTER
 0041 FA            192         MOV     A,R2            ;SAVED DATA WORD TO ACCUMULATOR
 0042 531F          193         ANL     A,#1FH          ;DEFINE CHARACTER LOOK-UP-TABLE LOC.
 0044 E3            194         MOVP3   A,@A            ;GET CHARACTER
 0045 AA            195         MOV     R2,A            ;SAVE CHARACTER
 0046 D37F          196         XRL     A,#7FH          ;IS CHARACTER DECIMAL POINT
 0048 C64E          197         JZ      DPOINT          ;
 004A FA            198         MOV     A,R2            ;SAVED CHARACTER TO ACCUMULATOR
 004B A0            199         MOV     @R0,A           ;CHARACTER TO DISPLAY MAP
 004C 0451          200         JMP     RETURN          ;
 004E FA            201 DPOINT  MOV     A,R2            ;SAVED CHARACTER TO ACCUMULATOR
 004F 50            202         ANL     A,@R0           ;"AND" WITH OLD CHARACTER
 0050 A0            203         MOV     @R0,A           ;BACK TO DISPLAY MAP
 0051 FF            204 RETURN. MOV     A,R7            ;RESTORE ACCUMULATOR
 0052 93            205         RETR
                    206 ;*************************************************************************
                    207 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0                PAGE    6

  LOC  OBJ          LINE        SOURCE STATEMENT

                     208 ;*****************************************************************
                     209 ;                          LOOK-UP TABLE
                     210 ; THIS LOOK-UP TABLE ORIGINATES IN PAGE 3 OF THE UPI-41 PROGRAM
                     211 ;MEMORY.  IT IS USED TO DEFINE THE CORRECT LEVEL OF EACH SEGMENT
                     212 ;AND DECIMAL POINT FOR A SELECTED CHARACTER FROM THE INPUT ROUTINE.
                     213 ;INVERSE LOGIC IS USED BECAUSE OF THE SPECIFIC DRIVER CIRCUITRY, THUS
                     214 ;A 1 ON A GIVEN SEGMENT MEANS IT IS OFF AND A 0 MEANS IT IS ON
                     215 ;
                     216                                  ;*******SEGMENTS********
  0300               217           ORG     300H          ;DP  G  F  E  D  C  B  A
  0300 C0            218 CH0       DB      0C0H          ;1   1  0  0  0  0  0  0
  0301 F9            219 CH1       DB      0F9H          ;1   1  1  1  1  0  0  1
  0302 A4            220 CH2       DB      0A4H          ;1   0  1  0  0  1  0  0
  0303 B0            221 CH3       DB      0B0H          ;1   0  1  1  0  0  0  0
  0304 99            222 CH4       DB      99H           ;1   0  0  1  1  0  0  1
  0305 92            223 CH5       DB      92H           ;1   0  0  1  0  0  1  0
  0306 82            224 CH6       DB      82H           ;1   0  0  0  0  0  1  0
  0307 F8            225 CH7       DB      0F8H          ;1   1  1  1  1  0  0  0
  0308 80            226 CH8       DB      80H           ;1   0  0  0  0  0  0  0
  0309 98            227 CH9       DB      98H           ;1   0  0  1  1  0  0  0
  030A 88            228 CHA       DB      88H           ;1   0  0  0  1  0  0  0
  030B 83            229 CHB       DB      83H           ;1   0  0  0  0  0  1  1
  030C C6            230 CHC       DB      0C6H          ;1   1  0  0  0  1  1  0
  030D A1            231 CHD       DB      0A1H          ;1   0  1  0  0  0  0  1
  030E 86            232 CHE       DB      86H           ;1   0  0  0  0  1  1  0
  030F 8E            233 CHF       DB      8EH           ;1   0  0  0  1  1  1  0
  0310 7F            234 CHDP      DB      7FH           ;0   1  1  1  1  1  1  1
  0311 C2            235 CHG       DB      0C2H          ;1   1  0  0  0  0  1  0
  0312 89            236 CHH       DB      89H           ;1   0  0  0  1  0  0  1
  0313 F8            237 CHI       DB      0F8H          ;1   1  1  1  1  0  1  1
  0314 E1            238 CHJ       DB      0E1H          ;1   1  1  0  0  0  0  1
  0315 C7            239 CHL       DB      0C7H          ;1   1  0  0  0  1  1  1
  0316 AB            240 CHN       DB      0ABH          ;1   0  1  0  1  0  1  1
  0317 A3            241 CHO       DB      0A3H          ;1   0  1  0  0  0  1  1
  0318 8C            242 CHP       DB      8CH           ;1   0  0  0  1  1  0  0
  0319 AF            243 CHR       DB      0AFH          ;1   0  1  0  1  1  1  1
  031A 87            244 CHT       DB      87H           ;1   0  0  0  0  1  1  1
  031B C1            245 CHU       DB      0C1H          ;1   1  0  0  0  0  0  1
  031C 91            246 CHY       DB      91H           ;1   0  0  1  0  0  0  1
  031D BF            247 CHDASH    DB      0BFH          ;1   0  1  1  1  1  1  1
  031E FD            248 CHAPOS    DB      0FDH          ;1   1  1  1  1  1  0  1
  031F FF            249 BLANK     DB      0FFH          ;1   1  1  1  1  1  1  1
                     250 ;*****************************************************************
                     251           END

USER SYMBOLS
BLANK  031F    BLKMAP 000E    CH0    0300    CH1    0301    CH2    0302    CH3    0303    CH4    0304    CH5    0305
CH6    0306    CH7    0307    CH8    0308    CH9    0309    CHA    030A    CHAPOS 031E    CHB    030B    CHC    030C
CHD    030D    CHDASH 031D    CHDP   0310    CHE    030E    CHF    030F    CHG    0311    CHH    0312    CHI    0313
CHJ    0314    CHL    0315    CHN    0316    CHO    0317    CHP    0318    CHR    0319    CHT    031A    CHU    031B
CHY    031C    DISPLA 001D    DPOINT 004E    INPUT  0036    RETURN 0051    SETIME 0030    START  0009    TIME   FFF1

ASSEMBLY COMPLETE,   NO ERRORS
```

AFN-01536A

```
F1 ASM48  F3 SENSOR NOOBJECT PRINT( LP )


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    1

 LOC  OBJ          LINE        SOURCE STATEMENT

                      1 $MOD41A
                      2 ;         ******************************************
                      3 ;         *      UPI-41A SENSOR MATRIX CONTROLLER      *
                      4 ;         ******************************************
                      5 ;
                      6 ,       THIS PROGRAM USES THE UPI-41A AS A SENSOR MATRIX CONTROLLER
                      7 ;IT HAS MONITORING CAPABILITIES OF UP TO 128 SENSORS   THE COORDINATE
                      8 ;AND SENSOR STATUS OF EACH DETECTED CHANGE IS AVAILABLE TO THE MASTER
                      9 ;MICROPROCESSOR IN A SINGLE BYTE.  A 40X8 FIFO QUEUE IS PROVIDED FOR
                     10 ;DATA BUFFERING   BOTH HARDWARE OR POLLED INTERRUPT METHODS CAN BE USED
                     11 ;TO NOTIFY THE MASTER OF A DETECTED SENSOR CHANGE
                     12 ,
                     13 ;*************************************************************************
                     14 ;
                     15 ;REGISTER DEFINITIONS.
                     16 ;        REGISTER             RB0                  RB1
                     17 ;        --------             ---                  ---
                     18 ,          R0          MATRIX MAP POINTER        NOT USED
                     19 ;          R1          FIFO POINTER              NOT USED
                     20 ;          R2          SCAN ROW SELECT           NOT USED
                     21 ,          R3          COLUMN COUNTER            NOT USED
                     22 ;          R4          FIFO-IN                   NOT USED
                     23 ;          R5          FIFO-OUT                  NOT USED
                     24 ;          R6          CHANGE WORD               NOT USED
                     25 ;          R7          COMPARE                   NOT USED
                     26 ;
                     27 ;*************************************************************************
                     28 ;
                     29 ,PORT PIN DEFINITIONS
                     30 ;
                     31 ;PIN       PORT 1 FUNCTION      PIN       PORT 2 FUNCTION
                     32 ,---       ---------------      ---       ---------------
                     33 ;P0-7      COLUMN LINE INPUTS   P0-3      ROW SELECT OUTPUTS
                     34 ;                               P4        FIFO NOT EMPTY INTERRUPT
                     35 ,                               P5        OBF INTERRUPT
                     36 ;                               P6-7      NOT USED
                     37 ,
                     38 ;*************************************************************************
                     39
                     40 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    2

 LOC  OBJ          LINE        SOURCE STATEMENT

                     41 ,*************************************************************************
                     42 ,
                     43 ,CHANGE WORD BIT DEFINITION
                     44 ,
                     45 ;                    BIT                 FUNCTION
                     46 ,                    ---                 --------
                     47 ;                    D0-6                SENSOR COORDINATE
                     48 ,                    D7                  SENSOR STATUS
                     49 ,
                     50 ,*************************************************************************
                     51 ;
                     52 ,STATUS REGISTER BIT DEFINITION
                     53 ,
                     54 ;                    BIT                 FUNCTION
                     55 ;                    ---                 --------
                     56 ;                    D0                  OBF
                     57 ;                    D1-3                IBF, F0, F1 (NOT USED)
                     58 ,                    D4                  FIFO NOT EMPTY
                     59 ,                    D5-7                USED DEFINED (NOT USED)
                     60 ;
                     61 ,*************************************************************************
                     62 ,
                     63 ,                            EQUATES
                     64 ,
                     65 ,THE FOLLOWING CODE DESIGNATES THREE VARIABLES, SCANTM,FIFOBA
                     66 ,AND FIFOTA   SCANTM ADJUSTS THE LENGTH OF A DELAY BETWEEN
                     67 ,SCANNING SWITCH   THIS SIMULATES DEBOUNCE FUNCTIONS   FIFOBA
                     68 ,IS THE BOTTOM ADDRESS OF THE FIFO   FIFOTA IS THE TOP ADDRESS
                     69 ,OF THE FIFO  THIS MAKES IT POSSIBLE TO HAVE A FIFO 3 TO 40
                     70 ,BYTES IN LENGTH
                     71 ,
                     72 ,*************************************************************************
                     73 ,
000F                 74 SCANTM  EQU     0FH             ,SCAN TIME ADJUST
0008                 75 FIFOBA  EQU     08H             ,FIFO BOTTOM ADDRESS
002F                 76 FIFOTA  EQU     2FH             ,FIFO TOP ADDRESS
                     77
                     78 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    3

 LOC  OBJ          LINE        SOURCE STATEMENT

                     79 ,*****************************************************************
                     80 ,
                     81 ,                          INITIALIZATION
                     82 ,
                     83 ,THE PROGRAM STARTS AT THE FOLLOWING CODE UPON RESET   WITHIN
                     84 ,THIS INITIALIZATION SECTION THE REGISTERS THAT MAINTAIN THE MATRIX
                     85 ,MAP,FIFO AND ROW SCANNING ARE SET UP   PORT 1 IS SET HIGH FOR USE
                     86 ,AS AN INPUT PORT FOR THE COLUMN STATUS   BIT 4 OF STATUS REGISTER IS
                     87 ,WRITTEN TO CONVEY A FIFO EMPTY CONDITION   THE INITIAL COLUMN STATUS
                     88 ;OF ALL THE ROWS IN THE SENSOR MATRIX IS THEN READ INTO THE MATRIX
                     89 ;MAP   ONCE THE MATRIX MAP IS FILLED THE OBF INTERRUPT (PORT 2-4) IS
                     90 ,ENABLED
                     91 ,
                     92 ;*****************************************************************
                     93 ,
 0000                94          ORG     0
 0000 B83F           95 INITMX   MOV     R0,#3FH         ;MATRIX MAP POINTER REGISTER,TOP ADDRESS
 0002 BA0F           96          MOV     R2,#0FH         ;SCAN ROW SELECT REGISTER,TOP ROW
 0004 BC08           97          MOV     R4,#FIFOBA      ,FIFO INPUT ADDRESS REGISTER,BOTTOM OF FIFO
 0006 BD2F           98          MOV     R5,#FIFOTA      ,FIFO OUTPUT ADDRESS REGISTER,TOP OF FIFO
 0008 B9FF           99          ORL     P1,#0FFH        ,INITIALIZE PORT 1 HIGH FOR INPUTS
 000A 2300          100          MOV     A,#00H          ,INITIALIZE STATUS REGISTER, FIFO EMPTY
 000C 90            101          MOV     STS,A           ,WRITE TO STATUS REGISTER, BITS 4-7
 000D FA            102 FILLMX   MOV     A,R2            ,SCAN ROW SELECT TO ACCUMULATOR
 000E 3A            103          OUTL    P2,A            ,OUTPUT SCAN ROW SELECT TO PORT 2
 000F 09            104          IN      A,P1            ,INPUT COLUMN STATUS PORT 1
 0010 A0            105          MOV     @R0,A           ,LOAD MATRIX MAP WITH COLUMN STATUS
 0011 FA            106          MOV     A,R2            ,CHECK SCAN ROW SELECT REGISTER VALUE FOR 0
 0012 C618          107          JZ      OBFINT          ,IF 0 ENABLE OBF INTERRUPT
 0014 C8            108          DEC     R0              ,DECREMENT TO NEXT MATRIX MAP ADDRESS
 0015 CA            109          DEC     R2              ,DECREMENT TO SCAN NEXT ROW
 0016 040D          110          JMP     FILLMX          ,FILL NEXT MATRIX MAP ADDRESS
 0018 BA10          111 OBFINT   MOV     R2,#10H         ,BIT 4 HIGH IN ROW SCAN SELECT REGISTER
 001A FA            112          MOV     A,R2            ,ROW SCAN SELECT VALUE TO ACCUMULATOR
 001B 3A            113          OUTL    P2,A            ,INITIALIZE PORT 2, BIT 4 FOR "EN FLAGS"
 001C F5            114          EN      FLAGS           ,ENABLE OBF INTERRUPT PORT 2, BIT 4
                    115
                    116 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    4

 LOC  OBJ          LINE        SOURCE STATEMENT

                    117 ,*************************************************************************
                    118 ,
                    119 ,                          SCAN AND COMPARE
                    120 ,
                    121 ,THE FOLLOWING CODE IS THE SCAN AND COMPARE SECTION OF THE PROGRAM.
                    122 ,UPON ENTERING THIS SECTION A CHECK IS MADE TO SEE IF THE ENTIRE MATRIX
                    123 ,HAS BEEN SCANNED.  IF SO THE REGISTERS THAT MAINTAIN THE MATRIX MAP AND ROW
                    124 ,SCANNING ARE RESET TO THE BEGINNING OF THE SENSOR MATRIX.  IF THE ENTIRE
                    125 ,MATRIX HASNT BEEN SCANNED THE REGISTERS INCREMENT TO SCAN THE NEXT ROW.
                    126 ,FROM THIS POINT ON THE ROW SCAN SELECT REGISTER IS USED FOR TWO FUNCTIONS.
                    127 ,BITS 0-3 FOR SCANNING AND BITS 4 AND 5 FOR THE EXTERNAL INTERRUPTS.  THUSLY
                    128 ,ALL USAGE OF THE REGISTERS IS DONE BY LOGICALLY MASKING IT SO AS TO ONLY
                    129 ,AFFECT THE FUNCTION DESIRED   ONCE THE REGISTERS ARE RESET, ONE ROW OF THE
                    130 ,SENSOR MATRIX IS SCANNED.  A DELAY IS EXECUTED TO ADJUST FOR SCAN TIME
                    131 ,(DEBOUNCE).  A BYTE OF COLUMN STATUS IS THEN READ INTO THE MATRIX MAP
                    132 ,AT THE TIME THE NEW COLUMN STATUS IS COMPARED TO THE OLD.  THE RESULT IS
                    133 ,STORED IN THE COMPARE REGISTER.  THE PROGRAM IS THEN ROUTED ACCORDING TO
                    134 ,WHETHER OR NOT A CHANGE WAS DETECTED.
                    135 ;
                    136 ,*************************************************************************
                    137
 001D FA            138 ADJREG  MOV     A,R2            ,SCAN ROW SELECT TO ACCUMULATOR
 001E 530F          139         ANL     A,#0FH          ,CHECK FOR 0 SCAN VALUE ONLY,NOT INTERRUPT
 0020 C626          140         JZ      RSETRG          ;IF 0 RESET REGISTERS
 0022 C8            141         DEC     R0              ;DECREMENT MATRIX MAP POINTER
 0023 CA            142         DEC     R2              ,DECREMENT SCAN ROW SELECT
 0024 042C          143         JMP     SCANMX          ,SCAN MATRIX
 0026 B83F          144 RSETRG. MOV     R0,#3FH         ,RESET MATRIX MAP POINTER REGISTER,TOP ADDRESS
 0028 FA            145         MOV     A,R2            ,SCAN ROW SELECT TO ACCUMULATOR
 0029 430F          146         ORL     A,#0FH          ;RESET SCAN ROW SELECT,NO INTERRUPT CHANGE
 002B AA            147         MOV     R2,A            ,SCAN ROW SELECT REGISTER
 002C FA            148 SCANMX  MOV     A,R2            ,SCAN ROW SELECT TO ACCUMULATOR
 002D 3A            149         OUTL    P2,A            ,OUTPUT SCAN ROW SELECT TO PORT 2
 002E BB0F          150         MOV     R3,#SCANTM      ,SET DELAY FOR OUTPUT SCAN TIME
 0030 EB30          151 DELAY2. DJNZ    R3,DELAY2       ,DELAY
 0032 09            152         IN      A,P1            ,INPUT COLUMN STATUS FROM PORT 1 TO ACCUMULATOR
 0033 20            153         XCH     A,@R0           ,STORE NEW COLUMN STATUS SAVE OLD IN ACCUMULATOR
 0034 D0            154         XRL     A,@R0           ,COMPARE OLD WITH NEW COLUMN STATUS
 0035 AF            155         MOV     R7,A            ,SAVE COMPARE RESULT IN COMPARE REGISTER
 0036 C669          156         JZ      CHFFUL          ,IF THE SAME, CHECK IF FIFO IS FULL
                    157
                    158 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3 0              PAGE    5

 LOC  OBJ          LINE        SOURCE STATEMENT

                    159 ,*****************************************************************
                    160 ,
                    161 ,                        CHANGE WORD ENCODING
                    162 ,
                    163 ,THE FOLLOWING CODE IS THE CHANGE WORD ENCODING SECTION.  THIS
                    164 ;SECTION IS ONLY EXECUTED IF A CHANGE WAS DETECTED   THE COLUMN COUNTER
                    165 ;IS SET AND DECREMENTED TO DESIGNATE EACH OF THE 8 COLUMNS.  THE COMPARE
                    166 ,REGISTER IS LOOKED AT ONE BIT AT A TIME TO FIND THE EXACT LOCATION OF
                    167 ,THE CHANGE(S)   WHEN A CHANGE IS FOUND IT IS ENCODED BY GIVING IT A
                    168 ,COORDINATE FOR ITS LOCATION   THIS IS DONE BY COMBINING THE PRESENT VALUE
                    169 ,IN THE ROW SCAN SELECT REGISTER AND THE COLUMN COUNTER.  THE ACTUAL STATUS
                    170 ,OF THAT SENSOR IS ESTABLISHED BY LOOKING AT THE CORRESPONDING BYTE IN
                    171 ,THE MATRIX MAP   THIS STATUS IS COMBINED WITH THE COORDINATE TO ESTABLISH
                    172 ,THE CHANGE WORD   THE CHANGE WORD IS THEN STORED IN THE CHANGE WORD REGISTER
                    173 ,
                    174 ,*****************************************************************
                    175 ,
 0038 B808          176          MOV     R3,#08H          ,SET COLUMN COUNTER REGISTER TO 8
 003A CB            177 RRLOOK   DEC     R3               ,DECREMENT COLUMN COUNTER
 003B F0            178          MOV     A,@R0            ;COLUMN STATUS TO ACCUMULATOR
 003C 77            179          RR      A                ,ROTATE COLUMN STATUS RIGHT
 003D A0            180          MOV     @R0,A            ,ROTATED COLUMN STATUS BACK TO MATRIX MAP
 003E FF            181          MOV     A,R7             ,COMPARE REGISTER VALUE TO ACCUMULATOR
 003F 77            182          RR      A                ,ROTATE COMPARE VALUE RIGHT
 0040 AF            183          MOV     R7,A             ,ROTATED COMPARE VALUE TO COMPARE REGISTER
 0041 F245          184          JB7     ENCODE           ,TEST BIT 7 IF CHANGE DETECTED ENCODE CHANGE WORD
 0043 0469          185          JMP     CHFFUL           ,IF NO CHANGE IS DETECTED CHECK FOR FIFO FULL
 0045 FA            186 ENCODE   MOV     A,R2             ,SCAN ROW SELECT TO ACCUMULATOR  0000XXXX
 0046 330F          187          ANL     A,#0FH           ,ROTATE ONLY SCAN VALUE
 0048 E7            188          RL      A                ,ROTATE LEFT                     000XXXX0
 0049 E7            189          RL      A                ,ROTATE LEFT                     00XXXX00
 004A E7            190          RL      A                ,ROTATE LEFT                     0XXXX000
 004B 4B            191          ORL     A,R3             ,ESTABLISH MATRIX COORDINANT     0XXXXXXX
                    192                                   ,(OR) COLUMN COUNTER VALUE WITH ACCUMULATOR
 004C AE            193          MOV     R6,A             ,SAVE COORDINANT IN CHANGE WORD REGISTER
 004D F0            194          MOV     A,@R0            ,COLUMN STATUS FROM MATRIX MAP TO ACCUMULATOR
 004E 5380          195          ANL     A,#80H           ,0 ALL BITS BUT BIT 7
 0050 4E            196          ORL     A,R6             ,(OR) SENSOR STATUS WITH COORDINATE FOR COMPLETED CHANGE WORD
 0051 AE            197          MOV     R6,A             ,SAVE CHANGE WORD                XXXXXXXX
                    198
                    199 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0                PAGE    6

 LOC  OBJ          LINE        SOURCE STATEMENT

                    200 ;*****************************************************************
                    201 ,
                    202 ,                    FIFO-DBBOUT MANAGEMENT
                    203 ;
                    204 ;THE FOLLOWING CODE IS THE FIFO-DBBOUT MANAGEMENT SECTION OF THE
                    205 ;PROGRAM   THIS SECTION TAKES AN ENCODED CHANGE WORD AND LOADS IT INTO
                    206 ,THE FIFO   THE FIFO NOT EMPTY INTERRUPT IS THEN SET AND THE FIFO-IN
                    207 ;POINTER GETS UPDATED  A FIFO FULL CONDITION IS THEN CHECKED FOR AND
                    208 ,ROUTED ACCORDINGLY   IF BOTH THE FIFO AND OBF HAVE CHANGE WORDS THE
                    209 ;PROGRAM LOCKS UP UNTIL THIS HAS CHANGED.  IF THE FIFO ISNT FULL COLUMN
                    210 ;COUNTER= 0, FIFO EMPTY AND OBF CONDITIONS ARE CHECKED   THE FIFO-OUT
                    211 ,POINTER IS SET AND DBBOUT IS LOADED IF THE FIFO ISNT EMPTY AND OBF ISNT
                    212 ;SET.  IF THIS ISNT THE SITUATION, PROGRAM FLOW IS ROUTED BACK TO THE
                    213 ;THE SCAN AND COMPARE SECTION TO SCAN THE NEXT ROW
                    214 ,
                    215 ;*****************************************************************
                    216 ;
0052 FC             217 LOADFF: MOV     A,R4            ;FIFO INPUT ADDRESS TO ACCUMULATOR
0053 A9             218         MOV     R1,A            ,FIFO POINTER USED FOR INPUT
0054 FE             219         MOV     A,R6            ,CHANGE WORD TO ACCUMULATOR
0055 A1             220         MOV     @R1,A           ,LOAD FIFO AT FIFO INPUT ADDRESS
0056 2310           221 STATNE: MOV     A,#10H          ,BIT 4 FOR FIFO NOT EMPTY
0058 90             222         MOV     STS,A           ,WRITE TO STATUS REGISTER, FIFO NOT EMPTY
0059 8A20           223 INTRH1: ORL     P2,#20H         ,FIFO NOT EMPTY INTERRUPT PORT 2-5 HIGH
005B FA             224         MOV     A,R2            ,ROW SCAN SELECT TO ACCUMULATOR
005C 4320           225         ORL     A,#20H          ,SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
005E AA             226         MOV     R2,A            ;ROW SCAN SELECT REGISTER
005F 232F           227 ADJFIN  MOV     A,#FIFOTA       ,FIFO TOP ADDRESS TO ACCUMULATOR
0061 DC             228         XRL     A,R4            ,COMPARE WITH CURRENT FIFO INPUT ADDRESS
0062 C667           229         JZ      RSFFIN          ,IF THE SAME RESET FIFO INPUT REGISTER
0064 1C             230         INC     R4              ,NEXT FIFO INPUT ADDRESS
0065 0469           231         JMP     CHFFUL          ,CHECK FIFO FULL
0067 BC08           232 RSFFIN  MOV     R4,#FIFOBA      ,RESET FIFO INPUT REGISTER,BOTTOM OF FIFO
0069 FC             233 CHFFUL  MOV     A,R4            ,FIFO INPUT ADDRESS TO ACCUMULATOR
006A DD             234         XRL     A,R5            ,COMPARE INPUT WITH OUTPUT FIFO ADDRESS
006B 967D           235         JNZ     CHCNTR          ,IF NOT SAME CHECK COLUMN COUNTER VALUE
006D 866D           236 CHOBF1  JOBF    CHOBF1          ,IF OBF IS 1 THEN CHECK OBF
006F 232F           237 ADJFOT  MOV     A,#FIFOTA       ,FIFO TOP ADDRESS TO ACCUMULATOR
0071 DD             238         XRL     A,R5            ,COMPARE TOP TO OUTPUT FIFO ADDRESS
0072 C677           239         JZ      RSFFOT          ,IF THE SAME RESET FIFO OUTPUT REGISTER
0074 1D             240         INC     R5              ,NEXT FIFO OUTPUT ADDRESS
0075 0479           241         JMP     LOADDB          ,LOAD DBBOUT
0077 BD08           242 RSFFOT  MOV     R5,#FIFOBA      ,RESET FIFO OUTPUT ADDRESS TO BOTTOM OF FIFO
0079 FD             243 LOADDB  MOV     A,R5            ,OUTPUT FIFO ADDRESS TO ACCUMULATOR
007A A9             244         MOV     R1,A            ,FIFO POINTER USED FOR OUTPUT
007B F1             245         MOV     A,@R1           ,CHANGE WORD TO ACCUMULATOR
007C 02             246         OUT     DBB,A           ,CHANGE WORD TO DBBOUT
007D FB             247 CHCNTR  MOV     A, R3           ,COLUMN COUNTER TO ACCUMULATOR
007E 963A           248         JNZ     RRLOOK          ,IF NOT 0 FINISH CHANGE WORD ENCODING
0080 2308           249 CHFFEM  MOV     A,#FIFOBA       ,FIFO BOTTOM ADDRESS TO ACCUMULATOR
                    250
                    251 $EJECT
```

AFN-01536A

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0                PAGE    7


  LOC  OBJ          LINE         SOURCE STATEMENT

  0082 DC            252           XRL     A,R4            ;COMPARE FIFO INPUT ADDRESS WITH FIFO BOTTOM ADD
  0083 C68C          253           JZ      ADJFEM          ;IF THE SAME, ADJUST TO CHECK FOR FIFO EMPTY
  0085 FC            254           MOV     A,R4            ;FIFO INPUT ADDRESS TO ACCUMULATOR
  0086 07            255           DEC     A               ;DECREMENT FIFO INPUT ADDRESS IN ACCUMULATOR
  0087 DD            256           XRL     A,R5            ;COMPARE INPUT TO OUTPUT FIFO ADDRESSES
  0088 C691          257           JZ      STATMT          ;IF SAME,WRITE STATUS REGISTER FOR FIFO EMPTY
  008A 049C          258           JMP     CHOBF2          ;CHECK OBF
  008C 232F          259 ADJFEM    MOV     A,#FIFOTA       ;FIFO TOP ADDRESS TO ACCUMULATOR
  008E DD            260           XRL     A,R5            ;COMPARE TOP TO OUTPUT FIFO ADDRESS
  008F 969C          261           JNZ     CHOBF2          ;IF NOT SAME THEN FIFO IS NOT EMPTY, CHECK OBF
  0091 2300          262 STATMT    MOV     A,#00H          ;CLEAR BIT 0 FOR FIFO EMPTY
  0093 90            263           MOV     STS,A           ;WRITE TO STATUS REGISTER
  0094 9ADF          264 INTRLO:   ANL     P2,#0DFH        ;FIFO EMPTY, INTERRUPT PORT 2-5 LOW
  0096 FA            265           MOV     A,R2            ;SCAN ROW SELECT TO ACCUMULATOR
  0097 53DF          266           ANL     A,#0DFH         ;SAVE INTERRUPT, NO CHANGE TO SCAN VALUE
  0099 AA            267           MOV     R2,A            ;SCAN ROW SELECT REGISTER
  009A 041D          268           JMP     ADJREG          ;ADJUST REGISTERS
  009C B61D          269 CHOBF2:   JOBF    ADJREG          ;IF OBF=1 THEN ADJUST REGISTERS
  009E 046F          270           JMP     ADJFOT          ;ADJUST FIFO OUT ADDRESS TO LOAD DBBOUT
                     271
                     272           END

USER SYMBOLS
ADJFEM 008C    ADJFIN 005F    ADJFOT 006F    ADJREG 001D    CHCNTR 007D    CHFFEM 0080    CHFFUL 0069    CHOBF1 006D
CHOBF2 009C    DELAY2 0030    ENCODE 0045    FIFOBA 0008    FIFOTA 002F    FILLMX 000D    INITMX 0000    INTRH1 0059
INTRLO 0094    LOADDB 0079    LOADFF 0052    OBFINT 0018    RRLOOK 003A    RSETRG 0026    RSFFIN 0067    RSFFOT 0077
SCANMX 002C    SCANTM 000F    STATMT 0091    STATNE 0056

ASSEMBLY COMPLETE,   NO ERRORS
```

PROGRAMMABLE KEYBOARD INTERFACE

- **Simultaneous Keyboard and Display Operations**
- **Interface Signals for Contact and Capacitive Coupled Keyboards**
- **128-Key Scanning Logic**
- **10.7msec Matrix Scan Time for 128 Keys and 6MHz Clock**
- **Eight Character Keyboard FIFO**
- **N-Key Rollover with Programmable Error Mode on Multiple New Closures**
- **Sixteen or Eight Character Seven-Segment Display Interface**
- **Right or Left Entry Display RAM**
- **Depress/Release Mode Programmable**
- **Interrupt Output on Key Entry**

This application is a general purpose programmable keyboard and display interface device designed for use with 8-bit microprocessors like the MCS-80 and MCS-85. The keyboard portion can provide a scanned interface to 128-key contact or capacitive-coupled keyboards. The keys are fully debounced with N-key rollover and programmable error generation on multiple new key closures. Keyboard entries are stored in an 8-character FIFO with overrun status indication when more than 8 characters are entered. Key entries set an interrupt request output to the master CPU.

The display portion of the UPI-41A provides a scanned display interface for LED, incandescent and other popular display technologies. Both numeric displays and simple indicators may be used. The UPI-41A has a 16×4 display RAM which can be

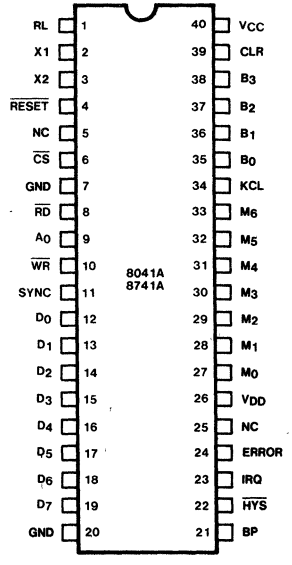


Figure 1. Pin Configuration

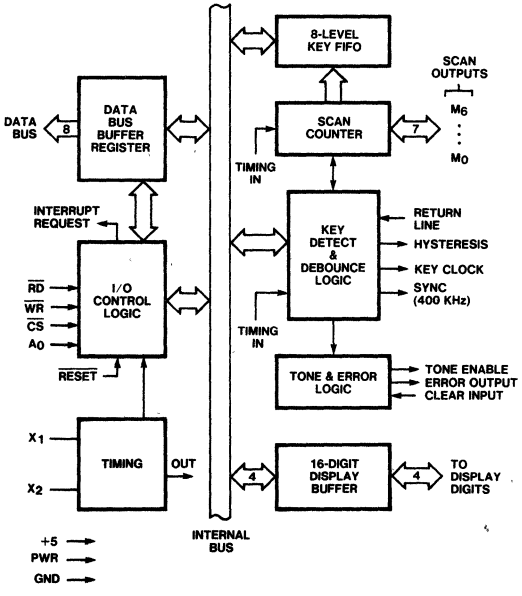


Figure 2. Block Diagram

loaded or interrogated by the CPU. Both right entry calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto increment of the display RAM address.

ORDERING INFORMATION:

This part may be ordered as an 8041A with ROM code number 8278. The source code is available through Insite.

Throughout this application of the UPI-41A, it will be referred to by its ROM code number, 8278. The 8278 is packaged in a 40-pin DIP. The following is a brief functional description of each pin.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 1.

I/O Control and Data Buffers

The I/O control section uses the $\overline{CS}$, A_0, $\overline{RD}$, and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers (see Table 2). All data flow to and from the 8278 is enabled by $\overline{CS}$. The 8-bits of information being transferred by the CPU is identified by A_0. A logic one means information is command or status. A logic zero means the information is data. $\overline{RD}$ and $\overline{WR}$ determine the direction of data flow through the Data Bus Buffer (DBB). The

Table 1. Pin Description

Signal	Pin. No.	Type	Name and Function
D_0-D_7	12-19	I/O	**Data Bus:** Three-state, bi-directional data bus lines used to transfer data and commands between the CPU and the 8278.
$\overline{WR}$	10	I	**Write:** Write strobe which enables the master CPU to write data and commands between the CPU and the 8278.
$\overline{RD}$	8	I	**Read:** Read strobe which enables the master CPU to read data and status from the 8278 internal registers.
$\overline{CS}$	6	I	**Chip Select:** Chip select input used to enable reading and writing to the 8278.
A_0	9	I	**Control/Data:** Address input used by the CPU to indicate control or data.
$\overline{RESET}$	4	I	**Reset:** A low signal on this pin resets the 8278.
X_1, X_2	2,3	I	**Freq. Reference Inputs:** Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency.
IRQ	23	O	**Interrupt Request:** Interrupt Request Output to the master CPU. In the keyboard mode the IRQ line goes low with each FIFO read and returns high if there is still information in the FIFO or an ERROR has occurred.
M_0-M_6	27-33	O	**Matrix Scan Lines:** Matrix scan outputs. These outputs control a decoder which scans the key matrix columns and the 16 display digits. Also, the Matrix scan outputs are used to multiplex the return lines from the key matrix.
RL	1	I	**Keyboard Return Line:** Input from the multiplexer which indicates whether the key currently being scanned is closed.
$\overline{HYS}$	22	O	**Hysteresis:** Hysteresis output to the analog detector. (Capacitive keyboard configuration). A "0" means the key currently being scanned has already been recorded.
KCL	34	O	**Key Clock:** Key Clock output to the analog detector (capacitive keyboard configuration) used to reset the detector before scanning a key.
SYNC	11	O	**Output Clock:** High frequency (400 kHz) output signal used in the key scan to detect a closed key (capacitive keyboard configuration).
B_0-B_3	35-38	O	**Display Outputs:** These four lines contain binary coded decimal display information synchronized to the keyboard column scan. The outputs are for multiplexed digital displays.
ERROR	24	O	**Error Signal:** This line is high whenever two new key closures are detected during a single scan or when too many characters are entered into the keyboard FIFO. It is reset by a system RESET pulse or by a "1" input on the CLR pin or by the CLEAR ERROR command.
CLR	39	I	**Clear Error:** Input used to clear an ERROR condition in the 8278.
BP	21	O	**Tone Enable:** Tone enable output. This line is high for 10ms following a valid key closure; it is set high and remains high during an ERROR condition.
V_{CC}, V_{DD}	40,26	I	**Power:** +5 volt power input: +5V ± 10%.
GND	20,7	I	**Ground:** Signal ground.

DBB register is a bi-directional 8-bit buffer register which connects the internal 8278 bus buffer register to the external bus. When the chip is not selected ($\overline{CS}$ = 1) the DBB is in the high impedance state. The DBB acts as an input when ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$) = (1, 0, 0) and an output when ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$) = (0, 1, 0).

Table 2. I/O Control and Data Buffers

$\overline{CS}$	A_0	$\overline{WR}$	$\overline{RD}$	Condition
0	0	1	0	Read DBB Data
0	1	1	0	Read STATUS
0	0	0	1	Write Data to DBB
0	1	0	1	Write Command to DBB
1	X	X	X	Disable 8278 Bus, High Impedance

Scan Counter

The scan counter provides the timing to scan the keyboard and display. The four MSB's (M_3-M_6) scan the display digits and provide column scan to the keyboard via a 4 to 16 decoder. The three LSB's (M_0-M_2) are used to multiplex the row return lines into the 8278.

Keyboard Debounce and Control

The 8278 system configuration is shown in Figure 3. The rows of the matrix are scanned and the outputs are multiplexed by the 8278. When a key closure is detected, the debounce logic waits about 12 msec to check if the key remains closed. If it does, the address of the key in the matrix is transferred into a FIFO buffer.

FIFO and FIFO Status

The 8278 contains an 8×8 FIFO character buffer. Each new entry is written into a successive FIFO location and each is then read out in the order of entry. A FIFO status register keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or key entries will be recognized as an error. The status can be read by a $\overline{RD}$ with $\overline{CS}$ low and A_0 high. The status logic also provides a IRQ signal to the master processor whenever the FIFO is not empty.

Display Address Registers and Display RAM

The Display Address registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The display RAM can be directly read by the CPU after the correct mode and address is set. Data entry to the display can be set to either left or right entry.

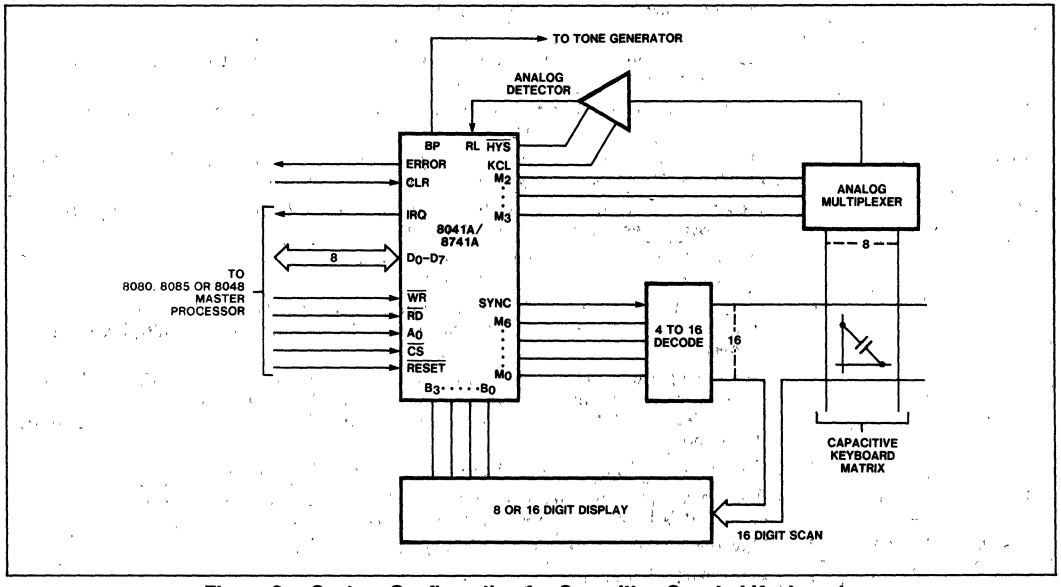


Figure 3. System Configuration for Capacitive-Coupled Keyboard

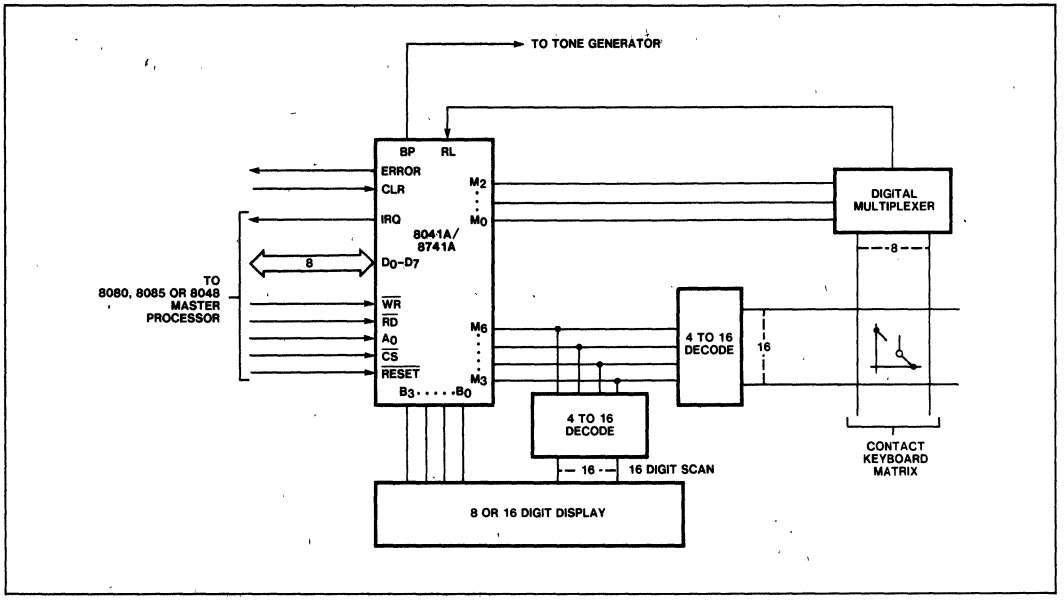
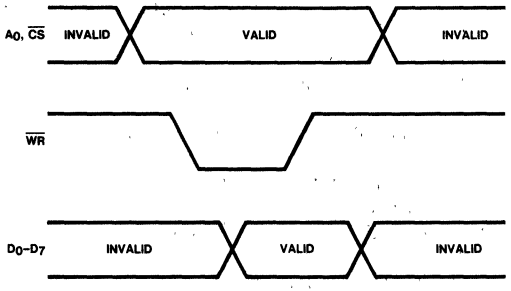


Figure 4. System Configuration for Contact Keyboard

COMMANDS

The 8278 operating mode is programmed by the master CPU using the A_0, WR and D_0-D_7 inputs as shown below:



The master CPU presents the proper command on the D_0-D_7 data lines with A_0 =1 and then sends a $\overline{WR}$ pulse. The command is latched by the 8278 on the rising edge of the $\overline{WR}$ and is decoded internally to set the proper operating mode. See the 8041A/8741A data sheet for timing details.

Command Summary

KEYBOARD/DISPLAY MODE SET

CODE	0	0	0	N	E	I	D	K

Where the mode set bits are defined as follows:

K—the keyboard mode select bit
- 0—normal key entry mode
- 1—special function mode: Entry on key closure and on key release

D—the display entry mode select bit
- 0—left display entry
- 1—right display entry

I—the interrupt request (IRQ) output enable bit.
- 0—enable IRQ output
- 1—disable IRQ output

E—the error mode select bit
- 0—error on multiple key depression
- 1—no error on multiple key depression

N—the number of display digits select
- 0—16 display digits
- 1—8 display digits

NOTE:
The default mode following a RESET input is all bits zero:

0	0	0	0	0	0	0	0

READ FIFO COMMAND

CODE	0	1	0	0	0	0	0	0

READ DISPLAY COMMAND

CODE	0	1	1	AI	A_3	A_2	A_1	A_0

Where AI indicates Auto Increment and A_3-A_0 is the address of the next display character to be read out.

AI = 1 AUTO increment
AI = 0 no AUTO increment

WRITE DISPLAY COMMAND

CODE	1	0	0	AI	A_3	A_2	A_1	A_0

Where AI indicates Auto Increment and A_3-A_0 is the address of the next display character to be written.

CLEAR/BLANK COMMAND

CODE	1	0	1	UD	BD	CD	CF	CE

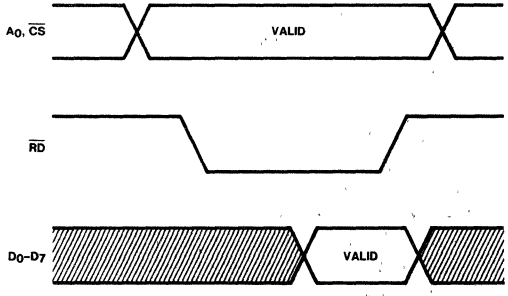
Where the command bits are defined as follows:

CE = Clear ERROR
CF = Clear FIFO
CD = Clear Display to all High
BD = Blank Display to all High
UD = Unblank Display

The display is cleared and blanked following a Reset.

Status Read

The status register in the 8278 can be read by the master CPU using the A_0, $\overline{RD}$, and D_0-D_7 inputs as shown below:



The 8278 places 8-bits of status information on the D_0-D_7 lines following (A_0, $\overline{CS}$, $\overline{RD}$) = 1, 0, 0 inputs from the master.

Status Format

S_3	S_2	S_1	S_0	B	KE	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Where the status bits are defined as follows:

IBF = Input Buffer Full Flag
OBF = Output Buffer Full Flag
KE = Keyboard Error Flag (multiple depression)
B = BUSY Flag
S_3-S_0 = FIFO Status

STATUS DESCRIPTION

The S_3-S_0 status bits indicate the number of entries (0 to 8) in the 8-level FIFO. A FIFO overrun will lock status at 1111. The overrun condition will prevent further key entries until cleared.

A multiple key closure error will set the KE flag and prevent further key entries until cleared.

The IBF and OBF flags signify the status of the 8278 data buffer registers used to transfer information (data, status or commands) to and from the master CPU.

The IBF flag is set when the master CPU writes Data or Commands to the 8278. The IBF flag is cleared by the 8278 during its response to the Data or Command.

The OBF flag is set when the 8278 has output data ready for the master CPU. This flag is cleared by a master CPU Data READ.

The Busy flag in the status register is used as a LOCKOUT signal to the master processor during response to any command or data write from the master.

The master must test the Busy flag before each read (during a sequence) to be sure that the 8278 is ready with valid DATA.

The ERROR and TONE outputs from the 8278 are set high for either type of error. Both types of error are cleared by the CLR input, by the CLEAR ERROR command, or by a reset. The FIFO and Display buffers are cleared independently of the Errors.

FIFO status is used to indicate the number of characters in the FIFO and to indiate whether an error has occurred. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO. The character read will be the last one entered. FIFO status will remain at 0000 and the error condition will not be set.

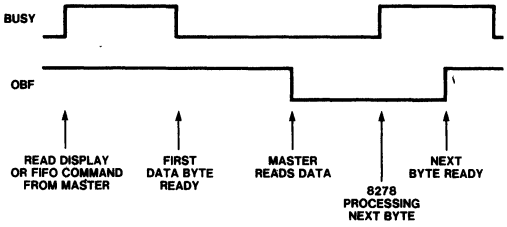
Data Read

The master CPU can read DATA from the 8278 FIFO or Display buffers by using the A_0, $\overline{RD}$, and D_0-D_7 inputs.

The master sends a $\overline{RD}$ pulse with $A_0 = 0$ and $CS = 0$ and the 8278 responds by outputting data on lines D_0-D_7. The data is strobed by the trailing edge of $\overline{RD}$.

DATA READ SEQUENCE

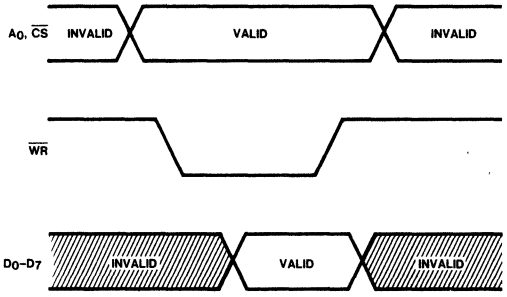
Before reading data, the master CPU must send a command to select FIFO or Display data. Following the command, the master must read STATUS and test the BUSY flag and the OBF flag to verify that the 8278 has responded to the previous command. A typical DATA READ sequence is as follows:



After the first read following a Read Display or Read FIFO command, successive reads may occur as soon as OBF rises.

Data Write

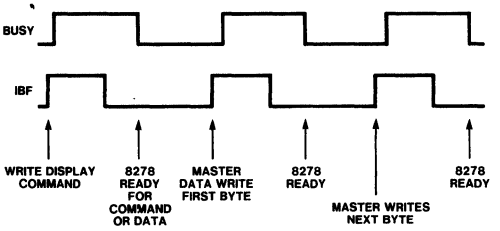
The master CPU can write DATA to the 8278 Display buffers by using the A_0, $\overline{WR}$ and D_0-D_7 inputs as follows:



The master CPU presents the Data on the D_0-D_7 lines with A_0=0 and then sends a $\overline{WR}$ pulse. The data is latched by the 8278 on the rising edge of $\overline{WR}$.

DATA WRITE SEQUENCE

Before writing data to the 8278, the master CPU must first send a command to select the desired display entry mode and to specify the address of the next data byte. Following the commands, the master must read STATUS and test the BUSY flag (B) and IBF flag to verify that the 8278 has responded. A typical sequence is shown below.



INTERFACE CONSIDERATIONS

Scanned Keyboard Mode

With N-key rollover each key depression is treated independently from all others. When a key is depressed the debounce logic waits for a full scan of 128 keys and then checks to see if the key is still down. If it is, the key is entered into the FIFO.

If two key closures occur during the same scan the ERROR output is set, the KE flag is set in the Status word, the TONE output is activated and IRQ is set, and no further inputs are accepted. This condition is cleared by a high signal on the CLEAR input or by a system $\overline{RESET}$ input or by the CLEAR ERROR command.

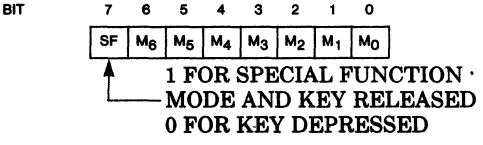
In the special function mode both the key closure and the key release cause an entry to the FIFO. The release is entered with the MSB=1.

Any key entry triggers the TONE output for 10ms.

The $\overline{HYS}$ and KCL outputs enable the analog multiplexer and detector to be synchronized for interface to capacitive coupled keyboards.

Data Format

In the scanned keyboard mode, the code entered into the FIFO corresponds to the position or address of the switch in the keyboard. The MSB is relevant only for special function keys in which code "0" signifies closure and "1" signifies release. The next four bits are the column count which indicates which column the key was found in. The last three bits are from the row counter.



Display

Display data is entered into a 16X4 display register and may be entered from the left, from the right or

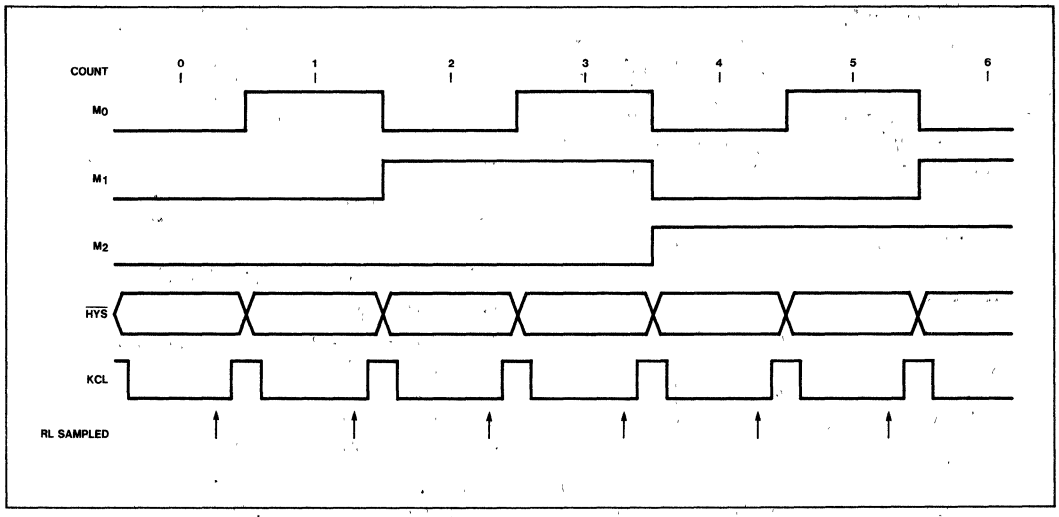


Figure 5. Keyboard Timing

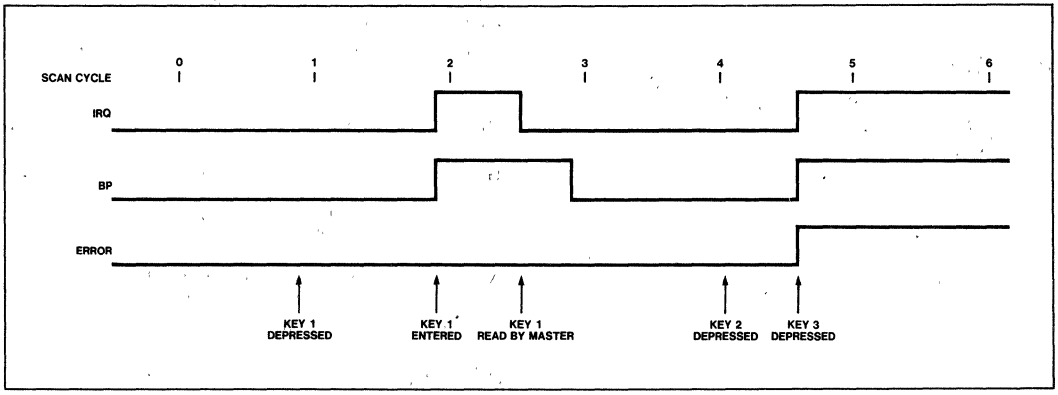


Figure 6. Key Entry and Error Timing

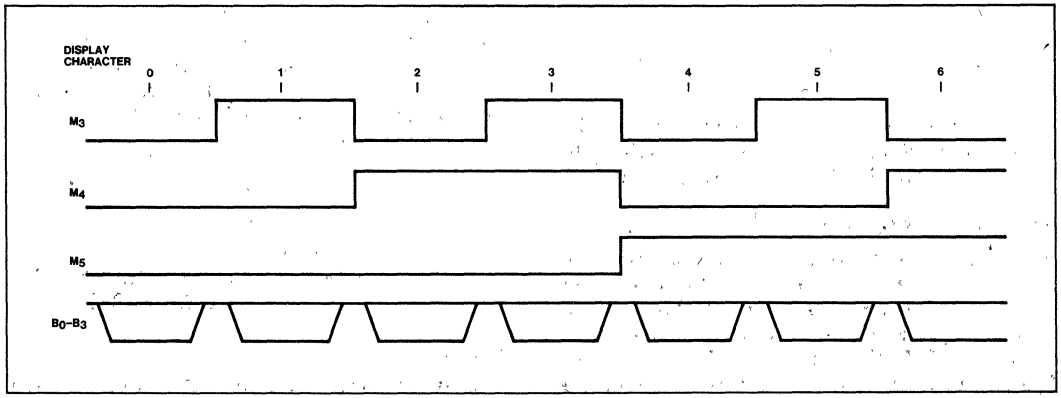


Figure 7. Display Timing

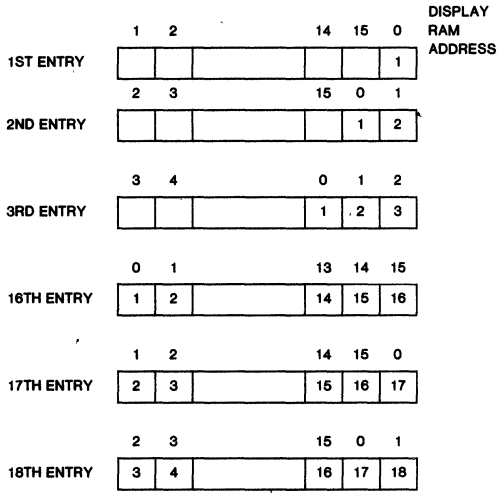
into specific locations in the display register. A new data character is put out on B_0-B_3 each time the M_6-M_3 lines change (i.e., once every 0.75ms with a 6 MHz crystal). Data is blanked during the time the column select lines change by raising the display outputs. Output data is positive true.

LEFT ENTRY

The left entry mode is the simplest display format in that each display position in the display corresponds to a byte (or nibble) in the Display RAM. ADDRESS 0 in the RAM is the left-most display character and ADDRESS 15 is the right-most display character. Entering characters from position zero causes the display to fill from the left. The 17th character is entered back in the left-most position and filling again proceeds from there.

RIGHT ENTRY

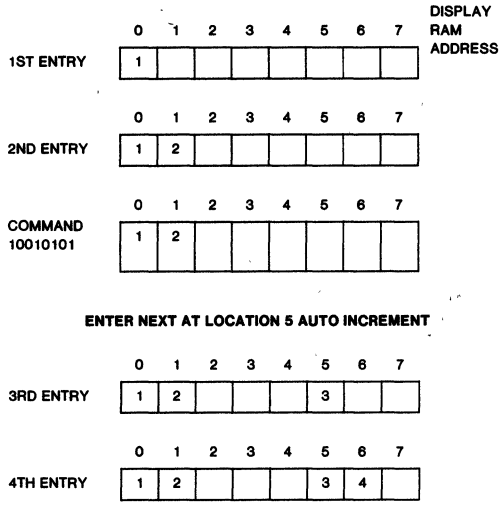
Right entry is the method used by most electronic calculators. The first entry is placed in the right-most display character. The next entry is also placed in the right-most character after the display is shifted left one character. The left-most character is shifted off the end and is lost.



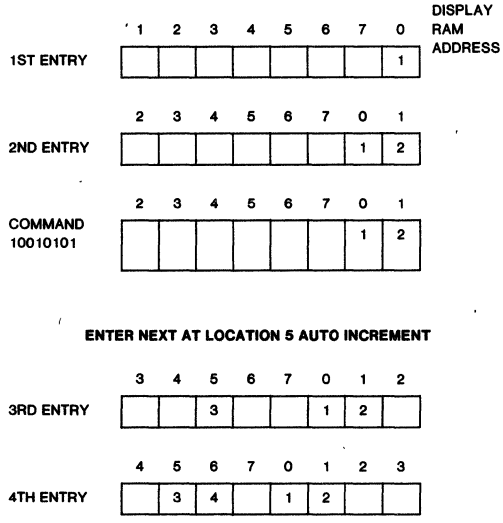
Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM ADDRESS 0 with sequential entry is recommended. A Clear Display command should be given before display data is entered if the number of data characters is not equal to 16 (or 8) in this mode.

AUTO INCREMENT

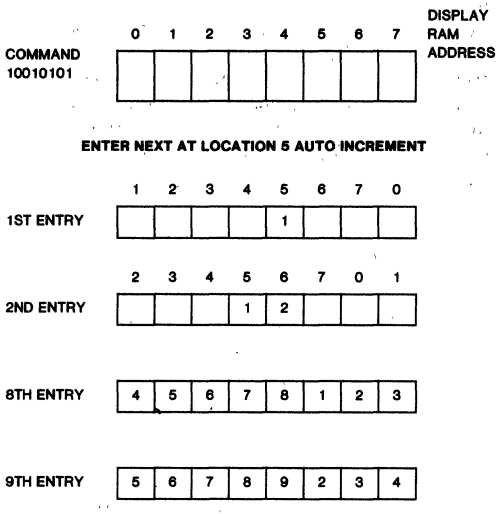
In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Left Entry—Auto Increment mode has no undesirable side effects and the result is predictable:



In the Right Entry mode, Auto Incrementing and non-Incrementing have the same effect as in the Left Entry except that the address sequence is interrupted.



Starting at an arbitrary location operates as shown below.



Entry appears to be from the initial entry point.

intel®

APPLICATION NOTE

AP-161

September 1983

Complex Peripheral Control with the UPI-42

Christopher Scott
Applications Engineer

NOVEMBER 1983
ORDER NUMBER: 230795-001

COMPLEX PERIPHERAL CONTROL WITH THE UPI-42

TABLE OF CONTENTS

FIGURES

230795-001

FLOW CHARTS

Addtional sources of information on Intel's UPI devices;

"UPI User's Manual"
Includes the following Application Notes;
Programmable Keyboard Interface
Using the 8295 Dot Matrix Printer Controller
An 8741A/8041A Digital Cassette Controller

"8048 Family Applications Handbook"

"1983 Microprocessor and Peripheral Handbook"

"MCS-48 and UPI-41A/42 Assembly Language Manual"

"Specifications for Impact Dot Matrix Printer Model-3210", Epson, Jan 8, 1981

INTRODUCTION

The UPI-42 is the newest member of Intel's Universal Peripheral Interface (UPI) microcomputer family. It represents a significant growth in UPI capabilities, resulting in a broader spectrum of applications. The UPI-42 incorporates twice the EPROM/ROM of the UPI-41A, 2048 vs 1024 bytes, twice the RAM, 128 vs 64 bytes, and operates at a maximum speed twice that of the UPI-41A, i.e. 12 MHz vs 6 MHz. The ROM based 8042 and the EPROM based 8742 provide more highly integrated solutions for complex stepping motor and dot matrix printer applications. Those applications previously requiring a microprocessor plus a UPI chip can now be implemented entirely with the UPI-42.

The software features of the UPI-42, such as indirect Data and Program Memory addressing, two independent and selectable 8 byte register banks, and directly software testable I/O pins, greatly simplify the external interface and software flow. The software and hardware design of the UPI-42 allows a complex peripheral to be controlled with a minimum of external hardware.

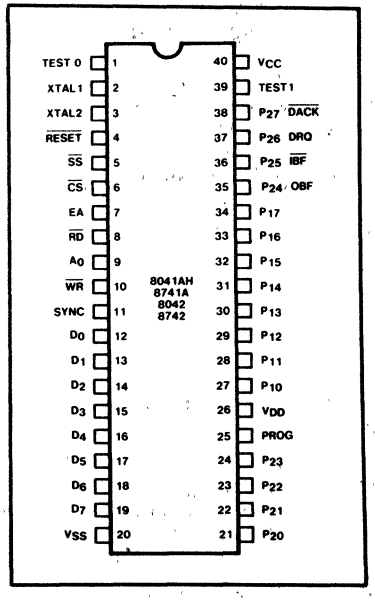


Figure 1. UPI-42 Pin Configuration

Many microcomputer systems need real time control of peripheral devices such as a printer, keyboard, complex motor control or process control. These medium speed but still time consuming tasks require a fair amount of system software overhead. This processing burden can be reduced by using a dedicated peripheral control processor

Until recently, the dedicated control processor approach was usually not cost effective due to the large number of components needed; CPU, RAM, ROM, I/O, and Timer/Counters. To help make the approach more cost effective, in 1977 Intel introduced the UPI-41 family of Universal Peripheral Interface controllers consisting of an 8041 (ROM) device and an 8741 (EPROM) device. These devices integrated the common microprocessor system functions into one 40 pin package. The UPI-42 family, consisting of the 8042 and 8742, further extends the UPI's cost effectiveness through more memory and higher speed.

Another member of the UPI family is the Intel 8243 Input/Output Expander chip. This chip provides the UPI-41A and UPI-42 with up to 16 additional independently programmable I/O lines, and interfaces directly to the UPI-41A/42. Up to seven 8243s can be cascaded to provide over 100 I/O lines.

The UPI is a single chip microcomputer with a standard microprocessor interface. The UPI's architecture, illustrated in Figure 3, features on-chip program memory, ROM (8041A/8042) or EPROM (8741A/8742), data memory (RAM), CPU, timer/counter, and I/O. Special interface registers are provided which enable the UPI to function as a peripheral to an 8-bit central processor.

Using one of the UPI devices, the designer simply codes his proprietary peripheral control algorithm into the UPI device itself, rather than into the main system software. The UPI device then performs the peripheral control task while the host processor simply issues commands and transfers data. With the proliferation of microcomputer systems, the use of UPIs or slave microprocessors to off load the main system microprocessor has become quite common.

This Application Note describes how the UPI-42 can be used to control dot matrix printing and the printer mechanism, using stepper motors for carriage/print head assembly and paper feed motion. Previous Intel Application Notes AP-27, AP-54, and AP-91 describe using intelligent processors and peripherals to control single solenoid driven printer mechanisms with 80 character line buffering and bidirectional printing. This Application Note expands on these previous themes and extends the concept of complex device control by incorporating full 80 character line buffering, bidirectional printing, as well as drive and feedback control of two four phase stepper motors.

The Application Note assumes that the reader is familiar with the 8042/8742 and 8243 Data Sheets, and UPI-41A/42 Assembly Language. Although some background information is included, it also assumes a basic understanding of stepper motors and dot matrix printer mechanisms. A complete software listing is included in Appendix A.

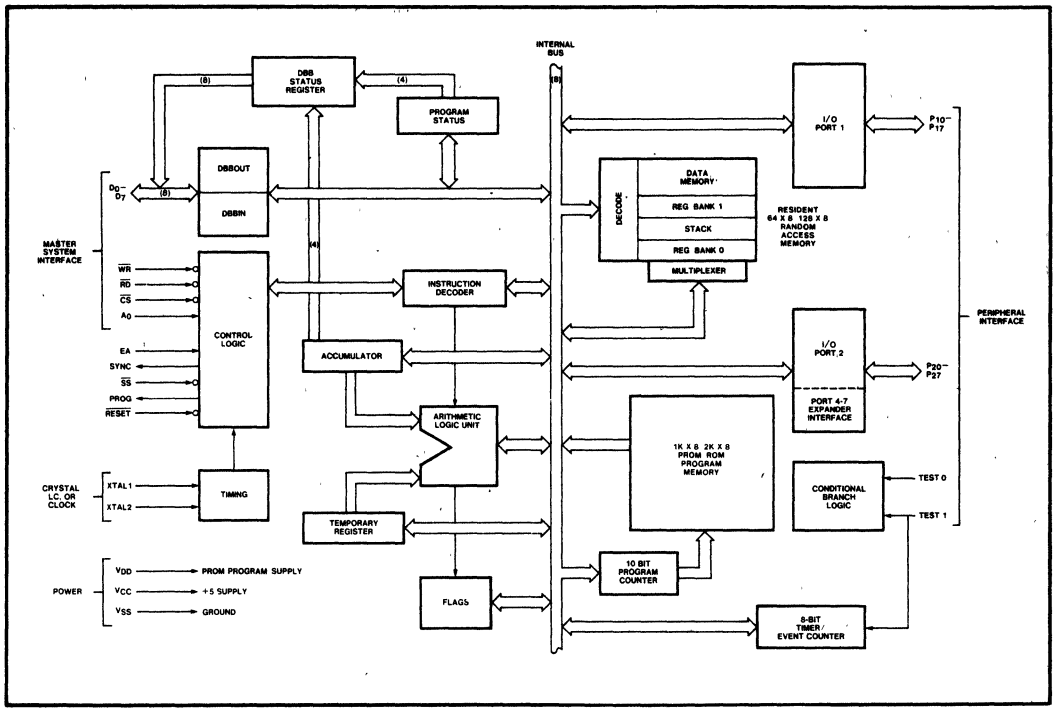


Figure 2. UPI-42 Block Diagram

DOT MATRIX PRINTING

A dot matrix printer print head typically consists of seven to nine solenoids, each of which drives a stiff wire, or hammer, to impact the paper through an inked ribbon. Characters are formed by firing the solenoids to form a matrix of "dots" (impacts of the wires). Figure 4 shows how the character "E" is formed using a 5 x 7 matrix. The columns are labeled C1 through C5, and the rows R1 through R7. The print head moves left-to-right across the paper, so that at time T1 the head is over column C1. The character is formed by activating the proper solenoids as the print head sweeps across the character position.

Dot matrix printers are a cost effective way of providing good quality hard copy output for microcomputer systems. There is an ever increasing demand for the moderately priced printer to provide more functionality with improved cost and performance. Using stepper motors to control the paper feed and carriage/print head assembly motion is one way of enabling the dot matrix printer to provide more capabilities, such as expanded or contracted characters, dot or line graphics, variable line and character spacing, and subscript or superscript printing.

However, stepper motors require fairly complex contol algorithms. Previous solutions involved the use of a main CPU, UPI, RAM, ROM, and I/O onboard the peripheral. The CPU acted as supervisor and used parallel processing to achieve accurate stepper motor control via a UPI, character buffering via the I/O device, RAM, and ROM. The CPU performed real-time decoding of each character into a dot matrix pattern. This Application Note demonstrates that the increased memory and performance of the UPI-42 facilitates integrating these control functions to reduce the cost and component count.

THE PRINTER MECHANISM

The printer mechanism used in this application is the Epson Model 3210. It consists of four basic subassemblies; the chassis or frame, the paper feed mechanism and stepper motor, the carriage motion mechanism and stepper motor, and the print head assembly.

The paper feed mechanism is a tractor feed type. It accomodates up to 8.5 inch wide paper (not including tractor feed portion). There is no platen as such; the paper is moved through the paper guide by two sprocketed wheels mounted on a center sprocket shaft. The sprocket shaft is driven by a four phase stepper motor. The rotation of the stepper motor is transmitted to the sprocket shaft through a series of four reduction gears.

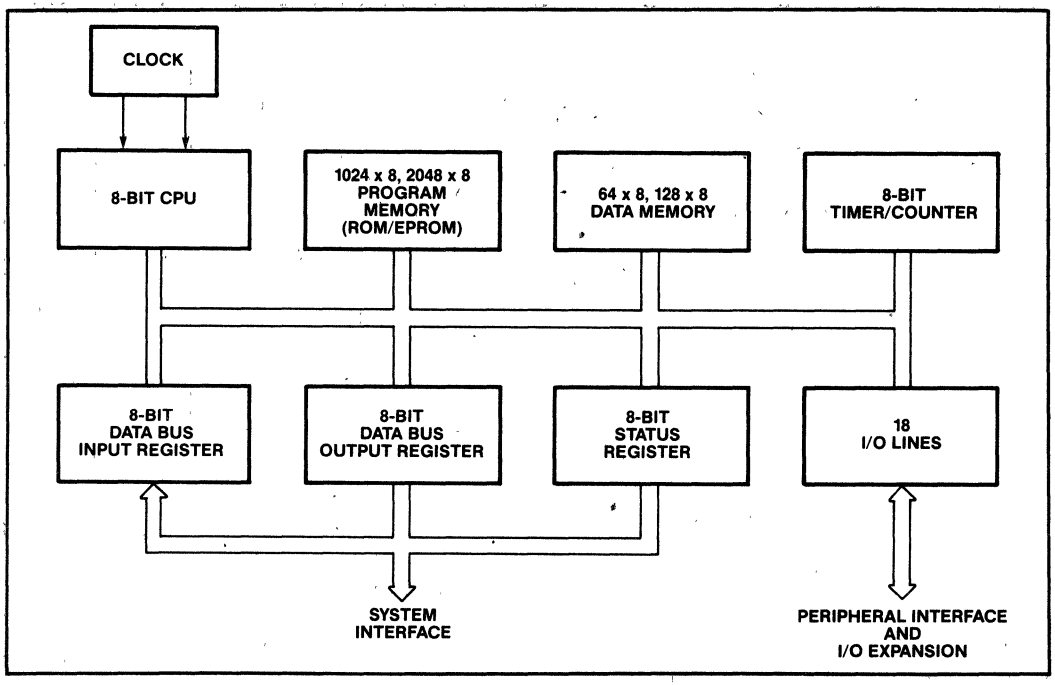


Figure 3. UPI-41A, 42 Functional Block Diagram

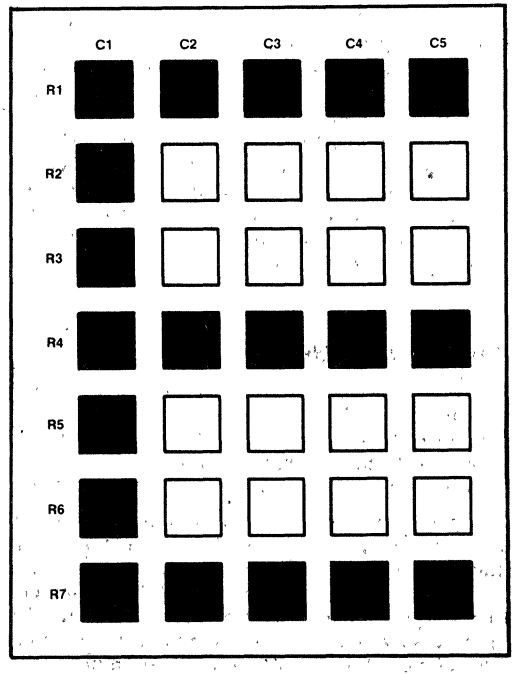


Figure 4. Character E in 5 x 7 Dot Matrix Format

The carriage motion mechanism consists of another four phase stepper motor which controls the left-to-right or right-to-left print head assembly motion. The print speed is 80 CPS maximum. Both the speed of the stepper motor and the movement of the print head assembly are independently controllable in either direction. The rotation of the stepper motor is converted to the linear motion of the print head assembly via a series of reduction gears and a toothed drive belt. The drive belt also controls a second set of reduction gears which advances the print ribbon as the print head assembly moves.

Two optical sensors provide feedback information on the carriage assembly position and speed. The first of these optical sensors, called the 'HOME RESET' or HR, is mounted near the left-most physical position to which the print head assembly can move. As the print head assembly approaches the left-most position, a flange on the print head assembly interferes with the light source and sensor, causing the output of the sensor to shift from a logic level one to zero. The right-most printer position is monitored in software rather than by another optical sensor. The right-most print position is a function of the number of characters printed and the distance required to print them.

The second optical sensor, called the 'PRINT TIMING SIGNAL' or PTS, provides feedback on carriage stepper motor velocity and relative position within a

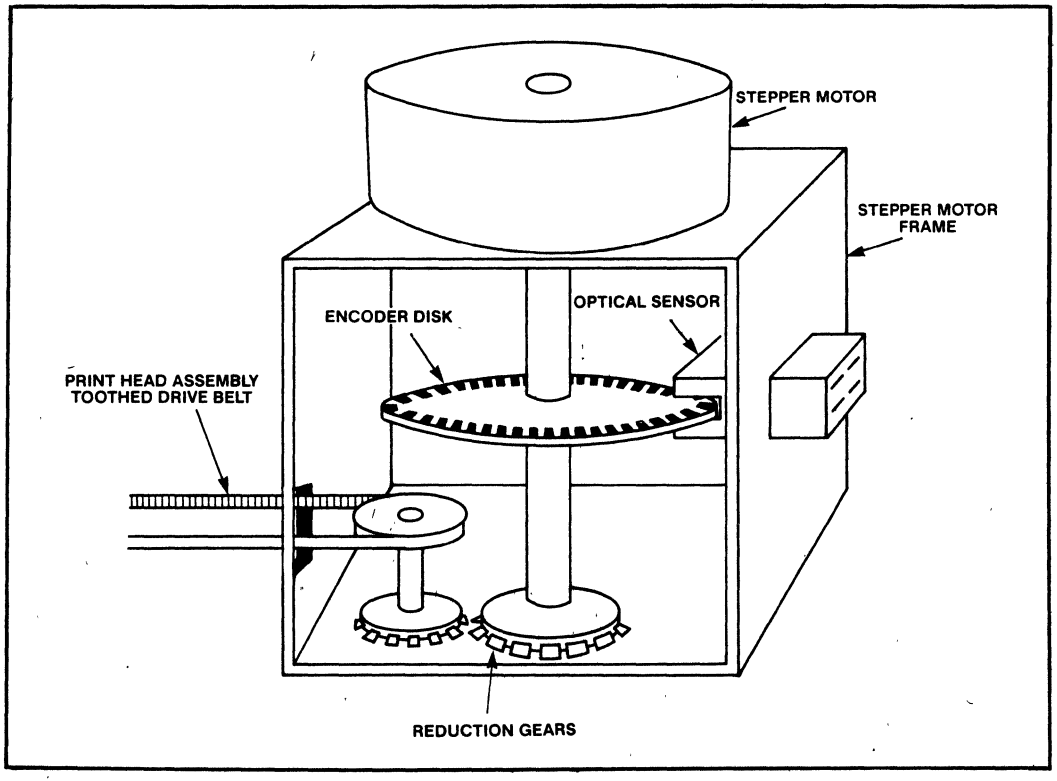


Figure 5. Carriage Stepper Motor Assembly

given step of the motor. The feedback is generated by the optical sensor as an "encoder disk" moves across it. Figure 5 illustrates the carriage stepper motor, optical sensor, encoder disk and reduction gears, and drive belt assembly. The optical sensor outputs a pulse train with the same period as the phase shift signal used to drive the stepper, but slightly out of phase with it when the motor is at a constant speed (see Software Functional Block: Phase Shift Data for additional details). The disk acts as a timing wheel, providing feedback to the UPI software of the carriage speed, position, and optimum position for energizing the print head solenoids. The two optical sensors are monitored under software and provide the critical feedback needed to control the print head assembly and paper feed motion accurately. The process of stepper motor drive and control via feedback signals is called "closed loop" stepper motor control, and is covered in more detail in the software discussion.

The print head assembly consists of nine solenoids and nine wires or hammers. Figure 6 illustrates a print head assembly. The available dot matrix measures 9 x 9. This large matrix enables the Epson 3210 print mechanism to print a variety of character fonts, such as expanded or contracted characters, as well as line or block graphics (see Appendix B, Printer Enhancements). It also facilitates printing lower case ASCII characters with "lower case descenders." That is to say, certain lower case letters (e.g. y, p, etc.) will print below the bottom part of all upper case letters.

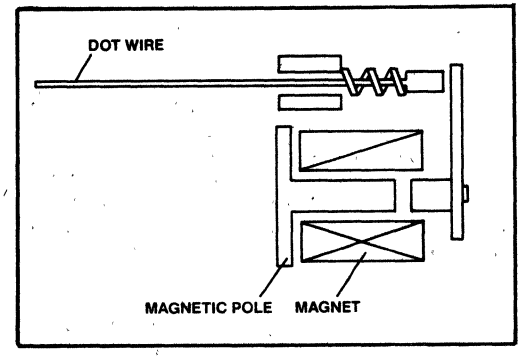


Figure 6. Print Head Solenoid Assembly

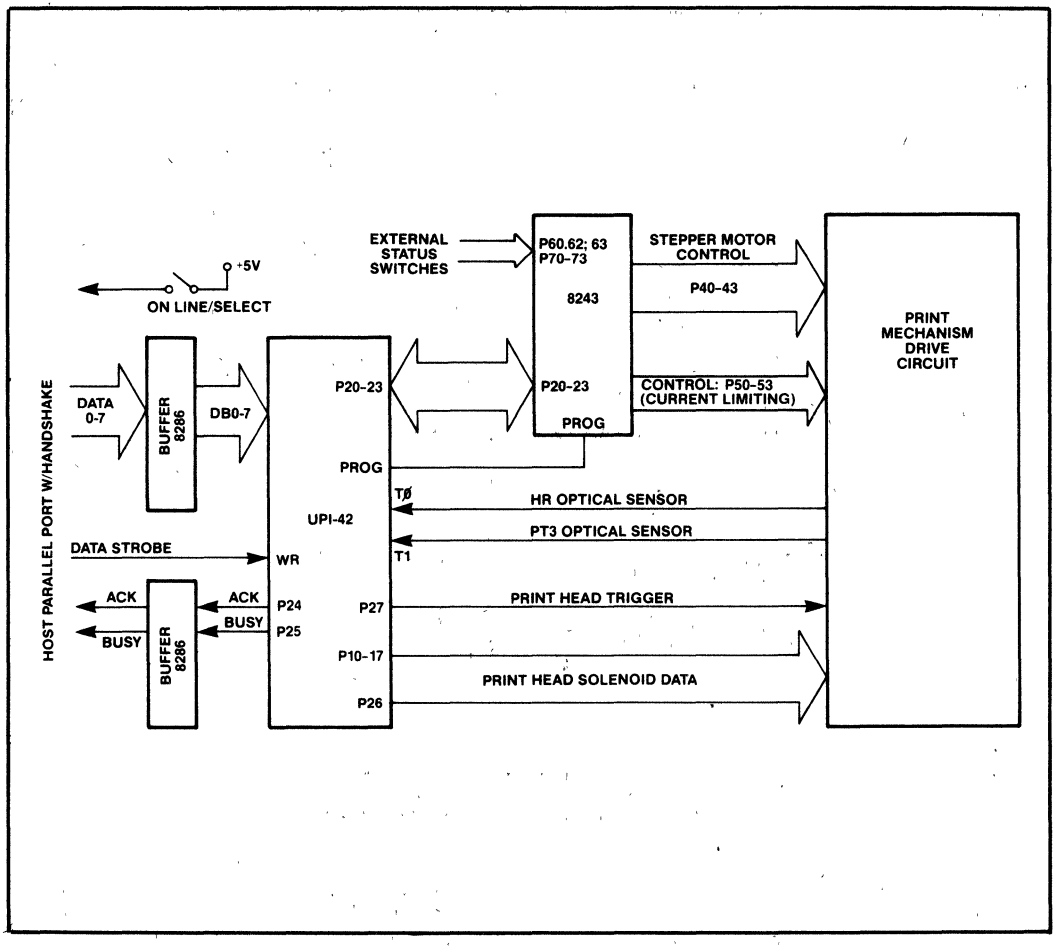


Figure 7. Hardware Interface Block Diagram

HARDWARE DESCRIPTION

Figure 7 shows a block diagram of the UPI-42 and 8243 interface to the printer mechanism drive circuit. A complete schematic is shown in Figure 8. The UPI-42 provides all signals necessary to control character buffering and handshaking, paperfeed and carriage motion stepper motor timing, print head solenoid activation, and monitoring of external status switches.

The Epson 3210 printer mechanism manual recommends a specific interface circuit to provide proper drive levels to the stepper motors windings and print head solenoids. The hardware interface used for this Application Note followed those recomendations exactly (see Appendix C, Printer Mechanism Drive Circuit Schematics).

I/O Ports

The lower half of the UPI-42 Port 2, pins 0-3, provides an interface to the 8243 I/O expander. The PROG pin of the UPI-42 is used as a strobe to clock address and data information via the Port 2 interface. The extra 16 I/O lines of the 8243 become PORTS 4, 5, 6, and 7 to the UPI software. Combined, the UPI-42 and 8243 provide a total of 28 independently programmable I/O line. These lines are used as follows:

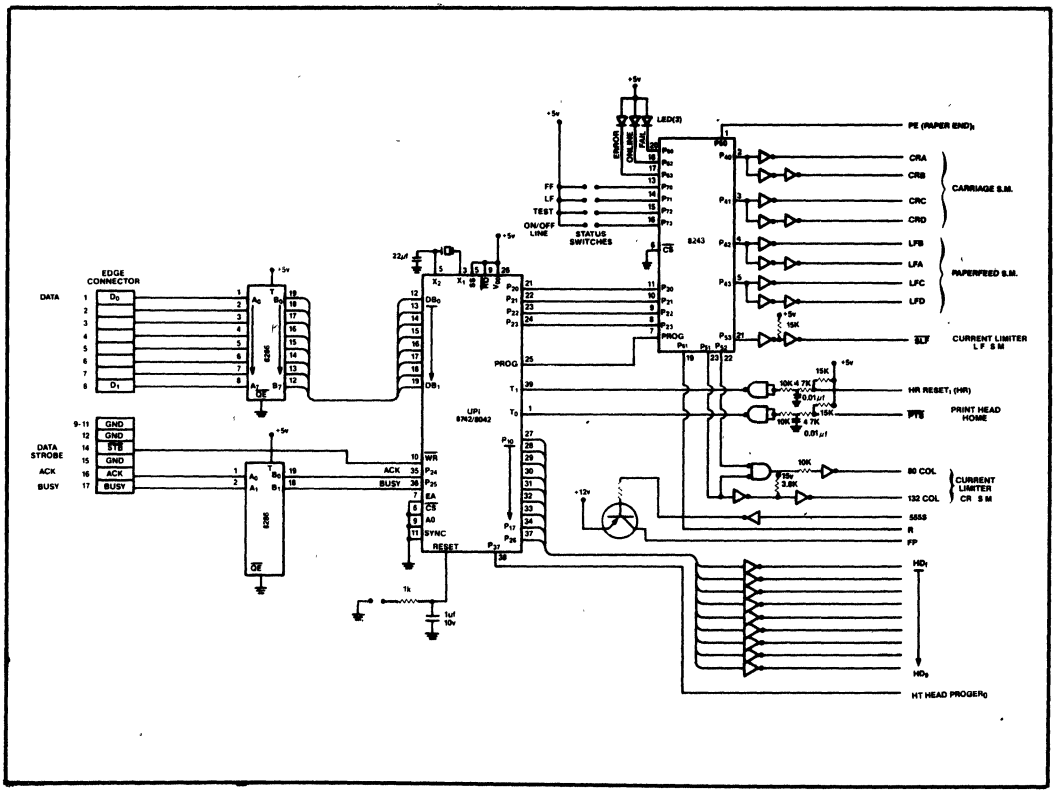


Figure 8. Hardware Interface Schematic

Port	No of lines	Bits	I/O	Description
1	8	0-7	O	Character dot column data to print head solenoids
2	1	6	O	(same)
2	1	7	O	Print head solenoid trigger
2	2	4,5	O	Host system data transfer handshaking (ACK/BUSY)
4	4	0-3	O	Carriage & paper feed stepper motors
5	3	1-3	O	Stepper motor select and current limiting
5	1	0	I	Paper End sense
6	1	1	O	Print head trigger reset
6	3	0,2,3	-	(unused)
7	5	0-3	I	External status switches; (LF, FF, TEST, ON/OFF Line)

Figure 9. UPI-42 and 8243 I/O Port Map

Note: The notation used in the balance of this Application Note, when referring to a port number and a particular pin or bit, is Port 23 rather than Port 2 bit 3.

The two printer mechanism optical sensors, discussed in the Printer Mechanism discription, are tied to the two "Test Input" pins, T0 and T1, of the UPI-42 through a buffer circuit for noise supression. These inputs are directly testable in software.

Host System Interface

The host system interfaces to the printer through a parallel port to the UPI-42 Data Bus. Four handshaking signals are used to control data transfer; Data Strobe (STB/), Acknowledge (ACK), Busy (BUSY), and Online or Select. The Data Strobe line of the host parallel port is tied directly to the UPI-42 WR/ pin. This provides a low going pulse on the UPI-42 WR/ pin whenever a data byte is written to the UPI-42. The ACK and BUSY handshake signals are tied to two UPI-42 I/O port lines for software control of data transfer. The "On Line" handshake signal is tied to a single-pole single-throw fixed position switch, which externally enables or disables character transfer from the host system. Characters transmitted to the UPI-42 by the host are loaded into the UPI-42 Data Bus Buffer In (DBBIN) register, and the Input Buffer Full (IBF) interrupt and UPI-42 status flag are set (see Figure 9. UPI-42 and 8243 I/O Ports).

Stepper Motor Interface

Port 4 (41-43) of the 8243, provides both carriage and paper feed stepper motor phase shift signals to the printer mechanism drive circuit. Each of the two stepper motors is driven by 2 two phase excitation signals (4 phases). Figure 10 shows the wave form for each stepper motor. Each signal consists of two components (Sig. 1 A/B & Sig. 2 C/D) 180 degrees out of phase with the other. Each of these signal pairs (A/B & C/D) is 90 degrees out of phase with the other pair. For each signal pair, one port line supplies both halves by using an inverter.

Each of the resulting eight stepper motor drive signals is interfaced to a discrete drive transistor through an inverter. The emitter of the drive transistor is tied to the open collector of the inverter to provide high current sinking capability for the drive transistor. Each half of the motor winding is tied to the collector of the drive transistor (see Appendix C, Printer Mechanism Drive Circuit Schematic).

Each stepper motor requires two current levels for operation. These levels are called "Rush" current and "Hold" current. Rush current refers to the high current required to cause the rotor to rotate within its windings as the polarity of the power applied to the windings is changing. Each change in the polarity of the power applied to the motor windings is called a step or phase shift. Hold current refers to the low level of current required to stabilize and maintain the rotor in a fixed position when the the polarity applied to the windings is not changing. Hold current is simply Rush current with a current limiting transistor switched in. Switching from Hold to Rush current "selects" or enables that stepper motor to move with the next step signal output. In the balance of this Application Note, the term "select" will be used to refer to turning on Rush current, and "deselect" will refer to switching to Hold currrent.

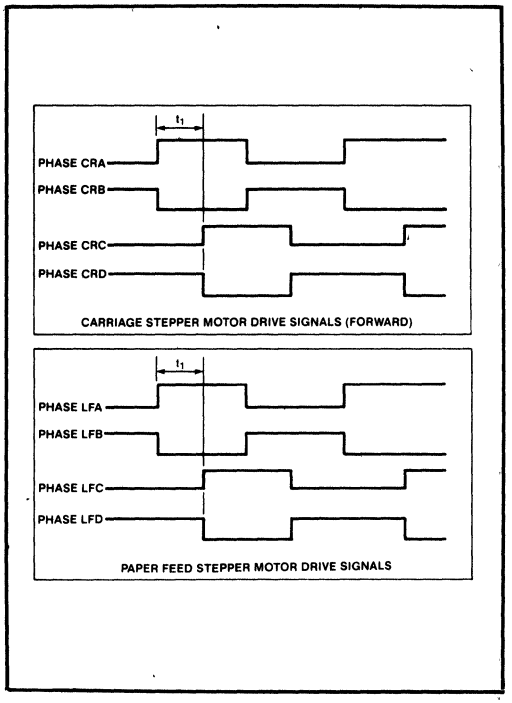


Figure 10. Stepper Motor Step Sequence Waveforms

Three 8243 port lines are dedicated to the select/deselect control of the two stepper motors. One line is for the paper feed stepper motor, and two lines are for the carriage motion stepper motor (80 and 132 column). These lines are labeled SLF, 80Col, and 132Col, and are 8243 PORT 53, 52, and 51, respectively.

By varying the voltage applied to the stepper motor biasing circuit and the current, it is possible to vary the distance the motor moves the print head assembly with each step. Enabling one of two different voltage biasing levels, and changing the timing rate at which the motor is stepped, facilitates either 80 or 132 character column printing. Only 80 character column printing is implemented in the software design. Appendix B, Printer Enhancements, details the software algorithm for handling 132 character printing.

Print Head Interface

A total of eleven I/O lines are used to control the print head solenoids and solenoid firing (see Figure 9 above). Nine are used for character dot data, one for the Print Head Trigger, and one for Reset of the Print Head Trigger circuit. Each of the nine character dot data lines is buffered by an open collector hex inverter.

The Print Head Trigger output is tied to the Trigger input of a 555 Monostable Multivibrator. The output pulse generated by the 555 triggers the print head solenoids to fire. The 555 Output pulse width is independent of the input trigger waveform. The pulse width is determined by an RC network across the 555 inputs and the voltage level applied to the Control Voltage 555 input. The 555 Output is tied to the base of a PNP transistor through an inverter, biased in a normally off configuration. The PNP transistor supplies enough drive to pull up the open collector inverter on each print head solenoid line, Port 10-17 and 26. The 555 output pulse momentarily enables the print head solenoid line open collector inverter output, turning on the solenoid drive transistor, and firing the print head hammer. The 555 Ouput pulse width is approximately 400 us. Further details of the print head firing operation can be found in the software description below.

Miscellaneous Interface Signals

The 8243 Port 5 pin 0 is tied to the Paper End Detector, a reed switch located on the printer paper guide. This sensor detects when the paper is nearly exhausted.

Three LED status lights complete the hardware interface design. One status light is used for each of the following: Power ON/OFF, On/Off Line, and Out of Paper.

BACKGROUND

Before a detailed discussion of the software begins, a few terms and software functions referenced throughout the software need introduction.

A. What is a Stepper Motor?

A stepper motor has the ability to rotate in either direction as well as start and stop at predetermined angular positions. The stepper motor's shaft (rotor) moves in precise angular increments for each input step. The displacement is repeated for each input step command, accurately positioning the rotor for a given number and sequence of steps.

The stepper motor controls position, velocity, and direction. The accuracy of stepper motors is generally 5 percent of one step. The number of steps in each revolution of the shaft varies, depending on the intended application.

B. Open/Closed Loop Stepper Motor Drive and Control

The carriage stepper motor is closed loop driven. The paper feed stepper motor is open loop driven.

There are two major types of stepper motor control known by the broad headings of open and closed loop. Open loop is simply continuous pulses to drive the motor at a predetermined rate based on the voltage, current, and the timing of the step pulses applied. Closed loop control is characterized by continuous monitoring of the stepper motor, through feedback signals, and adjusting the motor's operation based upon the feedback received.

C. Stepper Motor Drive Phase Shift or Step Sequence

Each change in the polarity of the power applied to the motor windings is called a step or phase shift. The sequence of the steps or phase shifts, and the pattern of polarity changes output to the stepper motor, determines the direction of rotation.

Figure 10 shows the waveforms for each of the two stepper motors. Figure 11 lists the step sequence for carriage motor clockwise rotation, which moves the print head assembly Left-to-Right. Figure 11 also lists the step sequence for counterclockwise rotations; the print head assembly moves Right-to-Left. Figure 12 lists the step sequence for the paper feed stepper motor clockwise drive. The phase sequence, for either stepper motor, may begin at any point within the sequence list, but must then continue in order.

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	Off	On
2	On	Off	On	Off
3	Off	On	On	Off
4	Off	On	Off	On

Carriage stepper motor rotates clockwise
Print head assembly moves from left to right

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Carriage stepper motor rotates counter clockwise
Print head assembly moves from right to left

Figure 11. Carriage Stepper Motor Step Sequence

Step No.	A-Step	B-Step	C-Step	D-Step
1	On	Off	On	Off
2	On	Off	Off	On
3	Off	On	Off	On
4	Off	On	On	Off

Figure 12. Paper Feed Stepper Motor Step Sequence

C. Acceleration and Deceleration of Stepper Motors

The carriage stepper motor starts from a fixed position, accelerates to a constant speed, maintains constant speed, and then decelerates to a fixed position. Printing may occur from the time and position the print head assembly reaches constant speed, until the time and position the print head assembly begins to decelerate from constant speed. Whether printing occurs during any carriage stepper motor drive sequence is controlled by software. Figure 18, below, illustrates these components of print head assembly line motion.

Due to inertia, a finite time interval and angular displacement is required to accelerate a stepper motor to its full speed. Conversely, deceleration must begin some time before the final angular position. The time interval and angular displacement of the carriage stepper motor translates into the distance the print head assembly travels before it reaches a constant speed. The distance traveled during acceleration is constant. The distance the print head assembly travels during deceleration must be the same as the distance traveled during acceleration in order to accurately align the character dot columns from one line to the next.

E. Stepper Motor Predetermined Time Constant

Whenever the stepper motor is stepped, or energized, the angular velocity of the rotor is greater than the constant speed which is ultimately required. This is called "overshoot." The frictional load of the carriage assembly (motor rotor, reduction gears, drive belt and print head assembly, or paper feed sprocket shaft and wheels) provides damping or frictional load. Damping slows the motor to less than the required constant speed and is called "undershoot" (see Figure 13, Carriage Stepper Motor Drive Timing). A constant rate of speed is achieved through the averaging of the overshoot and undershoot within each step.

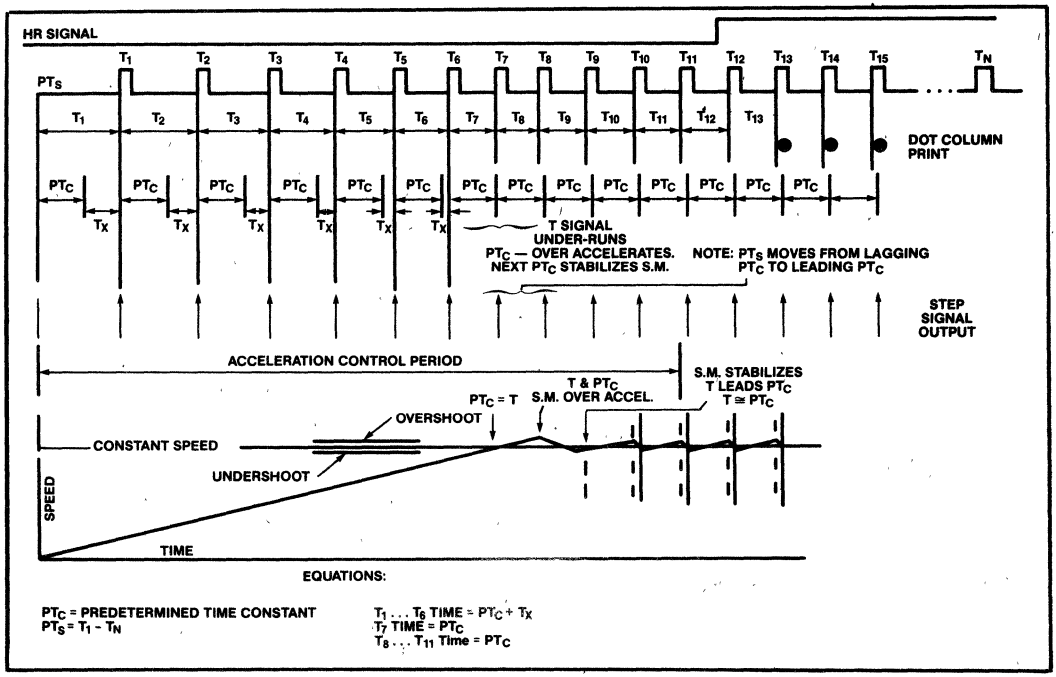


Figure 13. Carriage Stepper Motor Drive Timing

The Predetermined Time (PT) Constant is the time required to average the overshoot and undershoot of the particular stepper motor for a desired constant rate of speed. The PT also is the time required to move the print head assembly a specific distance, acounting for both overshoot and undershoot of the stepper motor.

Changing the Predetermined Time Constant changes the angular displacement of the stepper motor rotor, this in turn changes the output. Figure 14 lists the Time Constants for both standard and condensed character printing. Figure 15 lists the paper feed stepper motor Time Constants used for various line spacing formats. This Application Note implements standard character print and paper feed (6 lines per inch) Time Constants. See Appendix B, Printer Enhancements, for details on implementing non-standard Time Constants.

Character mode	Predetermined time	
Standard or Enlarged Character	2.08ms	+10% -4%
Condensed Character	4.16ms	+10% -4%

Figure 14. Carriage Stepper Motor Predetermined Time Constants

Paper feed pitch	
	0 12mm(1/216") /1 pulse
	4 23mm(1/6") /36 pulses
	3.18mm(1/8") /27 pulses
	2 82mm(1/9") /24 pulses
Paper feed time	
150ms/4 23mm	Approx 6.6 lines/s (continuous feed)
113ms/3 18mm	Approx 8 8 lines/s (continuous feed)
100ms/2.82mm	Approx. 10 lines/s (continuous feed)

Figure 15. Paper Feed Stepper Motor Predetermined Time Constants

D. Relationship Between PTS and PT

Figure 13 illustrates how PTS lags PT at the start of acceleration, and moves to lead PT as the motor achieves constant speed. Figure 13 also illustrates the relationship between HR, PTS, PT, acceleration, constant speed, and printing. Figure 16 and 17 illustrate the relationship between PTS and PT during acceleration and at constant speed.

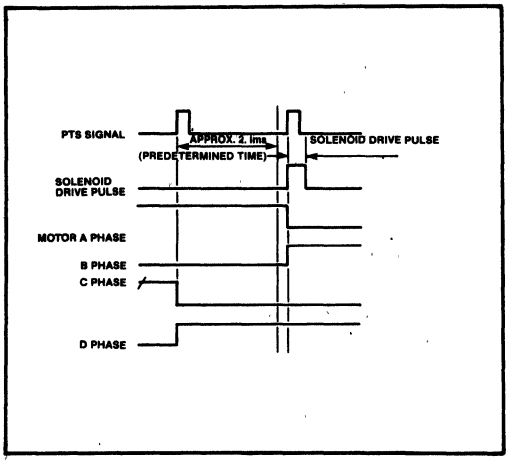


Figure 16. PTS Lags PT Timing

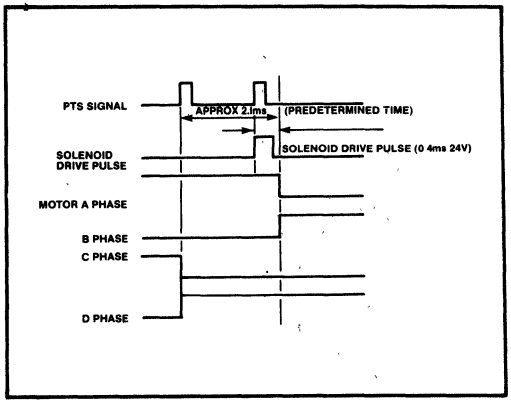


Figure 17. PTS Leads PT Timing

PTS is the point of peek angular velocity within a step of the motor. PTS is a function of the slot spacing on the encoder disk, shown in Figure 5. The spacing is determined by the mechanics of the printer mechanism.

When the carriage stepper motor is accelerated from a fixed position, the effects of damping slows the angular velocity of engergizing the stepper motor. This causes PTS to occur after the PT, or PTS lags PT. When PTS lags PT, the next step signal is output at PTS rather than at PT. If the step signal is outputted at PTS, the rotor could be midway through a rotation. Energizing the motor at PT could cause it to bind or shift in the wrong direction. When the carriage stepper motor is at a constant rate of speed, PTS leads PT and the step signal is output at PT (see Figure 13).G. Stored Time Constants.

The time between each step, for a constant number of steps, required for the motor to reach a constant speed, is calculated and stored in Data Memory during acceleration. The values stored are used, in reverse order, during deceleration as the Predetermined Time (PT) Constants. This ensures that the acceleration and deceleration distance traveled by the print head assembly is the same, and that it accurately aligns character dot columns from one line to the next during printing. The time values stored are called "Stored Time Constants." Steps T1 through T11 in Figure 13, represent the Stored Time Constants.

The equations for the Stored Time Constants are given at the bottom of Figure 13, Carriage Stepper Motor Drive Timing.

H. Print Head Assembly "Home" Position

The "logical" Home position for the print head assembly is the left-most position at which printing begins (for L-to-R motion) or ends (for R-to-L motion). The "physical" Home position is the logical HOME position, plus the distance required by the carriage stepper motor to fully accelerate the print head assembly to a constant speed. Printing can only occur when the print head is moving at a constant speed. The printer mechanism manual stipulates eleven step time periods are required to ensure the the print head assembly is at a constant speed. These eleven step time periods are the Stored Time Constants described above. Figure 18 illustrates the components of print head assembly line motion and character printing.

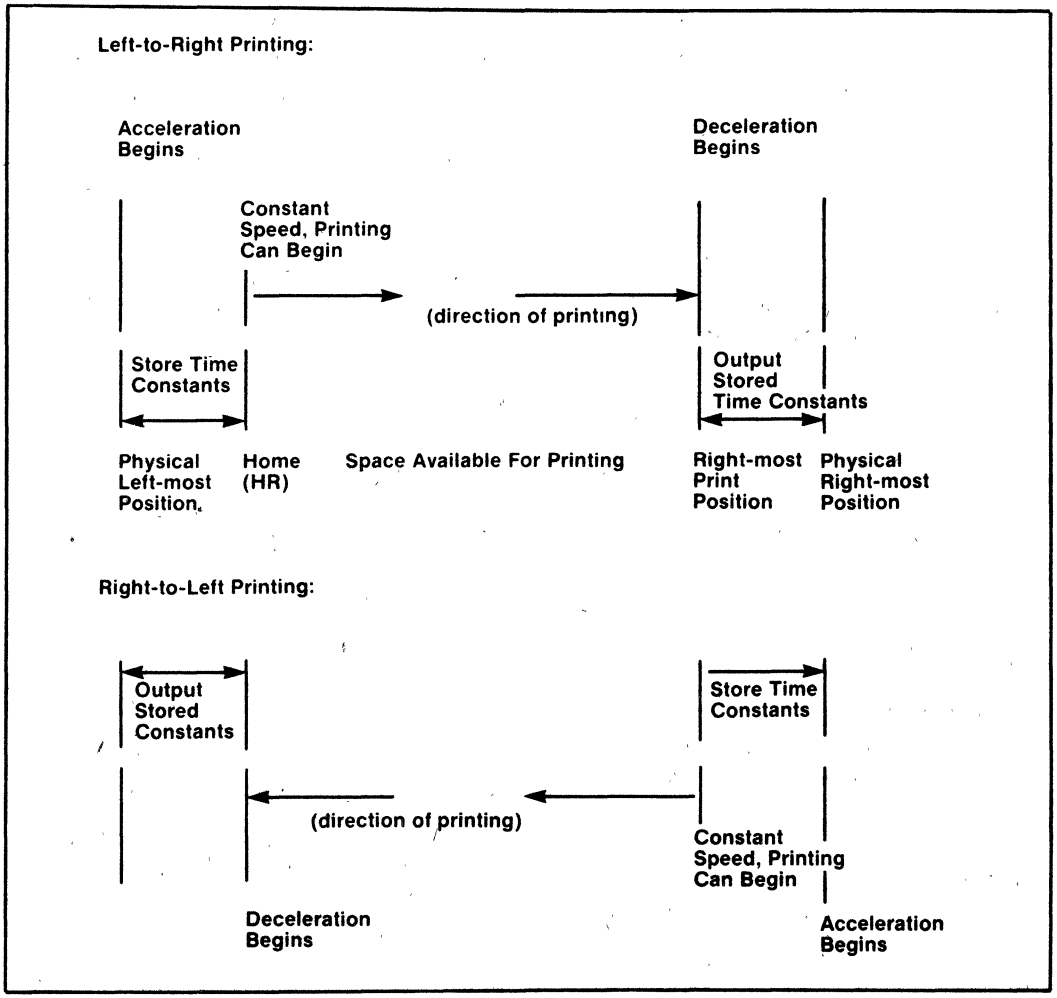


Figure 18. Components of Print Head Assembly Line Motion and Printing

SOFTWARE

Introduction

The software description is presented in three sections. First, a brief overview of the software to familiarize the reader with the interdependencies and overall program flow. Second, data and program memory allocation and status register flag definitions. And third, each of the ten software blocks is presented with its own flowchart.

Software Overview

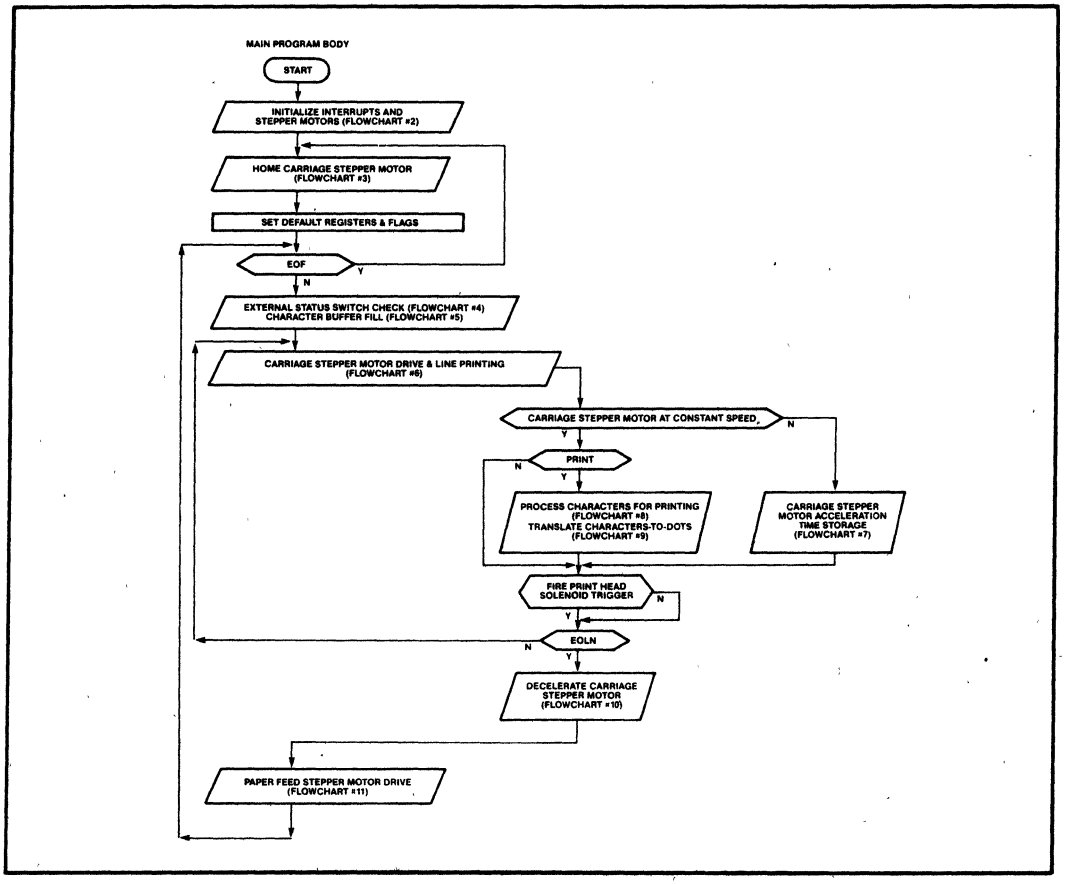
The software is written in Intel UPI-41A/42 Assembly Language. A block structure approach is used for ease of development, maintance, and comprehension. The software is divided into five principal parts.

1. Initialization
2. Character Buffering or Input
3. Stepper Motor Drive and Control
4. Character Processing
5. Character Printing or Output

The five principal parts are incorporated into ten software blocks, listed below.

1. Power On/Reset Initialization
2. Home Print Head Assembly
3. External Status Switch Check
4. Character Buffer Fill
5. Carriage Stepper Motor Drive and Line Printing
6. Accelerate Stepper Motor Time Storage
7. Process Characters for Printing
8. Translate Character-to-Dots
9. Decelerate Carriage Stepper Motor
10. Paperfeed Stepper Motor Drive

Flow Chart No. 1 illustrates the overall software algorithm. Below, is a description of the algorithm.



Flow Chart No. 1. Main Program Body

Upon power-on or reset, a software and hardware initialization is performed. This stablizes and sets inactive the printer hardware and electronics. The print head assembly is then moved to establish its HOME position. The default status registers are set for character buffering, carriage, and paper feed stepper motor drive. The External Status switches are checked; FORMFEED, LINEFEED, ON/OFF LINE, and Character Print TEST. If the printer is ON LINE, the software will loop on filling the Data Memory Character Buffer.

Character or data input to the UPI-42 is interrupt driven. Characters sent by the host system set the Input Buffer Full (IBF) interrupt and the IBF Program Status flag. Character input servicing (completed during the paper feed and carriage stepper motor drive end Delay subroutine) tests for various ASCII character codes, loads characters into the Character Buffer (CB), and repeats until one of several conditions sets the CB Full status flag. Once the CB Full flag is set, further character transmission by the host system is inhibited and printing can begin.

The carriage stepper motor is initialized, and drive begins for the direction indicated. The motor is accelerated to constant speed, printable character codes are translated to dot patterns and printed (if printing is enabled), and the motor is decelerated to a stop. Two timing loops guarantee both constant speed and protection (Failsafe Time) against stepper motor burn out due to high current overload. The two optical sensors, described in the Printer Mechanism section above, are constantly monitored to maintain constant speed, and trigger print head solenoid firing.

Once the line is printed and the carriage stepper motor drive routine has been completed, a Linefeed is forced. The paper feed stepper motor drive subroutine tests the number of lines to move, and energizes the paper feed stepper motor for the required distance. The lines per page default is 66; if 66 lines have been received, a Formfeed to Top-of-Next-Page is performed. The Top-Of-Page is set at Power On/Reset.

When the EOF code is received, the EOF status flag is set. When the last line has been printed, the EOF check will force the print head assembly to the HOME position. The EOF flag is tested following each Paper Feed stepper motor drive. The next entry to the External Status Check subroutine begins a loop which waits for input from either the external status switches or the host system.

The software character dot matrix used in this application is 5 x 7 of the available 9 x 9 print head solenoid matrix. Although lower case descenders and block/line graphics characters are not implemented, Appendix B, Printer Enhancements, discusses how and where these enhancements could be added. The software implements the full 95 ASCII printable characters set.

Memory and Register Allocation

Data Memory Allocation (RAM)

The UPI-42 has 128 bytes of Data Memory. Sixteen bytes are used by the two 8 byte register banks (RB0 and RB1). Sixteen additional bytes are used for the Program Stack. The Stored Time Constants utilize 11 bytes, while the stepper motor phase storage requires 4 bytes. Below is a detailed description of Data and Program Memory

Hex Address	Description

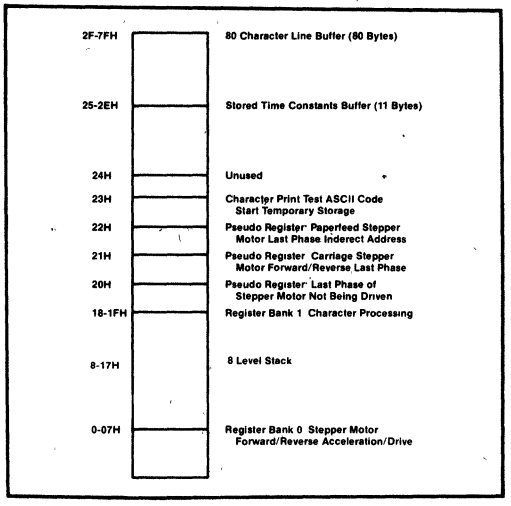


Figure 19. Data Memory Allocation Map

Register Bank 0 is used for stepper motor drive functions. Register Bank 1 is used for character processing. Each register bank's register assignments is listed in Figure 20 and 22, respectively. Each register bank has one register allocated as a Status Register. Figure 21 and 23 detail the Status Register flag assignments. Note that bit 7 of each Status Byte is used as a print head assembly motion direction flag. This saves coding of the Select Register Bank (SEL RBn) instruction at each point the flag is checked.

Register Bank 0

Register	Program Label	Description
R0	TmpR00	RB0 Temporary Register
R1	TStrR0	Store Time Register
R2	GStR20	General Status Register
R3	PhzR30	Stepper Motor Step Register
R4	CntR40	Count Register
R5	TConR0	Time Constant Register
R6	LnCtR0	Line Count Register
R7	OpnR70	Available, Scratch

Figure 20. Register Bank 0 Register Assignment

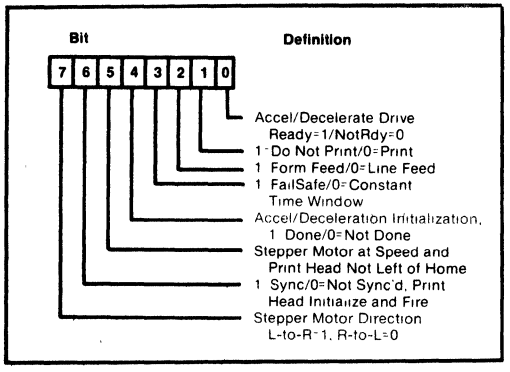


Figure 21. Register Bank 0 Status Byte Flag Assignments

Register	Program Label	Description
R0	TmpR10	RB0 Temporary Register
R1	CAdrR1	Character Data Memory Address Register
R2	ChStR1	Character Processing Status Byte Register
R3	CDtCR1	Character Dot Count Register
R4	CDotR1	Character Dot Temporary Storage Register
R5	CCntR1	Character Count Temporary Register
R6	StrCR1	Store Character Register
R7	OpnR71	Available/Scratch

Figure 22. Register Bank 1 Register Assignment

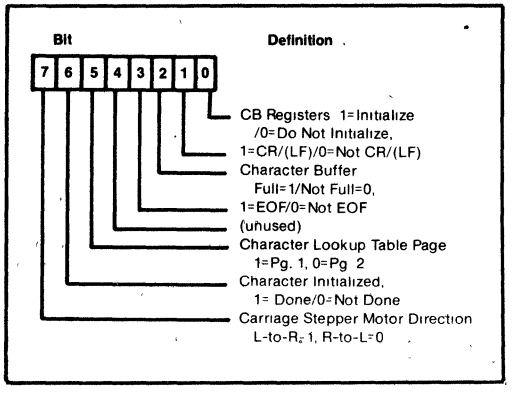


Figure 23. Register Bank 1 Status Byte Flag Assignments

Program Memory Allocation (EPROM/ROM)

The UPI-42 has 2048 bytes of Program Memory divided into eight pages, each 256 bytes. Figure 24 illustrates the Program Memory allocation map by page.

Page	Hex Address	Description
Page 7	1792-2047	Character to Dot Pattern Lookup Table; Page 2: ASCII 50H-7EH
Page 6	1536-1791	Character to Dot Pattern Lookup Table; Page 1: ASCII 20H-4FH (sp-M)
Page 5	1280-1535	Miscellaneous Subroutines: InitAl/AllOff Clear Data Memory Home Print Head Assembly Character Print Test Initialize Carriage Stepper Motor Delay Stepper Motor Deselect
Page 4	1024-1279	Paper Feed Stepper Motor Drive
Page 3	768-1023	Stepper Motor Step LookUp Table(Indexed) Character Processing and Translation Print Head Firing
Page 2	51–767	Carriage Stepper Motor Acceleration Time Calculation and Storage Stepper Motor Deceleration
Page 1	256-511	Carriage Stepper Motor Drive
Page 0	0-255	Initialization - Jump-on-Reset Main Program Body External Status Switch Check Character Buffer Fill

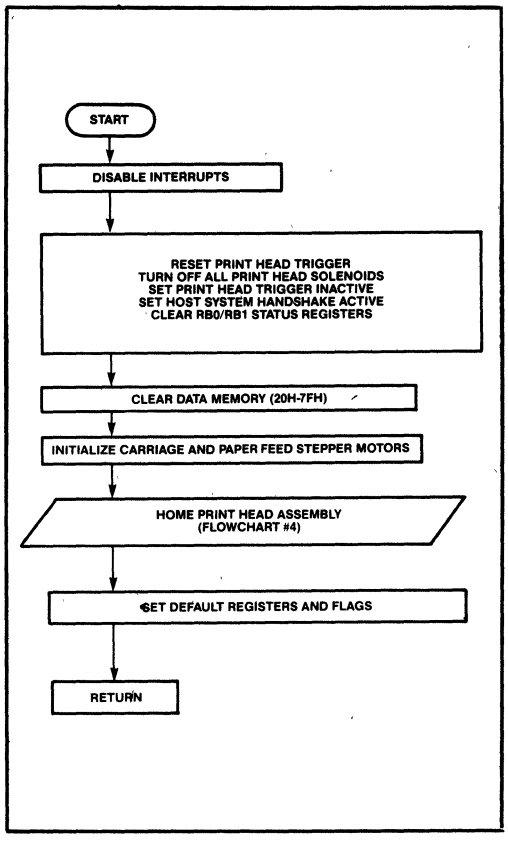
Figure 24. Program Memory Allocation Map

Software Functional Blocks

Below is a decription and flow chart for each of the ten software blocks listed above.

1. Power-On/Reset Initialization

The first operational part in Flow Chart No. 1 is the Power-On or Reset Initialization. Flowchart No. 2 illustrates the Initialization sequence in detail.



Flow Chart No. 2. Power-On/Reset Initialization

Initialization first disables both interrupts. This is done as a precaution to prevent the system software from hanging-up should an interrupt occur before the proper registers and Data Memory values are initialized.

Initialization then deactivates the system electronics. This is also a precaution to protect the printer mechanism and includes the print head solenoid (trigger and data) lines and the stepper motor select lines. The host system handshake signals are activated to inhibit data transfer from the host until the printer is ready to accept data.

Next, Data Memory is cleared from 20H to 7FH. This includes; the 80 byte Character Buffer, the 11 byte Stored Time Constants buffer, and the 4 bytes used as pseudo registers. The pseudo registers are Data Memory locations used as if they were registers. They serve as storage loacations for step data used in accurately reversing the direction of the carriage stepper motor, and stablizing either of the stepper motors not being driven.

The Data Memory locations 00H through 1FH are not cleared. These locations are Register Bank 0 (00H-07H), Program Stack (08H-17H), and Register Bank 1 (18H-1FH) (see Figure 19). Clearing the Program Registers or Stack would cause the initialization subroutine to become lost. The registers are used from the beginning of the program. Care is taken to initialize the registers and stack accurately prior to each program subroutine as required.

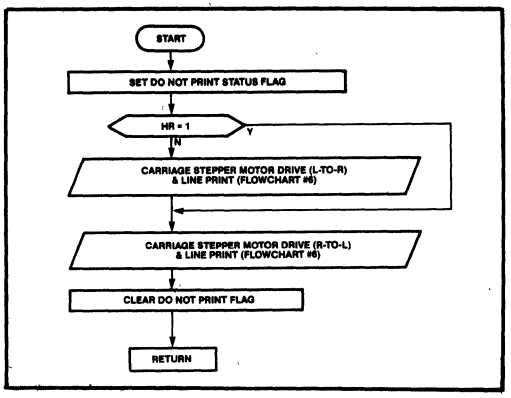
Upon power-on, it is necessary to initialize the two stepper motors, verify their operation, and locate the print head assembly in the left-most 'HOME' position. This sequence serves as a system checkout. If a failure occurs, the motors are deselected and the external status light is turned on. Each stepper motor is selected and energized for a sequence of four steps. This serves to align and stabilize each stepper motor's rotor position, preventing the rotor from skipping or binding when the first drive sequence begins.

At the end of each stepper motor's initialization, the last step data address is stored in one of the Data Memory pseudo registers. The last step data address is recalled at the beginning of the next corresponding stepper motor drive sequence, and used as the basis of the next step sequence. This ensures that the stepper motor always receives the exact next step data, in sequence, to garantee smooth stepper motor motion. This also garantees the motor never skips or jerks, which would misalign the start, stop, and character dot column positions. A stepper motor not being driven has its last phase data output held constant to stabilize it.

Following any stepper motor drive sequence of either motor, a delay of 30-60 ms occurs by switching the current to Hold Current, stabilizing the motor before it is deselected.

2. Home Print Head Assembly

At the end of the carriage stepper motor four step initialization, the output of the HR optical sensor is tested. The level of the HR signal indicates which drive sequence will be required to 'HOME' the print head assembly. If the print head assembly is to the right of HR, HR is high, the print head assembly need only be moved to from Right-to-Left until HR is low, then decelerated to locate the physical home position. If HR is low, the print head assembly must be moved first Left-to-Right until HR is high, then Right-to-Left to locate both the logical and physical 'HOME' positions. In each case, the software accelerates the carriage stepper motor, generating the Stored Time Constants then decelerates the stepper motor using the Stored Time Constants (see Background section above). Flow Chart No. 3 details the HOME print head assembly subroutine. Figures 13 and 18 illustrate the components of acceleration and print head assembly line motion.



Flow Chart No. 3. HOME Print Head Assembly

The carriage stepper motor drive subroutines used to HOME the print head assembly and to print, are the same. A status flag, called Do-Not-Print, determines whether the Character Processing subroutine is called. The flag is set by the subroutine which calls the Carriage Stepper Motor Drive subroutine. Details of the carriage and paper feed stepper motor drive and character processing subroutines are covered separately below.

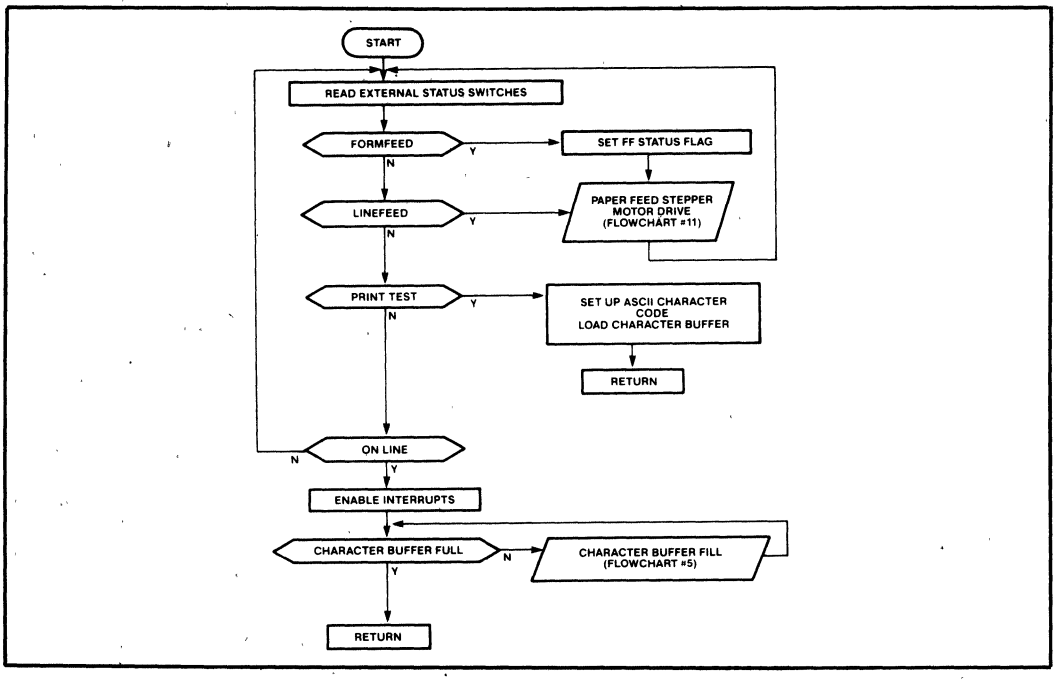
3. External Status Switch Check

Once the system is initialized and the print head is at the HOME position, the software enters a loop which continually monitors the four external status switches, and exits if any one is active. Flow Chart No. 4 details the External Status Switch Check subroutine.

Flow Chart No. 4. External Status Switch Check

If the LINEFEED or FORMFEED switch is set, the Paper Feed subroutine is called. The Paper Feed subroutine is discussed in detail below. If the ONLINE switch is set, the Character Buffer (CB) Fill subroutine is called.

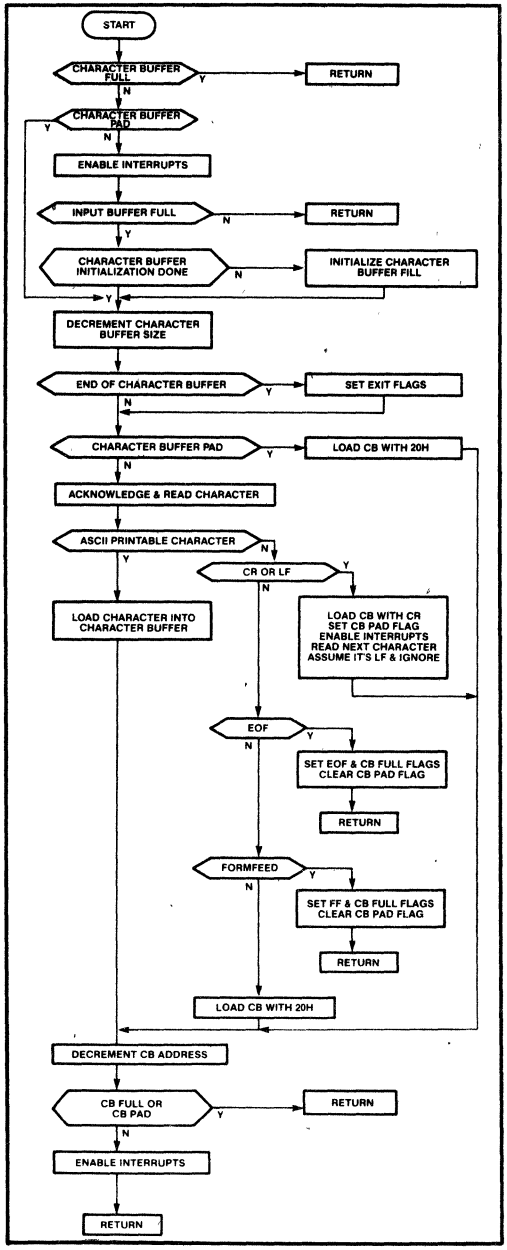
If the Character Print TEST switch is set, the Data Memory Character Buffer (CB) is automatically loaded with the ASCII code sequence, beginning at 20H (a Space character), the first ASCII printable character code. The software then proceeds as if the CB had been filled by characters received from the host system. The External Status Switch Check subroutine is exited and character printing begins. When the line has finished printing, a linefeed occurs (as shown in the main program Flow Chart No.1) and the program returns to the External Status Switch Check subroutine. If the TEST switch remains active, the ASCII character code is incremented and program continues as before. This will eventually print all 95 ASCII printable characters. An example of the TEST printer output, the complete ASCII character code printed, is shown in Figure 25.



Flow Chart No. 4. External Status Switch Check

4. Caracter Buffer Fill

The Character Buffer (CB) Fill subroutine is called from three points within the main program; External Status Switch subroutine, and the Delay subroutine following the carriage and paper feed stepper motor drive subroutines. Flowchart No. 5 details the Character Buffer Fill subroutine operation.



Flow Chart No. 5. Character Buffer Fill

The approximate 80 ms total pre-deselect delay at the end of each stepper motor drive sequence, 40 ms carriage and 40 ms paper feed stepper motor pre-deselect delay, is sufficient to load an entire 80 character line. Half the CB is filled at the end of printing the current line, and the second half is filled at the end of a paper feed. There is no time lost in printing throughput due to filling the character buffer.

Character input is interrupt driven. When the IBF interrupt is enabled, a transmitted character sets the IBF interrupt and IBF Program Status flag. Three instructions make up the IBF interrupt service routine. This short routine disables further interrupts, sets the BUSY handshake line active, inhibiting further transmission by the host, and returns. The subroutine can be executed at virtually any point in the software flow without effecting the printer mechanism operation. Processing of the received character takes place during one of the three program segments mentioned above. The BUSY line remains active until the character is processed by the CB Fill subroutine.

The CB is 80 bytes from the top of Data Memory (30H-7FH). It is a FIFO for forward, left-to-right printing, and a LIFO for reverse, right-to-left, printing. Loading the CB always begins at the top, 7FH. One character may be loaded into the buffer each time the CB Fill subroutine is called.

The CB is always filled with 80 bytes of data prior to printing. If the total number of characters input up to a Carriage Return (CR)/Linefeed (LF), does not completely fill the CB, the CR code is loaded into the CB and the balance of the CB is padded with 20H (Space Character) until the CB is full. A Linefeed (LF) character following a Carriage Return is ignored. A LF is always forced at the end of a printed line. When the CB is full, the CB Full status byte flag is set and printing can begin.

A LF character alone is treated as a CR/LF at the end of a full 80 character line. This is a special case of a printed line and is handled during character processing for printing (see No. 7, Processing Characters for Printing, below). A Formfeed (FF) character sets the FF status byte flag. The flag is tested at each paper feed stepper motor drive subroutine entry.

When the software is available to load the CB with a character, entry to the CB Fill subroutine checks three status flags; CB Full, CB Pad, and IBF flag. If the CB Full flag is set, the program returns without entering the body of the CB Fill subroutine. The CB Pad flag will cause another Space character to be loaded. If the IBF flag is not set, the program returns. If the IBF flag is set, the character is read from the Data Bus Buffer register, tested for printable or nonprintable ASCII code, and, if printable, loaded into the CB. If the character is a non-printable ASCII code and not an acceptable ASCII control code (CR, LF, FF, EOF), a 20H (Space Character) is loaded into the CB.

Exiting the CB Full subroutine with the CB Full or CB

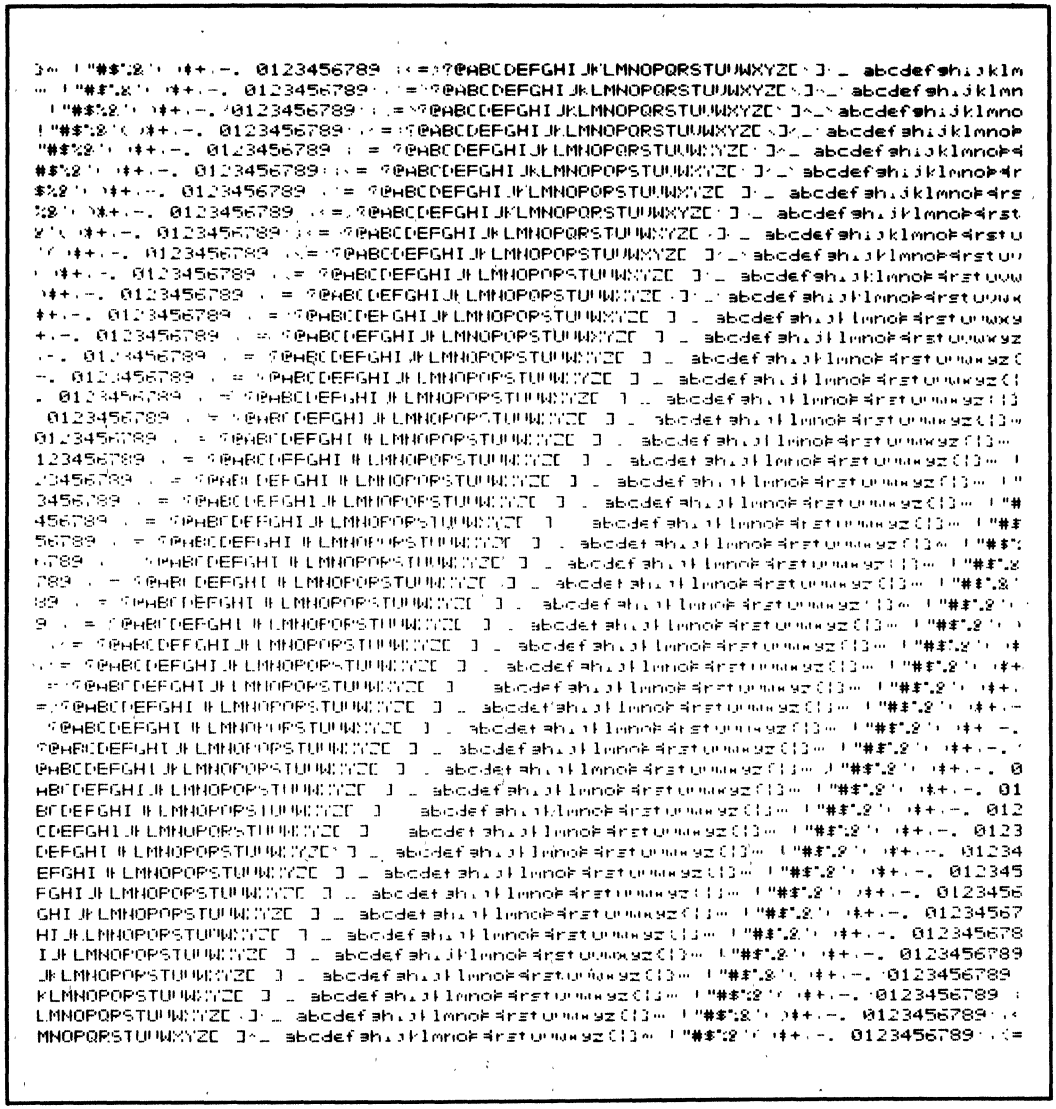


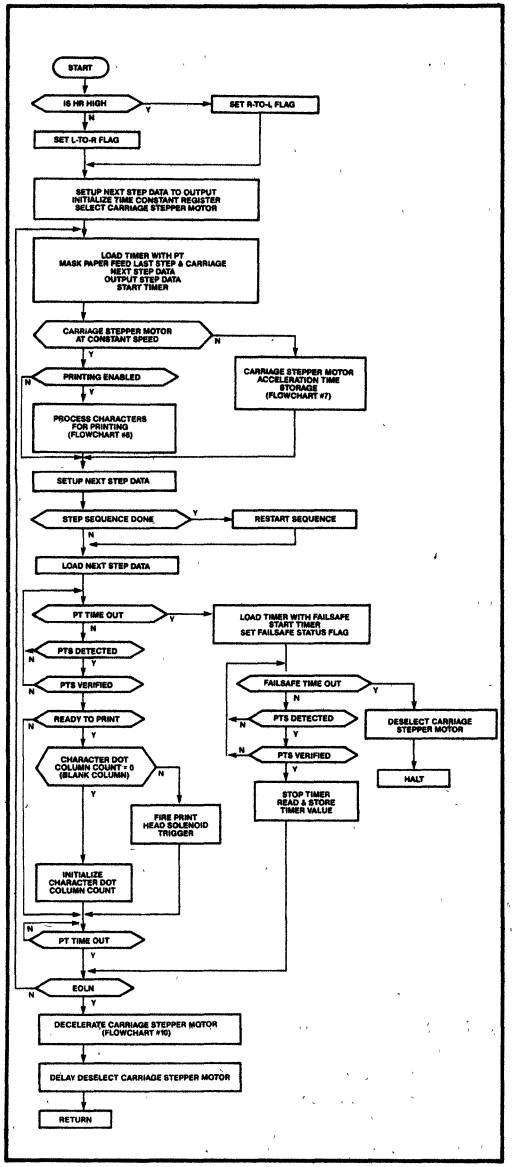
Figure 25. ASCII Character Code TEST Output and Print Example

Pad flag set does not re-enable IBF interrupts or reset the BUSY line. If neither of these flags is set, exiting the CB Fill subroutine sets BUSY inactive and IBF interrupts are enabled. Once the CB Full status byte flag is set, IBF interrupts are disabled until the CB has been entirely emptied, the line printed, or the system Reset.

5. Carriage Stepper Motor Drive and Line Printing

The carriage stepper motor drive subroutine controls both L-to-R and R-to-L print head assembly motion. Upon entering the subroutine, the HR signal level is tested to determine the direction of print head assembly motion and the Direction status flag is set. The default control register values are loaded and balance of the default status flags are set for stepper motor control and character processing. The default control register values include PT and the step sequence look-up table start address for the direction indicated.

The direction flag is tested throughout the carriage stepper motor drive and character processing subroutines. This enables the same subroutines to control activities for either direction, simplifying and shorting the overall program. Flow Chart No. 6 illustrates the carriage stepper motor drive subroutine.



Flow Chart No. 6. Carriage Stepper Motor Drive/Line Printing

Next, the carriage and paper feed stepper motor step data is initialized. The last step data output to the paper feed stepper motor is loaded into the Last Phase pseudo register. This data is masked with each step data output to the carriage stepper motor. Masking the step data in this manner guarantees the paper feed motor signals do not change as the carriage stepper motor is being driven.

Figure 26 illustrates the carriage stepper motor step sequence verses the actual step data output for clockwise rotation, Left-to-Right motion, and counterclockwise rotation, Right-to-Left print head assembly motion. An eight step sequence is depicted in the figure. Note that the sequence for Right-to-Left motion is the reverse of the sequence for Left-to-Right motion. Note also, that for the L-to-R sequence step 4 is the same as step 0, step 5 the same as step 1, etc., through step 7 matching step 3. The four step sequence simply repeats itself until the motor is stopped via the Deceleration subroutine.

L-to-R Motion Sequence	Phase/Step Data (3 2 1 0)	R-to-L Motion Sequence	BCD (3 2 1 0)
0	1 0 0 1	7	**0 0 0 0**
1	1 0 1 0	6	**0 0 0 1**
2	0 1 1 0	5	**0 0 1 0**
3	0 1 0 1	4	**0 0 1 1**
4	1 0 0 1	3	**0 1 0 0**
5	1 0 1 0	2	**0 1 0 1**
6	0 1 1 0	1	**0 1 1 0**
7	0 1 0 1	0	**0 1 1 1**

Figure 26. Carriage Stepper Motor Phase/Step Data

When the carriage stepper motor is driven for a specific direction of print head assembly motion, the step sequence must be consistant for the motion to be smooth and accurate. The same holds true for the transition from one direction of motion to the other. Since the sequence for one direction is the opposite for the other direction, incrementing the sequence for L-to-R and decrementing for R-to-L provides the needed step data flow. For example, referring to Figure 26, if the print head assembly moved L-to-R and the last step output was #1, the first step for R-to-L motion would be #7. Thus, when the carriage stepper motor is initialized for a clockwise (L-to-R) or counterclockwise (R-to-L) rotation, the last step sequence number is incremented or decremented to obtain the proper next step. In this way, the smooth motion of the stepper motors is assured.

The step data is referenced indirectly via the step sequence number. The step data is stored in a Program Memory look-up table whose addresses correspond to the step sequence numbers. For example, as shown in

Figure 26, at location 0 the step data "1001" is stored. This method is particularly well suited to the UPI-42 software. The UPI-42 features a number of instructions which perform an indirect move or data handling operation. One of these instructions, MOVP3 A,@A, unlike the others, allows data to be moved from Page 3 of Program Memory to any other page of Program Memory. This instruction allows the step data to be centrally located on Page 3 of Program Memory and accessed by various subroutines.

Each time the carriage stepper motor step data is output, the step data lookup table address is incremented or decremented, depending upon the direction of rotation, and tested for restart of the sequence. The address is tested because the actual step data, Figure 26, is not a linear sequence and thus is not an easily testable condition for restarting the sequence. The sequence number is tested for rollover of the sequence count from 03H to 04H and clockwise motor rotation via the Jump on Accumulator Bit instruction (JBn), with 00H loaded to restart the sequence. The same bit is tested when decrementing the sequence count for counterclockwise motor rotation, R-to-L motion, because the count rolls over from 00H to 0FFH, with 03H loaded to restart the sequence.

At this point the UPI-42 Timer/Counter is loaded, the step signal is output, and the timer started. The next step data to be output has been determined and the At-Speed flag is tested for entry to one of two subroutines; Stepper Motor Acceleration Time Storage or Character Processing.

The first entry to the Acceleration Time Storage subroutine initializes the subroutine and returns. All other entries to one of the two subroutines perform the necessary operations, detailed below (Blocks 6 and 7), and returns. The program loops until the PT times out or the PTS level change is detected. PTS is tied to T0 of the UPI-42. The level present on T0 is directly tested via conditional jump instrunctions. The software loops on polling the timer Time Out Program Status flag and the T0 input level.

As described in the Background section above (shown in Figure 13), if PT times out before PTS is detected, the software waits for PTS before outputing the next step signal. If PT times out before PTS, a second timer count value is loaded into the UPI-42 timer. The timer value is called "Failsafe." This is the maximum time the stepper motor can be selected, with no rotor motion, and not damage the motor. If PTS is not detected, either the carriage stepper motor is not rotating or the optical sensor is defective. In either case, program excution halts, the motor is deselected, and the external status light is turned on to indicate a malfunction. A system reset is required to recover from this condition. The Failsafe time is approximately 20 milliseconds, including PT.

The Failsafe time loop also serves as a means of tracking the elapsed time between PT time out and PTS. Entry to the Failsafe time loop sets the Failsafe/Constant Time Window status flag. This flag is tested by the Acceleration Time Storage subroutine for branching to the proper time storage calculation to be perform (see Figure 13 and Block 6 below for further description).

During the Failsafe timer loop, if PTS is detected and verified as true, the Failsafe timer value is read and stored in the Time Storage register. This value is used during the next Acceleration Time Storage subroutine call to calculate the Stored Time Constant (see Block 6 below). If PTS is invalid, the flow returns to the timer loop just exited, again waiting for PTS or Failsafe time out.

During the PT time loop, if PTS is detected and verified, the Sync flag is tested for entry to the print head solenoid firing subroutine. This flag is set by the first entry to the Character Processing subroutine. The flag synchronizes the solenoid firing with character processing. Only if characters are processed for printing will the solenoids be enabled, via the Snyc flag, for firing. This prevents the solenoids from being fired without valid character dot data present.

As described in the Background section "Relationship Between PTS and PT," PTS is the point of peek angular velocity within a step of the motor. After PTS is detected the motor speed ramps down, compensating for the overshoot of the rotor motion. PTS is the optimum time for print head solenoid firing, as shown in Figure 13. This is the most stable point of motor rotation and, thus, the print head assembly motion. If PTS is detected during PT, printing is enabled, the Sync flag is set, and the solenoid trigger is fired.

The firing of the solenoid trigger, following PTS, is very time critical. The time between PTS and solenoid firing must be consistant for accurate dot column alignment throughout the printed line. The software is designed to meet this requirement by placing all character processing and motor control overhead before the solenoid firing subroutine is called. The actual instruction sequence which fires the print head solenoid trigger is plus or minus one instruction for any call to the subroutine.

Once the timer loop is complete, the software tests for Exit conditions. If the Exit conditions fail, the software loops to output the next step signal, starts the PT timer, and continues to accelerate the carriage stepper motor, or process, and print characters. If the Exit test is true, the carriage stepper motor is decelerated to a fixed position, and the program returns to the main program flow (see Flowchart 1).

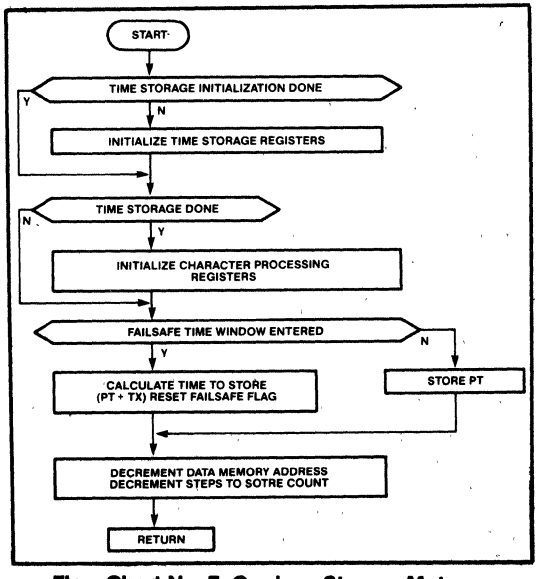
The exit conditions are different for the two directions of print head assembly motion. For L-to-R printing, if a Carriage Return (CR) character code is read from CB, the carriage stepper motor drive terminates and the motor is decelerated to a fixed position. There are two conditions for terminating carriage stepper motor drive upon detecting a CR during L-to-R motion. If less than half a character line (40 characters) has been printed,

the print head assembly returns to the HOME position to start the next printed line. Otherwise, the print head assembly continues to the right-most position for a full 80 character line, and then begins printing the next line from R-to-L. R-to-L printing always returns the print head assembly to the HOME position before the next line is printed L-to-R. When HR is high, character printing always stops and the carriage stepper motor drive subroutine exits to the deceleration subroutine.

6. Accelerate Stepper Motor Time Storage

As described above, when the carriage stepper motor is accelerated the step time required to guarantee the motor is at a constant rate of speed translates to a specific distance traveled by the print head assembly (see Figure 18). In order to position the print head assembly accurately for bi-directional printing, the distance traveled during deceleration must be the same as during acceleration. The Carriage Motor Acceleration Time Storage subroutine calculates the step times needed to accelerate the carriage stepper motor, and stores them in Data Memory for use as PT during deceleration.

The first call of the Carriage Stepper Motor Acceleration Time Storage subroutine initializes the required registers and status flags. The time calculation begins with the second carriage stepper motor step signal output. The program returns to the carriage stepper motor drive subroutine and loops on PT. Each subsequent call of the Acceleration Time Storage subroutine tests the Failsafe/Constant flag and branches accordingly (see Flow Chart 7). The Acceleration Time Storage subroutine has two parts which correspond to PTS leading or PTS lagging PT.



Flow Chart No. 7. Carriage Stepper Motor Acceleration Time Storage

If the Failsafe/Constant flag is set, PTS lagged PT. The time from PT time out to PTS, Tx (see Figure 13), must be added to the PT and stored in Data Memory. As described above, if PT lagged PT, the Failsafe time is loaded and PTS is again polled during the time loop. When PTS occurs within the Failsafe time, the timer is stopped and the timer value stored. The UPI-42 timer is an up timer, which means that the value stored is the time remaining of the Failsafe time when PTS occured. The elapsed time must be calculated by subtracting the time remaining (the value stored) from the Failsafe time constant. This is done in software by using two's complement arithmetic. If the Failsafe flag is not set PTS led PT, and PT is the Stored Time Constant stored.

Indirect addressing of Data Memory is used to reference the Stored Time Constant Data Memory location. The Data Memory location address is decremented each time the Acceleration Time Storage subroutine is exited and a Stored Time Constant has been generated.

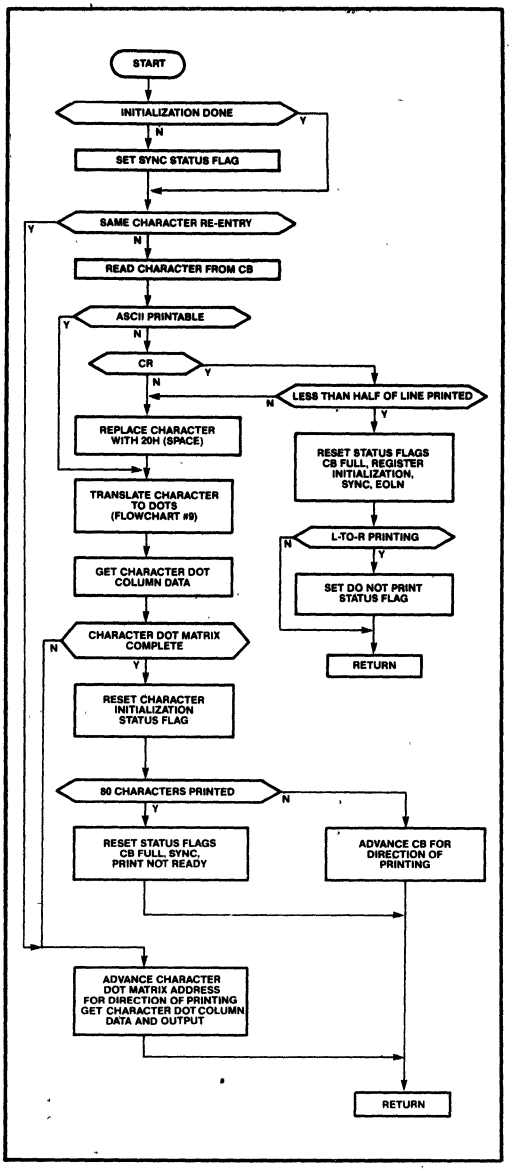
The last Acceleration Time Storage subroutine exit sets the At-Speed status flag and initializes the character processing registers and flags.

3. Process Characters for Printing

The Character Processing subroutine is entered only if the Home Reset (HR) optical sensor signal is high and printing is enabled. Otherwise, the software simply returns to the Carriage Stepper Motor Drive subroutine. There are two cases when printing is not enabled; during the HOME subroutine operation, and when the print head assembly returns to the HOME position after printing less than half an 80 character line. If printing is enabled, the Sync status flag is set.

All character processing operations use the second UPI-42 Data Memory Register Bank, RB1. Register Bank 1 is independent of Data Memory Register Bank 0, used for stepper motor control. The use of two independent register banks greatly simplfies the software flow, and helps to ensure the accuracy of event sequences that must be handled in parallel. Each register bank must be initialized only once for any entry to either the Carriage Stepper Motor Drive or Character Processing subroutines. A single UPI-42 Assembly Language instruction selects the appropriate register bank. Initializing the character processing registers includes loading the maximum character count (80), dot matrix size count (6), and CB start address. The CB start address is print direction dependant, as described in Block 4, above.

Character processing reads a character from the CB, tests for control codes, translates the character to dots, and conditionally exits, returning to the Carriage Stepper Motor Drive subroutine. Flow Chart 8 details the character processing subroutine.



Flow Chart No. 8. Process Characters for Printing

Each character requires six steps of the carriage stepper motor to print; five for the 5 character dot columns and 1 for the blank dot column between each character. Reading a character from the CB and character-to-dot pattern translation takes place during the last character dot column, or blank column, time.

The first character line entry to the Character Processing subroutine appears to the software as if a last character dot column (blank column) had been entered. The next character, in this case the first character in the line, is translated and printing can begin. This method of intiializing the Character Processing subroutine utilizes the same software for both start-up and normal character flow. Once a character code has been translated to a dot matrix pattern starting address in the look-up table, all subsequent entries to the Character Processing subroutine simply advance the dot column data address and outputs the data.

The decision to translate the character to dots during the blank column time was an arbitary one. As was the choice of the blank column following rather than preceding the actual character dot matrix printing.

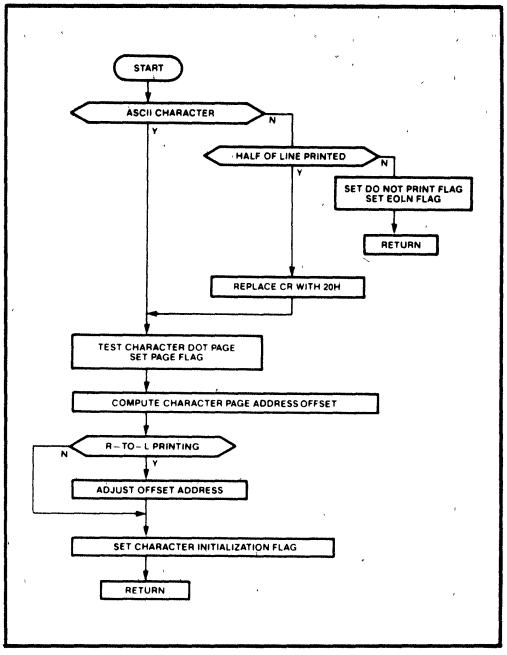
4. Translate Character-to-Dots

Character-to-dot pattern translation involves converting the ASCII code into a look-up table address, where the first of the five bytes of charcter dot column data is stored. The address is then incremented for the next column of dot pattern data until the full character has been printed.

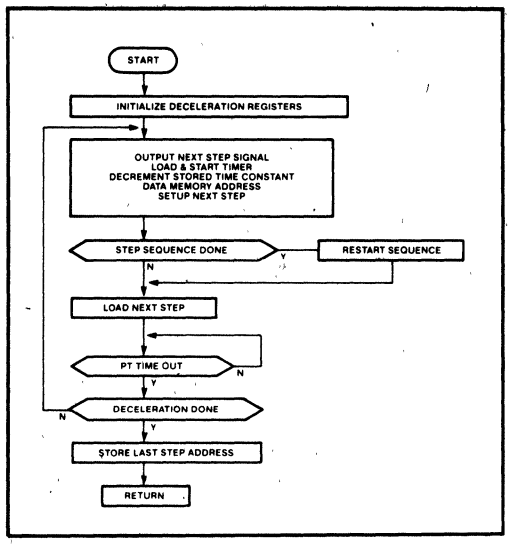
The dot pattern look-up table occupies two pages, or approximately 512 bytes of Program Memory. A printable ASCII character is tested for its dot pattern location page and the offset address, from zero, on that page. Both the page test and page offset calculations use two's complement arithmetic, with a jump on carry or not carry causing the appropriate branching. Once the pattern page and address are determined the indirect addressing and data move instructions are used to read and output the data to the print head solenoids. Flowchart 9 details the Character-to-Dots Translation subroutine.

In the case of R-to-L printing, although the translation operation is the same, the character is printed in reverse. This requires that the character dot pattern address be incremented by five, before printing begins, so that the first dot column data output is the last dot column data of the character. The dot pattern look-up table address is then decremented rather that incremented, as in L-to-R printing, for the balance of the character. Translation still takes place during the last character dot column, the blank column, and the blank column follows the character matrix.

Only one control code, a Carriage Return (CR), is encountered by the character translation subroutine. Linefeed (LF) characters are stripped off by the CB Fill subroutine. If a CR code is detected the software tests for a mid-line exit condition; less than half the line printed exits the stepper motor drive subroutine and HOMEs the print head assembly before printing the next line. If the test fails, more than half the line has been printed, the CR is replaced by a 20H (Space character) and printing continues for the balance of the line; the space characters padding the CB are printed.



Flow Chart No. 9. Translate Character-to-Dots



Flow Chart No. 10. Decelerate Carriage Stepper Motor

As mentioned above, the character dots are printed and the print head trigger is fired when the PTS signal is detected and verified and the carriage stepper motor is At Speed.

When the character to print test fails the CB Buffer size count equals zero, the Carriage Stepper Motor Drive subroutine exit flags are set, and the flow passes to the Deceleration and Delay subroutines and programs returns to the main program flow.

9. Decelerate Carriage Stepper Motor

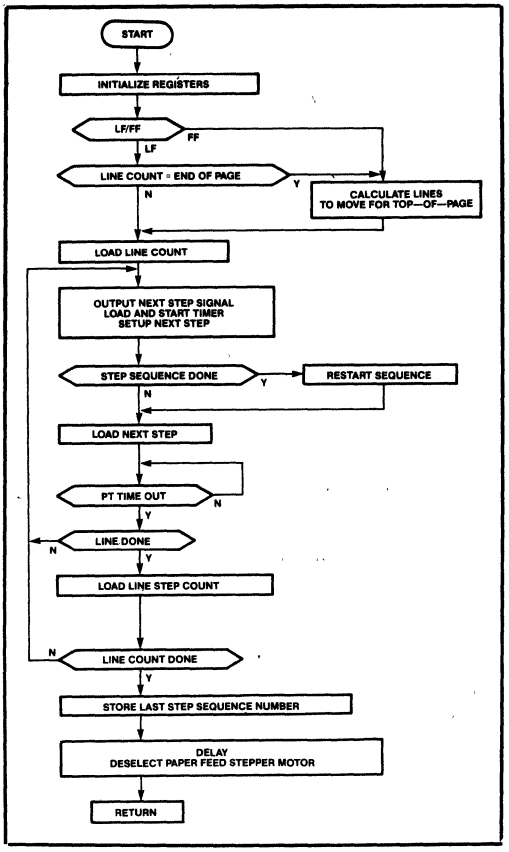
The transition from the Carriage Stepper Motor Drive subroutine to the Deceleration subroutine outputs the next step signal in sequence, and then initializes the Deceleration subroutine registers; Stored Time Constants Data Memory buffer end address and size. The Stored Time Constant Buffer is a LIFO for deceleration of the carriage stepper motor. The buffer size is used as the step count. When the step count decrements to zero, the step signal output is terminated, and the last step sequence number is stored in the carriage stepper motor Next Step pseudo register. The last step sequence number is recalled, during initialization of the next carriage stepper motor drive, as the basis of the next step data signal to be output. See Flow Chart 10.

When the carriage stepper motor is decelerated, Failsafe protection and PTS monitoring are not necessary. The Deceleration subroutine acts as its own failsafe mechanism. Should the stepper motor hang-up, the subroutine would exit and deselect the motor in sufficient time to protect the motor from burnout. Since neither Failsafe nor print head solenoid firing take place during deceleration, PTS is not needed. PT is replaced by the Stored Time Constant values in Data Memory. The Deceleration subroutine determines the next step signal to output, loads the Timer with the Stored Time Constant, starts the UPI-42 Timer, and loops until time out. The subroutine loops to output the next step until all of the Stored Time Constants have been used. The program returns to the Carriage Stepper Motor Drive subroutine and the motor is deselected following the Delay subroutine execution. The Delay subroutine is called to stablize the stepper motor before it is deselected. During the DELAY subroutine, the IBF interrupt is enabled and characters are processed. A paper feed is forced following the carriage stepper motor being deselected.

10. Paper Feed Stepper Motor Drive

The paper feed stepper motor subroutine outputs a predefined number of step signals to advance the paper, in one line increments, for the required number of lines. The number of step signals per line increment is a function of the defined number of lines per inch, given the distance the paper moves in one step. Figure 16 lists three step (or pulse) count and line spacing configura-

tions, as well as the distance the paper moves in one step. Standard 6 lines per inch spacing has been implemented in this Application Note (Appendix B details how variable line spacing could be implemented). Flowchart 11 illustrates the Paper Feed subroutine.



Flow Chart No. 11. Paper Feed Stepper Motor Drive

The number of lines the paper is to be moved is called the "Line Count." The Line Count defaults to one unless the Formfeed flag is set, or the total number of lines previously moved equals a full page. The default total lines per page for this application is 66. When the total number of lines moved equals 66, the paper is moved to the top of the next page. The Top-of-Page is set at power-on or reset.

If the Formfeed flag has been set in the Character Buffer Fill subroutine, the software calculates the number of lines needed for a top of next page paper feed. The resulting line count is loaded in the Line Count Register. The Paper Feed subroutine loops on the line count until done and then returns to the main program body.

Once the Paper Feed subroutine is complete, the software loops to test the End of File (EOF) Flag (see Flow Chart 1). If EOF is set, the print head assembly is moved to the HOME position, the program again enters the External Status Switch Test subroutine, and begins polling the external status switches. If EOF is not set, the program directly calls the External Status Switch Check subroutine, and the program repeats for the next line.

CONCLUSION

Although the full speed, 12 MHz, of the UPI-42 was used, the actual speed required is approximately 8-9 MHz. 1400 bytes of the available 2K bytes of Program Memory were used; 500 bytes for the 95 character ASCII code dot pattern look-up table, 900 bytes for operational software. This means that the UPI-42 has excess processing power and memory space for implementing the additional functions such as those listed below and discussed in Appendix B.

Special Characters or Symbols
Lower Case Descenders
Inline Control Codes
Different Character Formats
Variable Line Spacing

The software developed for this Application Note was not fully optimized and could be further packed by combining functions. This would require creating another status register, which could also serve to implement some of the features listed above. Since the full 16 byte stack is not used for subroutine nesting, there are 6-8 bytes of Program Stack Data Memory that could be used for this purpose. In several places, extra code was added for clarity of the Application Note. For example, each status byte flag is set with a separate instruction, using a equate label, rather than setting several flags simultaneously at the same point in the code.

This Application Note has demonstrated that the UPI-42 is easily capable of independently controlling a complex peripheral device requiring real time event monitoring. The moderate size of the program required to implement this application attests to the effectiveness of the UPI-42 for peripheral control.

APPENDIX A.
SOFTWARE LISTING

```
   1 $MOD42  TITLE('UPI 42 APP NOTE');
   2 $MACROFILE NOSYMBOLS NOGEN DEBUG
   3
   4 $INCLUDE(:F1:ANECD.OV1)
=  5 ; PG
=  6
=  7 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
=  8 ;               Complex Peripheral Control With the UPI-42
=  9
= 10 ;               Intel Corporation
= 11 ;               3065 Bowers Avenue
= 12 ;               Santa Clara, Ca. 95051
= 13
= 14
= 15 ;               Written By      Christopher Scott
= 16
= 17
= 18 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 19
= 20 ;               Notes and Comments
= 21
= 22 ;       Three Assembly Language files comprise the full Application
= 23 ;       Note source code;
= 24
= 25 ;       1. ANECD.OV1     App Note Equates, Constants, Declarations . Overlay
= 26
= 27 ;       2. 42ANC.SRC     UPI-42 App Note Code Source
= 28
= 29 ;       3. CHRTBL.OV1    Character Table . Overlay  (Character Lookup Tables)
= 30
= 31
= 32
= 33
= 34 ; PG
= 35 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 36 ;               Equates, Constants and System Definitions
= 37 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
= 38
= 39 ;       Data & Program Memory Allocations
= 40
= 41 ;       Program Memory
= 42
= 43 ;       Page No.   Hex Addr      Description
= 44 ; ---------------------------------------------------------------------------
= 45
= 46 ;       Page 7   1792-2047      Char to Dot pattern lookup table
= 47 ;                                 Page 2:  ASCII 50H-7FH (N-~)
= 48 ;       Page 6   1536-1791      Char to Dot pattern lookup table
= 49 ;                                 Page 1:  ASCII 20H-4FH (sp-M)
= 50 ;       Page 5   1280-1535      Misc called routines:
= 51 ;                                 InitAl/AllOff
= 52 ;                                 Clear Data Memory
= 53 ;                                 CR Home
= 54 ;                                 Char Print Test -  load Ascii char codes
= 55 ;                                 Initialize CR Stpr Mtr
= 56 ;                                 Delay: short/long/very long
= 57 ;                                 Stpr Mtr deselect
= 58 ;       Page 4   1024-1279      PaperFeed Stpr Mtr Init and Drive
= 59 ;       Page 3    768-1023      Stpr Mtr Phase LookUp Table - Indexed
= 60 ;                                 Character Translation and processing
= 61 ;                                 PrintHead firing
= 62 ;       Page 2    512-767       Stpr Mtr Accel. Time calc. and memorization
= 63 ;                                 Stpr Mtr Deceleration
= 64 ;       Page 1    256-511       SMDriv (FAccel/RAccel) - Forward & Reverse
= 65 ;                                   Stpr Mtr acceleration & drive
= 66 ;       Page 0      0-255       Initialization Jmp-on-Reset
= 67 ;                                 Program Body - all calls
= 68 ;                                 Character Input test and Char Buffer fill loop
= 69 ;                                 Interrupt service routines
= 70
```

```
                        71 ; PG
                        72 ; ------------------------------------------------------------------------------
                        73 ;       Data Memory
                        74 ; ------------------------------------------------------------------------------
                        75 ;         Dec.        Hex           Description
                        76 ;         -------     --------      ---------------------------------------------
                        77 ;    TOP
                        78 ;    ---------------------
                        79 ;       48-127      2F-7FH         80 Character Line Buffer
                        80 ;        37-47      25-2EH         Stpr Mtr Accel/Decel time, memorization
                        81 ;           36         24H         Unused
                        82 ;           35         23H         Char Print test ASCII code start tmp store
                        83 ;           34         22H         LF SM last Phz Inderect Addr psuedo reg
                        84 ;           33         21H         CR SM Forward/Reverse last Phz psuedo reg
                        85 ;           32         20H         Psuedo Reg:  Last Phase of stpr mtr not
                        86 ;                                              being driven
                        87 ;        24-31      18-1FH         Register Bank 1:  Character Handling
                        88 ;         8-23       8-17H         8 Level Stack
                        89 ;         0-7        0-07H         Register Bank 0:  Stpr Mtr F/R Accel/Drive
                        90 ;    ---------------------
                        91 ;    BOTTOM
                        92
                        93 ;         Data Memory Equates:
                        94 ; ------------------------------------------------------------------------------
                        95
0050                    96 CHBfSZ   EQU     50H             ;char buffer size 0-79 = 80
00D9                    97 H1fCpl   Equ     0D9H            ;Cpl(1/2 CbBfSz) => cpl of 27H = 0D9H
                        98
007F                    99 FCBfSt   Equ     7fH             ;start of char buffer
0080                   100 FCBfIS   Equ     80H             ;init CB strt-allows xtra Dec by 1
002F                   101 RCBfIS   Equ     2FH             ;init CB strt-allows xtra Inc by 1
0051                   102 ChBfIS   Equ     81              ;load char cnt reg w/char bufr Init Size
                       103
002F                   104 ENDBUF   EQU     2FH             ;END OF CHAR BUFFER
0008                   105 ASBfSz   EQU     08H             ;Accelerate stpr mtr buf count
000A                   106 DSBfSz   Equ     0AH             ;Decelerate stpr mtr buf count
002F                   107 SMBFST   EQU     2FH             ;STPR MTR BUFFER START
0025                   108 SMBEnd   Equ     25H             ;Stpr Mtr Data Memory Address end
007F                   109 DMTop    Equ     7FH             ;Data Memory Top
005D                   110 DMSize   Equ     93              ;Data Memory Size (less two working reg's)
                       111
0020                   112 LastPh   Equ     20H             ;last phz psuedo reg addr
0021                   113 CPSAdr   EQU     21H             ;CR phz psuedo reg
0022                   114 LPSAdr   Equ     22H             ;LF phz psuedo reg
0023                   115 PTAscS   Equ     23H             ;Char Print Test code start tmp store
                       116
                       117 ; PG
                       118 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       119 ;       Register allocation
                       120 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       121
                       122 ;         All Indirect Data Memory Addressing via @Rn Inst must use
                       123 ;           only registers 0 & 1 of either register bank.  Any other will
                       124 ;           be rejected by the Assembler
                       125 ;         Last character in lable indicates Register Bank referenced
                       126
                       127 ;       Register Bank 0
                       128 ; ------------------------------------------------------------------------------
0000                   129 TmpR0O   Equ     R0              ;RB0 Temporary Register
0001                   130 TStrRO   EQU     R1              ;Store Time Register RB0
0002                   131 GStR2O   EQU     R2              ;General Status Register RB0
0003                   132 PhzR3O   EQU     R3              ;Stpr Mtr Phase Register RB0
0004                   133 CntR4O   Equ     R4              ;Count Reg. Phase count-Stpr Mtr loops
                       134                                  ;   Accel/Decel Count
0005                   135 TConRO   Equ     R5              ;Time constant reg RB0
0006                   136 LnCtRO   Equ     R6              ;Line count
                       137
0007                   138 OpnR7O   EQU     R7              ;available
                       139
                       140 ;         Register Bank 0 Data Memory Address
                       141 ; ------------------------------------------------------------------------------
```

```
0000          = 142 TmpA00  Equ     00H             ;Temporary Register DM address
0001          = 143 TStrA0  EQU     01H             ;Time Store Register  DM address
0002          = 144 GStRAd  Equ     02H             ;RBO Char Status Reg  DM address
0003          = 145 PhzA20  EQU     03H             ;Stpr Mtr Phase Register DM address
0004          = 146 CntRA0  Equ     04H             ;Count Reg. Phase count-Stpr Mtr loops
              = 147                                 ;  Accel/Decel Count DM address
0005          = 148 TConA0  Equ     05H             ;Time constant reg DM address
0006          = 149 LnCtA0  Equ     06H             ;Line Count Register DM address
              = 150
0007          = 151 OpnA70  EQU     07H             ;available
              = 152
              = 153
              = 154 ; PG
              = 155 ; ------------------------------------------------------------------------
              = 156 ;       RBO  Status Byte Bit Definition
              = 157 ; ------------------------------------------------------------------------
              = 158
              = 159 ;       Bit             Definition
              = 160 ;       -----           ----------------------------------------------------
              = 161 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
              = 162 ;       6               1 = Sink / 0 = Not Sinked, Print Head Init and Fire
              = 163 ;       5               Stpr Mtr at speed and CR not left of Home
              = 164 ;       4               Accel/Decel Init, 1 = Done / 0 = Not Done
              = 165
              = 166 ;       3               1 = FailSafe / 0 = Constant, Time Window
              = 167 ;       2               1 = Form Feed / 0 = Line Feed
              = 168 ;       1               1 = Do Not Print / 0 = Print
              = 169 ;       0               FAccel/DAccel drive Ready = 1/NotRdy = 0 (exit
              = 170 ;                         drive & decel stpr mtr)
              = 171
              = 172 ;       Bit Masks:      RBO
              = 173 ;       Stepper Motor control bit masks function on GStR10
              = 174 ; ------------------------------------------------------------
              = 175
              = 176 LRPrnt  Equ     80H             ;Left to Right Printing (ORL)
              = 177 RLPrnt  Equ     7FH             ;Right to Left Printing (ANL)
              = 178 SnkSet  Equ     40H             ;Ready Print flag
              = 179 ClrSnk  Equ     0BFH            ;clear Ready to Print Bit
              = 180 AtSpdF  Equ     20H             ;Stpr Mtr at constant speed
              = 181 NAtSpd  Equ     0DFH            ;Stpr Mtr Not at speed
              = 182 ADIntD  Equ     10H             ;Accel/Decel Init Done
              = 183 ADIntN  Equ     0EFH            ;Accel/Decel Init Not Done
              = 184
              = 185 FsCTm   Equ     08H             ;FailSafe/Constant Time
              = 186 ClrFsC  Equ     0F7H            ;Clear FailSafe/Const time flag
              = 187 FrmFd   Equ     04H             ;do formfeed
              = 188 LineFd  Equ     0FBH            ;do line feed
              = 189 DoNotP  Equ     02H             ;set Do Not Print Stat bit
              = 190 OkPrnt  Equ     0FDH            ;Reset - Ok to Print
              = 191 Ready   Equ     01H             ;Ready drive Stpr Mtr
              = 192 NotRdy  Equ     0FEH            ;Not Ready exit Stpr Mtr drive
              = 193
              = 194
              = 195 ; PG
              = 196 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 197 ;       Register allocation    (cont)
              = 198 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 199
              = 200
              = 201 ;       Register Bank 1
              = 202 ; ------------------------------------------------------------
0000          = 203 TmpR10  Equ     R0
0001          = 204 CAdrR1  EQU     R1              ;char data memory addr register
0002          = 205 ChStR1  EQU     R2              ;char processing status byte register
0003          = 206 CDtCR1  EQU     R3              ;Char Dot count register
0004          = 207 CDotR1  Equ     R4              ;Char dot temp storage register
0005          = 208 CCntR1  Equ     R5              ;Char count temp register
0006          = 209 StrCR1  EQU     R6              ;Store Char Register
              = 210
0007          = 211 OpnR71  EQU     R7              ;Available
              = 212
              = 213 ;       Register Bank 1 Data Memory Address
              = 214 ; ------------------------------------------------------------
```

```
0018          = 215 TmpA10  Equ     24              ;temporary/scratch register
0019          = 216 ChARR1  EQU     25              ;char data memory addr register
001A          = 217 ChStAd  Equ     26              ;RB1 Char Status Reg address
001B          = 218 CDtCA1  EQU     27              ;Char Dot count register
001C          = 219 CDotA1  Equ     28              ;Char dot temp storage register
001D          = 220 CCntA1  Equ     29              ;Char count temp register
001E          = 221 StrCA1  EQU     30              ;Store Char Register
              = 222
001F          = 223 OpnA71  EQU     31              ;Available
              = 224
              = 225
              = 226 ; PG
              = 227 ; ------------------------------------------------------------------------
              = 228 ;       RB1  Status Byte Bit Definition
              = 229 ; ------------------------------------------------------------------------
              = 230
              = 231 ;       Bit             Definition
              = 232 ;       -----           ----------------------------------------
              = 233 ;       7               Stpr Mtr Direction:  L-to-R = 1, R-to-L = 0
              = 234 ;       6               Char Init, 1 =  Done / 0 = Not Done
              = 235 ;       5               Char Lookup Table Page: 1 = Pg1, 0 = Pg2
              = 236 ;       4               1 = Test / 0 = Normal char print/input
              = 237
              = 238 ;       3               1 = EOF / 0 = Not EOF
              = 239 ;       2               Full = 1/Not Full = 0, Line in Char Buffer
              = 240 ;       1               1 = CR/(LF) / 0 = Not CR/(LF)
              = 241 ;       0               1 = Init / 0 = Do Not Init, CB registers done
              = 242
              = 243 ;       Bit Masks:      RB1
              = 244 ;       Character printing bit masks function on ChStR1
              = 245 ; ------------------------------------------------------------
0080          = 246 ChrPrn  Equ     80H             ;Stpr Mtr Direction:  L-to-R = 1
007F          = 247 ClrCPr  Equ     7FH             ;Stpr Mtr Direction:  R-to-L = 0
0040          = 248 ChIntD  Equ     040H            ;Set Char Init Done
00BF          = 249 CIntND  Equ     0BFH            ;Reset Char Init Not Done
0020          = 250 ChOnP1  Equ     20H             ;Page 1 char, set rentry bit (ORL)
00DF          = 251 ChOnP2  Equ     0DFH            ;Page 2 char, reset rentry bit (ANL)
0010          = 252 TstPrn  Equ     10H             ;Char print test
00EF          = 253 NrmPrn  Equ     0EFH            ;Normal char input
              = 254
0008          = 255 EOF     Equ     08H             ;set EOF Flag
00F7          = 256 ClrEOF  Equ     0F7H            ;clear EOF flag - Not EOF
0004          = 257 CRLF    Equ     04H             ;CR/LF
00FB          = 258 ClrCR   Equ     0FBH            ;Clear CR/LF
0002          = 259 CBFLn   Equ     02H             ;Full Line in Char Buffer
00FD          = 260 NCBFLn  Equ     0FDH            ;Not Full Line in Char Buffer
0001          = 261 IntCBR  Equ     01H             ;Init of CB registers done
00FE          = 262 ClICBR  Equ     0FEH            ;Init of CB registers not done
              = 263
              = 264
              = 265 ; PG
              = 266 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 267 ;               Equates  (cont)
              = 268 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 269
              = 270 ;       Misc
              = 271 ; ------------------------------------------------------------
0004          = 272 RLPShf  Equ     04H             ;R-to-L print lookup table addr shift
              = 273
0020          = 274 Ascii   Equ     20H             ;hex nmbr of first Ascii Char
007F          = 275 AscLst  Equ     7FH             ;hex nmbr of last Ascii Char
              = 276
00F3          = 277 CRCpl   Equ     0F3H            ;ASCII control code 2's complement
00F6          = 278 LFCpl   Equ     0F6H            ; "
00F4          = 279 FFCpl   Equ     0F4H            ; "
00E5          = 280 EscCpl  Equ     0E5H            ; "
00E0          = 281 AscCpl  Equ     0E0H            ; "
00C8          = 282 FTCpl   Equ     0C8H            ; "
000D          = 283 CR      Equ     0DH             ;Ascii code (hex)
0020          = 284 Space   Equ     20H             ;Ascii code (hex)
              = 285
0081          = 286 LAsEnd  Equ     81H             ;Ascii End 2's cpl - test line start
0082          = 287 PAsEnd  Equ     82H             ;Ascii End 2's cpl - within line print
007F          = 288 AscStp  Equ     7fH             ;Ascii mask, strip off MSB
0042          = 289 PgLnCt  Equ     66              ;Page Line Count: Default = 66
00C4          = 290 PgLCpl  Equ     0C4H            ;Printed lines per page test
001B          = 291 EOFCpl  Equ     1BH             ;EOF ascii code cpl
              = 292
              = 293 ;       Loop count values
              = 294 ; ------------------------------------------------------------
```

```
0006          = 295 NDtCCt  Equ     06H             ;Normal Dot Column Count
000A          = 296 EDtCCt  Equ     0AH             ;Expanded Dot Column Count
0004          = 297 PHCnt1  EQU     04H             ;NUMBER OF SM PHASES ON INIT
              = 298
0004          = 299 ILFCnt  Equ     04              ;Init LF step/phz count
0024          = 300 LPI6p6  Equ     36              ;Lines Per Inch 6.6
001B          = 301 LPI8p8  Equ     27              ;Lines Per Inch 8.8
0018          = 302 LPI10   Equ     24              ;Lines Per Inch 10
              = 303
0001          = 304 LineCt  Equ     01              ;linefeed count
0042          = 305 FmFdCt  Equ     66              ;lines per formfeed count
0003          = 306 Status  EQU     03H             ;SEE BELOW FOR STATUS BYTE DEF.
              = 307                                 ;   TEST:  SET FOR CR STPR MTR CONTROL
              = 308
              = 309
              = 310 ; PG
              = 311 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 312 ;       TIMER VALUES - UPI Timer/Counter is UP Counter
              = 313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 314 ;       12 MHz Clk timings
              = 315 ; ---------------------------------------------------------------
0080          = 316 DLYCL   EQU     80H             ;DELAY COUNT Long
0030          = 317 DLYCS   EQU     30H             ;DELAY COUNT Short
00CC          = 318 DlyTim  EQU     256-52          ;TIME DELAY constant ~2.0mS
0000          = 319 FailTm  EQU     256-256         ;FailSafe TIME = ~17.0mS
00CC          = 320 CrTmr1  EQU     256-52          ;CR Stpr Mtr Phase TIME = ~2.08mS
008A          = 321 CrTmr2  EQU     256-70          ;CR Stpr Mtr Phase TIME = ~2.40mS
0092          = 322 CrTmr3  EQU     256-110         ;CR Stpr Mtr Phase TIME = ~4.16mS
00C0          = 323 IntTm2  EQU     256-64          ;Init Stpr Mtr Phase TIME = ~2.40mS
0098          = 324 LFTMR1  EQU     256-104         ;LF Stpr Mtr Phase TIME = ~4.16mS
              = 325
              = 326 ;       I/O port bit masks
00DF          = 327 NotBsy  Equ     0DFH            ;Not Busy
0020          = 328 Busy    Equ     20H             ;Busy
00EF          = 329 Ack     Equ     0EFH            ;Ack
0010          = 330 ReSAck  Equ     10H             ;ReSet Ack
              = 331
              = 332 ;       Misc bit Masks
000C          = 333 StrpLF  Equ     0CH             ;Strip off all bits but LF Stpr Mtr
0003          = 334 StrpCR  Equ     03H             ;Strip off all bits but CR Stpr Mtr
              = 335
              = 336 ;       Print Head fires on low going edge of Trigger
              = 337 ;         bit #9 in dot column is masked off, always:  P2, bit 6
0040          = 338 PTRQLO  EQU     40H             ;PH TRIGGER BIT - LOW
00C0          = 339 PTRQHI  EQU     0C0H            ;PH TRIGGER BIT - HIGH
              = 340
              = 341 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 342 ;       Stepper Motor Phase State Equates
              = 343 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              = 344
              = 345 ,       Stepper Motor Phase Shift Index Offset Offset
0000          = 346 FStCRP  EQU     00H             ;F CR stpr mtr phase data start addr
0003          = 347 RStCRP  EQU     03H             ;R CR stpr mtr phase data start addr
0008          = 348 STLFF   Equ     08H             ;Paper feed stpr mtr phase data start addr
              = 349
              = 350 ;       CARRIAGE STEPPER MOTOR PHASE EQUATES
              = 351 ;       Forward (1 thru 4)  &  Reverse (4 thru 1) :
0001          = 352 CRMFP1  EQU     01B             ;CR STPR MTR PHASE 1
0003          = 353 CRMFP2  EQU     11B             ;CR STPR MTR PHASE 2
0002          = 354 CRMFP3  EQU     10B             ;CR STPR MTR PHASE 3
0000          = 355 CRMFP4  EQU     00B             ;CR STPR MTR PHASE 4
              = 356
              = 357         ;LINE FEED STEPPER MOTOR PHASE EQUATES
              = 358 ;       Forward:
0004          = 359 LFMFP1  EQU     0100B           ;LF STPR MTR PHASE 1
000C          = 360 LFMFP2  EQU     1100B           ;LF STPR MTR PHASE 2
0008          = 361 LFMFP3  EQU     1000B           ;LF STPR MTR PHASE 3
0000          = 362 LFMFP4  EQU     0000B           ;LF STPR MTR PHASE 4
              = 363
              = 364 ; PG
```

```
                        = 365 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        = 366 ;       STEPPER MOTOR SELECT & CONTROL [CURRENT LIMITING]
                        = 367 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        = 368
                        = 369 ;       PORT BIT ASSIGNMENT:
                        = 370
                        = 371 ;               \ \ \
                        = 372 ;               S S S -
                        = 373 ;               L C C
                        = 374 ;               F R R
                        = 375 ;                 8 1
                        = 376 ;                 0 3
                        = 377 ;                   2
                        = 378 ;       ------------------
                        = 379 ;               5 5 5 5
                        = 380 ;               3 2 1 0
                        = 381 ;       ------------------
                        = 382 ; CODING:
                        = 383 ;       SLF     0 1 1 0         06H
                        = 384 ;       SCR80   1 0 0 0         0AH
                        = 385 ;       SCR132  1 1 0 0         0CH
                        = 386 ;       SMOFF   1 1 1 0         0EH
                        = 387 ;               W/SCR80 & SCR132 '0' [BOTH SELECTED]
                        = 388 ;                 DEFAULT IS TO 80 COL.
                        = 389 ;                 [DO NOT KNOW WHETHER SCR80='0' WILL
                        = 390 ;                 SELECT 80 COL ONLY] - REQUIRES TEST.
                        = 391
0008                    = 392 SCR80   EQU     08H             ;SELECT CR STPR MTR - 80 COL
                        = 393                                 ;  w/LF STPR MTR OFF
000C                    = 394 SCR132  EQU     0CH             ;SELECT CR STPR MTR - 132 COL
                        = 395                                 ;  w/LF STPR MTR OFF
0006                    = 396 SLF     EQU     06H             ;SELECT LF STPR MTR ON
                        = 397                                 ;  w/CR STPR MTR OFF
000E                    = 398 SMOFF   EQU     0EH             ;SELECT CR & LF STPR MTR OFF
                          399
                          400
                          401 ; PG
                          402 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                          403 ;               MAIN PROGRAM BODY
                          404 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                          405
                          406 ;       Power On / Reset Program Entry
                          407
                          408 ;       PROGRAM START
                          409
0000                      410         Org     00H
                          411
0000 040B                 412 START:  JMP     RESET
                          413
                          414 ;       INPUT BUFFER FULL INTERRUPT CALL ENTRY AND VECTOR
0003                      415         ORG     03H
0003 1425                 416 IBFIV:  Call    IBFIS
0005 93                   417         RETR
                          418 ;       TIMER OVERFLOW INTERRUPT CALL ENTRY AND VECTOR
0007                      419         ORG     07H
0007 1429                 420 TMRIV:  Call    TMRIS
0009 C5                   421         SEL     RB0
000A 83                   422         Ret
                          423
                          424 ;               INITIALIZATION
                          425
000B 15                   426 ReseT:  Dis     I
000C 35                   427         Dis     TCntI
000D B40F                 428         Call    InitAl          ;set all critical outputs inactive
000F B42F                 429         Call    ClrDM           ;clear all data memory - 93H to 7FH
                          430                                 ;  do not clear RB0, RB1 or Stack
0011 B44B                 431         Call    InitCR          ;CALL CR SM POWER ON INIT
0013 9400                 432         Call    InitLF          ;CALL LF SM POWER ON INIT
                          433
                          434 ;               MAIN PROGRAM LOOP
                          435 ;               All program segments are called from here
                          436
0015 B422                 437 Home:   Call    CRHome          ;Call Home CR routine -
                          438                                 ;  fixes logical and physical CR Home
0017 B400                 439         Call    Defalt          ;set default register values
0019 142C                 440 CBInpt: Call    ESCBfF          ;Stat Switch / CB Input Service Test
                          441                                 ; test for:  CB full/fill, LF, FF,
                          442                                 ;            Char Prnt Test
```

```
001B 3400          443 Repeat: Call    SMDriv          ;Call Forward Stpr Mtr Drive
001D 940D          444         Call    LFDriv          ;Call Linefeed Stpr Mtr Drive
001F D5            445         SEL     RB1
0020 FA            446         Mov     A,ChStR1        ;get the Char Status Register RB1
0021 7215          447         JB3     Home            ;jump to CR SM Home if EOF bit set
0023 0419          448         Jmp     CBInpt          ;loop to Char Buffer Input test
                   449
                   450 ; PG
                   451 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   452 ;                   Interrupt Service Routine
                   453 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   454
                   455 ; ---------------------------------------------------------------------
                   456 ;                   Input Buffer Full Interrupt Service Routine
                   457 ; ---------------------------------------------------------------------
                   458
                   459 IBFIS:
                   460 ; ---------------------------------------------------------------------
                   461 ;       Acknowledge Char input and set Hold/Busy Active
0025 8A20          462         ORL     P2,#Busy        ;get & set DBB ACK/Busy Bits
0027 15            463         Dis     I               ;disable IBF interrupts
0028 83            464         Ret
                   465
                   466 ; ---------------------------------------------------------------------
                   467 ;                   Timer / Counter Interrupt Service Routine
                   468 ; ---------------------------------------------------------------------
                   469 ;       ITF interrupt service routine disables all intr during
                   470 ;         stpr mtr phase shifting
0029 15            471 TMRIS.  Dis     I               ;disable IBF interrupts
002A 35            472         Dis     TCntI           ;dicable ITF interrupts
002B 83            473         Ret
                   474
                   475 ; PG
                   476 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   477 ;       External Status Switch Check/Char. Buffer Fill
                   478 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   479 ESCBfF:                         ;Prep for normal character handling/input
002C D5            480         SEL     RB1
002D FA            481         Mov     A,ChStR1        ;get the character stat reg byte
002E 53EF          482         ANL     A,#NrmPrn       ;set normal character input
0030 AA            483         Mov     ChStR1,A        ;store the stat byte
0031 C5            484         SEL     RB0
                   485
                   486 ;       Test External Status Port
0032 0F            487         MovD    A,P7            ;get the stat switch port bits
0033 123D          488         JB0     FormFd          ;  service Formfeed
0035 3245          489         JB1     LinFd           ;  service Linefeed
0037 5249          490         JB2     ChrTst          ;  service Character TEST
0039 725E          491         JB3     OnLine          ;  service Char Buffer Check/Fill
003B 042C          492         Jmp     ESCBfF          ;Loop
                   493 ; ---------------------------------------------------------------------
003D FA            494 FormFd: Mov     A,GStR20        ;get the status byte
003E 4304          495         ORL     A,#FrmFd        ;set the formfeed stat flag
0040 AA            496         Mov     GStR20,A        ;store trhe status byte
0041 940D          497         Call    LfDriv          ;do a formfeed
0043 042C          498         Jmp     ESCBfF
                   499 ; ---------------------------------------------------------------------
0045 940D          500 LinFd:  Call    LfDriv          ;do a line drive
0047 042C          501         Jmp     ESCBfF
                   502 ; ---------------------------------------------------------------------
0049 D5            503 ChrTst: SEL     RB1
004A FA            504         Mov     A,ChStR1        ;get the character stat reg byte
004B 4310          505         ORL     A,#TstPrn       ;set character test flag
004D AA            506         Mov     ChStR1,A        ;store the stat byte
004E B823          507         Mov     TmpR10,#PTAscS  ;load the psuedo Ascii code tmp reg addr
0050 F0            508         Mov     A,@TmpR10       ;get the inc'd ascii code
0051 0381          509         ADD     A,#LAsEnd       ;test for code end
0053 9657          510         JNZ     AscCLd          ;if not code end jmp to load
                   511                                 ;if end restart ascii at begining
0055 B020          512         Mov     @TmpR10,#Ascii  ;store the ascii code start
0057 F0            513 AscCLd: Mov     A,@TmpR10       ;get the ascii code again
0058 AF            514         Mov     OpnR71,A        ;place in the empty register
0059 10            515         Inc     @TmpR10         ;Inc start ASCII char in data memory
005A B439          516         Call    PrnTst          ;call the DM load procedure
005C C5            517         SEL     RB0             ;reselect reg bank 0
005D 83            518         Ret
                   519 ; ---------------------------------------------------------------------
```

```
005E D5          520 OnLine:  SEL     RB1             ;select char buffer registers
005F 05          521          EN      I               ;enable interrupts
0060 FA          522 CBfCk1:  Mov     A,ChStR1        ;get the Char Stat Byte
0061 3267        523          JB1     CBCkEx          ;if Chr Buf has full line exit
0063 146D        524 IBfCk:   Call    CBFill          ;read a char into Char Buffer
0065 0460        525          Jmp     CBFCk1          ;loop to Char Buf Ful test
0067 C5          526 CBCkEx:  SEL     RB0
0068 83          527          Ret
                 528
                 529 ; PG
                 530 ; -------------------------------------------------------------------------
                 531 ;       Character Input
                 532 ; -------------------------------------------------------------------------
                 533 ;       Input Buffer Full service routine:  test for Char buffer full-exit
                 534 ;         else load char into char buffer
0069 D5          535 IBFSrv:  SEL     RB1
006A FA          536          Mov     A,ChStR1        ;get the RB0 stat byte
006B 32EC        537          JB1     CBFull          ;if Do Not Print Bit Set - EXIT
006D 527C        538 CBFill:  JB2     CBPad           ;test for CB padding flag
                 539                                  ;  if not pad enable char input
                 540                                  ;  tell the host to send char's
006F 05          541          EN      I
0070 D6EC        542          JNIBF   CBF1Ex
                 543 ; -------------------------------------------------------------------------
                 544 ;       Acknowledge Char input and set Hold/Busy Active
0072 FA          545          Mov     A,ChStR1        ;get the RB1 Char Stat Byte
0073 127C        546          JBO     SkpInt          ;test for CB has been Initialized
                 547 ; -------------------------------------------------------------------------
                 548 ;       Init of all Char handling registers
0075 4301        549          ORL     A,#IntCBR       ;set CB Reg skip Initialization stat bit
0077 AA          550          Mov     ChStR1,A        ;save the altered stat byte
0078 B97F        551          Mov     CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
007A BD50        552          Mov     CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 553 CBPad:
007C ED86        554 SkpInt:  DJNZ    CCntR1,LdChar   ;DECREMENT BUFFER SIZE
007E FA          555          Mov     A,ChStR1        ;get the status byte
007F 4302        556          ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
0081 53FB        557          ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
0083 53FE        558          ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
0085 AA          559          Mov     ChStR1,A        ;store the status byte
0086 FA          560 LdChar:  Mov     A,ChStR1        ;get the status byte
0087 52E1        561          JB2     CBPad1          ;CB not full but CR/LF previously
                 562                                  ;  received so pad CB
0089 9AEF        563          ANL     P2,#Ack         ;output DBB Ack low
008B 22          564          In      A,DBB           ;read the Char
008C 537F        565          ANL     A,#AscStp       ;strip off MSB
008E A8          566          Mov     TmpR10,A        ;temp save char
008F 8A10        567          ORL     P2,#ReSAck      ;output DBB ACK High
                 568 ; -------------------------------------------------------------------------
                 569 ;       test for ASCII printable character
0091 03E0        570          ADD     A,#ASCCpl       ;test for Carriage Return
0093 F697        571          JC      AsciiC          ;jmp to service
0095 049C        572          Jmp     ChrChk
0097 97          573 AsciiC:  Clr     C               ;clear carry flag
0098 F8          574          Mov     A,TmpR10        ;get the char back
0099 A1          575          Mov     @CAdrR1,A       ;load data memory w/Char
009A 04E3        576          Jmp     IBFSrE
                 577 ; -------------------------------------------------------------------------
                 578 ;       test for CR/LF:  if CR/LF Strip off LF and exit setting
                 579 ;         Char Buffer Init Stat bit
009C F8          580 ChrChk:  Mov     A,TmpR10        ;get the char back
009D 03F3        581          ADD     A,#CRCpl        ;test for Carriage Return
009F C6C3        582          JZ      CRChr           ;  if CR go service it
00A1 F8          583          Mov     A,TmpR10        ;get the char back
00A2 031B        584          ADD     A,#EOFCpl       ;test for End Of File
00A4 96AA        585          JNZ     ChrCk1          ;if not EOF jmp to CB Pad
00A6 F8          586          Mov     A,TmpR10        ;if EOF, place it in CB
00A7 A1          587          Mov     @CAdrR1,A       ;load data memory w/CR Char
00A8 04B9        588          Jmp     ExtSet          ;Exit
00AA F8          589 ChrCk1:  Mov     A,TmpR10        ;get the status byte
00AB 03F4        590          ADD     A,#FFCpl        ;test for FormFeed
00AD 96E1        591          JNZ     CBPad1          ;if not FF Pad the CB
00AF C5          592          SEL     RB0
00B0 FA          593          Mov     A,GStR20        ;get the status byte
00B1 4304        594          ORL     A,#FrmFd        ;set the formfeed flag
00B3 AA          595          Mov     GStR20,A        ;store the status byte
00B4 D5          596          SEL     RB1
00B5 FA          597          Mov     A,ChStR1        ;get the status byte
00B6 4304        598          ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                 599                                  ;  with Spaces until fill
00B8 AA          600          Mov     ChStR1,A        ;store the status byte
00B9 FA          601 ExtSet:  Mov     A,ChStR1        ;get the status byte
```

```
00BA 4302          602          ORL     A,#CBFLn        ;set Char Buffer Full Line stat bit
00BC 53FB          603          ANL     A,#ClrCr        ;clear the CR/(LF) stat bit
00BE 53FE          604          ANL     A,#ClICBR       ;reset CB Init bit: init CB reg on entry
00C0 AA            605          Mov     ChStR1,A        ;store the status byte
00C1 04EC          606          Jmp     CBF1Ex          ;Exit
                   607 ; ---------------------------------------------------------------------
                   608 ;        Store CR char read in LF char (assume its always there) and ignor it
00C3 F8            609 CRChr:   Mov     A,TmpR1O        ;get the char back
00C4 A1            610          Mov     @CAdrR1,A       ;load data memory w/CR Char
00C5 C5            611          SEL     RBO
00C6 1E            612          INC     LnCtRO          ;inc the line count
00C7 FE            613          Mov     A,LnCtRO        ;get the line count
00C8 03C4          614          Add     A,#PgLCpl       ;test for page feed ln cnt
                   615                                  ;  if LnCt => PgLnCt set formfeed flag
00CA E6D0          616          JNC     NoFmFd          ;if not at end of page skip
00CC FA            617          Mov     A,GStR2O        ;get the status byte
00CD 4304          618          ORL     A,#FrmFd        ;set the form feed status flag
00CF AA            619          Mov     GStR2O,A        ;save the status byte
00D0 D5            620 NoFmFd:  SEL     RB1
00D1 05            621          En      I               ;enable the IBF service
00D2 9ADF          622          ANL     P2,#NotBsy      ;output a not busy to Host
00D4 D6D4          623 LFTest:  JNIBF   LFTest          ;loop to next char
00D6 9AEF          624          ANL     P2,#Ack         ;output DBB Ack low
00D8 22            625          In      A,DBB           ;get next Char - assume it's a LF
                   626                                  ;  and ignor it (LF is forced upon
                   627                                  ;  detection of CR at print time)
00D9 FA            628 SetPad:  Mov     A,ChStR1        ;get the status byte
00DA 4304          629          ORL     A,#CRLF         ;set CRLF stat bit:  pad balance of CB
                   630                                  ;  with Spaces until fill
00DC AA            631          Mov     ChStR1,A        ;store the status byte
00DD 8A10          632          ORL     P2,#ReSAck      ;output DBB ACK High
00DF 04E3          633          Jmp     IBfSrE          ;jmp to addr step & exit
                   634 ; ---------------------------------------------------------------------
                   635 ;        fill Char Buffer with space
00E1 B120          636 CBPad1:  Mov     @CAdrR1,#Space  ;load data memory w/Char
                   637 ; ---------------------------------------------------------------------
                   638 ;        step the char address test for CB full &/or pad
00E3 C9            639 IBFSrE:  DEC     CAdrR1          ;Decrement dat memory location
00E4 FA            640          Mov     A,ChStR1        ;get the status byte
00E5 32EC          641          JB1     CBFull          ;test for CB Full
00E7 52EC          642          JB2     CBF1Ex          ;test for CB pad - exit w/Busy set
                   643 ; ---------------------------------------------------------------------
                   644 ;        Set Busy Line Low - Not Busy
00E9 05            645          EN      I
00EA 9ADF          646          ANL     P2,#NotBsy      ;output a not busy to Host
                   647 ; ---------------------------------------------------------------------
                   648 ;        exit w/ Busy Still set high
                   649 CBFull:
00EC 83            650 CBF1Ex:  Ret
                   651
                   652 ; PG
                   653 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   654 ;        L-to-R/R-to-L Carriage Stepper Motor Drive
                   655 ;           and Line Printing
                   656 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   657
0100               658          ORG     100H
                   659
0100 3622          660 SMDriv:  JTO     RAccel          ;if Print Head at left drive right
                   661                                  ;  else drive left
                   662 ; F=============================================================
                   663 FAccel:                  ;L-to-R Accelerate Stepper Motor
                   664 ;        Set the Forward acceleration/drive Entry status bits
0102 FA            665          Mov     A,GStR2O        ;get the status byte
0103 53BF          666          ANL     A,#ClrSnk       ;set not at speed flag = 0
0105 53DF          667          ANL     A,#NAtSpd       ;set Not At Speed flag = 0
0107 4380          668          ORL     A,#LRPrnt       ;set L-to-R prnt stat bit = 1
0109 4301          669          ORL     A,#Ready        ;set stpr mtr ready - Drive On
010B 53EF          670          ANL     A,#ADIntN       ;set A/D Init Not Done
010D AA            671          Mov     GStR2O,A        ;store the status byte
010E D5            672 CBRDir:  SEL     RB1
010F FA            673          Mov     A,ChStR1        ;get the Char Stat Reg Data Mem Addr
0110 4380          674          ORL     A,#LRPrnt       ;Set L-to-R print bit
0112 AA            675          Mov     ChStR1,A        ;save the Char Stat byte
0113 C5            676          SEL     RBO
                   677
                   678 ;        Restore the phase register index addresses
0114 B821          679          Mov     TmpR0O,#CPSAdr  ;get Phz Storage Addr psuedo reg
0116 F0            680          Mov     A,@TmpR0O       ;  get stored CR last phase index addr
0117 AB            681          Mov     PhzR3O,A        ;place last LF phase index addr in Phz Reg
                   682
```

```
                        ;       Set up for next phase bit output before entering timing loops
0118 1B                 INC     PhzR30                  ;STEP PHASE DB ADDRESS
0119 FB                 MOV     A,PhzR30                ;CHECK THE PHASE COUNT REG
011A 521E               JB2     IAFZrP                  ;CHK FOR COUNT BIT ROLLOVER
011C 2440               JMP     SMDflt                  ;skip adr index reset
011E BB00       IAFZrP: MOV     PhzR30,#FStCRP          ;ZERO CR SM PHASE REGISTER
0120 2440               Jmp     SMDflt
                        ;R=============================================================
                        RAccel:                         ;R-to-L Accelerate Stepper Motor
                        ;---------------------------------------------------------------
                        ;       Set the Reverse acceleration/drive Entry status bits
0122 FA                 Mov     A,QStR20                ;get the status byte
0123 53BF               ANL     A,#ClrSnk               ;clear Print Ready bit
0125 53DF               ANL     A,#NAtSpd               ;set Not At Speed flag = 0
0127 537F               ANL     A,#RLPrnt               ;set R-to-L prnt status bit
0129 4301               ORL     A,#Ready                ;set stpr mtr ready - Drive On
012B 53EF               ANL     A,#ADIntN               ;set A/D Init Not Done
012D AA                 Mov     QStR20,A                ;store the status byte
012E D5                 SEL     RB1
012F FA                 Mov     A,ChStR1                ;get the Char Stat Reg Data Mem Addr
0130 537F               ANL     A,#RLPrnt               ;Set R-to-L print bit
0132 AA                 Mov     ChStR1,A                ;save the Char Stat byte
0133 C5                 SEL     RB0
                        ;---------------------------------------------------------------
                        ;       Restore the  phase register index address
0134 B821               Mov     TmpR00,#CPSAdr          ;get Phz Storage Addr psuedo reg
0136 F0                 Mov     A,@TmpR00               ; - get stored CR last phase index addr
0137 AB                 Mov     PhzR30,A                ;place last LF phase index addr in Phz Reg
                        ;       Set up for next phase bit output before entering timing loops
0138 CB                 Dec     PhzR30                  ;STEP PHASE DB ADDRESS
0139 FB                 MOV     A,PhzR30                ;CHECK THE PHASE COUNT REG
013A 523E               JB2     IARZrP                  ;CHK FOR COUNT BIT ROLLOVER
013C 2440               JMP     SMDflt
013E BB03       IARZrP: MOV     PhzR30,#RStCRP          ;ZERO CR SM PHASE REGISTER

                        SMDflt:
                        ;---------------------------------------------------------------
                        ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                        ;         store the unused stpr mtr current phase bits
0140 B822               Mov     TmpR00,#LPSAdr          ;get the CR phz storeage addr
0142 F0                 Mov     A,@TmpR00               ;get the byute stored there
0143 E3                 MovP3   A,@A                    ;get the phz data byte
0144 B820               Mov     TmpR00,#LastPh          ;load Last Phz psuedo reg to Temp Reg
0146 A0                 Mov     @TmpR00,A               ;store Last Phase bits - inderect

                        ;       SetUp Stpr Mtr Time Constant
0147 BDBA               MOV     TConR0,#CrTmr2          ;Load time constant Reg

                        Select:                         ;Select the Stpr Mtr
0149 2308               MOV     A,#SCR80                ;GET CR SM SELECT BITS
014B 3D                 MOVD    P5,A                    ;SELECT SM [SCR80]
                        ;---------------------------------------------------------------
                        ;       SetUp Stpr Mtr Phase Shift index address register
                        ;       Output next phase and init timer to Std Time constant
014C FD                 MOV     A,TConR0                ;get time constant from reg
014D 62                 MOV     T,A                     ;load the timer
014E FB                 MOV     A,PhzR30                ;get the phz reg indirect addr index
014F E3                 MovP3   A,@A                    ;do indirect get of phz bits

                        ;---------------------------------------------------------------
                        ;       patch together the CR last and LF next phase bits
0150 B820               Mov     TmpR00,#LastPh          ;load Last Phz psuedo reg to Temp Reg
0152 40                 ORL     A,@TmpR00               ;patch together CR existing & new LF
0153 3C                 MOVD    P4,A                    ;OUTPUT BITS
0154 55                 STRT    T                       ;START TIMER

                        ;       At start of timing loop do all Stpr Mtr Accel/Decel or
                        ;         Character SetUp overhead
0155 740C               Call    ADPTst                  ;call Accel/Decel/Print Test

                        ;       Set up for next phase bit output before entering timing loops
                        PNRdy1: ;test for forward / reverse phase start indirect index to load
0157 FA                 Mov     A,QStR20                ;store stat byte
0158 F264               JB7     AclF2

                        ; reverse:
                        ;       Set up for next phase bit output before entering timing loops
015A CB                 Dec     PhzR30                  ;STEP PHASE DB ADDRESS
015B FB                 MOV     A,PhzR30                ;CHECK THE PHASE COUNT REG
015C 5260               JB2     ARZroP                  ;CHK FOR COUNT BIT ROLLOVER
015E 2462               JMP     ARNxtP
0160 BB03       ARZroP: MOV     PhzR30,#RStCRP          ;ZERO CR SM PHASE REGISTER
0162 246C       ARNxtP: Jmp     ANxtPh
```

```
                        767
                        768 ; forward:
                        769 ;       Set up for next phase bit output before entering timing loops
0164 1B                 770 AclF2:  INC     PhzR30          ;STEP PHASE DB ADDRESS
0165 FB                 771         MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
0166 526A               772         JB2     AFZroP          ;CHK FOR COUNT BIT ROLLOVER
0168 246C               773         JMP     ANxtPh          ;skip adr index reset
016A BB00               774 AFZroP: MOV     PhzR30,#FStCRP  ;ZERO CR SM PHASE REGISTER
                        775 ANxtPh:
                        776 ; ---------------------------------------------------------------
                        777 ;       stage one timer loop - T occurs before Std timeout
                        778 ;         wait for time out
016C 1682               779 TLOOP2: JTF     FAILSF          ;JMP ON TIME OUT-t DOES NOT OCCUR 1ST
016E 5672               780         JT1     tCHK1           ;IS T HIGH-JMP TO tCHK
0170 246C               781         JMP     TLOOP2          ;LOOP FOR JT1 OR JTF
0172 00                 782 tCHK1:  NOP                     ;delay, then double check T signal
0173 5677               783         JT1     tTruW1          ;JUMP T TEST TRUE-WAIT FOR JTF
0175 246C               784         JMP     TLOOP2
                        785 tTruW1:
                        786 ;       test for Print Ready bit - was Print Head Fire Setup Done?
                        787 ;       insert acceleration time/store time count done/notdone flag bit
0177 FA                 788         Mov     A,@StR20        ;get the status byte - prep for prnt
0178 D27C               789         JB6     RdyPr2          ;if Ready Print bit set call PHFire
017A 247E               790         Jmp     SkpPHF          ;  else skip Print Head Fire
017C 74CA               791 RdyPr2: Call    PHFire          ;print head solenoid fire routine
                        792 PNRdy2:
                        793 SkpPHF:
017E 1698               794 tTruW2: JTF     NXTPHZ          ;JUMP TO SM ERROR
0180 247E               795         JMP     tTruW2          ;  LOOP TO TLOOP3
                        796 ; ---------------------------------------------------------------
                        797 ;       Step into failsafe/startup timer setup - T does not
                        798 ;         occurs before Std Time timeout, load failsafe SM protection
                        799 ;         time and wait for failsafe timeout or T.  If T occurs
                        800 ;         output phase immediately after T verify.
0182 2300               801 FAILSF: MOV     A,#FailTm       ;LOAD TIMER W/~15.0mS
0184 62                 802         MOV     T,A             ;  SM PROTECTION TIMEOUT
0185 55                 803         STRT    T               ;START TIMER
                        804 , ---------------------------------------------------------------
                        805 ;       set the Status bit for Store time test
0186 FA                 806         Mov     A,@StR20        ;get the status byte
0187 4308               807         ORL     A,#FSCTm        ;set Failsafe/constant time flag
0189 AA                 808         Mov     @StR20,A        ;store the status byte
018A 5690               809 TLOOP3: JT1     tCHK2           ;IS T HIGH
018C 16AC               810         JTF     DSLECT          ;IF TIME OUT GO SM ERROR
018E 248A               811         JMP     TLOOP3          ;LOOP UNTIL T HIGH OR T-OUT
0190 00                 812 tCHK2:  NOP                     ;WAIT
0191 5695               813         JT1     StrTm1          ;jump out and store elapsed time
0193 248A               814         JMP     TLOOP3          ;  JMP TO FAILSF LOOP
0195 65                 815 StrTm1: Stop    TCnt            ;stop the failSafe Timer
0196 42                 816         Mov     A,T             ;read the timer
0197 A1                 817         MOV     @TStrR0,A       ;Store the time read in indexed addr
                        818                                 ; - next entry to A/D Memorize Time
                        819                                 ; routine will add time constant to it
                        820
                        821 ;    Test is CR Stpr Mtr Drive is finished prior to next phase output
                        822 ; ---------------------------------------------------------------
                        823 NXTPHZ:
                        824 ;       test for forward / reverse phase start indirect index to load
0198 FA                 825         Mov     A,@StR20        ;store stat byte
0199 F2A7               826         JB7     FDrive
                        827 ; Reverse  --  test for Reverse Stpr Mtr Drive procedure exit
                        828 ;       ALWAYS drive the CR to the left most HOME position
019B 26AC               829         JNT0    EOLn            ;test if home position jmp stop
019D FA                 830         Mov     A,@StR20        ;get the status byte
019E 12AC               831         JB0     StrtT           ;test Ready stat bit:
                        832                                 ;  if bit 0 = 1 then Print More
01A0 4302               833         ORL     A,#DoNotP       ;set the do not print flag
01A2 53BF               834         ANL     A,#ClrSnk       ;clear Print Ready bit
01A4 AA                 835         Mov     @StR20,A        ;save the status byte
01A5 24AC               836         Jmp     StrtT           ;continue CR SM drive
                        837                                 ;  - only exit is HR
                        838 ; Forward  --  test for Forward Stpr Mtr Drive procedure exit
                        839 FDrive:
01A7 FA                 840         Mov     A,@StR20        ;get the status byte
01A8 12AC               841         JB0     StrtT           ;test Ready stat bit:
                        842                                 ;  if bit 0 = 1 then Print More
01AA 24AC               843         Jmp     EOLn            ;  else jmp to End Of Line exit
                        844                                 ;jump to start timer again
                        845 DSLECT:
01AC 5437               846 EOLn:   Call    DeclSM          ;call Sptr Mtr Deceleration
                        847 ; ---------------------------------------------------------------
                        848 ;       test for forward / reverse phase start indirect index to load
01AE FA                 849         Mov     A,@StR20        ;store stat byte
01AF F2B3               850         JB7     FDrvFS          ;jmp to f drive flag set
```

```
01B1 53FD          851          ANL     A,#OkPrnt       ;reset print flag - Ok Print
                   852                                  ; only if printing R-to-L
                   853
                   854 ;        update the status byte
01B3 53BF          855 FDrvFS:  ANL     A,#ClrSnk       ;clear Print Ready bit
                   856                                  ;set the Status bit for Store time test
01B5 53DF          857          ANL     A,#NAtSpd       ;Clear At Print Speed Bit
01B7 AA            858          Mov     @StR2O,A        ;save the status byte
01B8 83            859          RET
                   860
                   861 ; PG
                   862 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   863 ;        Stepper Motor Accel. Time Storeage
                   864 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                   865
0200               866          ORG     200H
                   867                                  ;  Entry has Gen Stat Byte in A
0200 920C          868 ADMmTS:  JB4     DADInt          ;is A/D init done - then jmp
                   869
                   870 ;        1st Entry initializes the A/D Time store working registers
0202 B92F          871          Mov     TStrR0,#SMBfSt  ;Load the Stpr Mtr Buffer Start Addr
0204 BC0B          872          Mov     CntR40,#ASBfSz  ;Load the Buffer Size
0206 FA            873          Mov     A,@StR2O        ;get the status byte
0207 4310          874          ORL     A,#ADIntD       ;set not 1st Accel Entry Flag
0209 AA            875          Mov     @StR2O,A        ;store the status byte
020A 4436          876          Jmp     ADExit          ;exit - 1st entry has not generated
                   877                                  ;  a closed time window
                   878
                   879 ;        Step the A/D Store count
020C EC26          880 DADInt:  DJNZ    CntR40,StorCt   ;dec Times to store count
                   881                                  ;if not 0 store the count
                   882                                  ;else at end-set done flag
020E FA            883          Mov     A,@StR2O        ;get the status byte
020F 4320          884          ORL     A,#AtSpdF       ;set at speed/no more to store flag
0211 AA            885          Mov     @StR2O,A        ;store the status byte
                   886
                   887 ;        Initialize Char Print Registers:  if printing enabled
0212 3226          888          JB1     StorCt          ;if Do Not Print stat bit set
                   889                                  ;  Skip the Char register init
                   890
                   891 ;        Initialize all Char Reg's
                   892 ;        Test for L-to-R (forward) or R-to-L (reverse) printing
0214 D5            893          SEL     RB1
0215 FA            894          Mov     A,ChStR1        ;get the status byte
0216 4340          895          ORL     A,#CHIntD       ;set Char Init Done flag - bypass
0218 AA            896          Mov     ChStR1,A        ;save the status byte
0219 F21F          897          JB7     LdCBR1          ;test Chr Stat Byte Returned
                   898                                  ;  if bit 7 = 1 then Print L-to-R
021B B92F          899 LdCBR:   Mov     CAdrR1,#RCBfIS  ;load char reg w/char bufr strt R-to-L
021D 4421          900          Jmp     LdCBR2
                   901
021F B980          902 LdCBR1:  Mov     CAdrR1,#FCBfIS  ;load char reg w/char bufr strt L-to-R
                   903
0221 BD51          904 LdCBR2:  Mov     CCntR1,#ChBfIS  ;load char cnt reg w/char bufr size
0223 BB01          905          Mov     CDtCR1,#01      ;set the chr dot column cnt
0225 C5            906          SEL     RB0
                   907
                   908 ;        Test for t > Tc or t < Tc
0226 722C          909 StorCt:  JB3     FailST          ;test for failsafe time switch
                   910
                   911 ;        t < Tc = store Time Constant in use
0228 FD            912          Mov     A,TConR0        ;Get time constant currently in use
0229 A1            913          Mov     @TStrR0,A       ;Memorize/Store the time - indirect addr
022A 4435          914          Jmp     ADPRet
                   915
                   916 ;        t > Tc = store Time Constant + FailSafe Time Elapsed
                   917 ;        [see Accel/Cnst Speed/Decel WaveForm]
                   918 ;        equation is:     Trd - FailSafe Time = Tx
                   919 ;                     => Trd + Cpl(FailSafe Time) = Tx
                   920 ;                        Tx + Tcnst = T
                   921 ;                        Store/Memorize T
                   922
022C F1            923 FailST:  Mov     A,@TStrR0       ;get the stored time
022D 03C8          924          Add     A,#FTCpl        ;2's cpl add
022F 6D            925          Add     A,TConR0        ;Add: Time stored + Time constant
                   926                                  ;  currently in use
0230 A1            927          Mov     @TStrR0,A       ;Memorize/Store the time
                   928 ;        Reset the Status bit for Store time test
                   929
0231 FA            930          Mov     A,@StR2O        ;get the status byte
0232 53F7          931          ANL     A,#ClrFSC       ;reset Failsafe/constant time flag
                   932                                  ;  assumes entry via constant time
```

```
0234 AA          933          Mov     GStR20,A          ;store the status byte
0235 C9          934 ADPRet:  Dec     TStrRO            ;step the A/D time data store addr
0236 83          935 ADExit:  Ret
                 936
                 937 ; PG
                 938 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 939 ;        Carriage Stepper Motor Deceleration
                 940 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 941
                 942 DeclSM:
                 943 ;        SetUp the Deceleration registers
0237 B925        944          Mov     TStrRO,#SMBEnd    ;Load the Stpr Mtr Buffer End Addr
0239 BC0A        945          Mov     CntR40,#DSBfSz    ;Load the Buffer Size
023B FB          946          MOV     A,PhzR30          ;get phase index address
023C E3          947          MovP3   A,@A              ;get phase from indexed address
                 948 ;        patch together the CR last and LF next phase bits
023D B820        949          Mov     TmpROO,#LastPh    ;load Last Phz psuedo reg to Temp Reg
023F 40          950          ORL     A,@TmpROO          ;patch together CR existing & new LF
0240 3C          951          MOVD    P4,A              ;OUTPUT BITS
0241 F1          952 StrtTD:  MOV     A,@TStrRO         ;get time from indexed data memory
0242 62          953          MOV     T,A               ;load timer
0243 55          954          STRT    T                 ;START TIMER
0244 19          955          Inc     TStrRO            ;step the Memorized time addr index reg
                 956 ;        test for forward / reverse phase start indirect index to load
0245 FA          957          Mov     A,GStR20          ;store stat byte
0246 F252        958          JB7     DclF2
                 959
                 960 ; reverse:
                 961 ;        Set up for next phase bit output before entering timing loops
0248 CB          962          Dec     PhzR30            ;decrement the phase addr
0249 FB          963          MOV     A,PhzR30          ;Get the phz data addr
024A 524E        964          JB2     DRZroP            ;CHK FOR COUNT BIT ROLLOVER
024C 445A        965          JMP     DNxtPh
024E BB03        966 DRZroP:  MOV     PhzR30,#RStCRP    ;ZERO CR SM PHASE REGISTER
0250 445A        967          Jmp     DclR2
                 968
                 969 ; forward:
                 970 ;        Set up for next phase bit output before entering timing loops
0252 1B          971 DclF2:   Inc     PhzR30            ;increment the phase addr
0253 FB          972          MOV     A,PhzR30          ;Get the phz data addr
0254 5258        973          JB2     DZroPh            ;CHK FOR COUNT BIT ROLLOVER
0256 445A        974          JMP     DNxtPh            ;skip adr index reset
0258 BB00        975 DZroPh:  MOV     PhzR30,#FStCRP    ;ZERO CR SM PHASE REGISTER
                 976 DNxtPh:                            ;set up for next phase shift
025A FB          977 DclR2:   MOV     A,PhzR30          ;get phase index address
025B E3          978          MovP3   A,@A              ;get phase from indexed address
                 979 ;        patch together the CR last and LF next phase bits
025C B820        980          Mov     TmpROO,#LastPh    ;load Last Phz psuedo reg to Temp Reg
025E 40          981          ORL     A,@TmpROO          ;patch together CR existing & new LF
025F 1663        982 TLoopD:  JTF     NxtPD2            ;JMP ON TIME OUT TO NEXT PH
0261 445F        983          JMP     TLoopD            ;LOOP UNTIL TIME OUT
0263 3C          984 NxtPD2:  MOVD    P4,A              ;OUTPUT BITS
0264 EC41        985          DJNZ    CntR40,StrtTD     ;Exit Test
                 986
                 987 ;        Set Storeage of next phase data in psuedo addr. This insures
                 988 ;        next phase is sequence correct for stpr mtr drive direction
0266 B821        989 SetRN:   Mov     TmpROO,#CPSAdr     ;get Phz Storeage Addr psuedo reg
0268 FB          990          MOV     A,PhzR30          ;get Phz data
0269 A0          991          Mov     @TmpROO,A         ;store CR Next phase index addr
026A B478        992 DMExit:  Call    DlyLng
026C B490        993          Call    DeSlSM
026E 83          994          RET
                 995
                 996 ; PG
                 997 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 998 ;        Stepper Motor Phase Shift Definitions
                 999 ;           All program procedures call this data.
                1000 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                1001
0300            1002          ORG     300H
                1003
                1004 ;        DEFINE PHASE ADDRESSES:
                1005 ;        THE PHASE DATA IS ENCODED TO THE ADDRESS CALLED DURING THE
                1006 ;        STPR MTR ENERGIZE SEQUENCE CORRESPONDING TO THE NEXT PHASE
                1007 ;        OF THE SEQUENCE REQUIRED.
                1008
                1009 ;        CARRAGE MOTOR ENCODING  FORWARD  -  LEFT-to-RIGHT
                1010 ;                                REVERSE  -  RIGHT-to-LEFT
                1011
```

```
                        1012 ;       Reverse direction ENCODING is the same bytes accessed in
                        1013 ;         reverse direction
                        1014
0300 01                 1015         DB      CRMFP1
0301 03                 1016         DB      CRMFP2
0302 02                 1017         DB      CRMFP3
0303 00                 1018         DB      CRMFP4
                        1019
                        1020 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1021
                        1022 ;       LF MOTOR PHASE ENCODE & DECODE: FORWARD (CLOCKWISE)
                        1023 ;       Forward direction ENCODING:
                        1024 ;       ---------------------------------------------
                        1025
0308                    1026         ORG     308H
                        1027
0308 04                 1028         DB      LFMFP1
0309 0C                 1029         DB      LFMFP2
030A 08                 1030         DB      LFMFP3
030B 00                 1031         DB      LFMFP4
                        1032
                        1033
                        1034 ; PG
                        1035 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1036 ;       Accel/Decel / Character Handling Test
                        1037 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1038 ;       TEST  >  Is CR Stpr Mtr At Speed ??
                        1039 ;         Yes - SetUp do Character Processing
                        1040 ;         No  - Calculate / Store the Acceleration Phase Shift Time (11)
                        1041 ; ---------------------------------------------------------------------------------
                        1042
030C FA                 1043 ADPTst: Mov     A,@StR2O        ;get the status byte
030D B211               1044         JB5     PHFSet          ;test if Stpr Mtr At Speed
                        1045                                 ;   jmp to Prnt Head Fire Setup
030F 4400               1046         Jmp     ADMmTS          ;else Call Accel/Decel Memory Time Store
                        1047
                        1048 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1049 ;       Process Characters for Printing
                        1050 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1051
                        1052 ;       Character dot matrix - normal char
                        1053 ;       d = Dot Column
                        1054 ;       b = Blank Column
                        1055
                        1056 ;       b d d d d d
                        1057 ;       (Char Matrix)
                        1058 ;       0 0 0 0  b
                        1059 ;       0 0 0 1  d
                        1060 ;       0 0 1 0  d
                        1061 ;       0 0 1 1  d
                        1062 ;       0 1 0 0  d
                        1063 ;       0 1 0 1  d
                        1064 ; ---------------------------------------------------------------------------------
                        1065
0311 2668               1066 PHFSet: JNTO    Retrn           ;if R=0 not ready to print-exit
0313 326A               1067         JB1     NPRet           ;if Do Not Print stat bit set - EXIT
0315 D21B               1068         JB6     SinkSt          ;if bit previously set-skip setting it
0317 FA                 1069         Mov     A,@StR2O        ;get the status byte
0318 4340               1070         ORL     A,#SnkSet       ;set Prnt Ready Sink bit
031A AA                 1071         Mov     @StR2O,A        ;save the status byte
031B D5                 1072 SinkSt: SEL     RB1
031C FA                 1073         Mov     A,ChStR1        ;get char status register addr
031D D23A               1074         JB6     PageCk          ;test Char Init Done, 1 = Print Dot
                        1075                                 ;   0 = Get Char
                        1076
                        1077 ; PG
                        1078 ; ---------------------------------------------------------------------------------
                        1079 ;       Call for Individual character processing:  mid line test if CR/(LF)
                        1080 ; ---------------------------------------------------------------------------------
                        1081 GetChr:
                        1082 ;       test for CR/(LF) if it is the test position in the line
031F F1                 1083 CRChCk. Mov     A,@CAdrR1       ;get character
0320 03F3               1084         ADD     A,#CRCpl        ;test for Carriage Return
0322 C626               1085         JZ      CrLnCk          ;   if CR go service it
0324 6437               1086         Jmp     AsciC1          ;if not CR Insert Space Char
0326 FA                 1087 CRLnCk: Mov     A,ChStR1        ;get char status register addr
0327 F22B               1088         JB7     HlfLn           ;test Chr Stat Byte Returned
                        1089                                 ;   if bit 7 = 1 then Print L-to-R
0329 6432               1090         Jmp     SpFill          ;if R-to-L print skip exit upon CR detect
                        1091 ; ---------------------------------------------------------------------------------
```

```
                1092 ;       if L-to-R printing exit the line if less than 1/2 line printed
032B FD         1093 H1fLn:  Mov     A,CCntR1        ;load char cnt reg w/char bufr size
032C 03D9       1094         ADD     A,#H1fCpl       ;add the 2's cpl of 1/2 chr buf size
032E F632       1095         JC      LnPad           ;if CB>1/2 full set CR/LF stat bit for pad
                1096                                 ;If CB<1/2 set buffer full stat bit
0330 648A       1097         Jmp     MdLnEx          ;mid-line exit
                1098 SpFill.
0332 97         1099 LnPad:  Clr     C               ;clear carry flag
0333 2320       1100         Mov     A,#Space        ;insert a space char
0335 6438       1101         Jmp     ChIsrt          ;char inserted jmp over get char
                1102 ; ---------------------------------------------------------------------
0337 F1         1103 AsciC1: Mov     A,@CAdrR1       ;get character
0338 7498       1104 ChIsrt: Call    QChar1          ;call the char lookup/trns table
                1105 ; ---------------------------------------------------------------
                1106 ;       fetch the char dot column data
                1107 PageCk.                         ;page test for balance of char
033A FA         1108         Mov     A,ChStR1        ;get the status byte
033B B241       1109         JB5     FxJmp1          ;fix jmp over page boundries
033D F4EB       1110         Call    ChrPg2          ;Ascii char 50 - 7F Hex
033F 6443       1111         Jmp     MtxTst          ;jump to Matrix Test
0341 D4F0       1112 FxJmp1: Call    ChrPg1          ;Ascii char 20 - 4F Hex
                1113                                 ;  fall thru to print matrix
                1114                                 ;  and CB count tests
                1115
                1116 ; PG
                1117 ; ---------------------------------------------------------------------
                1118 ;       test the Char dot column print matrix count and Char buffer count
                1119 ; ---------------------------------------------------------------------
0343 EB61       1120 MtxTst: DJNZ    CDtCR1,PrntDt   ;test for dot col or blank
                1121                                 ;status byte in A upon entry here
0345 FA         1122         Mov     A,ChStR1        ;get the status byte
0346 53BF       1123         ANL     A,#CIntND       ;set Char Init NotDone stat Flag
0348 AA         1124         Mov     ChStR1,A        ;store the status byte
0349 ED58       1125         DJNZ    CCntR1,NotLCh   ;dec char cnt-jmp if Not Last Char
034B 53FD       1126         ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
034D 53FE       1127         ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
034F AA         1128         Mov     ChStR1,A        ;save the status byte
                1129
0350 C5         1130         SEL     RB0
0351 FA         1131         Mov     A,GStR20        ;get Gen Status register addr
0352 53FE       1132         ANL     A,#NotRdy       ;clear the ready bit
0354 AA         1133         Mov     GStR20,A        ;store the General Status Byte
0355 D5         1134         SEL     RB1
0356 6468       1135         Jmp     Retrn           ;EXit
                1136
                1137 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                1138 ;         (see QChar1 ASCII char code translation procedure)
                1139 ; ---------------------------------------------------------------------
                1140 NotLCh:                         ;A contains LR/RL bit properly set
0358 FA         1141         Mov     A,ChStR1        ;get char status register addr
0359 F25E       1142         JB7     StpCh2          ;test Chr Stat Byte Returned
                1143                                 ;  if bit 7 = 1 then Print L-to-R
035B 19         1144 StpCh1: Inc     CAdrR1          ;Increment char data memory addr.
035C 6468       1145         Jmp     Retrn
035E C9         1146 StpCh2: Dec     CAdrR1          ;Decrement char data memory addr.
035F 6468       1147         Jmp     Retrn           ;  fall thru to Get Char
                1148
                1149
                1150 ; ---------------------------------------------------------------------
                1151 ;       Re-Entry Exit point for same char:
                1152 ;         (before returning step the matrix)
                1153 ; ---------------------------------------------------------------------
                1154 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                1155 ;         (see QChar1 ASCII char code translation procedure)
                1156 ; ---------------------------------------------------------------------
                1157
                1158 PrntDt:
0361 FA         1159 PrnDir: Mov     A,ChStR1        ;get char status byte
0362 F267       1160         JB7     StpCD2          ;test Chr Stat Byte Returned
                1161                                 ;  if bit 7 = 1 then Print L-to-R
0364 CC         1162 StpCD1: Dec     CDotR1          ;reverse step char dot col index
                1163                                 ;  addr if R-to-L print
0365 6468       1164         Jmp     Retrn           ;skip over L-to-R print addr inc
0367 1C         1165 StpCD2: INC     CDotR1          ;forward step char dot col index
                1166                                 ;  addr if L-to-R print
                1167                                 ;EXIT
                1168
                1169 ; PG
```

```
                        1170 ; -------------------------------------------------------------------------
                        1171 ;       Character Print SetUp Exit Procedures
                        1172 ; -------------------------------------------------------------------------
                        1173 ;       Clean Standard Exit
                        1174 ; -------------------------------------------------------------------------
0368 C5                 1175 Retrn:  SEL     RB0
0369 83                 1176         Ret                     ;EXIT - return w/ Reg Bank 0 Reset
                        1177
                        1178 ;       Do Not Print exit:  set Stpr Mtr drive routine count loop
036A D5                 1179 NPRet:  SEL     RB1
036B FA                 1180         Mov     A,ChStR1        ;get the status byte
036C F27C               1181         JB7     SkpNPI          ;test print direction
                        1182 ; Reverse
036E C5                 1183         SEL     RB0
036F FA                 1184         Mov     A,QStR20        ;get the status byte
0370 53BF               1185         ANL     A,#ClrSnk       ;reset the print ready bit- skips PHFire call
0372 83                 1186         Ret
                        1187 ; Forward
0373 D27C               1188         JB6     SkpNPI          ;test for first PHFSet entry reg init
                        1189 ;       Initialize register variables upon first entry
                        1190 ;         end of count clears char to print bit in status byte
0375 4340               1191         ORL     A,#ChIntD       ;set Char Reg Init Done stat bit
0377 AA                 1192         Mov     ChStR1,A        ;save the status byte
0378 B807               1193         Mov     TmpR10,#07H     ;load CR stpr mtr count during NoPrnt
037A 6488               1194         Jmp     NPExit
037C E888               1195 SkpNPI: DJNZ    TmpR10,NPExit
037E FA                 1196         Mov     A,ChStR1        ;get the status byte
037F 53BF               1197         ANL     A,#CIntND       ;reset - char init not done
0381 AA                 1198         Mov     ChStR1,A        ;save the status byte
0382 C5                 1199         SEL     RB0
0383 FA                 1200         Mov     A,QStR20        ;get Gen Status register addr
0384 53FE               1201         ANL     A,#NotRdy       ;clear the ready bit
0386 AA                 1202         Mov     QStR20,A        ;store the General Status Byte
0387 83                 1203 NSetEx: Ret
0388 C5                 1204 NPExit: SEL     RB0
0389 83                 1205         Ret
                        1206
                        1207 ;       Mid-Line Exit
                        1208 ; -------------------------------------------------------------------------
                        1209 ;       EXIT - if CR and not > 1/2 line done during L-to-R print
038A FA                 1210 MdLnEx: Mov     A,ChStR1        ;get the status byte
038B 53FD               1211         ANL     A,#NCBFln       ;if 0 reset stat bit Not CB Full Line
038D 53FE               1212         ANL     A,#ClICBR       ;reset CB Reg Init Flag - do Init
038F AA                 1213         Mov     ChStR1,A        ;save the status byte
0390 C5                 1214         SEL     RB0
0391 FA                 1215         Mov     A,QStR20        ;get the RB0 status byte
0392 4302               1216         ORL     A,#DoNotP       ;set the Do Not Print Flag(for RAccel)
0394 53BF               1217         ANL     A,#ClrSnk       ;reset the print ready bit-exit FAccel
0396 AA                 1218         Mov     QStR20,A        ;save the status byte
0397 83                 1219         Ret
                        1220
                        1221 ; PG
                        1222 ; -------------------------------------------------------------------
                        1223 ;       Character Dot Generator Math
                        1224 ;         Look-up Table Page Vectoring
                        1225 ;         Print Head Firing
                        1226 ; -------------------------------------------------------------------
                        1227
0398 AE                 1228 QCHAR1: MOV     StrCR1,A                ;STORE THE CHAR
                        1229
                        1230 ;  screen for printable char [char +(cpl 20 Hex + 1 = E0 Hex)]
0399 03E0               1231         ADD     A,#0E0H
039B F69F               1232         JC      PrntCh
039D 64C9               1233         jmp     Cntlch          ;jmp to control char lookup table
039F 97                 1234 PrntCh: Clr     C               ;clear carry flag
03A0 FE                 1235         Mov     A,StrCR1        ;get the char again
                        1236
                        1237 ;  screen for char page [char +(cpl 50 Hex + 1 = B0 Hex)]
                        1238 ;    if carry char on page 2 else page 1
03A1 03B0               1239         ADD     A,#0B0H
03A3 F6AE               1240         JC      Page2
                        1241
                        1242 ;       Page 1 Character -- ASCII 20 Hex thru 4F Hex
                        1243 ;         Correct offset for lookup table page
                        1244 ;         {(char + E0 Hex)*5 = Page 1 index addr}
                        1245 ; -------------------------------------------------------------------
03A5 FA                 1246 Page1:  Mov     A,ChStR1        ;get the status byte
03A6 4320               1247         OrL     A,#ChOnP1       ;set the page rentry flag bit
03A8 AA                 1248         Mov     ChStR1,A        ;store the status byte
03A9 FE                 1249         Mov     A,StrCR1        ;get the char agian
03AA 03E0               1250         ADD     A,#0E0H         ;set page 1 relative 00 offset
03AC 64B8               1251         Jmp     Multi5          ;jump to address math function
                        1252
```

```
                        1253 ;       Page 2 Character -- ASCII 20 Hex thru 4F Hex
                        1254 ;         Correct offset for lookup table page two's complement
                        1255 ;         of ASCII chr code LookUp Table page base char of 50H plus
                        1256 ;         char * 5 {(char + 80 Hex)*5 = Page 2 index addr}
                        1257 ; ---------------------------------------------------------------
03AE 97                 1258 Page2:  Clr     C                  ;clear carry flag
03AF FA                 1259         Mov     A,ChStR1           ;get the status byte
03B0 53DF               1260         AnL     A,#ChOnP2          ;set the page rentry flag bit
03B2 AA                 1261         Mov     ChStR1,A           ;store the status byte
03B3 FE                 1262         Mov     A,StrCR1           ;get the char agian
03B4 03B0               1263         ADD     A,#0B0H            ;set page 2 relative 00 offset
03B6 64B8               1264         Jmp     Multi5             ;fall thru to address math function
                        1265
                        1266 ;       Compute character page offset dot pattern index address
03B8 AE                 1267 MULTI5: Mov     StrCR1,A           ;store the zero offset char
03B9 E7                 1268         RL      A                  ;MULTIPLY CHR BY 5 TO
03BA E7                 1269         RL      A                  ;  FIND THE ADDRESS
03BB 6E                 1270         ADD     A,StrCR1           ;ADD 1 TO COMPLETE 5X
03BC AC                 1271         MOV     CDotR1,A           ;SAVE THE ADDRESS
                        1272
                        1273 ;       Test for L-to-R (forward) or R-to-L (reverse) printing
                        1274 ;         (see GChar1 ASCII char code translation procedure)
                        1275 ; ---------------------------------------------------------------
03BD FA                 1276         Mov     A,ChStR1           ;get char status byte
03BE F2C4               1277         JB7     LRPrn              ;test Chr Stat Byte Returned
                        1278                                    ;  if bit 7 = 1 then Print L-toR
03C0 FC                 1279         MOV     A,CDotR1           ;get the char index addr
03C1 0304               1280         ADD     A,#RLPShf          ;add char offset - start at end
                        1281                                    ;of char, print it R-to-L
03C3 AC                 1282         MOV     CDotR1,A           ;SAVE THE ADDRESS
                        1283
                        1284 ;       Set the status byte for Character SetUp done
                        1285 ; ---------------------------------------------------------------
03C4 FA                 1286 LRPrn:  Mov     A,ChStR1           ;get the status byte
03C5 4340               1287         ORL     A,#ChIntD          ;set 1st char col test bit = 0
03C7 AA                 1288         Mov     ChStR1,A           ;store the status byte
03C8 83                 1289         Ret                        ;return w/status byte in A
                        1290 ;       test for non printable characters goes here
03C9 83                 1291 Cntlch: Ret
                        1292
                        1293 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1294 ;       Print Head Fire
                        1295 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1296
                        1297 ;       Entry point for print head solenoid firing
                        1298 ;         - test for status byte for dot/blank column position
03CA D5                 1299 PHFire: SEL     RB1
03CB FB                 1300         Mov     A,CDtCR1           ;set the chr dot column cnt
03CC 96D2               1301         JNZ     Fire               ;if char cnt not 0 - Fire Head Sol.
                        1302                                    ;if Chr Dot Cnt 0, reset the
03CE BB06               1303 SetCnt: Mov     CDtCR1,#NDtCCt     ;  char dot column count
03D0 64D8               1304         Jmp     Retrn1             ;skip PH Fire
03D2 2340               1305 Fire:   MOV     A,#PTrgLo          ;get the Prnt Head Trigger byte
03D4 3A                 1306         OUTL    P2,A               ;FIRE PRINT HEAD
03D5 23C0               1307         MOV     A,#PTrgHi          ;get the Prnt Head Trigger byte
03D7 3A                 1308         OUTL    P2,A               ;FIRE PRINT HEAD
03D8 C5                 1309 Retrn1: SEL     RB0
03D9 83                 1310         Ret                        ;EXIT - return w/ Reg Bank 0 Reset
                        1311
                        1312 ; PG
                        1313 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1314 ;       PaperFeed Stpr Mtr Drive
                        1315 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        1316
0400                    1317         ORG     400H
                        1318
                        1319 ;       Init psuedo register with LF inderect addr start - subsequent
                        1320 ;         exchanges of the psuedo register will yield correct value
0400 BC04               1321 InitLF: MOV     CntR40,#ILFCNT     ;INIT PHASE COUNT REG
0402 B822               1322         Mov     TmpR00,#LPSAdr     ;get Phz Inderect Addr psuedo reg
0404 2308               1323         MOV     A,#StLF            ;get LF starting addr
0406 A0                 1324         Mov     @TmpR00,           ;store LF phase index addr start
                        1325                                    ;  in psuedo register
0407 BE01               1326         Mov     LnCtR0,#LineCt     ;set line count reg for 1 ln
                        1327                                    ; enables exit following LF SM init
0409 841B               1328         Jmp     LfDrv1             ;jump over if  form feed amd variable
                        1329                                    ;  line spacing tests & setups
                        1330
                        1331 ;       LineFeed / FormFeed Drive
```

```
                    1332 ; ---------------------------------------------------------------------
                    1333
                    1334 ;       load step count constant for standard line spacing
                    1335 ; ---------------------------------------------------------------------
                    1336 ;       test for various line/inch spacing would go here
                    1337 ;         (and removal of constant setup below)
040B BC1B           1338          MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
                    1339
                    1340 ;       LineFeed/FormFeed Test
040D FA             1341 LfDriv:  Mov     A,GStR20        ;get the status byte
040E 5214           1342          JB2     FmFd            ;if linefeed jmp to cnt load
0410 BE01           1343 LnCtLd:  Mov     LnCtR0,#LineCt  ;set line count reg for 1 line
0412 841B           1344          Jmp     LfDrv1          ;jmp to Start of Drive
0414 FE             1345 FmFd:    Mov     A,LnCtR0        ;get the line count
0415 37             1346          Cpl     A               ;2's cpl Line Count
0416 0301           1347          Add     A,#01
0418 0342           1348          Add     A,#PgLnCt       ;Add 2's cpl for Paging
                    1349                                  ;  PgLnCt - LnCt = n Lines to move
                    1350                                  ;  PgLnCt+(cpl(LnCt) = n lines to move
041A AE             1351          Mov     LnCtR0,A        ;set the line count for FF
                    1352
                    1353 ;       for stablization of unused stpr mtr during CR stpr mtr drive,
                    1354 ;         store the unused stpr mtr current phase bits
041B B821           1355 LFDrv1:  Mov     TmpR00,#CPSAdr   ;get the CR phz storeage addr
041D F0             1356          Mov     A,@TmpR00       ;get the byute stored there
041E E3             1357          MovP3   A,@A            ;get the phz data byte
041F B820           1358          Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0421 A0             1359          Mov     @TmpR00,A       ;store Last Phase bits - inderect
                    1360 ;       exchange/store the phase register index addresses
0422 B822           1361          Mov     TmpR00,#LPSAdr  ;get Phz Inderect Addr psuedo reg
0424 F0             1362          Mov     A,@TmpR00       ;get LF last phase index addr
0425 AB             1363          Mov     PhzR30,A        ;place last LF phase index addr in Phz Reg
0426 BD98           1364          MOV     TConR0,#LFTMR1  ;Load time constant Reg
                    1365
                    1366 ;       Select the Stpr Mtr
0428 2306           1367          MOV     A,#SLF          ;GET CR SM SELECT BITS
042A 3D             1368          MOVD    P5,A            ;SELECT SM [SCR80]
                    1369
                    1370 ; ---------------------------------------------------------------------
                    1371 ;       LineFeed / FormFeed Drive Loop
                    1372 ; ---------------------------------------------------------------------
042B FB             1373          MOV     A,PhzR30        ;get the phz reg indirect addr index
042C E3             1374          MovP3   A,@A            ;do indirect get of phz bits
                    1375 ;       patch together the CR last and LF next phase bits
042D B820           1376          Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
042F 40             1377          ORL     A,@TmpR00       ;patch together CR existing & new LF
                    1378 ;       start timer and step motor
0430 3C             1379          MOVD    P4,A            ;OUTPUT BITS
                    1380
                    1381 StrtLF:
0431 FD             1382 STRLFT:  MOV     A,TConR0        ;get time constant from reg
0432 62             1383          MOV     T,A             ;load the timer
0433 55             1384          STRT    T               ;START TIMER
                    1385 ;       setup the next phase to output
0434 1B             1386          INC     PhzR30          ;STEP PHASE DB ADDRESS
0435 FB             1387          MOV     A,PhzR30        ;get the phase index address
0436 523A           1388          JB2     ZROPHL          ;test phase
0438 843C           1389          JMP     NXTPHL
043A B808           1390 ZROPHL:  MOV     PhzR30,#STLFF   ;re-init phase register
                    1391
043C FB             1392 NXTPHL:  MOV     A,PhzR30        ;get the phz reg indirect addr index
043D E3             1393          MovP3   A,@A            ;do indirect get of phz bits
                    1394 ;       patch together the CR last and LF next phase bits
043E B820           1395          Mov     TmpR00,#LastPh  ;load Last Phz psuedo reg to Temp Reg
0440 40             1396          ORL     A,@TmpR00       ;patch together CR existing & new LF
                    1397
0441 1645           1398 TLoopL:  JTF     NXPHLF          ;jmp on time out to output nxt phz
0443 8441           1399          JMP     TLOOPL          ;loop until timer times out
                    1400
0445 3C             1401 NXPHLF:  MOVD    P4,A            ;step motor - OUTPUT BITS
0446 EC31           1402          DJNZ    CntR40,StrLFT   ;test for end of phase count for line
                    1403                                  ;prep for next line
                    1404
                    1405 ;       test for various line/inch spacing would go here
0448 BC1B           1406          MOV     CntR40,#LPI8p8  ;init cnt reg for standard line feed
044A EE31           1407          DJNZ    LnCtR0,StrtLF   ;test for end of line count
                    1408
044C FA             1409          Mov     A,GStR20        ;Get the status byte
044D 53FB           1410          ANL     A,#LineFd       ;reset for line feed
044F AA             1411          Mov     GStR20,A        ;save the status byte
                    1412
                    1413 ;       store the phase register index addresses
                    1414 ;       Set LineFeed Stpr Mtr Next Phase index address
0450 B822           1415 SetLRN:  Mov     TmpR00,#LPSAdr  ;get Phz Storage Addr psuedo reg
```

```
0452 FB          1416          Mov     A,PhzR30        ;get the phase index address
0453 A0          1417          Mov     @TmpR00,A       ;store LF Next phase index addr
0454 B478        1418          Call    DlyLng
0456 B490        1419          Call    DeSlSM
                 1420
                 1421 ;        Check if Char Buffer contains full line (80 char or nChar & CR)
                 1422 ;          exit otherwise continue to read in characters
                 1423 ;         Mov     A,QStR20        ;get the stat byte
                 1424 ;         JB1     ByPas1          ;if Do Not Print Bit Set - EXIT
                 1425 ;         Call    CBFck
0458 83          1426 ByPas1: Ret
                 1427
                 1428 ; PG
                 1429 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1430 ;        Minor Software Subroutines
                 1431 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1432
0500             1433          ORG     500H
                 1434
                 1435 ; ---------------------------------------------------------------
                 1436 ;        System initialization subroutines
                 1437 ; ---------------------------------------------------------------
                 1438 Defalt:
                 1439 ; ---------------------------------------------------------------
                 1440 ;        reset/set EOF status flag bit = 0
0500 D5          1441          SEL     RB1
0501 FA          1442          Mov     A,ChStR1        ;get the char status byte
0502 53F7        1443          ANL     A,#ClrEOF       ;clear the EOF flag bit
0504 AA          1444          Mov     ChStR1,A        ;store the char status byte
0505 B823        1445          Mov     TmpR10,#PTAscS  ;get the Ascii code tmp store addr
0507 B020        1446          Mov     @TmpR10,#Ascii  ;load the tmp stor reg w/ascii start
0509 C5          1447          SEL     RB0
                 1448 ; ---------------------------------------------------------------
                 1449 ;        reset/set Ok-to-Print status flag bit = 0
050A FA          1450          Mov     A,QStR20        ;get the status byte
050B 53FD        1451          ANL     A,#OkPrnt       ;reset print flag - Ok Print
050D AA          1452          Mov     QStR20,A        ;save the status byte
050E 83          1453          RET
                 1454 InitAl:
                 1455 AllOff:
                 1456 ; ---------------------------------------------------------------
                 1457 ;        CLEAR all outputs
050F C5          1458          SEL     RB0
0510 230F        1459          MOV     A,#0FH          ;FORCE PORT HI - R/ OF 555
0512 3E          1460          MOVD    P6,A
0513 23FF        1461          MOV     A,#0FFH         ;TURN ALL PRNT SOL's OFF
0515 39          1462          OUTL    P1,A
0516 23C0        1463          MOV     A,#PTRGHI       ;print head fire tirgger inactive
0518 3A          1464          OUTL    P2,A
0519 8A03        1465          ORL     P2,#03          ;set comm hdsk to ACK hi/Busy hi
051B BA00        1466          Mov     QStR20,#00H     ;clear the status registers
051D D5          1467          SEL     RB1
051E BA00        1468          Mov     ChStR1,#00H
0520 C5          1469          SEL     RB0
0521 83          1470          RET                     ;RETURN TO INIT ROUTINE
                 1471
                 1472 ; PG
                 1473 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1474 ;        Home Carriage / Print Head Assembly
                 1475 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1476
0522 FA          1477 CRHome: Mov     A,QStR20        ;get the status byte
0523 4302        1478          ORL     A,#DoNotP       ;set the do not print flag
0525 AA          1479          Mov     QStR20,A        ;save the status byte
0526 362A        1480          JTO     RtoL            ;test for position of PH assembly
                 1481                                  ;  drive accordingly
0528 3402        1482          Call    FAccel          ;drive CR Stpr Mtr
052A 3422        1483 RtoL:    Call    RAccel          ;find the logical left home CR position
                 1484                                  ;delay a long time before continuing
052C B474        1485          Call    DlyVLg
052E 83          1486          Ret
                 1487
                 1488 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1489 ;        Clear Data Memory
                 1490 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1491
                 1492 ;        At PowerUp or Reset, following CR & LF Stpr Mtr Init, this
                 1493 ;        procedure clears data memory above RB0, Stack and RB1.
052F B87F        1494 ClrDM:   MOV     R0,#DMTop       ;GET BUFFER START LOCATION [HEX]
0531 B95D        1495          MOV     R1,#DMSIZE
0533 B000        1496 ClrDM1:  MOV     @R0,#00H        ;ZERO MEMORY LOCATION
```

```
0535 C8          1497          DEC     R0
0536 E933        1498          DJNZ    R1,ClrDM1       ;dec buffer, loop if not zero[end]
0538 83          1499          RET                     ;RETURN TO INIT ROUTINE
                 1500
                 1501 ; PG
                 1502 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1503 ;       Character Print TEST
                 1504 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1505
                 1506 PrnTst:
                 1507 ;  TEST ---  load the char buffer with successive increments of
                 1508 ;             the ascii code start.  test for end of ascii
                 1509 ;             printable chars and reinit the char stream loaded.
                 1510
0539 B97F        1511 CTInt:   Mov     CAdrR1,#FCBfSt  ;load char reg w/char bufr strt
053B BD50        1512          Mov     CCntR1,#ChBfSz  ;load char cnt reg w/char bufr size
                 1513 ChTst:                           ;Test char buffer fill with ASCII Char Code
053D FF          1514          Mov     A,opnr71        ;get the ascii char
053E A1          1515          Mov     @CAdrR1,A       ;load data memory w/Char
053F C9          1516          DEC     CAdrR1          ;Decrement dat memory location
0540 1F          1517          INC     opnr71          ;Increment Ascii char number
0541 0382        1518          ADD     A,#PAsEnd       ;test for ascii code end
0543 9647        1519          JNZ     ChrTQo          ;if not end jmp over code restart
0545 BF20        1520          Mov     OpnR71,#Ascii
0547 ED3D        1521 ChrTQo:  DJNZ    CCntR1,ChTst    ;dec buffer, loop if not zero[end]
0549 C5          1522          SEL     RB0
054A 83          1523          RET                     ;ELSE RETURN TO INIT ROUTINE
                 1524
                 1525 ; PG
                 1526 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1527 ;       CR Stpr Mtr Power On Initialization and
                 1528 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1529 ;       This routine drives the CR or LF stpr mtr for four phase
                 1530 ;         shifts for initialization.
                 1531 INITCR:                          ;POWER ON INIT STPR MTR
054B BC04        1532          MOV     CntR40,#PhCnt1  ;load phase cnt reg for INIT
054D 2308        1533          MOV     A,#SCR80        ;GET CR SM SELECT BITS
054F 3D          1534          MOVD    P5,A            ;SELECT SM [SCR80]
0550 BDC0        1535          MOV     TConR0,#IntTm2  ;Load time constant Reg
0552 BB00        1536          MOV     PhzR30,#FStCRP  ;zero SM phase reg - forward
0554 FB          1537          MOV     A,PhzR30        ;get phase index register byte
0555 E3          1538          MovP3   A,@A            ;load indexed phase shift byte
0556 3C          1539          MOVD    P4,A            ;OUTPUT BITS
0557 FD          1540 STRTTR:  MOV     A,TConR0        ;GET TIMER CONSTANT
0558 62          1541          MOV     T,A
0559 55          1542          STRT    T               ;START TIMER
055A 1B          1543          INC     PhzR30          ;step phase index register
055B FB          1544          MOV     A,PhzR30        ;CHECK THE PHASE COUNT REG
055C 5260        1545          JB2     ZroRg2
055E A462        1546          JMP     NxtPhR
0560 BB00        1547 ZroRg2:  MOV     PhzR30,#FStCRP  ;zero SM phase reg - forward
                 1548 NxtPhR:
0562 FB          1549          MOV     A,PhzR30        ;get phase index register byte
0563 E3          1550          MovP3   A,@A            ;load indexed phase shift byte
0564 1669        1551 TLoopR:  JTF     NXPHR1          ;JMP ON TIME OUT TO NEXT PH
0566 A464        1552          JMP     TLoopR          ;LOOP UNTIL TIME OUT
0568 3C          1553          MOVD    P4,A            ;OUTPUT BITS
0569 EC57        1554 NXPHR1:  DJNZ    CntR40,STRTTR
                 1555 ; ---------------------------------------------------------------
                 1556 ;       store the last phase register index addresses
056B B821        1557          Mov     TmpR00,#CPSAdr  ;get Phz Storage Addr psuedo reg
056D FB          1558          Mov     A,PhzR30        ;place last CR phase index addr in Phz Reg
056E A0          1559          Mov     @TmpR00,A       ;  store CR last phase index addr
056F B478        1560          Call    DlyLng
0571 B490        1561          Call    DeSlSM
0573 83          1562          RET
                 1563
                 1564 ; PG
                 1565 ; ---------------------------------------------------------------
                 1566 ;       Time Delay Subroutines
                 1567 ; ---------------------------------------------------------------
                 1568
                 1569 ;       Very Long
0574 B87F        1570 DlyVLg:  MOV     TmpR00,#7FH     ;LOAD DELAY COUNT IN REG.
0576 A47E        1571          Jmp     DlyST
                 1572
                 1573 ;       Long
0578 B880        1574 DlyLng:  MOV     TmpR00,#DlyCL   ;LOAD DELAY COUNT IN REG.
057A A47E        1575          Jmp     DlyST
                 1576
```

```
                        1577 ;       Not So Long - Short
057C B830               1578 DlySht: MOV     TmpR00,#DlyCS  ;LOAD DELAY COUNT IN REG.
                        1579
                        1580 ;       Start Delay
057E 23CC               1581 DlyST:  MOV     A,#DlyTim      ;GET MAX TIMER DELAY
0580 62                 1582 NxtTLd  MOV     T,A            ;LOAD TIMER
0581 55                 1583         STRT    T              ;START TIMER
                        1584
0582 168D               1585 DlyLop: JTF     DlyTO          ;LOOP
                        1586
                        1587 ;       Char buffer fill during time loop:
0584 D5                 1588         SEL     RB1
0585 FA                 1589         Mov     A,ChStR1       ;get the character stat reg byte
0586 928A               1590         JB4     SkpCI          ;  test for normal char input
                        1591                                ;  or skip if char prnt test
0588 1469               1592         Call    IBFSrv         ;service the char buffer fill
058A C5                 1593 SkpCI:  SEL     RB0
058B A482               1594         JMP     DlyLOP
058D E880               1595 DlyTO:  DJNZ    TmpR00,NxtTLd   ;dec delay count & test for exit
058F 83                 1596         RET
                        1597 ; ---------------------------------------------------------------
                        1598 ;       Stpr Mtr Deselect
                        1599 ; ---------------------------------------------------------------
                        1600 ;       Stepper Motor DeSelect Routine
                        1601 DESLSM:                       ;DESELECT LF/CR SM
0590 230E               1602 SMERDR. MOV     A,#SMOFF       ;GET LF/CR SM DE-SELECT BITS
0592 3D                 1603         MOVD    P5,A           ;DE-SELECT CR SM
0593 83                 1604         RET
                        1605
                        1606 $INCLUDE(:F1:CHRTBL.OV1)
                       =1607
                       =1608 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       =1609 ;       Character Dot Generator Look-up Table Page 1
                       =1610 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                       =1611
                       =1612
                       =1613 ;              Character Table Page 1, contains
                       =1614
                       =1615 ;     20H ----------------------------------------> 4FH
                       =1616
                       =1617 ;   " (sp)!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLM "
                       =1618
                       =1619 ; ---------------------------------------------------------------
                       =1620
0600                   =1621         ORG     600H
                       =1622
                       =1623 ; ---------------------------------------------------------------
                       =1624 ;       Page 1  --  Character Dot Pattern Fetch
                       =1625 ;       <<<  actual assembled character table code not listed  >>>
                       =1626 ; ---------------------------------------------------------------
                       =1627 $NoList
                       =1676 $List
                       =1677 ;       Listing below is for reference only, actual code is not listed
                       =1678 ;         at assembly time.
                       =1679
                       =1680 ; ---------------------------------------------------------------
                       =1681 ; asc20:        DB      7FH, 7FH, 7FH, 7FH, 7FH         ;SPACE
                       =1682 ; asc21:        DB      7FH, 7FH, 20H, 7FH, 7FH         ;!
                       =1683 ; asc22:        DB      7FH, 7FH, 78H, 7FH, 78H         ;"
                       =1684 ; asc23:        DB      6BH, 00H, 6BH, 00H, 6BH         ;#
                       =1685 ; asc24:        DB      5BH, 55H, 00H, 55H, 6DH         ;$
                       =1686 ; asc25:        DB      5CH, 6CH, 77H, 1BH, 1DH         ;%
                       =1687 ; asc26:        DB      19H, 26H, 26H, 59H, 2FH         ;&
                       =1688 ; asc27:        DB      7FH, 7FH, 7CH, 7FH, 7FH         ;'
                       =1689 ; asc28:        DB      63H, 5DH, 3EH, 7FH, 7FH         ;(
                       =1690 ; asc29:        DB      7FH, 7FH, 3EH, 5DH, 63H         ;)
                       =1691 ; asc2A:        DB      5DH, 6BH, 00H, 6BH, 5DH         ;*
                       =1692 ; asc2B:        DB      77H, 77H, 41H, 77H, 77H         ;+
                       =1693 ; asc2C:        DB      7FH, 3FH, 4FH, 7FH, 7FH         ;,
                       =1694 ; asc2D:        DB      77H, 77H, 77H, 77H, 77H         ;-
                       =1695 ; asc2E:        DB      7FH, 1FH, 1FH, 7FH, 7FH         ;.
                       =1696 ; asc2F:        DB      5FH, 6FH, 77H, 7BH, 7DH         ;/
                       =1697 ; asc30:        DB      41H, 2EH, 36H, 3AH, 41H         ;0
                       =1698 ; asc31:        DB      7FH, 3DH, 00H, 3FH, 7FH         ;1
                       =1699 ; asc32:        DB      3DH, 1EH, 2EH, 36H, 39H         ;2
                       =1700 ; asc33:        DB      5DH, 3EH, 36H, 36H, 49H         ;3
                       =1701 ; asc34:        DB      67H, 6BH, 6DH, 00H, 6FH         ;4
                       =1702 ; asc35:        DB      58H, 3AH, 3AH, 3AH, 46H         ;5
                       =1703 ; asc36:        DB      43H, 35H, 36H, 36H, 4EH         ;6
                       =1704 ; asc37:        DB      7EH, 0EH, 76H, 7AH, 7CH         ;7
                       =1705 ; asc38:        DB      49H, 36H, 36H, 36H, 49H         ;8
```

```
=1706 ; asc39:          DB      39H, 36H, 36H, 56H, 61H         ;9
=1707 ; asc3A:          DB      7FH, 7FH, 6BH, 7FH, 7FH         ;:
=1708 ; asc3B:          DB      7FH, 3FH, 4BH, 7FH, 7FH         ;;
=1709 ; asc3C:          DB      77H, 6BH, 5DH, 3EH, 7FH         ;<
=1710 ; asc3D:          DB      6BH, 6BH, 6BH, 6BH, 6BH         ;=
=1711 ; asc3E:          DB      7FH, 3EH, 5DH, 6BH, 77H         ;>
=1712 ; asc3F:          DB      79H, 7EH, 26H, 7AH, 7DH         ;?
=1713 ; asc40:          DB      41H, 3EH, 22H, 36H, 71H         ;@
=1714 ; asc41:          DB      03H, 6DH, 6EH, 6DH, 03H         ;A
=1715 ; asc42:          DB      00H, 36H, 36H, 36H, 49H         ;B
=1716 ; asc43:          DB      41H, 3EH, 3EH, 3EH, 5DH         ;C
=1717 ; asc44:          DB      00H, 3EH, 3EH, 5DH, 63H         ;D
=1718 ; asc45:          DB      00H, 36H, 36H, 36H, 36H         ;E
=1719 ; asc46:          DB      00H, 76H, 76H, 76H, 76H         ;F
=1720 ; asc47:          DB      41H, 3EH, 3EH, 2EH, 0DH         ;G
=1721 ; asc48:          DB      00H, 77H, 77H, 77H, 00H         ;H
=1722 ; asc49:          DB      7FH, 3EH, 00H, 3EH, 7FH         ;I
=1723 ; asc4A:          DB      5FH, 3FH, 3FH, 3FH, 40H         ;J
=1724 ; asc4B:          DB      00H, 77H, 6BH, 5DH, 3EH         ;K
=1725 ; asc4C:          DB      00H, 3FH, 3FH, 3FH, 3FH         ;L
=1726 ; asc4D:          DB      00H, 7DH, 73H, 7DH, 00H         ;M
=1727 ; asc4E:          DB      0aaH, 0dfH, 0efH, 0f7H, 0aaH    ;test
=1728 ; asc4F:          DB      55H, 0dfH, 0efH, 0f7H, 55H      ;test
=1729 ; asc4E:          DB      00H, 7BH, 77H, 6FH, 00H         ;N
=1730 ; asc4F:          DB      41H, 3EH, 3EH, 3EH, 41H         ;O
=1731 ; ---------------------------------------------------------------------------
=1732 ;         End Page 1  --  Character Dot Pattern Fetch
=1733
=1734 ; ---------------------------------------------------------------------------
=1735 ;         Character Dot Pattern Fetch
=1736 ; ---------------------------------------------------------------------------
=1737
06F0 FC         =1738 ChrPg1: MOV     A,CDotR1          ;get char index address offset
06F1 A3         =1739         MOVP    A,@A              ;get column dot patern byte
=1740
=1741 ;         this bit fix necessary to not underline each character
=1742 ;           this saves fixing each bit in the look up table
=1743
06F2 4380       =1744         ORL     A,#80H            ;char bit fix
06F4 39         =1745         OutL    P1,A              ;output the dot pattern
06F5 83         =1746         RET                       ;exit with byte in acc
=1747
=1748 ; ---------------------------------------------------------------------------
=1749 ;         END Page 1  --  Character Dot Pattern Fetch
=1750 ; ---------------------------------------------------------------------------
=1751
=1752
=1753 ;         PAGE 2  -- Character Dot Generator Look-Up Table
=1754 ; ---------------------------------------------------------------------------
=1755
=1756 ;                 Character Table Page 2, contains
=1757
=1758 ;      50H ------------------------------------------> 7EH
=1759
=1760 ;    " NOPQRSTUVWXYZ[\]^_(?)abcdefghijklmnopqrstuvwxyz{|}~ "
=1761
=1762 ; ---------------------------------------------------------------------------
=1763
0700            =1764         ORG     700H
=1765
=1766 ; ---------------------------------------------------------------------------
=1767 ;         Page 2  --  Character Dot Pattern Fetch
=1768 ;         <<<  Actual assembled character table code not listed  >>>
=1769 ; ---------------------------------------------------------------------------
=1770 $NoLIST
=1818 $List
=1819 ;         Listing below is for reference only, actual code is not listed
=1820 ;           at assembly time.
=1821
=1822 ; ---------------------------------------------------------------------------
=1823 ; asc50:          DB      00H, 76H, 76H, 76H, 79H         ;P
=1824 ; asc51:          DB      41H, 3EH, 2EH, 5EH, 21H         ;Q
=1825 ; asc52:          DB      00H, 76H, 66H, 56H, 39H         ;R
=1826 ; asc53:          DB      59H, 36H, 36H, 36H, 4DH         ;S
=1827 ; asc54:          DB      7EH, 7EH, 00H, 7EH, 7EH         ;T
=1828 ; asc55:          DB      40H, 3FH, 3FH, 3FH, 40H         ;U
=1829 ; asc56:          DB      60H, 5FH, 3FH, 5FH, 60H         ;V
=1830 ; asc57:          DB      00H, 5FH, 67H, 5FH, 00H         ;W
=1831 ; asc58:          DB      1CH, 6BH, 77H, 6BH, 1CH         ;X
=1832 ; asc59:          DB      7CH, 7BH, 07H, 7BH, 7CH         ;Y
```

```
                =1833 ;  asc5A:         DB      1EH, 2EH, 36H, 3AH, 3CH          ;Z
                =1834 ;  asc5B:         DB      00H, 3EH, 3EH, 3EH, 7FH          ;[
                =1835 ;  asc5C:         DB      7DH, 7BH, 77H, 6FH, 5FH          ;\
                =1836 ;  asc5D:         DB      7FH, 3EH, 3EH, 3EH, 00H          ;]
                =1837 ;  asc5E:         DB      6FH, 77H, 7BH, 77H, 6FH          ;^
                =1838 ;  asc5F:         DB      3FH, 3FH, 3FH, 3FH, 3FH          ;_
                =1839 ;  asc60:         DB      7DH, 7BH, 77H, 0FFH, 0FFH        ;\
                =1840 ;  asc61:         DB      0DFH, 0ABH, 0ABH, 0ABH, 087H     ;a
                =1841 ;  asc62:         DB      080H, 0B7H, 0B7H, 0B7H, 0CFH     ;b
                =1842 ;  asc63:         DB      0C7H, 0BBH, 0BBH, 0BBH, 0BBH     ;c
                =1843 ;  asc64:         DB      0CFH, 0B7H, 0B7H, 0B7H, 080H     ;d
                =1844 ;  asc65:         DB      0C7H, 0ABH, 0ABH, 0ABH, 0B7H     ;e
                =1845 ;  asc66:         DB      0F7H, 081H, 0F6H, 0FEH, 0FDH     ;f
                =1846 ;  asc67:         DB      0F7H, 0ABH, 0ABH, 0ABH, 0C3H     ;g
                =1847 ;  asc68:         DB      080H, 0F7H, 0FBH, 0FBH, 087H     ;h
                =1848 ;  asc69:         DB      0FFH, 0BFH, 0BBH, 0BFH, 0FFH     ;i
                =1849 ;  asc6A:         DB      0DFH, 0BFH, 0BBH, 0C2H, 0FFH     ;j
                =1850 ;  asc6B:         DB      0FFH, 080H, 0EFH, 0D7H, 0BBH     ;k
                =1851 ;  asc6C:         DB      0FFH, 0BEH, 080H, 0BFH, 0FFH     ;l
                =1852 ;  asc6D:         DB      087H, 0FBH, 0E7H, 0FBH, 087H     ;m
                =1853 ;  asc6E:         DB      083H, 0F7H, 0FBH, 0FBH, 087H     ;n
                =1854 ;  asc6F:         DB      0C7H, 0BBH, 0BBH, 0BBH, 0C7H     ;o
                =1855 ;  asc70:         DB      084H, 0EBH, 0EBH, 0EBH, 0F7H     ;p
                =1856 ;  asc71:         DB      0F7H, 0EBH, 0EBH, 0EBH, 084H     ;q
                =1857 ;  asc72:         DB      0FFH, 083H, 0F7H, 0FBH, 0FBH     ;r
                =1858 ;  asc73:         DB      0B7H, 0ABH, 0ABH, 0ABH, 0DBH     ;s
                =1859 ;  asc74:         DB      0FBH, 0C1H, 0BBH, 0DFH, 0FFH     ;t
                =1860 ;  asc75:         DB      0C3H, 0BFH, 0BFH, 0BFH, 0C3H     ;u
                =1861 ;  asc76:         DB      0E3H, 0DFH, 0BFH, 0DFH, 0E3H     ;v
                =1862 ;  asc77:         DB      0C3H, 0BFH, 0CFH, 0BFH, 0C3H     ;w
                =1863 ;  asc78:         DB      0BBH, 0C7H, 0EFH, 0C7H, 0BBH     ;x
                =1864 ;  asc79:         DB      0FFH, 0B3H, 0AFH, 0AFH, 0C3H     ;y
                =1865 ;  asc7A:         DB      0BBH, 09BH, 0ABH, 0B3H, 0BBH     ;z
                =1866 ;  ASC7B:         DB      07FH, 077H, 049H, 03EH, 03EH     ;{
                =1867 ;  ASC7C:         DB      0FFH, 0FFH, 088H, 0FFH, 0FFH     ;|
                =1868 ;  ASC7D:         DB      03EH, 03EH, 009H, 077H, 07FH     ;}
                =1869 ;  ASC7E:         DB      067H, 07BH, 067H, 05FH, 067H     ;~
                =1870
                =1871 ; ---------------------------------------------------------------
                =1872 ;       Character Dot Pattern Fetch
                =1873 ; ---------------------------------------------------------------
                =1874
07EB FC         =1875 ChrPg2: MOV     A,CDotR1          ;get char index address offset
07EC A3         =1876         MOVP    A,@A              ;get column dot patern byte
                =1877
                =1878 ;       this bit fix necessary to not underline each character
                =1879 ;         this saves fixing each bit in the look up table
                =1880
07ED 4380       =1881         ORL     A,#80H            ;char bit fix
07EF 39         =1882         OutL    P1,A              ;output the dot pattern
07F0 83         =1883         RET                       ;exit with byte in acc
                 1884
                 1885
                 1886
                 1887 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1888 ;               Program End
                 1889 ; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                 1890
                 1891          END

ASSEMBLY COMPLETE,   NO ERRORS
```

APPENDIX B. SOFTWARE PRINTER ENHANCEMENTS

This section describes several software enhancements which could be implemented as additions to the software developed for this Application Note. Space is available for most of the items described. Approximately 5 bytes of Data Memory would be required to implement most of the features. Two bytes would be used for status flags, and two bytes for temporary data or count storage. It is possible to use less than five bytes, but this would require the duplicate use of some flags, or other Data Memory storage, which will significantly complicate the software coding and debug tasks.

Special Characters or Symbols

Dot matrix printing lends itself well to the creation of custom characters and symbols. There are two aspects to implementing special characters. First, a character look-up table, and second, additional software for dec oding and processing the special characters or symbols. Special characters might be scientific notation, mathematical symbols, unique language characters, or block and line graphics characters.

The character look-up table could be an additional page of Program Memory dedicated to the special characters, or replace part, or all, of the existing look-up tables. If an additional look-up table is used, a third page test would be needed at the beginning of the Character Translation subroutine. There is fundamentally no difference between the processing of special characters and standard ASCII printable characters. If the characters require the same 5 x 7 dot matrix, the balance of the software would remain the same. If, however, the special characters require a different matrix, or the manipulation of the matrix, the software becomes more complex.

In general, the major software modification required to implement special characters is the size of the dot matrix printed or the dot matrix configuration used. In the case of scientific characters, it would often be necessary to shift the 5 x 7 matrix pattern within the available 9 x 9 matrix. Block or line graphics characters, on-the-other-hand, would require using the entire 9 x 9 print head matrix and printing during normally blank dot columns. This would require suspending the blank column blanking mechanism implemented in this Application Note. This would be the most complex aspect of implementing special characters. It would possibly change the number of required instructions, and thus the timing between PTS detection and print head solenoid trigger firing. This could cause the dot columns to be misaligned within a printed line and between lines.

In the case of a matrix change, two approaches are possible: dynamically changing the matrix, in line, as standard ASCII characters are being printed, or isolating the special characters to a separate processing flow where special characters are handled as a unique and complete line of characters only. A discussion of in line matrix changes for special characters is beyond the scope of this Appendix. It is sufficient to say that the changes would require the conditions setting the EOLN flag, character count, and dot column count software be modified during character processing and printing.

Lower Case Descenders

The general principle of implementing lower case descenders is to shift the 5 x 7 character dot matrix within the available 9 x 9 print head solenoid matrix. Implementing lower case descenders requires two software modifications and the creation of status flag for the purpose. First, the detection of characters needing descenders and setting a dedicated status flag during the character code to dot pattern translation subroutine. Second, the character dot column data output to the print head solenoids must be shifted for each dot column of the character. At the end of the character, the flag would be reset.

Inline Control Codes

Inline control codes are two to three character sequences, which indicate special hardware conditions or software flow control and branching. The first character indicates that the control code sequence is beginning and is typically an ASCII Escape character (ESC), 1BH. Termination of the inline code sequence would be indicated by a default number of code sequence characters. This would decrease the buffer size available for characters. Full 80 character line buffering would require loading the Character Buffer with a received character as a character is removed from it and processed.

The Inline Control Code test would be performed in two places: in the Character Buffer Fill subroutine and in the Character Processing (translation) subroutine. The test would be performed in the same manner that a Carriage Return (CR) character code test is implemented. Examples are horizontal tabs and expanded or condensed character fonts. In the case of horizontal tabs, 20H (Space Character) would have to be placed in the Character Buffer for inline processing during character processing and printing. Unless fixed position tabs are used, a minimum of a nibble of Data Memory would be required to maintain a "spaces-to-tab" count. Fixed tab positions could be set via another inline control code, by default of the printer software, or through the use of external hardware switch settings. The control code method of setting the tab positions is the most desireable, but the most complex to implement.

Different Character Formats

Figure B1 illustrates three different character fonts; standard, condensed, and enlarged or expanded characters. As the the figure illustrates, condensed and

enlarged characters are variations in either the number of dots and/or the space used to print them. Thus, each character is a variation of the stepper motor and/or print head solenoid trigger timings. Figure B2 illustrates the timings required to implement the additional character printing.

In addition to the three character fonts shown, it is possible to print each in bold face by printing each dot twice per dot column position. This would require little software modification, but would require a status flag. Again, care must be used to ensure that the delay in retriggering the solenoids is precisely the same for each type of event. Without this precise timing the dot column alignment will not be accurate. The software modifications needed to implement enlarged or condensed characters is essentially the same. The carriage and print head solenoid firing software flow is the same, but the timing for each changes. For condensed characters, the step Time Constant is doubled to approximately 4.08 ms, and the solenoids are fired four times within each step time. The step rate actually becomes a multiple of the solenoid firing time, and a counter incrementing once for each solenoid firing would be needed. At the count of four, the carriage stepper motor is stepped and the counter reset.

In the case of condensed characters, PTS does not play the same roll as in standard or enlarged character printing. PTS is not used to indicate the optimum print head solenoid firing time. Solenoid firing is purely a time function for condensed characters. PTS would only be used for Failsafe protection.

Enlarged characters would require the solenoids be fired twice per dot column data, in two sequential dot columns, at the same rate as standard characters. The character dot column data and dot column count would not be incremented at each output but at every other output. A flag could be used for this purpose.

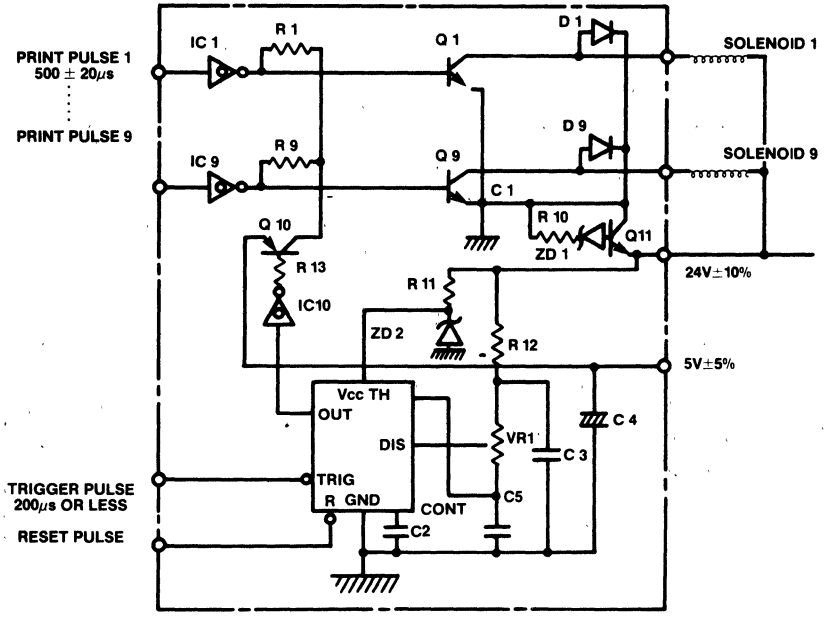
When printing either condensed or enlarged characters, the maximum character count would have to compensate for the increased or decreased characters per line count. When printing enlarged characters, the maximum characters per line would be 40. The Character Buffer could hold two complete lines of characters. But, condensed characters presents a quite different situation. The available character per line increases to 132, well beyond the 80 character Character Buffer size. The solution is to re-initialize the Character Buffer Size Count register count during condensed character processing. This will effectively inhibit the carriage stepper motor drive EOLN detection.

Two status flags would be required; one for standard or enlarged characters, and the second for condensed characters. A third status flag would be required to implement bold face printing. Activating one of the alternate character fonts could be either through the use of external status switches or through inline control code sequences, as detailed above. Note, that if the alternate character fonts are implemented in such a way that format changing is to occur dynamically during any single line being printed, the same control code problems described above also apply. In addition, the effect on the timing and dot column alignment must also be investigated.

Variable Line Spacing

Variable line spacing is another feature which could be implemented either through the use of external status switches or inline control codes. The line spacing is a function of the number of steps the stepper motor rotates for a given line. Figure 15, Paper Feed Stepper Motor Predetermined Time Constants, in the Background section above, lists the Time Constants required for three different line spacings; 6, 8, and 10 lines per inch. At the beginning of the Paper Feed Stepper Motor Drive subroutine, the default line step count is loaded. The software required is a conditional load for the line spacing, indicated by a status flag set in the External Status Switch Check subroutine or the Character Buffer Fill subroutine. Implementing the three different line spacings would require two additional status flags.

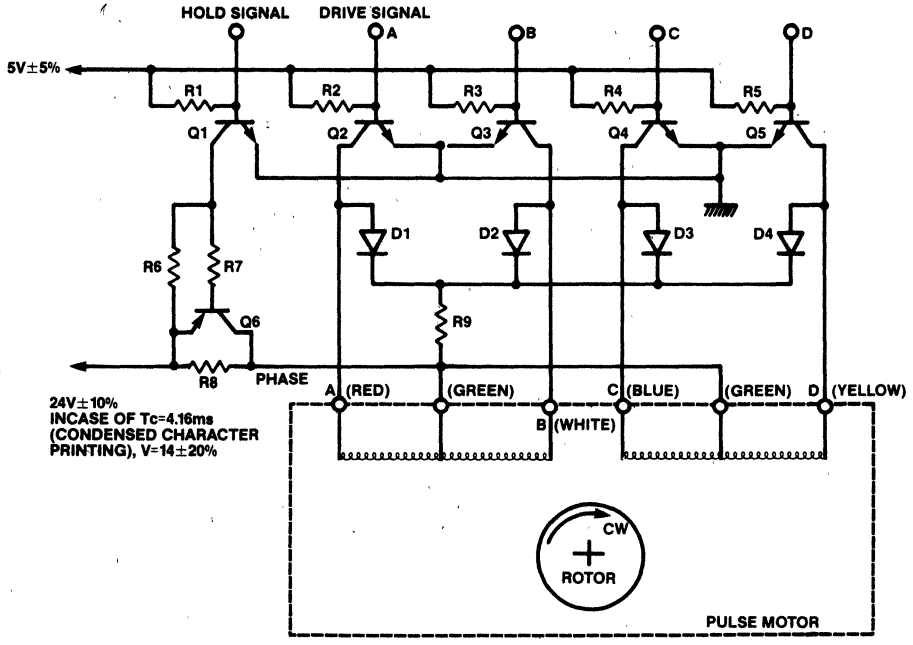
APPENDIX C.
PRINTER MECHANISM DRIVE CIRCUIT



Recommended Solenoid Drive Circuit

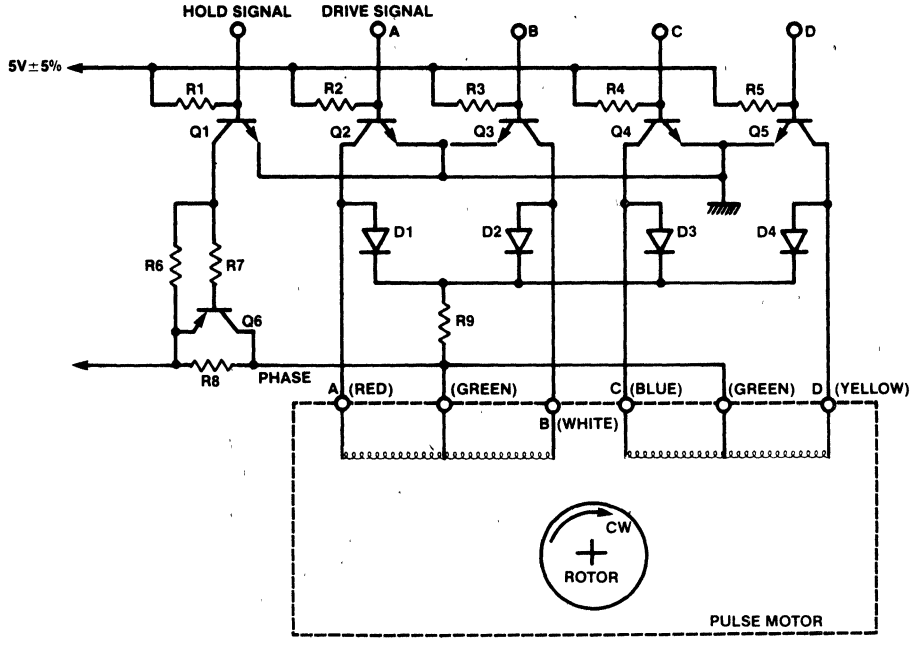
PARTS NO.		TYPE	MAKER
IC1~IC10		SN7406	TI
IC11		μA555	Fairchild
D1~D9	DIODE	S5277B	Toshiba
Q1~Q9	TRANSISTOR	2SD986	NEC
Q10	TRANSISTOR	2SA1015	Toshiba
Q11	TRANSISTOR	2SD633	Toshiba
R1~R9	RESISTOR	1.2kΩ ¼	
R10	RESISTOR	22Ω ¼	
R11	RESISTOR	580Ω 2	
R12	RESISTOR	15kΩ ¼ Carbon fil=	
R13	RESISTOR	1.2kΩ ¼	
VR1	VARIABLE RESISTOR	20kΩ ¼	
C1	CAPACITOR	1μF 100V	
C2	CAPACITOR	0.01μF	
C3	CAPACITOR	0.001μF	
C4	CAPACITOR	10μF 16V	
C5	CAPACITOR	0.1μF fil=	
ZD1	ZENOR DIODE	HZ24	Hitachi
ZD2	ZENOR DIODE	HZ5C1	Hitachi

Recommended Carriage Motor Drive Circuit

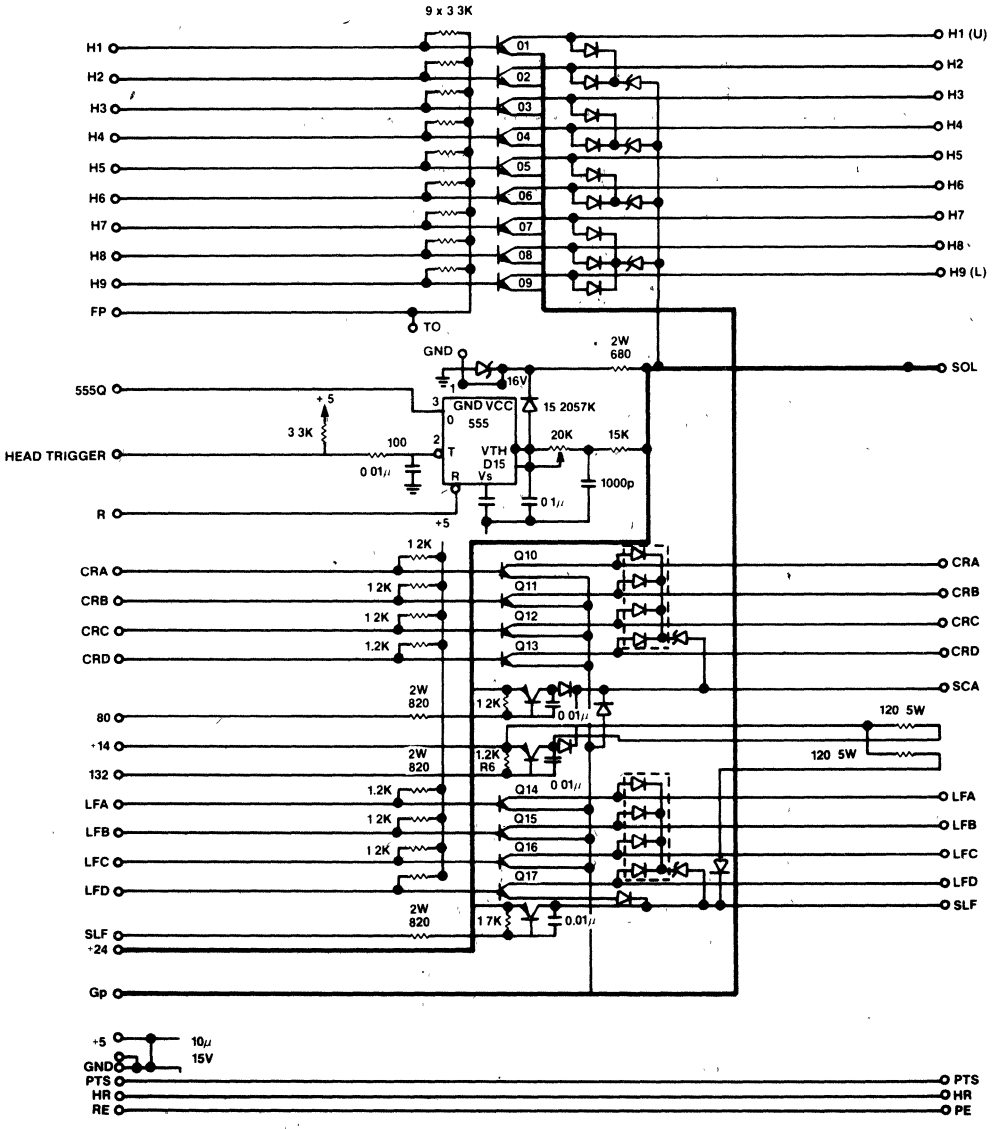


PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4

Recommended Paper Feed Motor Drive Circuit



PARTS NO.		TYPE	MAKER	QTY
R1	Resistor	1kΩ±10% ¼		1
R2-R5	Resistor	220Ω±10% ¼		4
R6	Resistor	10kΩ±10% ¼		1
R7	Resistor	470Ω±10% 3		1
R8	Resistor	130Ω±10% 7		1
R9	Resistor	330Ω±10% 3		1
Q1	Transistor	2SC1815	Toshiba	1
Q2~Q5	Transistor	2SD526—Y	Toshiba	4
Q6	Transistor	2SB669	Matsushita	1
D1~D4	Diode	1S954	NEC	4


9 x 3 3K
H1
H2
H3
H4
H5
H6
H7
H8
H9
FP
TO
01
02
03
04
05
06
07
08
09
H1 (U)
H2
H3
H4
H5
H6
H7
H8
H9 (L)
GND
2W
680
SOL
16V
555Q
+5
3 3K
HEAD TRIGGER
100
0 01μ
GND VCC
555
15 2057K
20K
15K
VTH
D15
Vs
1000p
0 1μ
R
+5
1 2K
CRA
CRB
CRC
CRD
Q10
Q11
Q12
Q13
2W
820
80
+14
132
LFA
LFB
LFC
LFD
SLF
+24
Gp
1.2K
R6
0 01μ
Q14
Q15
Q16
Q17
1 7K
0.01μ
120 5W
SCA
+5
GND
PTS
HR
RE
10μ
15V
PE

230795-001

May 1980

An 8741A/8041A Digital Cassette Controller

John Beaston, Jim Kahn
Peripheral Applications

INTRODUCTION

The microcomputer system designer requiring a low-cost, non-volatile storage medium has a difficult choice. His options have been either relatively expensive, as with floppy discs and bubble memories, or non-transportable, like battery backed-up RAMs. The full-sized digital cassette option was open but many times it too was too expensive for the application. Filling this void of low-cost storage is the recently developed digital mini-cassette. These mini-cassettes are similar to, but not compatible with, dictation cassettes. The mini-cassette transports are inexpensive (well under $100 in quantity), small (less than 25 cu. in.), low-power (one watt), and their storage capacity is a respectable 200K bytes of unformatted data on a 100-foot tape. These characteristics make the mini-cassette perfect for applications ranging from remote datalogging to program storage for hobbyist systems.

The only problem associated with mini-cassette drives is controlling them. While these drives are relatively easy to interface to a microcomputer system, via a parallel I/O port, they can quickly overburden a CPU if other concurrent or critical real-time I/O is required. The cleanest and probably the least expensive solution in terms of development cost is to use a dedicated single-chip controller. However, a quick search through the literature turns up no controllers compatible with these new transports. What to do? Enter the UPI-41A family of Universal Peripheral Interfaces.

The UPI-41A family is a group of two user-programmable slave microcomputers plus a companion I/O expander. The 8741A is the "flag-chip" of the line. It is a complete microcomputer with 1024 bytes of EPROM program memory, 64 bytes of RAM data memory, 16 individually programmable I/O lines, an 8-bit event counter and timer, and a complete slave peripheral interface with two interrupts and Direct Memory Access (DMA) control. The 8041A is the masked ROM, pin compatible version of the 8741A. Figure 2 shows a block diagram common to both parts. The 8243 I/O port expander completes the family. Each 8243 provides 16 programmable I/O lines.

Using the UPI concept, the designer can develop a custom peripheral control processor for his particular I/O problem. The designer simply develops his peripheral control algorithm using the UPI-41A assembly language and programs the EPROM of

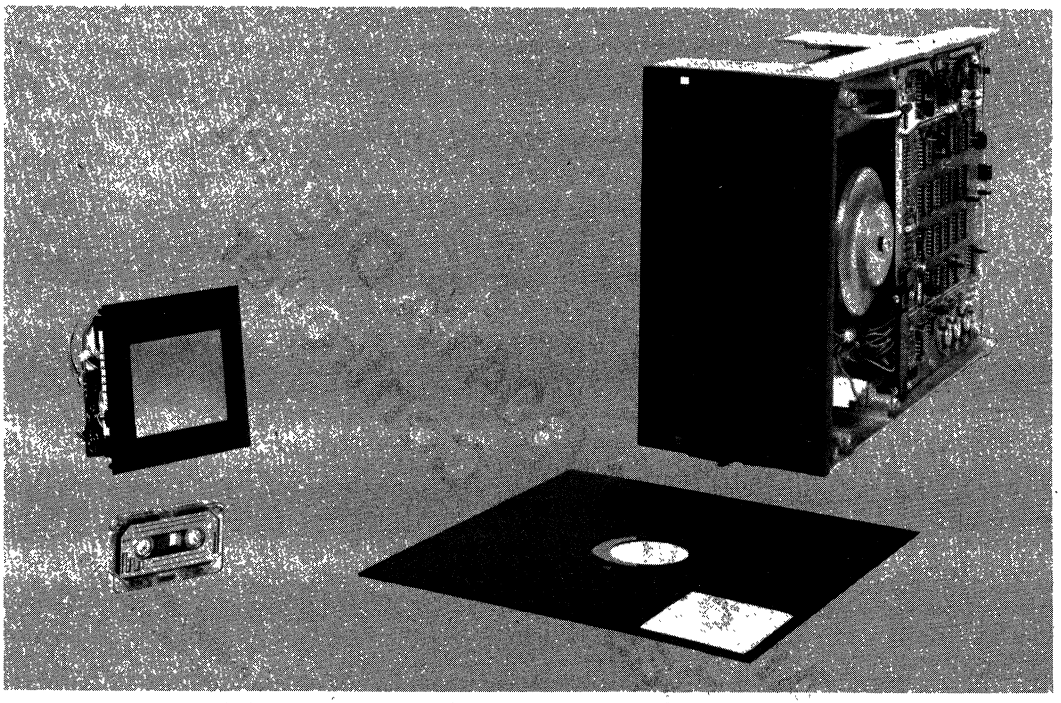


Figure 1. Comparison of Mini-Cassette and Floppy Disk Transports and Media.

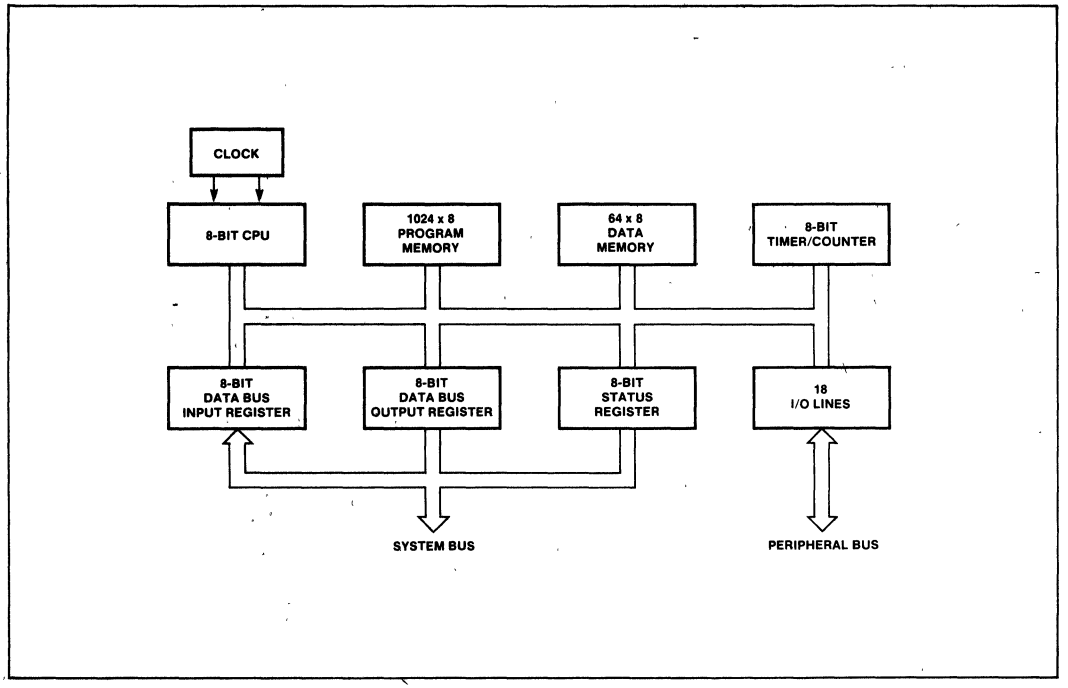


Figure 2. 8741A/8041A Block Diagram

the 8741A. Voila! He has a single-chip dedicated controller. Testing may be accomplished using either an ICE-41A or the Single-step mode of the 8741A. UPI-41A peripheral interfaces are being used to control printers, keyboards, displays, custom serial interfaces, and data encryption units. Of course, the UPI family is perfect for developing a dedicated controller for digital mini-cassette transports. To illustrate this application for the UPI family let's consider the job of controlling the Braemar CM-600 Mini-Dek®.

THE CM-600 MINI-DEK*

The Braemar CM-600 is representative of digital mini-cassette transports. It is a single-head, single-motor transport which operates entirely from a single 5-volt power supply. Its power requirements, including the motor, are 200ma for read or write and 700ma for rewind. Tapes speeds are 3 inches per second (IPS) during read or write, 5 IPS fast forward, and 15 IPS rewind. With these speeds and a maximum recording density of 800 bits per inch (BPI), the maximum data rate is 2400 bits per second (BAUD). The data capacity using both sides of a 100-foot tape is 200K bytes. On top of this, the transport occupies only 22.5 cubic inches (3"x3"x2.5").

All I/O for the CM-600 is TTL-compatible and can be divided into three groups: motor control, data control, and cassette status. The motor group controls are GO/STOP, FAST/SLOW, and FORWARD/REVERSE. The data controls are READ/ WRITE, DATA IN, and DATA OUT. The remaining group of outputs give the transport's status: CLEAR LEADER, CASSETTE PRESENCE, FILE PROTECT, and SIDE SENSOR. These signals, shown schematically in figure 3 and table 1, give the pin definition of the CM-600 16-pin I/O connector.

RECORDING FORMAT

The CM-600 does not provide either encoding or decoding of the recorded data; that task is left for the peripheral interface. A multitude of encoding techniques from which the user may choose are available. In this single-chip dedicated controller application, a "self-clocking" phase encoding scheme similar to that used in floppy discs was chosen. This scheme specifies that a logic "0" is a bit cell with no transition; a cell with a transition is a logic "1."

Table 1. CM-600* I/O Pin Definition

Pin	I/O	Function
1	—	Index pin—not used
2	—	Signal ground
3	O	Cassette side (0—side B, 1—side A)
4	I	Data input (0—space, 1—mark)
5	O	Cassette presence (0—cassette, 1—no cassette)
6	I	Read/Write (0—read, 1—write)
7	O	File protect (0—tab present, 1—tab removed)
8	—	+5v motor power
9	—	Power ground
10	—	Chassis ground
11	I	Direction (0—forward, 1—rewind)
12	I	Speed (0—fast, 1—slow)
13	O	Data output (0—space, 1—mark)
14	O	Clear leader (0—clear leader, 1—off clear leader)
15	I	Motion (0—go, 1—stop)
16	—	+5v logic power

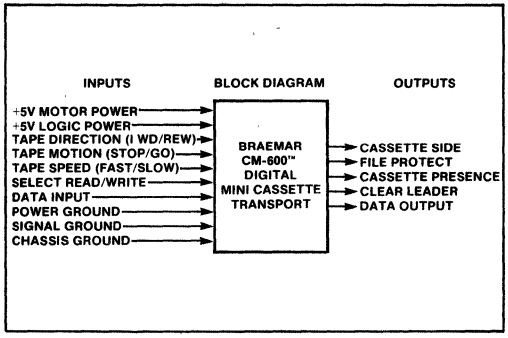


Figure 3. Braemar CM-600* Block Diagram

Figure 4 illustrates the encoding of the character 3AH assuming the previous data ended with the data line high. (The least significant bit is sent first.) Notice that there is always a "clocking" transition at the beginning of each cell. Decoding is simply a matter of triggering on this "clocking" transition, waiting 3/4 of a bit cell time, and determining whether a mid-cell transition has occurred. Cells with no mid-cell transitions are data 0's; cells with transitions are data 1's. This encoding technique has all the benefits of Manchester encoding with the added advantage that the encoded data may be "decoded by eyeball:" long cells are always 0's, short cells are always 1's.

Besides the encoding scheme, the data format is also up to the user. This controller uses a variable byte length, checksum protected block format. Every block starts and ends with a SYNC character

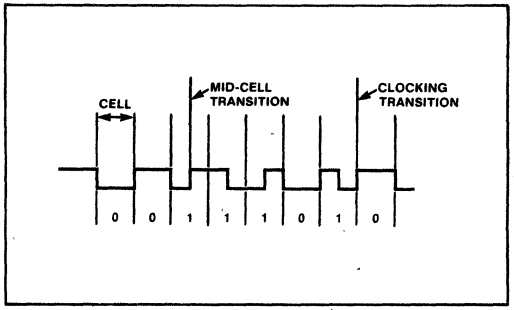


Figure 4. Modified Phase Encoding of Character 3A Hex

(AAH), and the character immediately preceeding the last SYNC is the checksum. The checksum is capable of catching 2 bit errors. The number of data characters within a block is limited to 64K bytes. Blocks are separated by an Inter-Record Gap (IRG). The IRG is of such a length that the transport can stop and start within an IRG, as illustrated in the data block timing, figure 5. Braemar specifies a maximum start or stop time of 150ms for the transport, thus the controller uses 450ms for the IRG. This gives plenty of margin for controlling the transport and also for detecting IRGs while skipping blocks.

THE UPI-41A™ CONTROLLER

The goal of the UPI software design for this application was to make the UPI-41A microcomputer into an intelligent cassette control processor. The host processor (8086, 8088, 8085A, etc.) simply issues a high-level command such as READ-a-block or WRITE-a-block. The 8741A accepts the command, performs the requested operation, and returns to the host system a result code telling the outcome of the operation, eg. Good-Completion, Sync Error, etc. Table 2 shows the command and result code repertoire. The 8741A completely manages all the data transfers for reading and writing.

As an example, consider the WRITE-a-block command. When this command is issued, the UPI-41A expects a 16-bit number from the host telling how many data bytes to write (up to 64K bytes per block). Once this number is supplied in the form of two bytes, the host is free to perform other tasks; a bit in the UPI's STATUS register or an interrupt output will notify the host when a data transfer is required. The 8741A then checks the transport's status to be sure that a cassette is present and not file protected. If either is false, a result code is

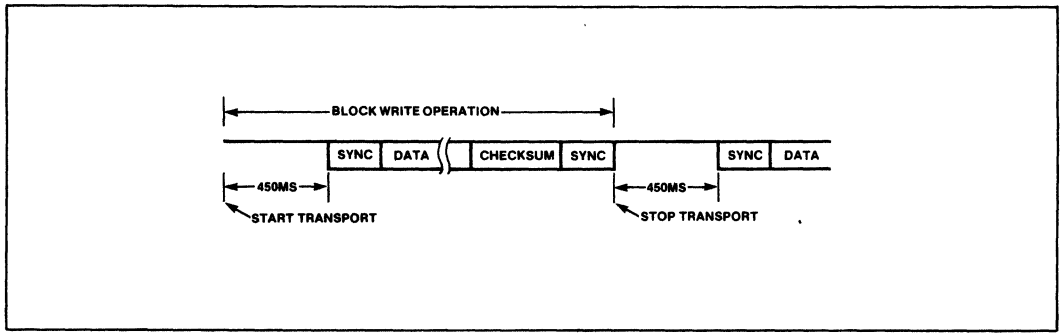


Figure 5. IRG/Block Timing Diagram (not to scale)

Table 2. Controller Command/Result Code Set

Command	Result
Read (01H)	Good-Completion (00H) Buffer Overrun Error (41H) Bad Synch1 Error (42H) Bad Synch2 Error (43H) Checksum Error (44H) Command Error (45H) End of Tape Error (46H)
Rewind (04H)	Good-Completion (00H)
Skip (03H)	Good-Completion (00H) End of Tape Error (47H) Beginning of Tape Error (48H)
Write (02H)	Good-Completion (00H) Buffer Underrun Error (81H) Command Error (82H) End of Tape Error (83H)

returned to the host; otherwise the transport is started. After the peripheral controller checks to make sure that the tape is off the clear leader and past the hole in the tape, it writes a 450ms IRG, a SYNC character, the block of data, the checksum, and the final SYNC character. (The tape has a clear leader at both ends and a small hole 6 inches from the end of each leader.) The data transfers from the host to the UPI-41A slave microcomputer are double buffered. The controller requests only the desired number of data bytes by keeping track of the count internally.

If nothing unusual happened, such as finding clear leader while writing, it returns a Good-Completion result code to the host. If clear leader was encountered, the transport is stopped immediately and an End-of-Tape result code is returned to the host. Another possible error would be if the host is late in supplying data. If this occurs, the controller writes an IRG, stops the drive, and returns the appropriate Data-Underrun result code.

The READ-a-block command also provides error checking. Once this command is issued by the host, the controller checks for cassette presence. If present, it starts the transport. The data output from the transport is then examined and decoded continuously. If the first character is not a SYNC, that's an error and the controller returns a Bad-First-SYNC result code (42H) after advancing to the next IRG. If the SYNC is good, the succeeding characters are read into an on-chip 30 character circular buffer. This continues until an IRG is encountered. When this occurs, the transport is stopped. The controller then tests that the last character. If it is a SYNC, the controller then compares the accumulated internal checksum to the block's checksum, the second to the last character of the block. If they match, a Good-Completion result code (00H) is returned to the host. If either test is bad, the appropriate error result code is returned. The READ command also checks for the End-of-Tape (EOT) clear leader and returns the appropriate error result code if it is found before the read operation is complete.

The 30 character circular buffer allows the host up to 30 character times of response time before the host must collect the data. All data transfers take place thru the UPI-41A Data Bus Buffer Output register (DBBOUT). The controller continually monitors the status of this register and moves characters from the circular buffer to the register whenever it is empty.

The SKIP-n-blocks command allows the host to skip the transport forward or backward up to 127 blocks. Once the command is issued, the controller expects one data byte specifying the number of

blocks to skip. The most significant bit of this byte selects the direction of the skip (0=forward, 1=reverse). SKIP is a dual-speed operation in the forward direction. If the number of blocks to skip is greater than 8, the controller uses fast-forward (5 IPS) until it is within 8 blocks of the desired location. Once within 8 blocks, the controller switches to the normal read speed (3 IPS) to allow accurate placement of the tape. The reverse skip uses only the rewind speed (15 IPS). Like the READ and WRITE commands, SKIP also checks for EOT and beginning-of-tape (BOT) depending upon the tape's direction. An error result code is returned if either is encountered before the number of blocks skipped is complete.

The REWIND command simply rewinds the tape to the BOT clear leader. The ABORT command allows the termination of any operation in progress, except a REWIND. All commands, including ABORT, always leave the tape positioned on an IRG.

THE HARDWARE INTERFACE

There's hardly any hardware design effort required for the controller and transport interface in figure 6. Since the CM-600 is TTL compatible, it connects directly to the I/O ports of the UPI controller. If the two are separated (i.e. on different PC cards), it is recommended that TTL buffers be provided.) The only external circuitry needed is an LED driver for the DRIVE ACTIVE status indicator.

The 8741A-to-host interface is equally straightforward. It has a standard asynchronous peripheral interface: 8 data lines (D_0-D_7), read (RD), write (WR), register select (AO), and chip select (CS). Thus it connects directly to an 8086, 8088, 8085A, 8080, or 8048 bus structure. Two interrupt outputs are provided for data transfer requests if the particular system is interrupt-driven. DMA transfer capability is also available. The clock input can be driven from a crystal directly or with the system clock (6MHz max). The UPI-41A clock may be asynchronous with respect to other clocks within the system.

This application was developed on an Intel iSBC 80/30 single board computer. The iSBC 80/30 is controlled by an 8085A microprocessor, contains 16K bytes of dual-ported dynamic RAM and up to 8K bytes of either EPROM or ROM. Its I/O complement consists of an 8255A Programmable Parallel Interface, an 8251A Programmable Communica-

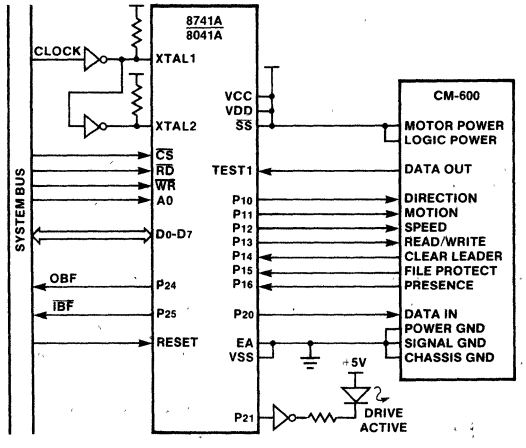


Figure 6. Controller/Transport System Schematic

tions Interface, an 8253 Programmable Interval Timer, and an 8259A Programmable Interrupt Controller. The iSBC 80/30 is especially convenient for UPI development since it contains an uncommitted socket dedicated to either an 8041A or 8741A, complete with buffering for its I/O ports. The iSBC 80/30 to 8741A interface is reflected in figure 8. (Optionally, an iSBC 569 Digital Controller board could be used. The iSBC 569 board contains three uncommitted UPI sockets with an interface similar to that in figure 8.)

Looking at the host-to-controller interface, the host sees the 8741A as three registers in the host's I/O address space: the data register, the command register, and the status register. The decoding of these registers is shown in figure 7. All data and commands for the controller are written into the Data Bus Buffer Input register (DBBIN). The state of the register select input, A0, determines whether a command or data is written. (Writes with A0 set to 1 are commands by convention.) All data and results from the controller are read by the host from the Data Bus Buffer Output register (DBBOUT).

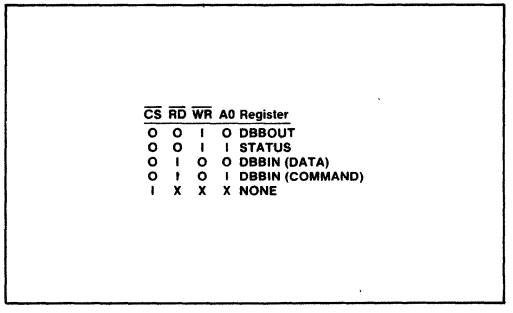


Figure 7. 8741A/8041A Interface Register Decoding

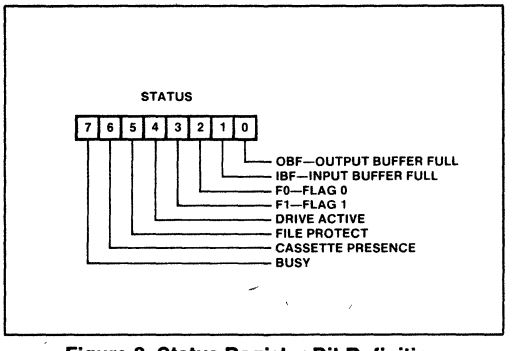


Figure 8. Status Register Bit Definition

The Status register contains flags which give the host the status of various operations within the controller. Its format is given in figure 8. The Input Buffer Full (IBF) and Output Buffer Full (OBF) flags show the Status of the DBBIN and DBBOUT registers respectively. IBF indicates when the DBBIN register contains data written by the host. The host may write to DBBIN only when IBF is 0. Likewise, the host may read DBBOUT only when OBF is set to a 1. These bits are handled automatically by the UPI-41A internal hardware. FLAG 0 (F_0) and FLAG 1 (F_1) are general purpose flags used internally by the controller which have no meaning externally.

The remaining four bits are user-definable. For this application they are DRIVE ACTIVE, FILE PROTECT, CASSETTE PRESENCE, and BUSY flags. The FILE PROTECT and CASSETTE PRESENCE flags reflect the state of the corresponding I/O lines from the transport. DRIVE ACTIVE is set whenever the transport motor is on and the controller is performing an operation. The BUSY flag indicates whether the contents of the DBBOUT register is data or a result code. The BUSY flag is set whenever a command is issued by the host and accepted by the controller. As long as BUSY is set, any character found in DBBOUT is a result code. Thus whenever the host finds OBF set, it should test the BUSY flag to determine whether the character is data or a result code.

Notice the OBF and $\overline{\text{IBF}}$ are available as interrupt outputs to the host processor, figure 6. These outputs are self-clearing, that is, OBF is set automatically upon the controller loading DBBOUT and cleared automatically by the host reading DBBOUT. Likewise $\overline{\text{IBF}}$ is cleared to a 0 by the host writing into DBBIN: set to a 1 when the controller reads DBBIN into the accumulator.

The flow charts of figure 9 show the flow of sample host software assuming a polling software interface between the host and the controller. The WRITE command requires two additional count bytes which form the 16-bit byte count. These extra bytes are "handshaked" into the controller using the IBF flag in the STATUS register. Once these bytes are written, the host writes data in response to IBF being cleared. This continues until the host finds OBF set indicating that the operation is complete and reads the result code from DBBOUT. No testing of BUSY is needed since only the result code appears in the DBBOUT register.

The READ command does require that BUSY be tested. Once the READ command is written into the

controller, the host must test BUSY whenever OBF is set to determine whether the contents of DBBOUT is data from the tape or the result code.

THE CONTROLLER SOFTWARE

The UPI-41A software to control the cassette can be divided up into various commands such as WRITE, READ and ABORT. In a previous version of this application note (May 1980), software was described that implemented these commands. This code however did not adequately compensate for speed variations of the motor during record and playback nor for data distortion caused by the magnetic media. Since then, new code has been written to include these effects. This revised software is now available through the INTEL User's Library, INSITE. For more information on this software or INSITE, contact your local INTEL Sales Office.

8041A/8641A/8741A
UNIVERSAL PERIPHERAL INTERFACE
8-BIT MICROCOMPUTER

- **8-Bit CPU plus ROM, RAM, I/O, Timer and Clock in a Single Package**
- **One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface**
- **DMA, Interrupt, or Polled Operation Supported**
- **1024 x 8 ROM/EPROM, 64 x 8 RAM, 8-Bit Timer/Counter, 18 Programmable I/O Pins**
- **Fully Compatible with All Microprocessor Families**
- **Interchangeable ROM and EPROM Versions**
- **3.6 MHz 8741A-8 Available**
- **Expandable I/O**
- **RAM Power-Down Capability**
- **Over 90 Instructions: 70% Single Byte**
- **Available in EXPRESS**
  - **—Standard Temperature Range**
  - **—Extended Temperature Range**

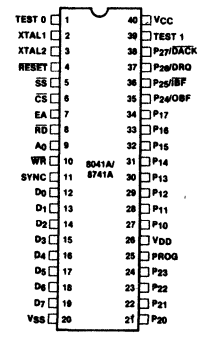
The Intel® 8041A/8741A is a general purpose, programmable interface device designed for use with a variety of 8-bit microprocessor systems. It contains a low cost microcomputer with program memory, data memory, 8-bit CPU, I/O ports, timer/counter, and clock in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in MCS-48™, MCS-80™, MCS-85™, MCS-86™, and other 8-bit systems.

The UPI-41A™ has 1K words of program memory and 64 words of data memory on-chip. To allow full user flexibility the program memory is available as ROM in the 8041A version or as UV-erasable EPROM in the 8741A version. The 8741A and the 8041A are fully pin compatible for easy transition from prototype to production level designs. The 8641A is a one-time programmable (at the factory) 8741A which can be ordered as the first 25 pieces of a new 8041A order. The substitution of 8641A's for 8041A's allows for very fast turnaround for initial code verification and evaluation results.

The device has two 8-bit, TTL compatible I/O ports and two test inputs. Individual port lines can function as either inputs or outputs under software control. I/O can be expanded with the 8243 device which is directly compatible and has 16 I/O lines. An 8-bit programmable timer/counter is included in the UPI device for generating timing sequences or counting external inputs. Additional UPI features include: single 5V supply, low power standby mode (in the 8041A), single-step mode for debug (in the 8741A), and dual working register banks.

Because it's a complete microcomputer, the UPI provides more flexibility for the designer than conventional LSI interface devices. It is designed to be an efficient controller as well as an arithmetic processor. Applications include keyboard scanning, printer control, display multiplexing and similar functions which involve interfacing peripheral devices to microprocessor systems.

PIN CONFIGURATION



BLOCK DIAGRAM

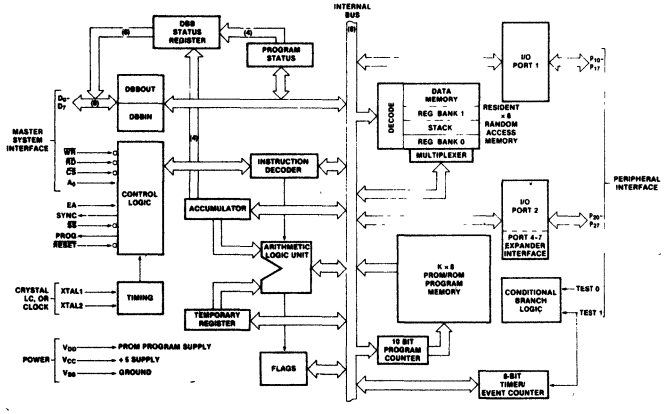


Table 1. Pin Description

Signal	Description
D_0–D_7 (BUS)	Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-41A to an 8-bit master system data bus.
P_{10}–P_{17}	8-bit, PORT 1 quasi-bidirectional I/O lines.
P_{20}–P_{27}	8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}–P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4–7 access. The upper 4 bits (P_{24}–P_{27}) can be programmed to provide Interrupt Request and DMA Handshake capability. Software control can configure P_{24} as OBF (Output Buffer Full), P_{25} as $\overline{IBF}$ (Input Buffer Full), P_{26} as DRQ (DMA Request), and P_{27} as $\overline{DACK}$ (DMA ACKnowledge).
$\overline{WR}$	I/O write input which enables the master CPU to write data and command words to the UPI-41A INPUT DATA BUS BUFFER.
$\overline{RD}$	I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
$\overline{CS}$	Chip select input used to select one UPI-41A out of several connected to a common data bus.
A_0	Address input used by the master processor to indicate whether byte transfer is data or command. During a write operation flag F_1 is set to the status of the A_0 input.
TEST 0, TEST 1	Input pins which can be directly tested using conditional branch instructions. T_1 also functions as the event timer input (under software control). T_0 is used during PROM programming and verification in the 8741A.

Signal	Description
XTAL1, XTAL2	Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
SYNC	Output signal which occurs once per UPI-41A instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
EA	External access input which allows emulation, testing and PROM/ROM verification.
PROG	Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243.
$\overline{RESET}$	Input used to reset status flip-flops and to set the program counter to zero. $\overline{RESET}$ is also used during PROM programming and verification. $\overline{RESET}$ should be held low for a minimum of 8 instruction cycles after power-up.
$\overline{SS}$	Single step input used in the 8741A in conjunction with the SYNC output to step the program through each instruction.
V_{CC}	+5V main power supply pin.
V_{DD}	+5V during normal operation. +25V during programming operation. Low power standby pin in ROM version.
V_{SS}	Circuit ground potential.

PROGRAMMING, VERIFYING, AND ERASING THE 8741A EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input (1 to 6MHz)
$\overline{\text{Reset}}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:

An attempt to program a missocketed 8741A will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. A_0 = 0V, $\overline{CS}$ = 5V, EA = 5V, $\overline{RESET}$ = 0V, TEST0 = 5V, V_{DD} = 5V, clock applied or internal oscillator operating, BUS and PROG floating.
2. Insert 8741A in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 23V (activate program mode)
5. Address applied to BUS and P20-1
6. $\overline{RESET}$ = 5v (latch address)
7. Data applied to BUS
8. V_{DD} = 25v (programming power)
9. PROG = 0v followed by one 50ms pulse to 23V
10. V_{DD} = 5v
11. TEST 0 = 5v (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0v
14. $\overline{RESET}$ = 0v and repeat from step 5
15. Programmer should be at conditions of step 1 when 8741A is removed from socket.

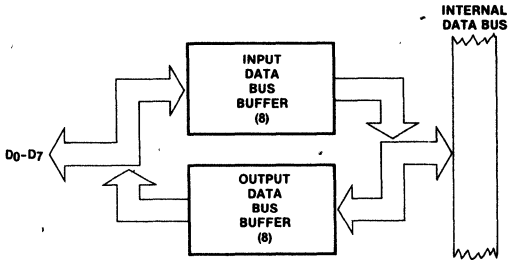
8741A Erasure Characteristics

The erasure characteristics of the 8741A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8741A in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8741A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8741A window to prevent unintentional erasure.

The recommended erasure procedure for the 8741A is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity x exposure time) for erasure should be a minimum of 15 w-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8741A should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

UPI-41A™ FEATURES AND ENHANCEMENTS

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.



2. 8 Bits of Status

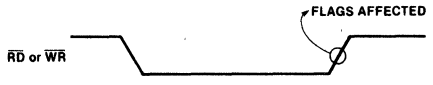
ST_7	ST_6	ST_5	ST_4	F_1	F_0	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

ST_4-ST_7 are user definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4-7 of the accumulator are moved to bits 4-7 of the status register. Bits 0-3 of the status register are not affected.

MOV STS, A Op Code: 90H

1	0	0	1	0	0	0	0
D_7							D_0

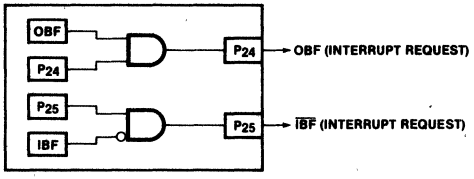
3. $\overline{RD}$ and $\overline{WR}$ are edge triggered. IBF, OBF, F_1 and INT change internally after the trailing edge of $\overline{RD}$ or $\overline{WR}$.



4. P_{24} and P_{25} are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

   If the "EN FLAGS" instruction has been executed, P_{24} becomes the OBF (Output Buffer Full) pin. A "1" written to P_{24} enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P_{24} disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI-41A (in Output Data Bus Buffer).

   If "EN FLAGS" has been executed, P_{25} becomes the $\overline{IBF}$ (Input Buffer Full) pin. A "1" written to P_{25} enables the $\overline{IBF}$ pin (the pin outputs the inverse of the IBF Status Bit). A "0" written to P_{25} disables the $\overline{IBF}$ pin (the pin remains low). This pin can be used to indicate that the UPI-41A is ready for data.



DATA BUS BUFFER INTERRUPT CAPABILITY

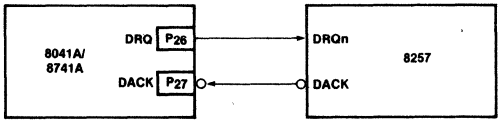
EN FLAGS Op Code: 0F5H

1	1	1	1	0	1	0	1
D_7							D_0

5. P_{26} and P_{27} are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

   If the "EN DMA" instruction has been executed, P_{26} becomes the DRQ (DMA ReQuest) pin. A "1" written to P_{26} causes a DMA request (DRQ is activated). DRQ is deactivated by DACK·RD, DACK·WR, or execution of the "EN DMA" instruction.

   If "EN DMA" has been executed, P_{27} becomes the $\overline{DACK}$ (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.



DMA HANDSHAKE CAPABILITY

EN DMA Op Code: 0E5H

1	1	1	0	0	1	0	1
D_1							D_0

APPLICATIONS

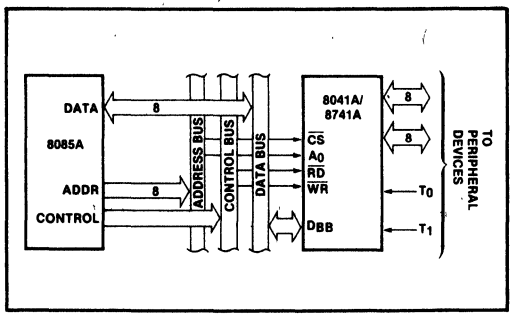


Figure 1. 8085A-8041A Interface

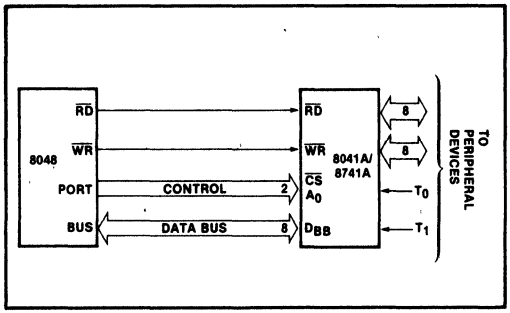


Figure 2. 8048-8041A Interface

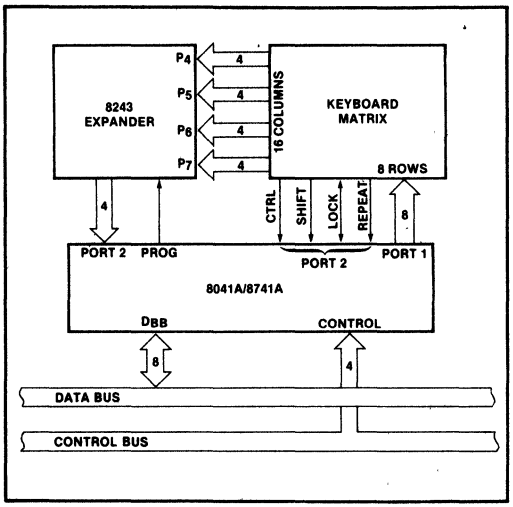


Figure 3. 8041A-8243 Keyboad Scanner

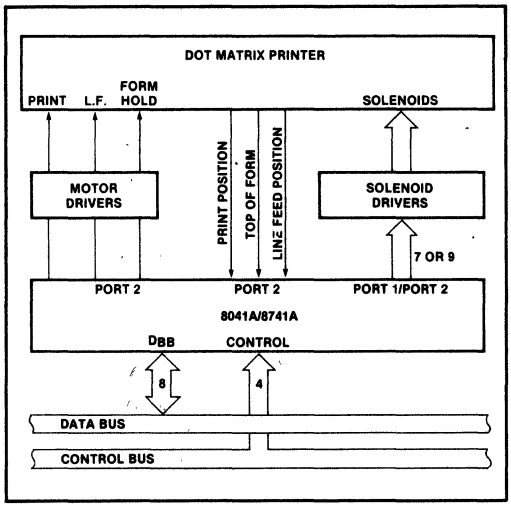


Figure 4. 8041A Matrix Printer Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin With Respect to Ground . . . . . 0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

*COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS

T_A=0°C to 70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V ±10%*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage (Except XTAL1, XTAL2, $\overline{RESET}$)	−0.5	0.8	V	
V_{IL1}	Input Low Voltage (XTAL1, XTAL2, $\overline{RESET}$)	−0.5	0.6	V	
V_{IH}	Input High Voltage (Except XTAL1, XTAL2, $\overline{RESET}$)	2.2	V_{CC}		
V_{IH1}	Input High Voltage (XTAL1, XTAL2, $\overline{RESET}$)	3.8	V_{CC}	V	
V_{OL}	Output Low Voltage (D_0-D_7)		0.45	V	I_{OL}=2.0 mA
V_{OL1}	Output Low Voltage ($P_{10}P_{17}$, $P_{20}P_{27}$, Sync)		0.45	V	I_{OL}=1.6 mA
V_{OL2}	Output Low Voltage (Prog)		0.45	V	I_{OL}=1.0 mA
V_{OH}	Output High Voltage (D_0-D_7)	2.4		V	I_{OH}=−400 μA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4		V	I_{OH}=−50 μA
I_{IL}	Input Leakage Current (T_0, T_1, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_0, EA)		±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{OZ}	Output Leakage Current (D_0-D_7, High Z State)		±10	μA	$V_{SS}+0.45 \le V_{IN} \le V_{CC}$
I_{LI}	Low Input Load Current ($P_{10}P_{17}$, $P_{20}P_{27}$)		0.5	mA	V_{IL}=0.8V
I_{LI1}	Low Input Load Current ($\overline{RESET}$, $\overline{SS}$)		0.2	mA	V_{IL}=0.8V
I_{DD}	V_{DD} Supply Current		15	mA	Typical =5 mA
$I_{CC}+I_{DD}$	Total Supply Current		125	mA	Typical=60 mA

A.C. CHARACTERISTICS

T_A=0°C to 70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V ±10%*

DBB READ

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		225	ns	C_L=150 pF
t_{RD}	$\overline{RD}$↓ to Data Out Delay		225	ns	C_L=150 pF
t_{DF}	$\overline{RD}$↑ to Data Float Delay		100	ns	
t_{CY}	Cycle Time (Except 8741A-8)	2.5	15	μs	6.0 MHz XTAL
t_{CY}	Cycle Time (8741A-8)	4.17	15	μs	3.6 MHz XTAL

DBB WRITE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	$\overline{CS}$, A_0 Setup to $\overline{WR}$↓	0		ns	
t_{WA}	$\overline{CS}$, A_0 Hold After $\overline{WR}$↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$↑	150		ns	
t_{WD}	Data Hold After $\overline{WR}$↑	0		ns	

A.C. TIMING SPECIFICATION FOR PROGRAMMING

T_A=0°C to 70°C, V_{CC}=+5V ±10% *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{RESET}$ ↑	4tcy			
t_{WA}	Address Hold Time After $\overline{RESET}$ ↑	4tcy			
t_{DW}	Data in Setup Time to PROG ↑	4tcy			
t_{WD}	Data in Hold Time After PROG ↓	4tcy			
t_{PH}	$\overline{RESET}$ Hold Time to Verify	4tcy			
t_{VDDW}	V_{DD} Setup Time to PROG ↑	4tcy			
t_{VDDH}	V_{DD} Hold Time After PROG ↓	0			
t_{PW}	Program Pulse Width	50	60	mS	
t_{TW}	Test 0 Setup Time for Program Mode	4tcy			
t_{WT}	Test 0 Hold Time After Program Mode	4tcy			
t_{DO}	Test 0 to Data Out Delay		4tcy		
t_{WW}	$\overline{RESET}$ Pulse Width to Latch Address	4tcy			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0 5	2 0	μs	
t_{CY}	CPU Operation Cycle Time	5.0		μs	
t_{RE}	$\overline{RESET}$ Setup Time Before EA ↑.	4tcy			

Note: If TEST 0 is high, t_{DO} can be triggered by $\overline{RESET}$ ↑.

* For Extended Temperature EXPRESS, use M8741A electrical parameters.

D.C. SPECIFICATION FOR PROGRAMMING

T_A = 25°C ±5°C, V_{CC} = 5V ±5%, V_{DD} = 25V ±1V

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{DDH}	V_{DD} Program Voltage High Level	24.0	26 0	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	21.5	24.5	V	
V_{PL}	PROG Voltage Low Level		0.2	V	
V_{EAH}	EA Program or Verify Voltage High Level	21.5	24.5	V	
V_{EAL}	EA Voltage Low Level		5.25	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		16.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

A.C. CHARACTERISTICS—PORT 2

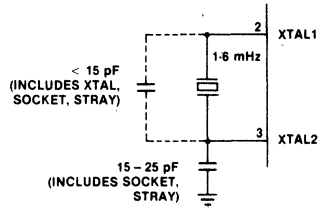
T_A=0°C to 70°C, : V_{CC}= +5V ±10%

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CP}	Port Control Setup Before Falling Edge of PROG	110		ns	
t_{PC}	Port Control Hold After Falling Edge of PROG	100		ns	
t_{PR}	PROG to Time P2 Input Must Be Valid		810	ns	
t_{PF}	Input Data Hold Time	0	150	ns	
t_{DP}	Output Data Setup Time	250		ns	
t_{PD}	Output Data Hold Time	65		ns	
t_{PP}	PROG Pulse Width	1200		ns	

A.C. CHARACTERISTICS—DMA

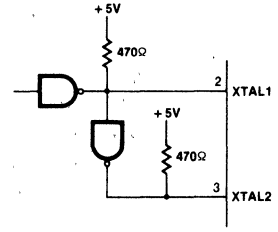
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{ACC}	$\overline{DACK}$ to $\overline{WR}$ or $\overline{RD}$	0		ns	
t_{CAC}	$\overline{RD}$ or $\overline{WR}$ to $\overline{DACK}$	0		ns	
t_{ACD}	$\overline{DACK}$ to Data Valid		225	ns	$C_L = 150$ pF
t_{CRQ}	$\overline{RD}$ or $\overline{WR}$ to DRQ Cleared		200	ns	

CRYSTAL OSCILLATOR MODE



CRYSTAL SERIES RESISTANCE SHOULD BE <75Ω AT 6 MHz; <180Ω AT 3.6 MHz.

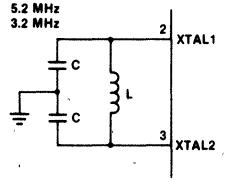
DRIVING FROM EXTERNAL SOURCE



BOTH XTAL1 AND XTAL2 SHOULD BE DRIVEN RESISTORS TO V_{CC} ARE NEEDED TO ENSURE $V_{IH} = 3.8V$ IF TTL CIRCUITRY IS USED.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz



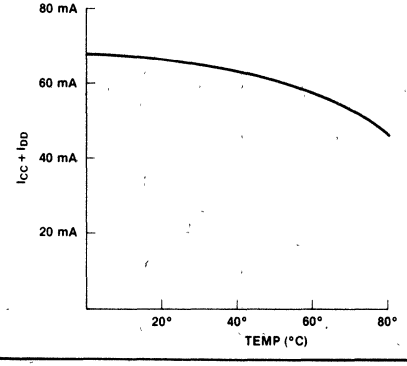
$$f \approx \frac{1}{2\pi\sqrt{LC'}}$$

$$C' = \frac{C + 3C_{PP}}{2}$$

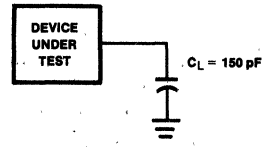
$C_{PP} \approx 5 - 10$ pF PIN-TO-PIN CAPACITANCE

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE.

TYPICAL 8041/8741A CURRENT

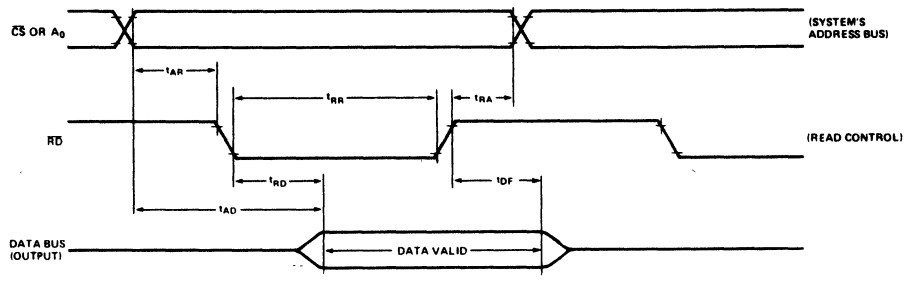


A.C. TESTING LOAD CIRCUIT

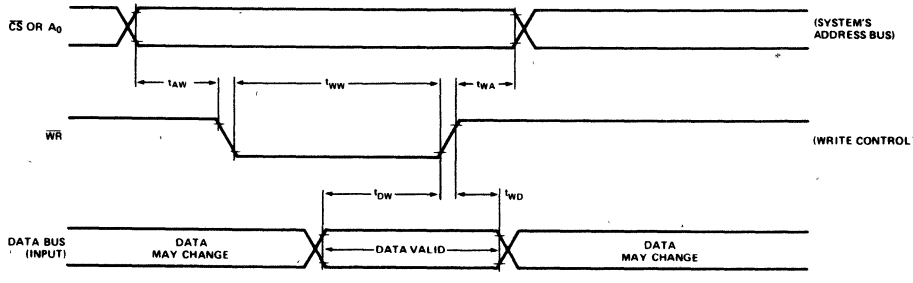


WAVEFORMS

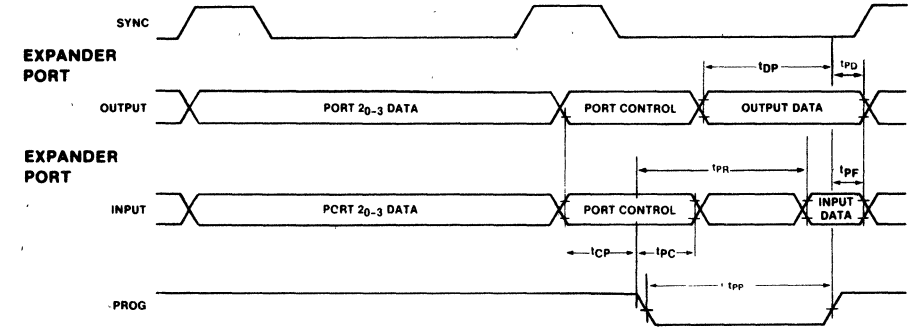
READ OPERATION—DATA BUS BUFFER REGISTER.



WRITE OPERATION—DATA BUS BUFFER REGISTER.

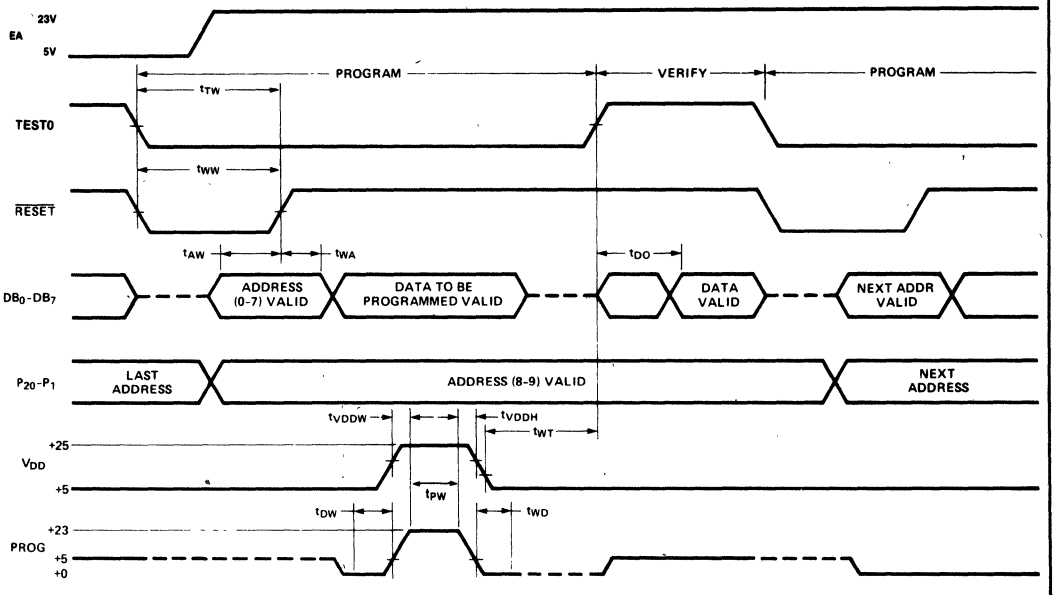


PORT 2 TIMING

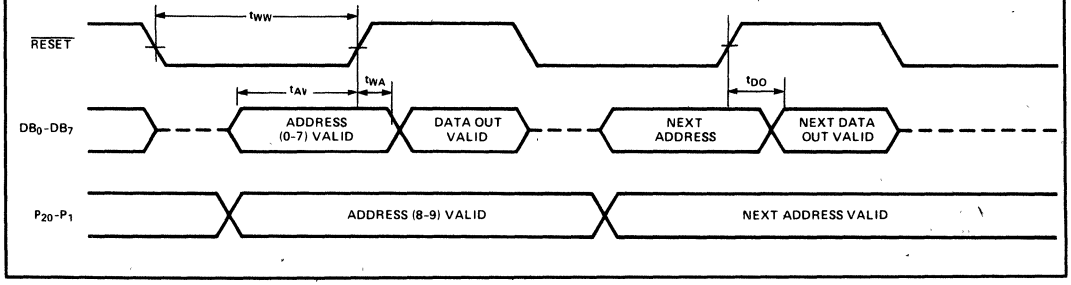


WAVEFORMS FOR PROGRAMMING

COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



VERIFY MODE (ROM/EPROM)

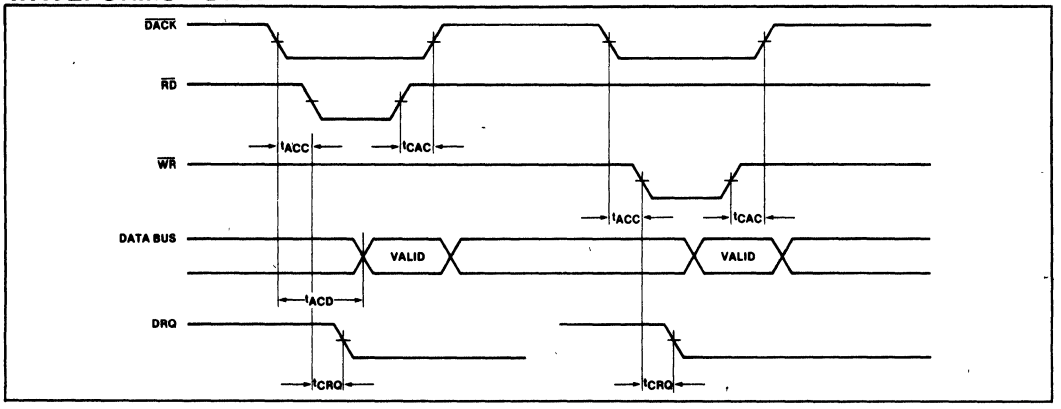


NOTES:
1. **PROG MUST FLOAT IF EA IS LOW (i.e., ≠23V), OR IF T0 = 5V FOR THE 8741A. FOR THE 8041A PROG MUST ALWAYS FLOAT.**
2. **XTAL1 AND XTAL 2 DRIVEN BY 3.6 MHz CLOCK WILL GIVE 4.17 μsec t_{CY}. THIS IS ACCEPTABLE FOR 8741A-8 PARTS AS WELL AS STANDARD PARTS.**
3. **AO MUST BE HELD LOW (i.e., = 0V) DURING PROGRAM/VERIFY MODES.**

The 8741A EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

WAVEFORMS—DMA



INPUT AND OUTPUT WAVEFORMS FOR A.C. TESTS

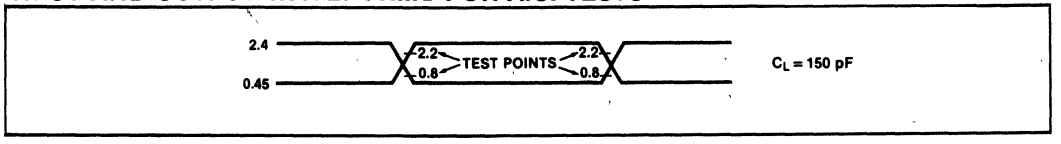


Table 2. UPI™ Instruction Set

Mnemonic	Description	Bytes	Cycles
Accumulator			
ADD A,Rr	Add register to A	1	1
ADD A,@Rr	Add data memory to A	1	1
ADD A,#data	Add immediate to A	2	2
ADDC A,Rr	Add register to A with carry	1	1
ADDC A,@Rr	Add data memory to A with carry	1	1
ADDC A, #data	Add immed. to A with carry	2	2
ANL A,Rr	AND register to A	1	1
ANL A,@Rr	AND data memory to A	1	1
ANL A,#data	AND immediate to A	2	2
ORL A,Rr	OR register to A	1	1
ORL A,@Rr	OR data memory to A	1	1
ORL A,#data	OR immediate to A	2	2
XRL A,Rr	Exclusive OR register to A	1	1
XRL A,@Rr	Exclusive OR data memory to A	1	1
XRL A,#data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RCL A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Table 2. UPI™ Instruction Set (Cont'd.)

Mnemonic	Description	Bytes	Cycles
Input/Output			
In A,Pp	Input port to A	1	2
OUTL Pp,A	Output A to port	1	2
ANL Pp,#data	AND immediate to port	2	2
ORL Pp,#data	OR immediate to port	2	2
In A,DBB	Input DBB to A, clear IBF	1	1
OUT DBB,A	Output A to DBB, set OBF	1	1
MOV STS,A	A_4-A_7 to Bits 4-7 of Status	1	1
MOVD A,Pp	Input Expander port to A	1	2
MOVD Pp,A	Output A to Expander port	1	2
ANLD Pp,A	AND A to Expander	1	2
ORLD Pp,A	OR A to Expander port	1	2
Data Moves			
MOV A,Rr	Move register to A	1	1
MOV A,@Rr	Move data memory to A	1	1
MOV A,#data	Move immediate to A	2	2
MOV Rr,A	Move A to register	1	1
MOV @Rr,A	Move A to data memory	1	1
MOV Rr,#data	Move immediate to register	2	2
MOV @Rr, #data	Move immediate to data memory	2	2
MOV A,PSW	Move PSW to A	1	1
MOV PSW,A	Move A to PSW	1	1
XCH A,Rr	Exchange A and register	1	1
XCH A,@Rr	Exchange A and data memory	1	1
XCHD A,@Rr	Exchange digit of A and register	1	1
MOVP A,@A	Move to A from current page	1	2
MOVP3, A,@A	Move to A from page 3	1	2
Timer/Counter			
MOV A,T	Read Timer/Counter	1	1
MOV T,A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter	1	1
DIS TCNTI	Disable Timer/ Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycles
Control			
EN DMA	Enable DMA Hand-shake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
Registers			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ Rr,addr	Decrement register and jump	2	2
JC addr	Jump on Carry=1	2	2
JNC addr	Jump on Carry=0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0=1	2	2
JNT0 addr	Jump on T0=0	2	2
JT1 addr	Jump on T1=1	2	2
JNT1 addr	Jump on T1=0	2	2
JF0 addr	Jump on F0 Flag=1	2	2
JF1 addr	Jump on F1 Flag=1	2	2
JTF addr	Jump on Timer Flag=1, Clear Flag	2	2
JN1BF addr	Jump on IBF Flag=0	2	2
JOBF addr	Jump on OBF Flag=1	2	2
JBb addr	Jump on Accumulator Bit	2	2

PRELIMINARY

8042/8742
UNIVERSAL PERIPHERAL INTERFACE 8-BIT MICROCOMPUTER

- 8042/8742: 12 MHz
- Pin, Software and Architecturally Compatible with 8041A/8741A
- 8-Bit CPU plus ROM, RAM, I/O, Timer and Clock in a Single Package
- 2048 x 8 ROM/EPROM, 128 x 8 RAM, 8-Bit Timer/Counter, 18 Programmable I/O Pins
- One 8-Bit Status and Two Data Registers for Asynchronous Slave-to-Master Interface
- DMA, Interrupt, or Polled Operation Supported
- Fully Compatible with all Intel and Most Other Microprocessor Families
- Interchangeable ROM and EPROM Versions
- Expandable I/O
- RAM Power-Down Capability
- Over 90 Instructions: 70% Single Byte
- Available in EXPRESS —Standard Temperature Range

The Intel 8042/8742 is a general-purpose Universal Peripheral Interface that allows the designer to grow his own customized solution for peripheral device control. It contains a low-cost microcomputer with 2K of program memory, 128 bytes of data memory, 8-bit CPU, I/O ports, 8-bit timer/counter, and clock generator in a single 40-pin package. Interface registers are included to enable the UPI device to function as a peripheral controller in the MCS-48™, MCS-51™, MCS-80™, MCS-85™, iAPX-88, iAPX-86 and other 8-, 16-bit systems.

The 8042/8742 is software, pin, and architecturally compatible with the 8041A, 8741A. The 8042/8742 doubles the on-chip memory space to allow for additional features and performance to be incorporated in upgraded 8041A/8741A designs. For new designs, the additional memory and performance of the 8042/8742 extends the UPI concept to more complex motor control tasks, 80-column printers and process control applications as examples.

To allow full user flexibility, the program memory is available as ROM in the 8042 version or as UV-erasable EPROM in the 8742 version. The 8742 and the 8042 are fully pin compatible for easy transition from prototype to production level designs.

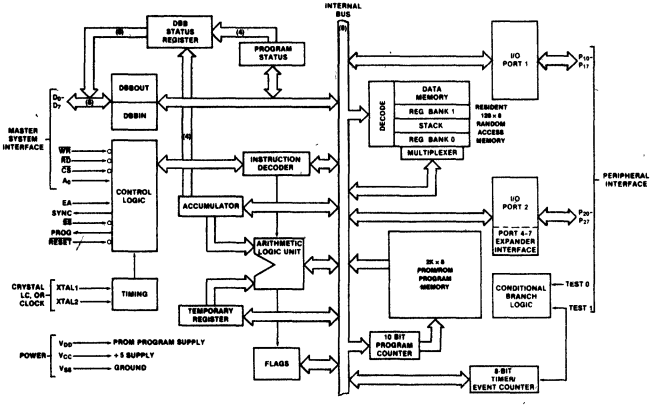


Figure 1. Block Diagram

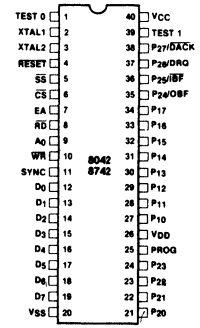


Figure 2. Pin Configuration

FEBRUARY 1983
ORDER NUMBER: 210393-001

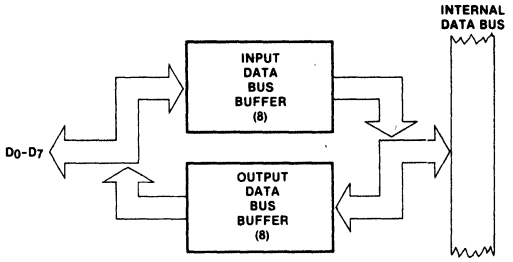
Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
TEST 0, TEST 1	1 39	I	**Test Inputs:** Input pins which can be directly tested using conditional branch instructions. **Frequency Reference:** TEST 1 (T_1) also functions as the event timer input (under software control). TEST 0 (T_0) is used during PROM programming and verification in the 8742.
XTAL 1, XTAL 2	2 3	I	**Inputs:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
$\overline{\text{RESET}}$	4	I	**Reset:** Input used to reset status flip-flops and to set the program counter to zero. $\overline{\text{RESET}}$ is also used during PROM programming and verification.
$\overline{\text{SS}}$	5	I	**Single Step:** Single step input used in conjunction with the SYNC output to step the program through each instruction. (8742 only)
$\overline{\text{CS}}$	6	I	**Chip Select:** Chip select input used to select one UPI microcomputer out of several connected to a common data bus.
EA	7	I	**External Access:** External access input which allows emulation, testing and PROM/ROM verification. This pin should be tied low if unused.
$\overline{\text{RD}}$	8	I	**Read:** I/O read input which enables the master CPU to read data and status words from the OUTPUT DATA BUS BUFFER or status register.
A_0	9	I	**Command/Data Select:** Address input used by the master processor to indicate whether byte transfer is data ($A_0 = 0$, F1 is reset) or command ($A_0 = 1$, F1 is set).
$\overline{\text{WR}}$	10	I	**Write:** I/O write input which enables the master CPU to write data and command words to the UPI INPUT DATA BUS BUFFER.

Symbol	Pin No.	Type	Name and Function
SYNC	11	O	**Output Clock:** Output signal which occurs once per UPI-42 instruction cycle. SYNC can be used as a strobe for external circuitry; it is also used to synchronize single step operation.
D_0-D_7 (BUS)	12-19	I/O	**Data Bus:** Three-state, bidirectional DATA BUS BUFFER lines used to interface the UPI-42 microcomputer to an 8-bit master system data bus.
P_{10}-P_{17}	27-34	I/O	**Port 1:** 8-bit, PORT 1 quasi-bidirectional I/O lines.
P_{20}-P_{27}	21-24 35-38	I/O	**Port 2:** 8-bit, PORT 2 quasi-bidirectional I/O lines. The lower 4 bits (P_{20}-P_{23}) interface directly to the 8243 I/O expander device and contain address and data information during PORT 4-7 access. The upper 4 bits (P_{24}-P_{27}) can be programmed to provide interrupt Request and DMA Handshake capability. Software control can configure P_{24} as Output Buffer Full (OBF) interrupt, P_{25} as Input Buffer Full ($\overline{\text{IBF}}$) interrupt, P_{26} as DMA Request (DRQ), and P_{27} as DMA ACKnowledge ($\overline{\text{DACK}}$).
PROG	25	I/O	**Program:** Multifunction pin used as the program pulse input during PROM programming. During I/O expander access the PROG pin acts as an address/data strobe to the 8243. This pin should be tied high if unused.
V_{CC}	40		**Power:** +5V main power supply pin.
V_{DD}	26		**Power:** +5V during normal operation. +21V during programming operation. Low power standby pin in ROM version.
V_{SS}	20		**Ground:** Circuit ground potential.

UPI-42 FEATURES

1. Two Data Bus Buffers, one for input and one for output. This allows a much cleaner Master/Slave protocol.



2. 8 Bits of Status

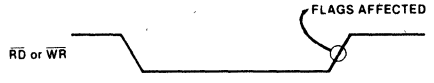
ST_7	ST_6	ST_5	ST_4	F_1	F_0	IBF	OBF
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

ST_4-ST_7 are user definable status bits. These bits are defined by the "MOV STS, A" single byte, single cycle instruction. Bits 4-7 of the accumulator are moved to bits 4-7 of the status register. Bits 0-3 of the status register are not affected.

MOV STS, A Op Code 90H

1	0	0	1	0	0	0	0
D_7							D_0

3. $\overline{RD}$ and $\overline{WR}$ are edge triggered. IBF, OBF, F_1 and INT change internally after the trailing edge of $\overline{RD}$ or $\overline{WR}$.

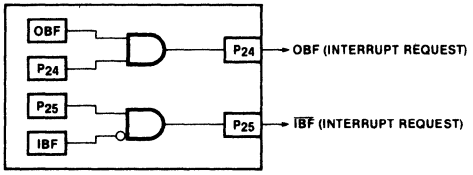


During the time that the host CPU is reading the status register, the 8042/8742 is prevented from updating this register or is 'locked out.'

4. P_{24} and P_{25} are port pins or Buffer Flag pins which can be used to interrupt a master processor. These pins default to port pins on Reset.

   If the "EN FLAGS" instruction has been executed, P_{24} becomes the OBF (Output Buffer Full) pin. A "1" written to P_{24} enables the OBF pin (the pin outputs the OBF Status Bit). A "0" written to P_{24} disables the OBF pin (the pin remains low). This pin can be used to indicate that valid data is available from the UPI-41A (in Output Data Bus Buffer).

   If "EN FLAGS" has been executed, P_{25} becomes the $\overline{IBF}$ (Input Buffer Full) pin. A "1" written to P_{25} enables the $\overline{IBF}$ pin (the pin outputs the inverse of the IBF Status Bit). A "0" written to P_{25} disables the $\overline{IBF}$ pin (the pin remains low). This pin can be used to indicate that the UPI-42 is ready for data.



DATA BUS BUFFER INTERRUPT CAPABILITY

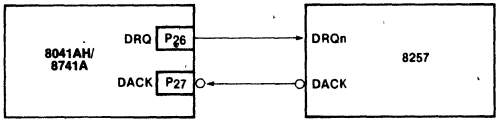
EN FLAGS Op Code: 0F5H

1	1	1	1	0	1	0	1
D_7							D_0

5. P_{26} and P_{27} are port pins or DMA handshake pins for use with a DMA controller. These pins default to port pins on Reset.

   If the "EN DMA" instruction has been executed, P_{26} becomes the DRQ (DMA ReQuest) pin. A "1" written to P_{26} causes a DMA request (DRQ is activated). DRQ is deactivated by DACK·RD, DACK·WR, or execution of the "EN DMA" instruction.

   If "EN DMA" has been executed, P_{27} becomes the $\overline{DACK}$ (DMA ACKnowledge) pin. This pin acts as a chip select input for the Data Bus Buffer registers during DMA transfers.



DMA HANDSHAKE CAPABILITY

EN DMA Op Code: 0E5H

1	1	1	0	0	1	0	1
D_1							D_0

6. The RESET input on the 8042/8742 includes a 2-stage synchronizer to support reliable reset operation for 12 MHz operation.

7. When EA is enabled on the 8042/8742, the program counter is placed on Port 1 and the lower three bits of Port 2 (MSB = P_{22}, LSB = P_{10}). On the 8042/8742 this information is multiplexed with PORT DATA (see port timing diagrams at end of this data sheet).

APPLICATIONS

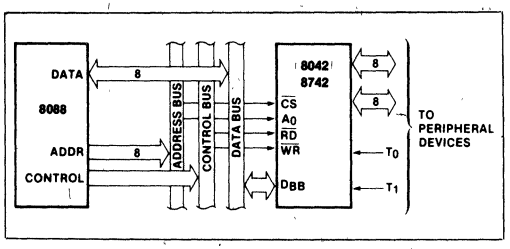


Figure 3. 8088-8042/8742 Interface

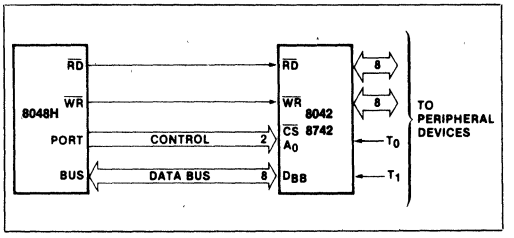


Figure 4. 8048H-8042/8742 Interface

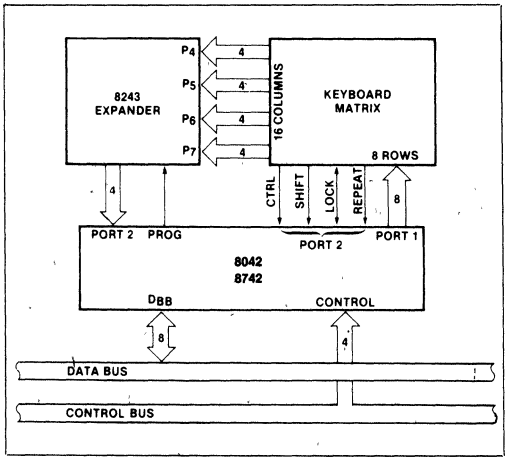


Figure 5. 8042/8742-8243 Keyboard Scanner

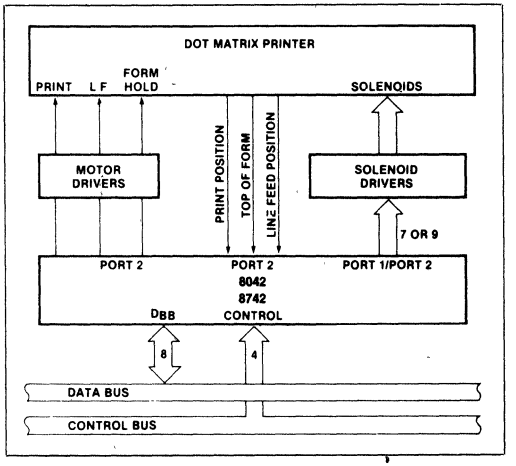


Figure 6. 8042/8742 80-Column Matrix Printer Interface

PROGRAMMING, VERIFYING, AND ERASING THE 8742 EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL 1	Clock Input
$\overline{\text{Reset}}$	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-12	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING

An attempt to program a missocketed 8742 will result in severe damage to the part. An indication of a properly socketed part is the appearance of the SYNC clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. A_0 = 0V, CS = 5V, EA = 5V, RESET = 0V, TESTO = 5V, V_{DD} = 5V, clock applied or internal oscillator operating, BUS floating, PROG = 5V.
2. Insert 8742 in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 18V (active program mode)*
5. Address applied to BUS and P_{20-22}
6. $\overline{\text{RESET}}$ = 5v (latch address)
7. Data applied to BUS**
8. V_{DD} = 21V (programming power)**
9. PROG = V_{CC} followed by one 50 ms pulse to 18V**
10. V_{DD} = 5v
11. TEST 0 = 5v (verify mode)

12. Read and verify data on BUS
13. TEST 0 = 0v
14. $\overline{RESET}$ = 0v and repeat from step 5
15. Programmer should be at conditions of step 1 when 8742 is removed from socket

*When verifying ROM, EA = 12V.
**Not used in verifying ROM procedure.

8742 Erasure Characteristics

The erasure characteristics of the 8742 are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8742 in approximately 3 years while it would take approximately one week to cause erasure when exposed to direct sunlight. If the 8742 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8742 window to prevent unintentional erasure.

The recommended erasure procedure for the 8742 is exposure to shortwave ultraviolet light which has a wavelength of 2537Å. The integrated dose (i.e., UV intensity x exposure time) for erasure should be a minimum of 15 w-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12,000 μW/cm^2 power rating. The 8742 should be placed within one inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin With Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0° to +70°C, V_{CC} = V_{DD} = +5V ± 10%)

Symbol	Parameter	8042/8742 Min.	8042/8742 Max.	Units	Notes
V_{IL}	Input Low Voltage (Except XTAL1, XTAL2, RESET)	−0.5	0.8	V	
V_{IL1}	Input Low Voltage (XTAL1, XTAL2, RESET)	−0.5	0.6	V	
V_{IH}	Input High Voltage (Except XTAL1, XTAL2, RESET)	2.0	V_{CC}	V	
V_{IH1}	Input High Voltage (XTAL1, RESET)	3.5	V_{CC}	V	
V_{IH2}	Input High Voltage (XTAL2)	2.2	V_{CC}	V	
V_{OL}	Output Low Voltage (D_0–D_7)		0.45	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage (P_{10}–P_{17}, P_{20}–P_{27}, Sync)		0.45	V	I_{OL} = 1.6 mA
V_{OL2}	Output Low Voltage (PROG)		0.45	V	I_{OL} = 1.0 mA
V_{OH}	Output High Voltage (D_0–D_7)	2.4		V	I_{OH} = −400 μA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			I_{OH} = −50 μA
I_{IL}	Input Leakage Current (T_0, T_1, RD, WR, CS, A_0, EA)		± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Output Leakage Current (D_0–D_7, High Z State)		± 10	μA	V_{SS} + 0.45 $\leq V_{OUT} \leq V_{CC}$
I_{LI}	Low Input Load Current (P_{10}–P_{17}, P_{20}–P_{27})		0.3	mA	V_{IL} = 0.8V
I_{LI1}	Low Input Load Current (RESET, SS)		0.2	mA	V_{IL} = 0.8V
I_{DD}	V_{DD} Supply Current		20	mA	Typical = 5 mA
I_{CC} + I_{DD}	Total Supply Current		135	mA	Typical = 60 mA
I_{IH}	Input Leakage Current (P_{10}–P_{17}, P_{20}–P_{27})		100	μA	V_{IN} = V_{CC}
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

D.C. CHARACTERISTICS—PROGRAMMING (T_A = 25°C ± 5°C, V_{CC} = 5V ± 5%, V_{DD} = 21V ± 0.5V

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{DOH}	V_{DD} Program Voltage High Level	20.5	21.5	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	17.5	18.5	V	
V_{PL}	PROG Voltage Low Level	V_{CC}−0.5	V_{CC}	V	
V_{EAH}	EA Program or Verify Voltage High Level	17.5	18.5	V	
V_{EAL}	EA Voltage Low Level		5.25	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		1.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

A.C. CHARACTERISTICS (T_A=0°C to +70°C, V_{SS}=0V, V_{CC}=V_{DD}=+5V ± 10%)

DBB READ

Symbol	Parameter	8042		8742		Units
		Min.	Max.	Min.	Max.	
t_{AR}	CS, A_0 Setup to RD↓	0		0		ns
t_{RA}	CS, A_0 Hold After RD↑	0		0		ns
t_{RR}	RD Pulse Width	160		160		ns
t_{AD}	CS, A_0 to Data Out Delay		130		130	ns
t_{RD}	RD↓ to Data Out Delay		130		130	ns
t_{DF}	RD↑ to Data Float Delay		85		85	ns

DBB WRITE

Symbol	Parameter	Min.	Max.	Min.	Max.	Units
t_{AW}	CS, A_0 Setup to WR↓	0		0		ns
t_{WA}	CS, A_0 Hold After WR↑	0		0		ns
t_{WW}	WR Pulse Width	160		160		ns
t_{DW}	Data Setup to WR↑	130		130		ns
t_{WD}	Data Hold After WR↑	0		0		ns

CLOCK

Symbol	Parameter	8042		8742		Units
		Min.	Max.	Min.	Max.	
t_{CY}	Cycle Time	1 25	9.20	1.25	9.20	$\mu s^{[1]}$
t_{CYC}	Clock Period	83 3	613	83.3	613	ns
t_{PWH}	Clock High Time	33		38		ns
t_{PWL}	Clock Low Time	33		38		ns
t_R	Clock Rise Time		10		10	ns
t_F	Clock Fall Time		10		10	ns

NOTE:
1. t_{CY} = 15/f(XTAL)

A.C. CHARACTERISTICS (T_A=25°C±5°C, V_{CC}=5V±5%, V_{DD}=21V ±0.5V)

PROGRAMMING

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	Address Setup Time to RESET↑	$4t_{CY}$			
t_{WA}	Address Hold Time After RESET↑	$4t_{CY}$			
t_{DW}	Data in Setup Time to PROG↑	$4t_{CY}$			
t_{WD}	Data in Hold Time After PROG↓	$4t_{CY}$			
t_{PH}	RESET Hold Time to Verify	$4t_{CY}$			
t_{VDDW}	V_{DD} Setup Time to PROG↑	0	1.0	mS	
t_{VDDH}	V_{DD} Hold Time After PROG↑	0	1.0	mS	
t_{PW}	Program Pulse Width	50	60	mS	
t_{TW}	Test 0 Setup Time for Program Mode	$4t_{CY}$			
t_{WT}	Test 0 Hold Time After Program Mode	$4t_{CY}$			
t_{DO}	Test 0 to Data Out Delay		$4t_{CY}$		
t_{WW}	RESET Pulse Width to Latch Address	$4t_{CY}$			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0.5	100	μs	
t_{CY}	CPU Operation Cycle Time	4.0		μs	
t_{RE}	RESET Setup Time Before EA↑	$4t_{CY}$			

NOTE:
If TEST 0 is high, t_{DO} can be triggered by RESET↑.

A.C. CHARACTERISTICS DMA

Symbol	Parameter	8042		8742		Units
		Min.	Max.	Min.	Max.	
t_{ACC}	DACK to WR or RD	0		0		ns
t_{CAC}	RD or WR to DACK	0		0		ns
t_{ACD}	DACK to Data Valid		130		130	ns
t_{CRQ}	RD or WR to DRQ Cleared		110		130	ns[1]

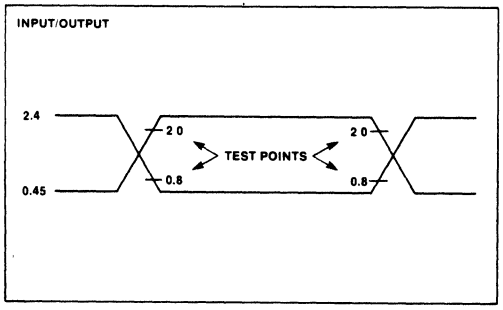
NOTE:
1. C_L = 150 pF.

A.C. CHARACTERISTICS PORT 2 (T_A = 0°C to +70°C, V_{CC} = +5V ±10%)

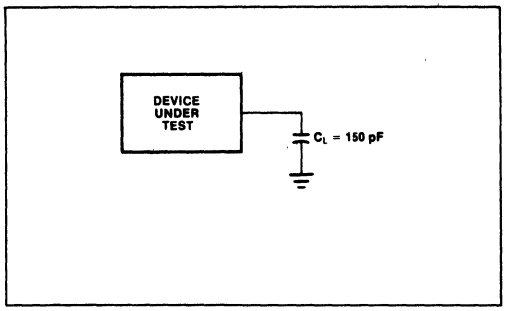
Symbol	Parameter	f(t_{CY})	8042/8742 [3]		Units
			Min.	Max.	
t_{CP}	Port Control Setup Before Falling Edge of PROG	1/15 t_{CY}−28	55		ns[1]
t_{PC}	Port Control Hold After Falling Edge of PROG	1/10 t_{CY}	125		ns[2]
t_{PR}	PROG to Time P2 Input Must Be Valid	(8/15 t_{CY}−16		650	ns[1]
t_{PF}	Input Data Hold Time		0	150	ns[2]
t_{DP}	Output Data Setup Time	2/10 T_{CY}	250		ns[1]
t_{PD}	Output Data Hold Time	1/10 t_{CY}−80	45		ns[2]
t_{PP}	PROG Pulse Width	6/10 t_{CY}	750		ns

NOTES:
1. C_L=80 pF.
2. C_L=20 pF.
3. t_{CY}=1.25 μs.

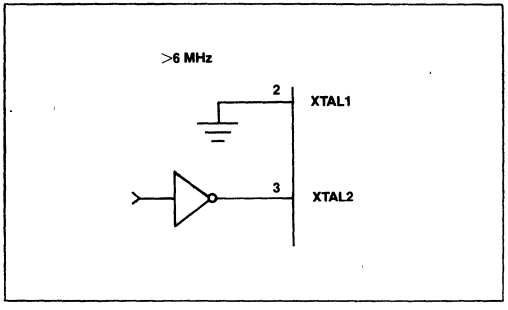
A.C. TESTING INPUT, OUTPUT WAVEFORM

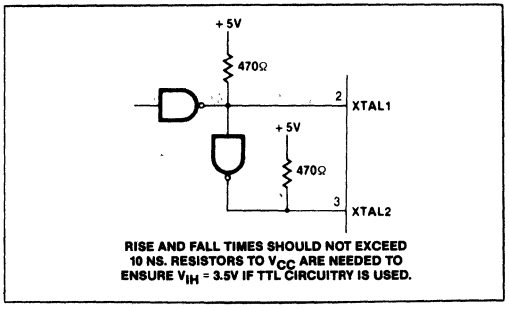


A.C. TESTING LOAD CIRCUIT



DRIVING FROM EXTERNAL SOURCE-TWO OPTIONS

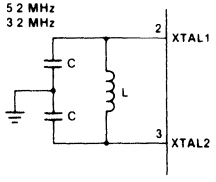




RISE AND FALL TIMES SHOULD NOT EXCEED 10 NS. RESISTORS TO V_{CC} ARE NEEDED TO ENSURE V_{IH} = 3.5V IF TTL CIRCUITRY IS USED.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz



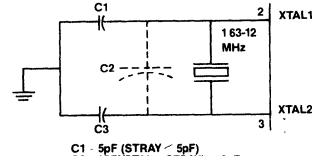
$$f \approx \frac{1}{2\pi\sqrt{LC'}}$$

$$C' = \frac{C + 3C_{PP}}{2}$$

$C_{PP} \approx 5 - 10$ pF PIN TO PIN CAPACITANCE

EACH C SHOULD BE APPROXIMATELY 20 pF INCLUDING STRAY CAPACITANCE

CRYSTAL OSCILLATOR MODE

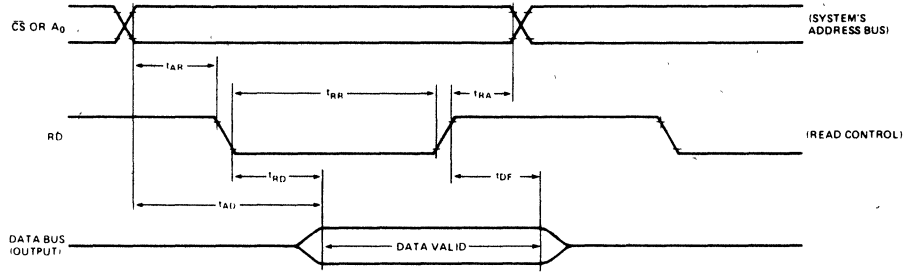


C1 = 5pF (STRAY < 5pF)
C2 = (CRYSTAL + STRAY) < 8pF
C3 = 20-30pF INCLUDING STRAY

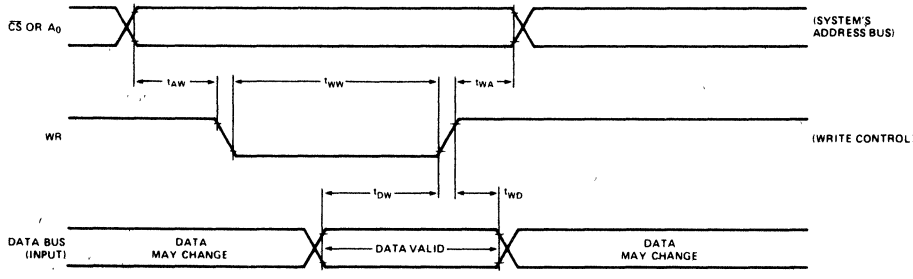
CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 30Ω AT 12 MHz; LESS THAN 75Ω AT 6 MHz; LESS THAN 160Ω AT 3.6 MHz.

WAVEFORMS

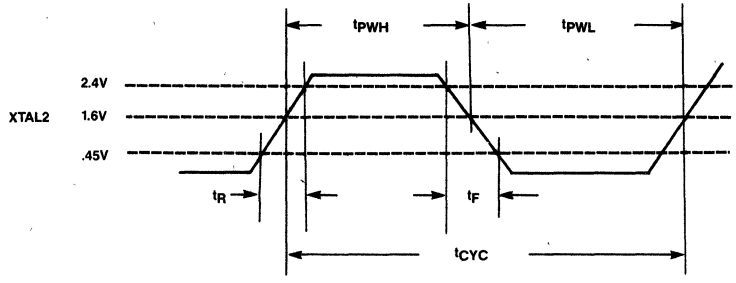
READ OPERATION—DATA BUS BUFFER REGISTER



WRITE OPERATION—DATA BUS BUFFER REGISTER



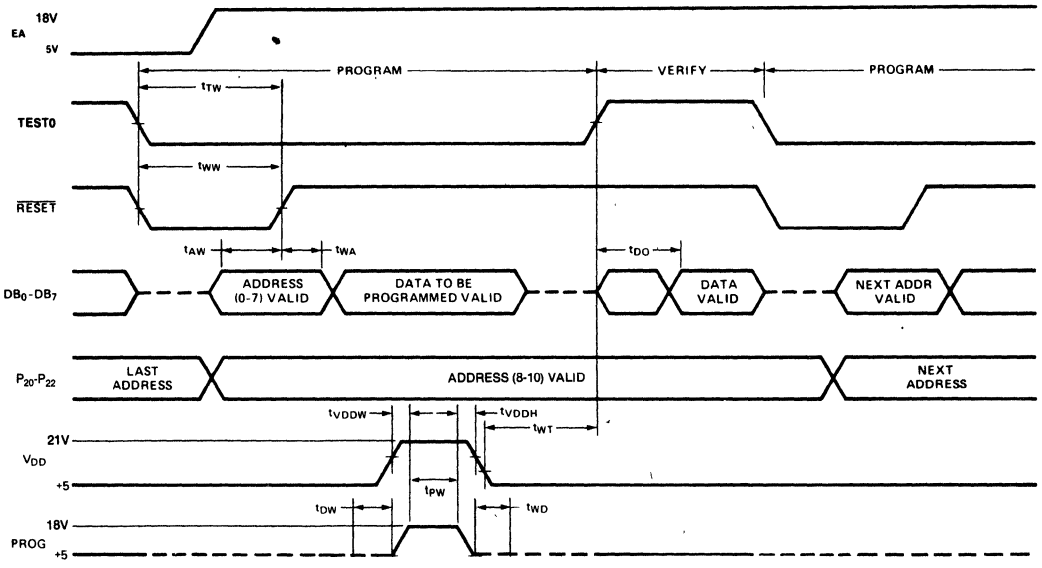
CLOCK TIMING



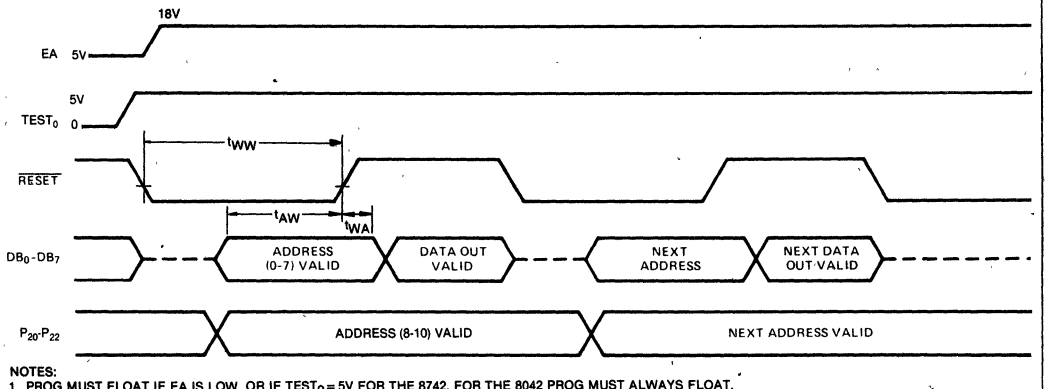
AFN-01832A

WAVEFORMS (Continued)

COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



VERIFY MODE (ROM/EPROM)



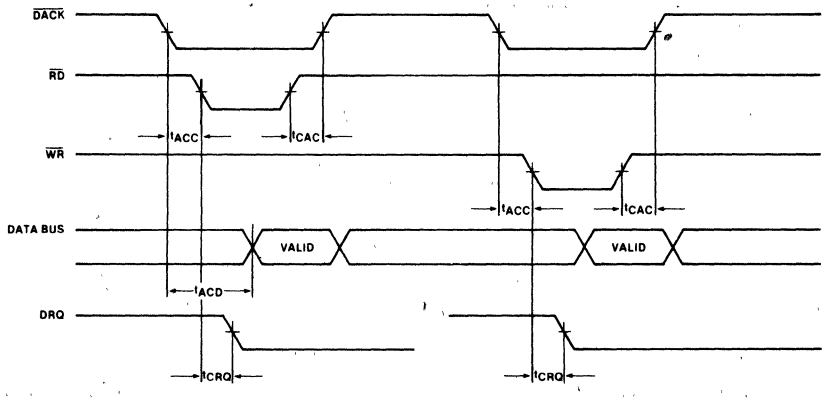
NOTES:
1. PROG MUST FLOAT IF EA IS LOW OR IF $TEST_0$ = 5V FOR THE 8742. FOR THE 8042 PROG MUST ALWAYS FLOAT.
2. A_0 MUST BE HELD LOW (i.e., = 0V) DURING PROGRAM/VERIFY MODES.
3. TEST 0 MUST BE HELD HIGH.

The 8742 EPROM can be programmed by the following Intel products:

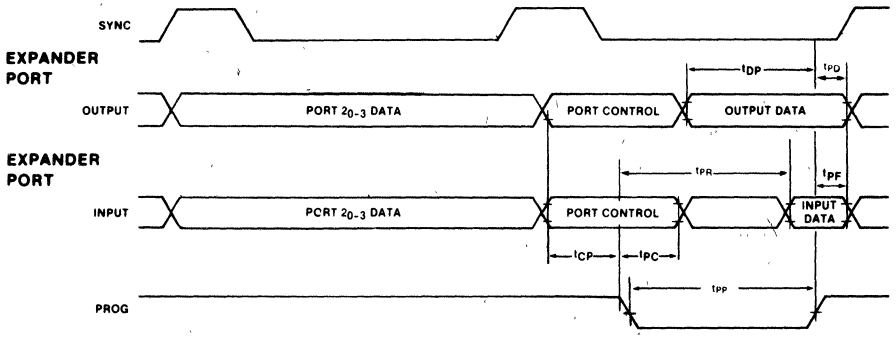
1. Universal PROM Programmer (UPP 103) peripheral of the Intellec® Development System with a UPP-549 Personality Card.
2. iUP-200/iUP-201 PROM Programmer with the iUP-F87/44 Personality Module.

WAVEFORMS (Continued)

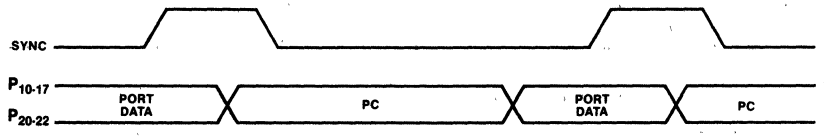
DMA



PORT 2



PORT TIMING DURING EA



ON THE RISING EDGE OF SYNC AND EA IS ENABLED, PORT DATA IS VALID AND CAN BE STROBED ON THE TRAILING EDGE OF SYNC THE PROGRAM COUNTER CONTENTS ARE AVAILABLE

Table 2. UPI™ Instruction Set

Mnemonic	Description	Bytes	Cycles
ACCUMULATOR			
ADD A, Rr	Add register to A	1	1
ADD A, @Rr	Add data memory to A	1	1
ADD A, #data	Add immediate to A	2	2
ADDC A, Rr	Add register to A with carry	1	1
ADDC A, @Rr	Add data memory to A with carry	1	1
ADDC A, #data	Add immediate to A with carry	2	2
ANL A, Rr	AND register to A	1	1
ANL A, @Rr	AND data memory to A	1	1
ANL A, #data	AND immediate to A	2	2
ORL A, Rr	OR register to A	1	1
ORL A, @Rr	OR data memory to A	1	1
ORL A, #data	OR immediate to A	2	2
XRL A, Rr	Exclusive OR register to A	1	1
XRL A, @Rr	Exclusive OR data memory to A	1	1
XRL A, #data	Exclusive OR immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
INPUT/OUTPUT			
IN A, Pp	Input port toA	1	2
OUTL Pp, A	Output A to port	1	2
ANL Pp, #data	AND immediate to port	2	2
ORL Pp, #data	OR immediate to port	2	2
IN A, DBB	Input DBB to A, clear IBF	1	1
OUT DBB, A	Output A to DBB, set OBF	1	1
MOV STS, A	A_4–A_7 to Bits 4–7 of Status	1	1
MOVD A, Pp	Input Expander port to A	1	2
MOVD Pp, A	Output A to Expander port	1	2
ANLD Pp, A	AND A to Expander port	1	2
ORLD Pp, A	OR A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
DATA MOVES			
MOV A, Rr	Move register to A	1	1
MOV A, @Rr	Move data memory to A	1	1
MOV A, #data	Move immediate TO A	2	2
MOV Rr, A	Move A to register	1	1
MOV @Rr, A	Move A to data memory	1	1
MOV Rr, #data	Move immediate to register	2	2
MOV @Rr, #data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, Rr	Exchange A and register	1	1
XCH A, @Rr	Exchange A and data memory	1	1
XCHD A, @Rr	Exchange digit of A and register	1	1
MOVP A, @A	Move to A from current page	1	2
MOVP3, A, @A	Move to A from page 3	1	2
TIMER/COUNTER			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/ Counter Interrupt	1	1
DIS TCNTI	Disable Timer/ Counter Interrupt	1	1
CONTROL			
EN DMA	Enable DMA Handshake Lines	1	1
EN I	Enable IBF Interrupt	1	1
DIS I	Disable IBF Interrupt	1	1
EN FLAGS	Enable Master Interrupts	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
NOP	No Operation	1	1
REGISTERS			
INC Rr	Increment register	1	1
INC @Rr	Increment data memory	1	1
DEC Rr	Decrement register	1	1
SUBROUTINE			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Table 2. UPI™ Instruction Set (Continued)

Mnemonic	Description	Bytes	Cycles
FLAGS			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear F1 Flag	1	1
CPL F1	Complement F1 Flag	1	1
BRANCH			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ Rr, addr	Decrement register and jump	2	2
JC addr	Jump on Carry=1	2	2
JNC addr	Jump on Carry=0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0=1	2	2
JNT0 addr	Jump on T0=0	2	2
JT1 addr	Jump on T1=1	2	2
JNT1 addr	Jump on T1=0	2	2
JF0 addr	Jump on F0 Flag=1	2	2
JF1 addr	Jump on F1 Flag=1	2	2
JTF addr	Jump on Timer Flag =1, Clear Flag	2	2
JNIBF addr	Jump on IBF Flag =0	2	2
JOBF addr	Jump on OBF Flag =1	2	2
JBb addr	Jump on Accumulator Bit	2	2

8243
MCS-48® INPUT/OUTPUT EXPANDER

- **Low Cost**
- **Simple Interface to MCS-48® Microcomputers**
- **Four 4-Bit I/O Ports**
- **AND and OR Directly to Ports**
- **24-Pin DIP**
- **Single 5V Supply**
- **High Output Drive**
- **Direct Extension of Resident 8048 I/O Ports**

The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48® family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.

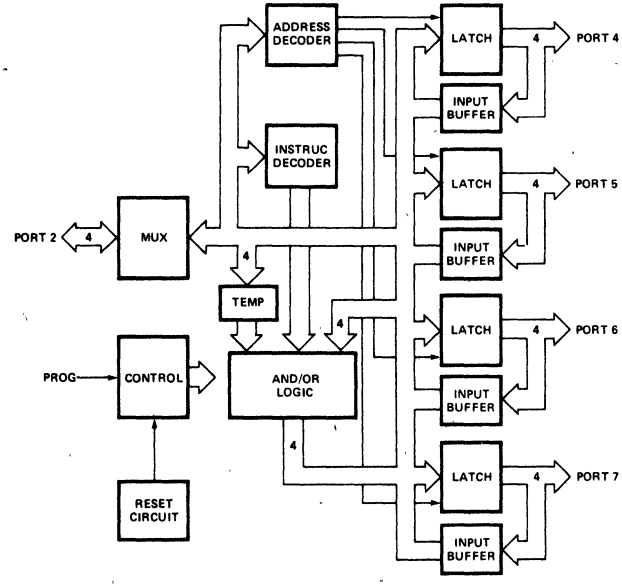


Figure 1. 8243 Block Diagram

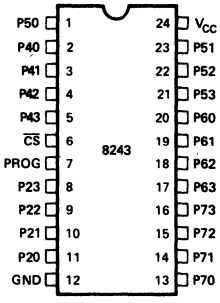


Figure 2. 8243 Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.
$\overline{CS}$	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20-P23	11-8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40-P43 P50-P53 P60-P63 P70-P73	2-5 1, 23-21 20-17 13-16	Four (4) bit bi-directional I/O ports. May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data
V_{CC}	24	+5 volt supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048 and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048/8748/8035.

Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if VCC drops below 1V.

P21	P20	Address Code	P23	P22	Instruction Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

Write Modes

The device has three write modes. MOVD Pi, A directly writes new data into the selected port and old data is lost. ORLD Pi, A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi, A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... -0.5 V to +7V
Power Dissipation ..... 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

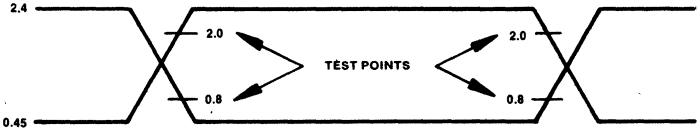
D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+0.5	V	
V_{OL1}	Output Low Voltage Ports 4-7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4-7	2.4			V	I_{OH} = 240μA
I_{IL1}	Input Leakage Ports 4-7	-10		20	μA	V_{in} = V_{CC} to OV
I_{IL2}	Input Leakage Port 2, CS, PROG	-10		10	μA	V_{in} = V_{CC} to OV
V_{OL3}	Output Low Voltage Port 2			45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100μA
I_{OL}	Sum of all I_{OL} from 16 Outputs			72	mA	4.5 mA Each Pin

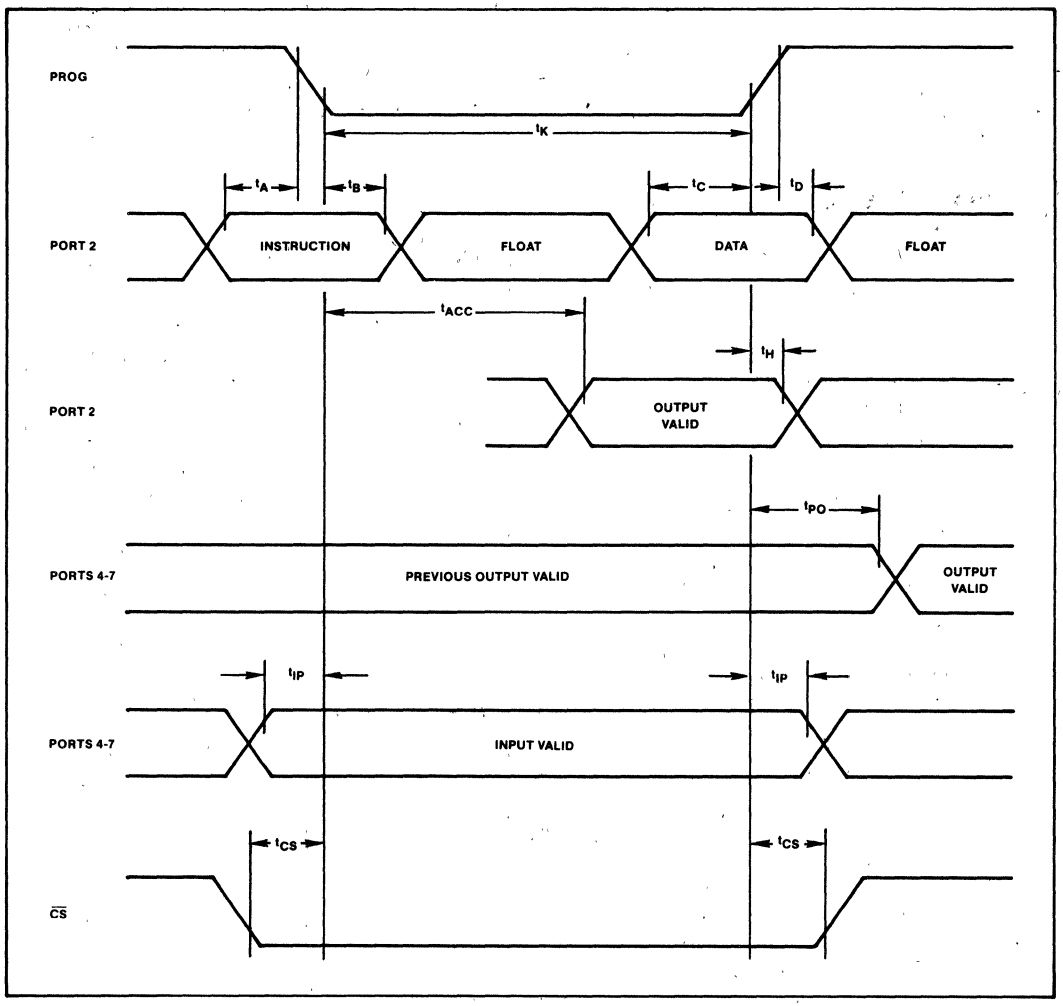
*See following graph for additional sink current capability

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

Symbol	Parameter	Min	Max	Units	Test Conditions
tA	Code Valid Before PROG	100		ns	80 pF Load
tB	Code Valid After PROG	60		ns	20 pF Load
tC	Data Valid Before PROG	200		ns	80 pF Load
tD	Data Valid After PROG	20		ns	20 pF Load
tH	Floating After PROG	0	150	ns	20 pF Load
tK	PROG Negative Pulse Width	700		ns	
tCS	CS Valid Before/After PROG	50		ns	
tPO	Ports 4-7 Valid After PROG		700	ns	100 pF Load
tLP1	Ports 4-7 Valid Before/After PROG	100		ns	
tACC	Port 2 Valid After PROG		650	ns	80 pF Load



WAVEFORMS



AFN-00214A-04

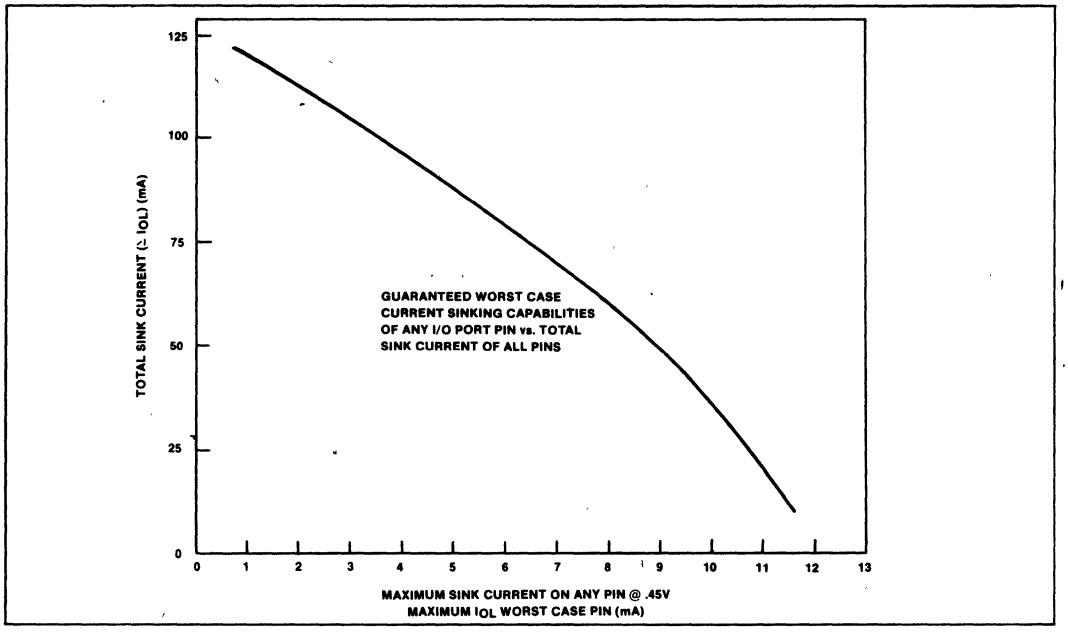


Figure 3

Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total IOL must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

I_{OL} = 5 x 1.6 mA = 8 mA
ϵI_{OL} = 60 mA from curve
pins = 60 mA ÷ 8 mA/pin = 7.5 = 7

In this case, 7 lines can sink 8 mA for a total of 56mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads—20 mA @ 1V (port 7 only)
8 loads—4 mA @ .45V
6 loads—3.2 mA @ .45V
Is this within the specified limits?

ϵI_{OL} = (2 x 20) + (8 x 4) + (6 x 3.2) = 91.2 mA.
From the curve: for I_{OL} = 4 mA, $\epsilon I_{OL} \approx$ 93 mA.
since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating ϵI_{OL}, it is the largest current required @ .45V which determines the maximum allowable ϵI_{OL}.

NOTE: A10 to 50KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

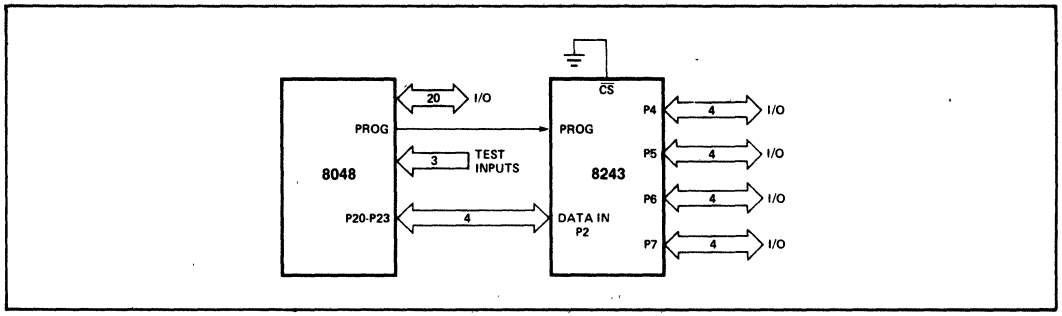


Figure 4. Expander Interface

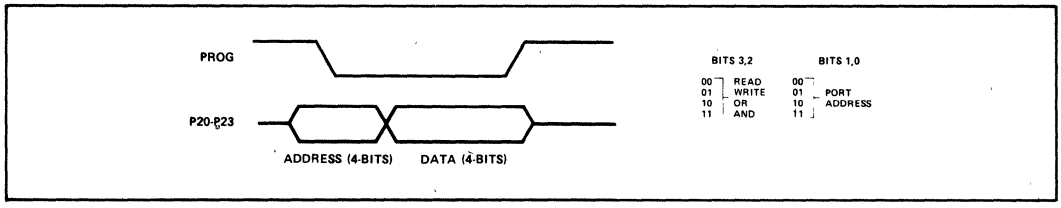


Figure 5. Output Expander Timing

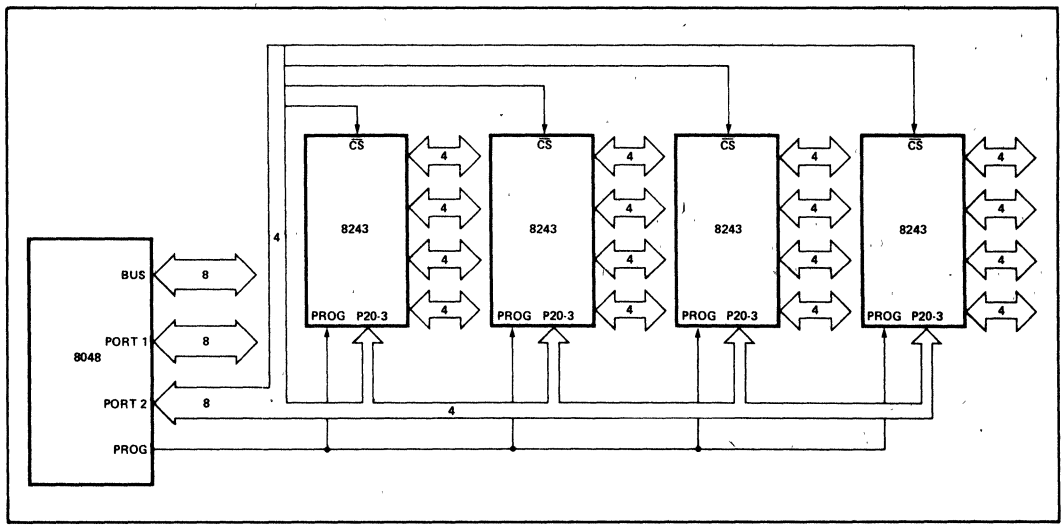


Figure 6. Using Multiple 8243's

8295
DOT MATRIX PRINTER CONTROLLER

- **Interfaces Dot Matrix Printers to MCS-48™, MCS-80/85™, MCS-86™ Systems**
- **40 Character Buffer On Chip**
- **Serial or Parallel Communication with Host**
- **DMA Transfer Capability**
- **Programmable Character Density (10 or 12 Chararcters/Inch)**
- **Programmable Print Intensity**
- **Single or Double Width Printing**
- **Programmable Multiple Line Feeds**
- **3 Tabulations**
- **2 General Purpose Outputs**

The Intel® 8295 Dot Matrix Printer Controller provides an interface for microprocessors to the LRC 7040 Series dot matrix impact printers. It may also be used as an interface to other similar printers.

The chip may be used in a serial or parallel communication mode with the host processor. In parallel mode, data transfers are based on polling, interrupts, or DMA. Furthermore, it provides internal buffering of up to 40 characters and contains a 7 x 7 matrix character generator accommodating 64 ASCII characters.

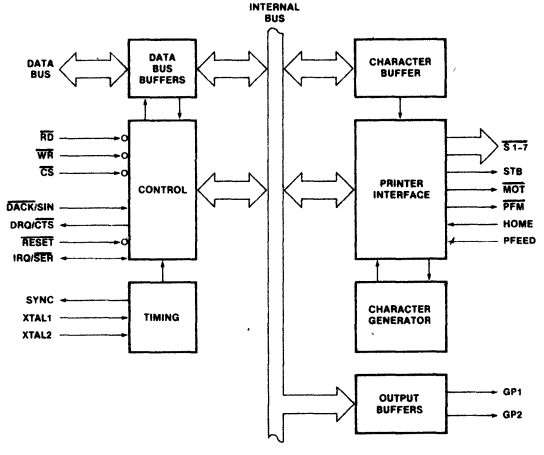


Figure 1. Block Diagram

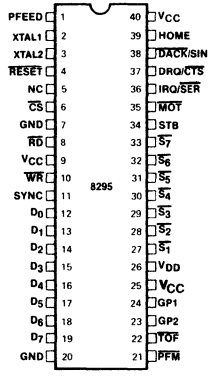


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
PFEED	1	I	**Paper Feed:** Paper feed input switch.
XTAL1 XTAL2	2 3	I	**Crystal:** Inputs for a crystal to set internal oscillator frequency. For proper operation use 6 MHz crystal.
$\overline{RESET}$	4	I	**Reset:** Reset input, active low. After reset the 8295 will be set for 12 characters/inch single width printing, solenoid strobe at 320 msec.
NC	5		**No Connection:** No connection or tied high.
$\overline{CS}$	6	I	**Chip Select:** Chip select input used to enable the RD and WR inputs except during DMA.
GND	7		**Ground:** This pin must be tied to ground.
$\overline{RD}$	8	I	**Read:** Read input which enables the master CPU to read data and status. In the serial mode this pin must be tied to V_{CC}.
V_{CC}	9		**Power:** +5 volt power input: +5V ± 10%.
$\overline{WR}$	10	I	**Write:** Write input which enables the master CPU to write data and commands to the 8295. In the serial mode this pin must be tied to V_{SS}.
SYNC	11	O	**Sync:** 2.5 μs clock output. Can be used as a strobe for external circuitry.
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7	12 13 14 15 16 17 18 19	I/O	**Data Bus:** Three-state bidirectional data bus buffer lines used to interface the 8295 to the host processor in the parallel mode. In the serial mode D_0—D_2 sets up the baud rate.
GND	20		**Ground:** This pin must be tied to ground.
V_{CC}	40		**Power:** +5 volt power input: +5 ± 10%.

Symbol	Pin No.	Type	Name and Function
HOME	39	I	**Home:** Home input switch, used by the 8295 to detect that the print head is in the home position.
$\overline{DACK}$/SIN	38	I	**DMA Acknowledge/Serial Input:** In the parallel mode used as DMA acknowledgment; in the serial mode, used as input for data.
DRQ/$\overline{CTS}$	37	O	**DMA Request/Clear to Send:** In the parallel mode used as DMA request output pin to indicate to the 8257 that a DMA transfer is requested; in the serial mode used as clear-to-send signal.
IRQ/$\overline{SER}$	36	O	**Interrupt Request/Serial Mode:** In parallel mode it is an interrupt request input to the master CPU; in serial mode it should be strapped to V_{SS}.
$\overline{MOT}$	35	O	**Motor:** Main motor drive, active low.
STB	34	O	**Solenoid Strobe:** Solenoid strobe output. Used to determine duration of solenoids activation.
$\overline{S_7}$ $\overline{S_6}$ $\overline{S_5}$ $\overline{S_4}$ $\overline{S_3}$ $\overline{S_2}$ $\overline{S_1}$	33 32 31 30 29 28 27	O	**Solenoid:** Solenoid drive outputs; active low.
V_{DD}	26		**Power:** +5V power input (+5V ± 10%). Low power standby pin.
V_{CC}	25		**Power:** Tied high.
GP1 GP2	24 23	O O	**General Purpose:** General purpose output pins.
$\overline{TOF}$	22	I	**Top of Form:** Top of form input, used to sense top of form signal for type T printer.
$\overline{PFM}$	21	O	**Paper Feed Motor Drive:** Paper feed motor drive, active low.

FUNCTIONAL DESCRIPTION

The 8295 interfaces microcomputers to the LRC 7040 Series dot matrix impact printers, and to other similar printers. It provides internal buffering of up to 40 characters. Printing begins automatically when the buffer is full or when a carriage return character is received. It provides a modified 7x7 matrix character generator. The character set includes 64 ASCII characters.

Communication between the 8295 and the host processor can be implemented in either a serial or parallel mode. The parallel mode allows for character transfers into the buffer via DMA cycles. The serial mode features selectable data rates from 110 to 4800 baud.

The 8295 also offers two general purpose output pins which can be set or cleared by the host processor. They can be used with various printers to implement such functions as ribbon color selection, enabling form release solenoid, and reverse document feed.

COMMAND SUMMARY

Hex Code	Description
00	Set GP1. This command brings the GP1 pin to a logic high state. After power on it is automatically set high.
01	Set GP2. Same as the above but for GP2.
02	Clear GP1. Sets GP1 pin to logic low state, inverse of command 00.
03	Clear GP2. Same as above but for GP2. Inverse command 01.
04	Software Reset. This is a pacify command. This command is not effective immediately after commands requiring a parameter, as the Reset command will be interpreted as a parameter.
05	Print 10 characters/in. density.
06	Print 12 characters/in. density.
07	Print double width characters. This command prints characters at twice the normal width, that is, at either 17 or 20 characters per line.
08	Enable DMA mode; must be followed by two bytes specifying the number of data characters to be fetched. Least significant byte accepted first.
09	Tab character.
0A	Line feed.
0B	Multiple Line Feed; must be followed by a byte specifying the number of line feeds.
0C	Top of Form. Enables the line feed output until the Top of Form input is activated.
0D	Carriage Return. Signifies end of a line and enables the printer to start printing.
0E	Set Tab #1, followed by tab position byte.
0F	Set Tab #2, followed by tab position byte. Should be greater than Tab #1.
10	Set Tab #3, followed by tab position byte. Should be greater than Tab #2.
11	Print Head Home on Right. On some printers the print head home position is on the right. This command would enable normal left to right printing with such printers.
12	Set Strobe Width; must be followed by strobe width selection byte. This command adjusts the duration of the strobe activation.

PROGRAMMABLE PRINTING OPTIONS

CHARACTER DENSITY

The character density is programmable at 10 or 12 characters/inch (32 or 40 characters/line). The 8295 is automatically set to 12 characters/inch at power-up. Invoking the Print Double-Width command halves the character density (5 or 6 characters/inch). The 10 char/in or 12 char/in command must be re-issued to cancel the Double-Width mode. Different character density modes may not be mixed within a single line of printing.

PRINT INTENSITY

The intensity of the printed characters is determined by the amount of time during which the solenoid is on. This on-time is programmable via the Set Strobe-Width command. A byte following this command sets the solenoid on-time according to Table 2. Note that only the three least significant bits of this byte are important.

Table 2. Solenoid On-Time

D7—D3	D2	D1	D0	Solenoid On (microsec)
x	0	0	0	200
x	0	0	1	240
x	0	1	0	280
x	0	1	1	320
x	1	0	0	360
x	1	0	1	400
x	1	1	0	440
x	1	1	1	480

TABULATIONS

Up to three tabulation positions may be specified with the 8295. The column position of each tabulation is selected by issuing the Set Tab commands, each fol-

lowed by a byte specifying the column. The tab positions will then remain valid until new Set Tab commands are issued.

Sending a tab character (09H) will automatically fill the character buffer with blanks up to the next tab position. The character sent immediately after the tab character will thus be stored and printed at that position.

CPU TO 8295 INTERFACE

Communication between the CPU and the 8295 may take place in either a serial or parallel mode. However, the selection of modes is inherent in the system hardware; it is not software programmable. Thus, the two modes cannot be mixed in a single 8295 application.

PARALLEL INTERFACE

Two internal registers on the 8295 are addressable by the CPU: one for input, one for output. The following table describes how these registers are accessed.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Register
1	0	0	Input Data Register
0	1	0	Output Status Register

Input Data Register—Data written to this register is interpreted in one of two ways, depending on how the data is coded.

1. A command to be executed (0XH or 1XH).
2. A character to be stored in the character buffer for printing (2XH, 3XH, 4XH, or 5XH). See the character set, Table 2.

Output Status Register—8295 status is available in this register at all times.

STATUS BIT:	7	6	5	4	3	2	1	0
FUNCTION:	x	x	PA	DE	x	x	IBF	x

PA—Parameter Required; PA = 1 indicates that a command requiring a parameter has been received. After the necessary parameters have been received by the 8295, the PA flag is cleared.

DE—DMA Enabled; DE = 1 whenever the 8295 is in DMA mode. Upon completion of the required DMA transfers, the DE flag is cleared.

IBF—Input Buffer Full; IBF = 1 whenever data is written to the Input Data Register. No data should be written to the 8295 when IBF = 1.

A flow chart describing communication with the 8295 is shown in Figure 3.

The interrupt request output (IRQ, Pin 36) is available on the 8295 for interrupt driven systems. This output is asserted true whenever the 8295 is ready to receive data.

To improve bus efficiency and CPU overhead, data may be transferred from main memory to the 8295 via DMA cycles. Sending the Enable DMA command (08H) activates the DMA channel of the 8295. This command must be followed by two bytes specifying the length of the data string to be transferred (least significant byte first). The 8295 will then assert the required DMA requests to the 8257 DMA controller without further CPU intervention. Figure 4 shows a block diagram of the 8295 in DMA mode.

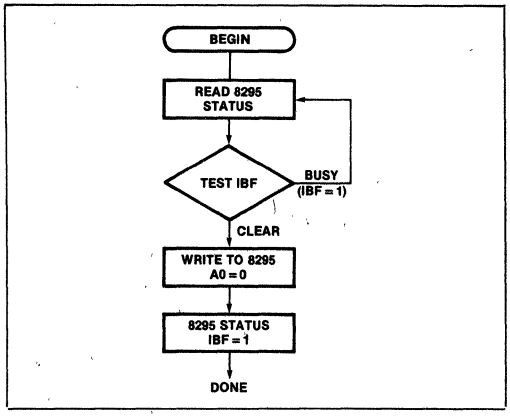


Figure 3. Host to 8295 Protocol Flowchart

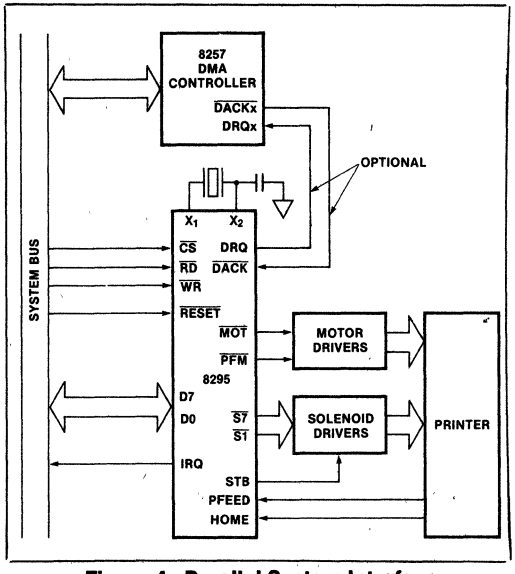


Figure 4. Parallel System Interface

Data transferred in the DMA mode may be either commands or characters or a mixture of both. The procedure is as follows:

1. Set up the 8257 DMA controller channel by sending a starting address and a block length.
2. Set up the 8295 by issuing the "Enable DMA" command (08H) followed by two bytes specifying the block length (least significant byte first).

The DMA enabled flag (DE) will be true until the assigned data transfer is completed. Upon completion of the transfer, the flag is cleared and the interrupt request (IRQ) signal is asserted. The 8295 then returns to the non-DMA mode of operation.

SERIAL INTERFACE

The 8295 may be hardware programmed to operate in a serial mode of communication. By connecting the IRQ/$\overline{SER}$ pin (pin 36) to logic zero, the serial mode is enabled immediately upon power-up. The serial Baud rate is also hardware programmable; by strapping pins 14, 13, and 12 according to Table 3, the rate is selected. $\overline{CS}$, $\overline{RD}$, and $\overline{WR}$ must be strapped as shown in Figure 5.

Table 3. Serial Baud Rate

Pin 14	Pin 13	Pin 12	Baud Rate
0	0	0	110
0	0	1	150
0	1	0	300
0	1	1	600
1	0	0	1200
1	0	1	2400
1	1	0	4800
1	1	1	4800

The serial data format is shown in Figure 5. The CPU should wait for a clear to send signal ($\overline{CTS}$) from the 8295 before sending data.

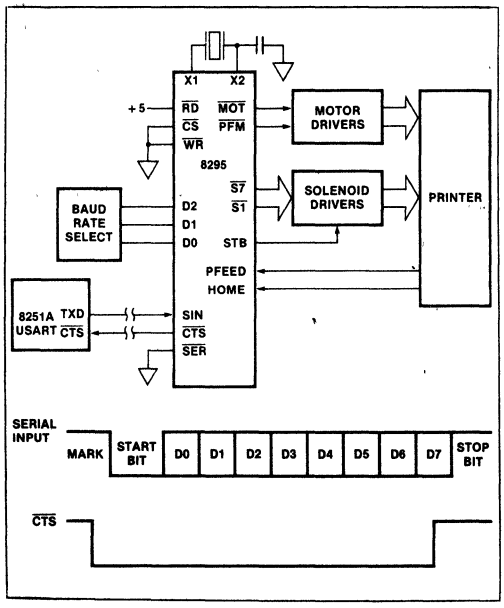


Figure 5. Serial Interface to UART (8251A)

8295 TO PRINTER INTERFACE

The strobe output signal of the 8295 determines the duration of the solenoid outputs, which hold the data to the printer. These solenoid outputs cannot drive the printer solenoids directly. They should be buffered through solenoid drivers as shown in Figure 6. Recommended solenoid and motor driver circuits may be found in the printer manufacturer's interface guide.

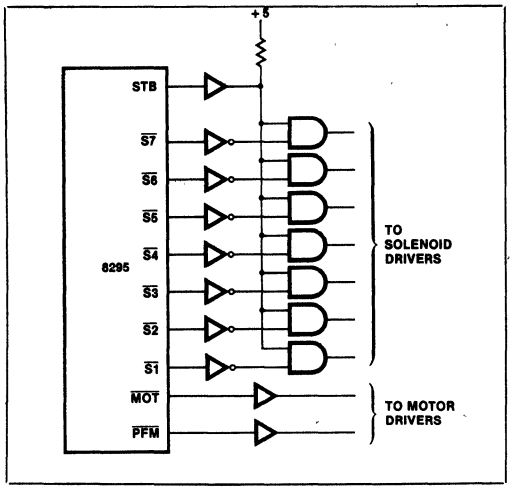


Figure 6. 8295 To Printer Solenoid Interface

OSCILLATOR AND TIMING CIRCUITS

The 8295's internal timing generation is controlled by a self-contained oscillator and timing circuit. A 6 MHz crystal is used to derive the basic oscillator frequency. The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 7. The recommended crystal connection is shown in Figure 8.

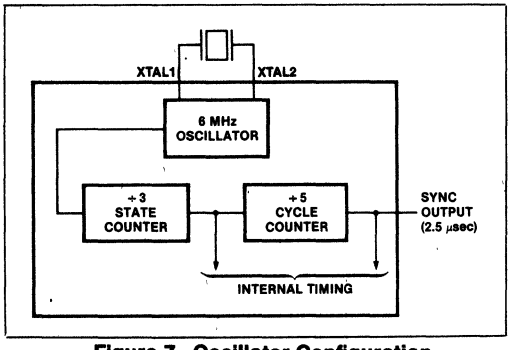


Figure 7. Oscillator Configuration

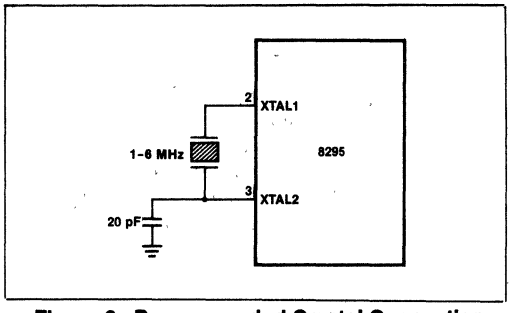


Figure 8. Recommended Crystal Connection

8295 CHARACTER SET

Hex Code	Print Char.	Hex Code	Print Char.	Hex Code	Print Char.	Hex Code	Print Char.
20	space	30	0	40	@	50	P
21	!	31	1	41	A	51	Q
22	"	32	2	42	B	52	R
23	#	33	3	43	C	53	S
24	$	34	4	44	D	54	T
25	%	35	5	45	E	55	U
26	&	36	6	46	F	56	V
27	'	37	7	47	G	57	W
28	(	38	8	48	H	58	X
29	)	39	9	49	I	59	Y
2A	*	3A	:	5A	J	5A	Z
2B	+	3B	;	4B	K	5B	[
2C	,	3C	<	4C	L	5C	\
2D	-	3D	=	4D	M	5D	]
2E	.	3E	>	4E	N	5E	↑
2F	/	3F	?	4F	O	5F	—

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65° to +150°C
Voltage on Any Pin With
Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. AND OPERATING CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except X_1, X_2, $\overline{RESET}$)	−0.5		0.8	V	
V_{IL1}	Input Low Voltage (X_1, X_2, $\overline{RESET}$)	−0.5		0.6	V	
V_{IH}	Input High Voltage (All Except X_1, X_2, $\overline{RESET}$)	2.2		V_{CC}	V	
V_{IH1}	Input High Voltage (X_1, X_2, $\overline{RESET}$)	3.8		V_{CC}	V	
V_{OL}	Output Low Voltage (D_0-D_7)			0.45	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage (All Other Outputs)			0.45	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage (D_0-D_7)	2.4			V	I_{OH} = −400 μA
V_{OH1}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = −50 μA
I_{IL}	Input Leakage Current ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_0)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Output Leakage Current (D_0-D_7, High Z State)			±10	μA	$V_{SS}+0.45 \leq V_{OUT} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		5	15	mA	
$I_{DD}+I_{CC}$	Total Supply Current		60	125	mA	
I_{LI}	Low Input Load Current (Pins 24, 27-38)			0.5	mA	V_{IL} = 0.8 V
I_{LI1}	Low Input Load Current ($\overline{RESET}$)			0.2	mA	V_{IL} = 0.8 V
I_{IH}	Input High Leakage Current (Pins 22, 38)			100	μA	V_{IN} = V_{CC}
C_{IN}	Input Capacitance			10	pF	
$C_{I/O}$	I/O Capacitance			20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ± 10%, V_{SS} = 0V)

DBB READ

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$ ↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$ ↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		225	ns	C_L = 150 pF
t_{RD}	$\overline{RD}$ ↓ to Data Out Delay		225	ns	C_L = 150 pF
t_{DF}	$\overline{RD}$ ↑ to Data Float Delay		100	ns	
t_{CY}	Cycle Time	2.5	15	μs	

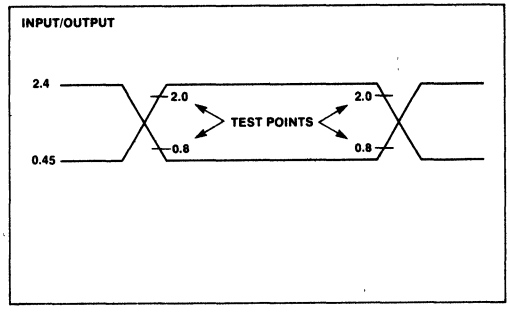
DBB WRITE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	$\overline{CS}$, A_0 Setup to $\overline{WR}$ ↓	0		ns	
t_{WA}	$\overline{CS}$, A_0 Hold After $\overline{WR}$ ↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$ ↑	150		ns	
t_{WD}	Data Hold to $\overline{WR}$ ↑	0		ns	

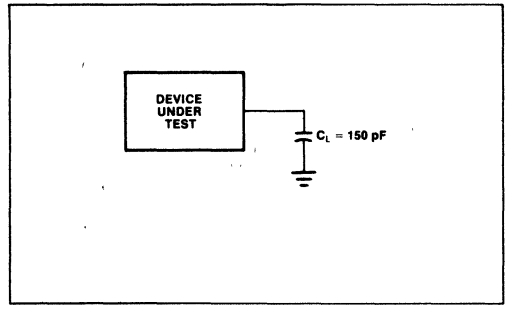
DMA AND INTERRUPT TIMING

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{ACC}	$\overline{DACK}$ Setup to Control	0		ns	
t_{CAC}	$\overline{DACK}$ Hold After Control	0		ns	
t_{CRQ}	$\overline{WR}$ to DRQ Cleared		200	ns	
t_{ACD}	$\overline{DACK}$ to Data Valid		225	ns	C_L = 150 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM

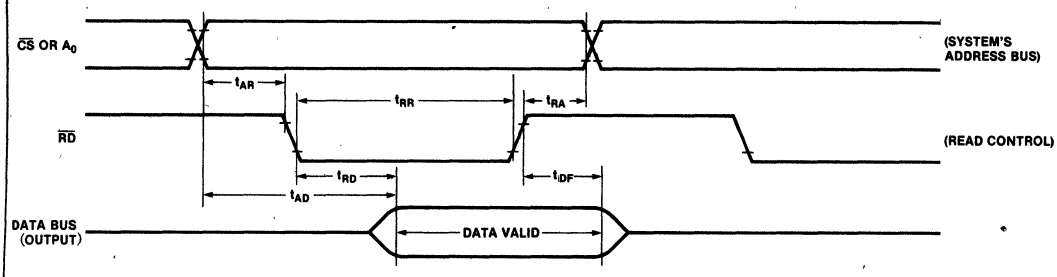


A.C. TESTING LOAD CIRCUIT

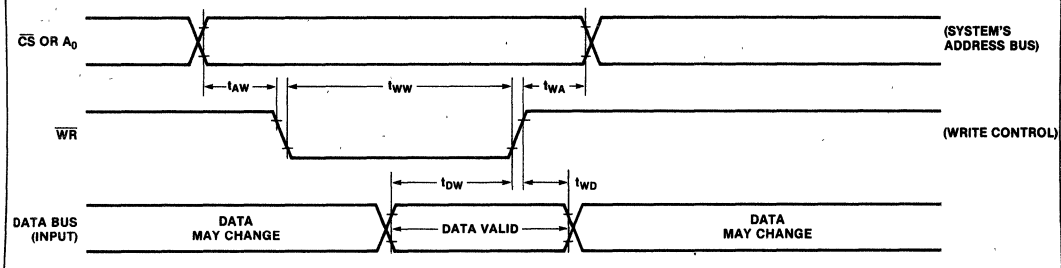


WAVEFORMS

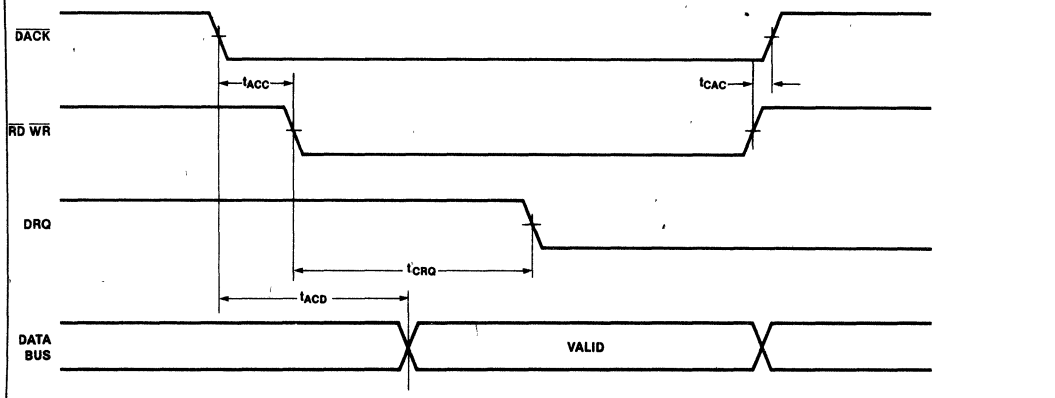
READ OPERATION—OUTPUT BUFFER REGISTER



WRITE OPERATION—INPUT BUFFER REGISTER

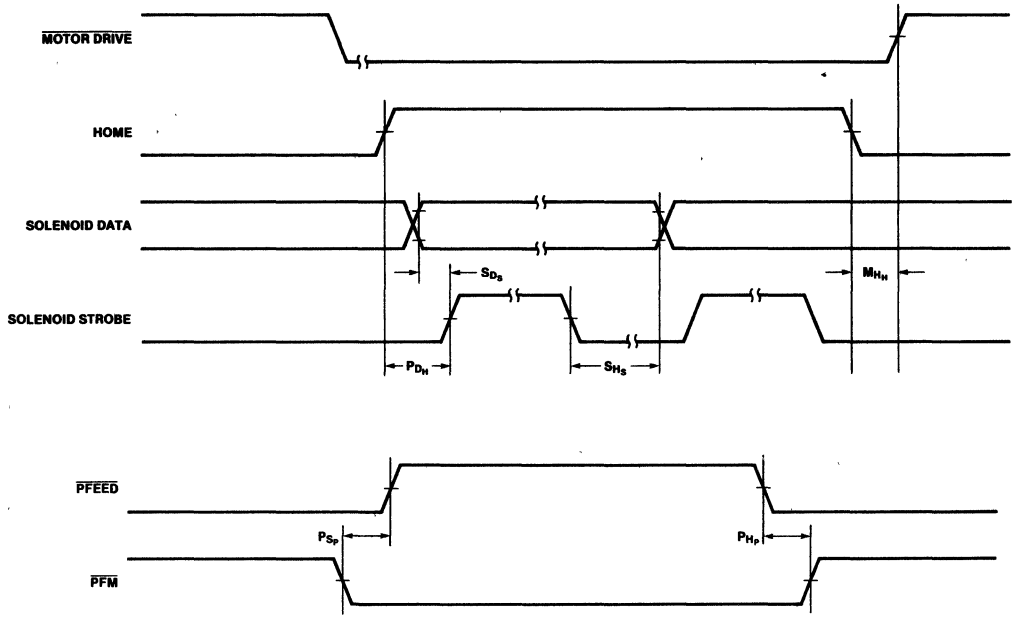


DMA AND INTERRUPT TIMING



WAVEFORMS (Continued)

PRINTER INTERFACE TIMING

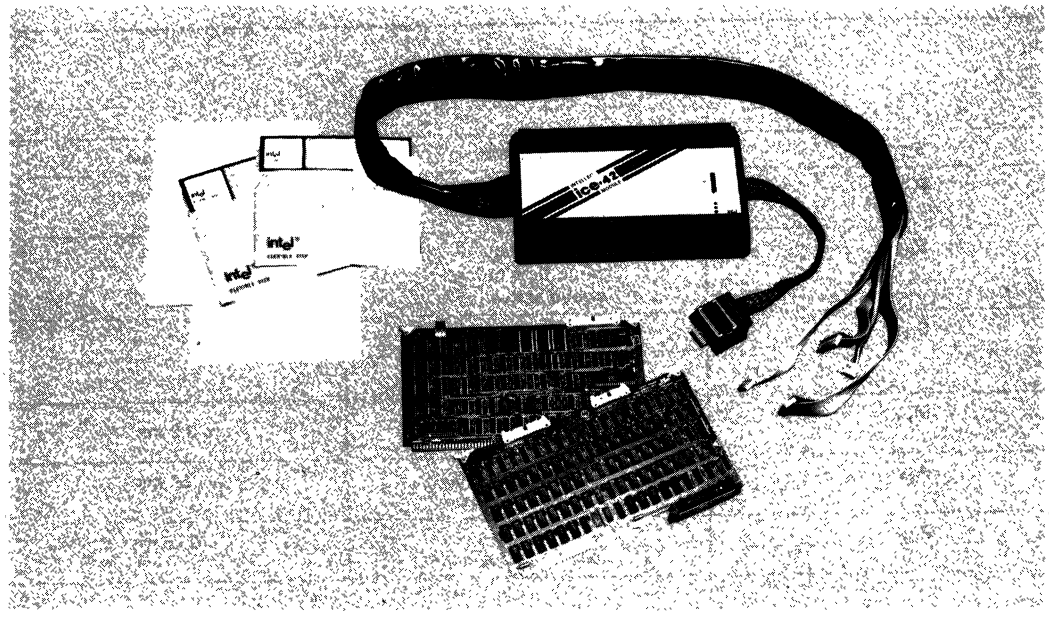


Symbol	Parameter	Typical
P_{DH}	Print delay from home inactive	1.8 ms
S_{DS}	Solenoid data setup time before strobe active	25 μs
S_{HS}	Solenoid data hold after strobe inactive	>1 ms
M_{HA}	Motor hold time after home active	3.2 ms
P_{SP}	PFEED setup time after PFM active	58 ms
P_{HP}	PFM hold time after PFEED active	9.75 ms

ICE™-42
8042 IN-CIRCUIT EMULATOR

- **Precise, full-speed, real-time emulation**
  - **Load, drive, timing characteristics**
  - **Full-speed program RAM**
  - **Parallel ports**
  - **Data Bus**
- **User-specified breakpoints**
- **Execution trace**
  - **User-specified qualifier registers**
  - **Conditional trigger**
  - **Symbolic groupings and display**
  - **Instruction and frame modes**
- **Emulation timer**
- **Full symbolic debugging**
- **Single-line assembly and disassembly for program instruction changes**
- **Macro commands and conditional block constructs for automated debugging sessions**
- **HELP facility: ICE™-42 command syntax reference at the console**
- **User confidence test of ICE™-42 hardware**

The ICE™-42 module resides in the Intellec Microcomputer Development System and interfaces to any user-designed 8042 or 8041A system through a cable terminating in an 8042 emulator microprocessor and a pin-compatible plug. The emulator processor, together with 2K bytes of user program RAM located in the ICE-42 buffer box, replaces the 8042 device in the user system while maintaining the 8042 electrical and timing characteristics. Powerful Intellec debugging functions are thus extended into the user system. Using the ICE-42 module, the designer can emulate the system's 8042 chip in real-time or single-step mode. Breakpoints allow the user to stop emulation on user-specified conditions, and a trace qualifier feature allows the conditional collection of 1000 frames of trace data. Using the single-line 8042 assembler the user may alter program memory using the 8042 assembler mnemonics and symbolic references, without leaving the emulator environment. Frequently used command sequences can be combined into compound commands and identified as macros with user-defined names.



MAY 1983
Order Number: 210818-001

FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The ICE-42 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-42 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-42 emulator assists four stages of development:

SOFTWARE DEBUGGING

This emulator operates without being connected to the user's system before any of the user's hardware is available. In this stage ICE-42 debugging capabilities can be used in conjunction with the Intellec text editor and 8042 macroassembler to facilitate program development.

HARDWARE DEVELOPMENT

The ICE-42 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware.

SYSTEM INTEGRATION

Integration of software and hardware begins when any functional element of the user system hardware is connected to the 8042 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is "system" tested in real-time operation as it becomes available.

SYSTEM TEST

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-42 module is then used for real-time emulation of the 8042 chip to debug the system as a completed unit.

The final product verification test may be performed using the 8742 EPROM version of the 8042 microcomputer. Thus, the ICE-42 module provides the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

Symbolic Debugging

The ICE-42 emulator permits the user to define and to use symbolic, rather than absolute, references to program and data memory addresses. Thus, there is no need to recall or look up the addresses of key locations in the program, or to become involved with machine code.

When a symbol is used for memory reference in an ICE-42 emulator command, the emulator supplies the corresponding location as stored in the ICE-42 emulator symbol table. This table can be loaded with the symbol table produced by the assembler during application program assembly. The user obtains the symbol table during software preparation simply by using the "DEBUG" switch in the 8042 macroassembler. Furthermore, the user interactively modifies the emulator symbol table by adding new symbols or changing or deleting old ones. This feature provides great flexibility in debugging and minimizes the need to work with hexadecimal values.

Through symbolic references in combination with other features of the emulator, the user can easily:

- Interpret the results of emulation activity collected during trace.
- Disassemble program memory to mnemonics, or assemble mnemonic instructions to executable code.
- Reference labels or addresses defined in a user program.

Automated Debugging and Testing

MACRO COMMAND

A macro is a set of commands given a name. A group of commands executed frequently can be defined as a macro. The user executes the group of commands by typing a colon followed by the macro name. Up to ten parameters may be passed to the macro.

Macro commands can be defined at the beginning of a debug session and then used throughout the whole session. One or more macro definitions can be saved on diskette for later use. The Intellec text editor may be used to edit the macro file. The macro definitions are easy to include in any later emulation session.

The power of the development system can be applied to manufacturing testing as well as development by writing test sequences as macros. The macros are stored on diskettes for use during system test.

COMPOUND COMMAND

Compound commands provide conditional execution of commands (IF command) and execution of commands repeatedly until certain conditions are met (COUNT, REPEAT commands).

Compound commands may be nested any number of times, and may be used in macro commands.

Example:

```
*DEFINE .I=0          ; Define symbol .I to 0
*COUNT 100H           ; Repeat the following
                        commands 100H times.
.*IF .I AND 1 THEN    ; Check if .I is odd
..*CBYTE.I=.I         ; Fill the memory at
                        location .I to value .I
..*END
.*.I=.I+1             ; Increment .I by 1.
.*END                 ; Command executes
                        upon carriage-return
                        after END
```

(The Asterisks are system prompts; the dots indicate the nesting level of compound commands.)

Operating Modes

The ICE-42 software is an Intellec RAM-based program that provides easy-to-use commands for initiating emulation, defining breakpoints, controlling trace data collection, and displaying and controlling system parameters. ICE-42 commands are configured with a broad range of modifiers that provide maximum flexibility in describing the operation to be performed.

EMULATION

The ICE-42 module can emulate the operation of prototype 8042 system, at real-time speed (up to 12MHz) or in single steps. Emulation commands to the ICE-42 module control the process of setting up, running, and halting an emulation of the user's 8042-based system. Breakpoints and tracepoints enable the ICE-42 emulator to halt emulation and provide a detailed trace of execution in any part of the user's program. A summary of the emulation commands is shown in Table 1.

Table 1 Major Emulation Commands

Command	Description
GO	Begins real-time emulation and optionally specifies break conditions.
BR0, BR1, BR	Sets or displays either or both Breakpoint Registers used for stopping real-time emulation.
STEP	Performs single-step emulation.
QR0, QR1	Specifies match conditions for qualified trace.
TR	Specifies or displays trace-data collection conditions and optionally sets Qualifier Register (QR0, QR1).
Synchronization Line Commands	Sets and displays status of synchronization line outputs or latched inputs. Used to allow real-time emulation or trace to start and stop synchronously with external events.

Breakpoints

The ICE-42 hardware includes two breakpoint registers that allow halting of emulation when specified conditions are met. The emulator continuously compares the values stored in the breakpoint registers with the status of specified address, opcode, operand, or port values, and halts emulation when this comparison is satisfied. When an instruction initiates a break, that instruction is executed completely before the break takes place. The ICE-42 emulator then regains control of the console and enters the interrogation mode. With the breakpoint feature, the user can request an emulation break when the program:

- Executes an instruction at a specific address or within a range of addresses.

- Executes a particular opcode.
- Receives a specific signal on a port pin.
- Fetches a particular operand from the user program memory.
- Fetches an operand from a specific address in program memory.

Trace and Tracepoints

Tracing is used with real-time and single-step emulation to record diagnostic information in the trace buffer as a program is executed. The information collected includes opcodes executed, port values, and memory addresses. The ICE-42 emulator collects 1000 frames of trace data.

If desired this information can be displayed as assembler instruction mnemonics for analysis during interrogation or single-step mode. The trace-collection facility may be set to run conditionally or unconditionally. Two unique trace qualifier registers, specified in the same way as breakpoint registers, govern conditional trace activity. The qualifiers can be used to condition trace data collection to take place as follows:

- Under all conditions (forever).
- Only while the trace qualifier is satisfied.
- For the frames or instructions preceding the time when a trace qualifier is first satisfied (pre-trigger trace).
- For the frames or instructions after a trace qualifier is first satisfied (post-triggered trace).

Table 2 shows an example of trace display.

INTERROGATION AND UTILITY

Interrogation and utility commands give convenient access to detailed information about the

Table 2 Trace Display (Instruction Mode)

FRAME	LOC	OBJ	INSTRUCTION	P1	P2	T0	T1	DBYIN	YOUT	YSTS	TOVF
0000:	100H	2355	MOV A,#55H	FFH	FFH	0	0	66H	DFH	02H	0
0004:	102H	39	OUTL P1,A	FFH	FFH	0	0	66H	DFH	02H	0
0008:	103H	3A	OUTL P2,A	55H	FFH	0	0	66H	DFH	02H	0
0012:	104H	22	IN A,DBB	55H	55H	0	0	66H		02H	0
0014:	105H	37	CPL A	55H	55H	0	0		DFH	02H	0
0016:	106H	02	OUT DBB,A	55H	55H	0	0	66H		00H	0
0018:	107H	BA03	MOV R2,#03H	55H	55H	0	0	66H	99H	00H	0
0022:	109H	B840	MOV R0,#.TABLE0	55H	55H	0	0	66H	99H	01H	0
0026:	10BH	B960	MOV R1,#.TABLE1	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0030:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0032:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0034:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0036:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0038:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0042:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0044:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0046:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0048:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0050:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
.LOOP											
0054:	10DH	F0	MOV A,@R0	55H	55H	0	0		99H	01H	0
0056:	10EH	A1	MOV @R1,A	55H	55H	0	0	66H		01H	0
0058:	10FH	18	INC R0	55H	55H	0	0		99H	01H	0
0060:	110H	19	INC R1	55H	55H	0	0	66H		01H	0
0062:	111H	EA0D	DJNZ R2,.LOOP	55H	55H	0	0	66H	99H	01H	0
0066:	113H	00	NOP	55H	55H	0	0		99H	01H	0

user program and the state of the 8042 that is useful in debugging hardware and software. Changes can be made in memory and in the 8042 registers, flags, and port values. Commands are also provided for various utility operations such as loading and saving program files, defining symbols, displaying trace data, controlling system synchronization and returning control to ISIS-II. A summary of the basic interrogation and utility commands is shown in Table 3. Two additional time-saving emulator features are discussed below.

Single-Line Assembler/Disassembler

The single-line assembler/disassembler (ASM and DASM commands) permits the designer to examine and alter program memory using assembly language mnemonics, without leaving the emulator environment or requiring time-consuming program reassembly. When assembling new mnemonic instructions into program memory, previously defined symbolic references (from the original program assembly, or subsequently defined during the emulation session)

Table 3 Major Interrogation and Utility Commands

Command	Description
HELP	Displays help messages for ICE-42 emulator command-entry assistance.
LOAD	Loads user object program (8042 code) into user-program memory, and user symbols into ICE-42 emulator symbol table.
SAVE	Saves ICE-42 emulator symbol table and/or user object program in ISIS-II hexadecimal file.
LIST	Copies all emulator console input and output to ISIS-II file.
EXIT	Terminates ICE-42 emulator operation.
DEFINE	Defines ICE-42 emulator symbol or macro.
REMOVE	Removes ICE-42 emulator symbol or macro.
ASM	Assembles mnemonic instructions into user-program memory.
DASM	Disassembles and displays user-program memory contents.
Change/Display Commands	Change or display value of symbolic reference in ICE-42 emulator symbol table, contents of key-word references (including registers, I/O ports, and status flags), or memory references.
EVALUATE	Evaluates expression and displays resulting value.
MACRO	Displays ICE-42 macro or macros.
INTERRUPT	Displays contents for the Data Bus and timer interrupt registers.
SECONDS	Displays contents of emulation timer, in microseconds.
Trace Commands	Position trace buffer pointer and select format for trace display.
PRINT	Displays trace data pointed to by trace buffer pointer.
MODE	Sets or displays the emulation mode, 8041A or 8042.

Table 4 HELP Command

```
*HELP
Help is available for the following items. Type HELP followed by the item name.
The help items cannot be abbreviated. (For more information, type HELP HELP or
HELP INFO.)
Emulation:   Trace Collection:          Misc:          <address>
GO GR SYO    TR   QR   QR0  QR1  SY1   BASE           <CPU*keyword>
BR BR0BR1                              DISABLE        <expr>
STEP         Trace Display:            ENABLE         <ICE42*keyword>
             TRACE   MOVE    PRINT     ERROR          <identifier>
             OLDEST NEWEST             EVALUATE       <instruction>
                                       HELP           <masked*constant>
Change/       Display/ Define/ Remove: INFO           <match*cond>
<CHANGE>     REMOVE   CBYTE            <LIGHTS>       <numeric*constant>
<DISPLAY>    SYMBOL   DBYTE   DASM     LIST           <partition>
REGISTER     RESET            ASM      LOAD           <string>
                                       MODE
SECONDS      WRITE                     SAVE           <string*constant>
DEFINE       STACK            SY       SUFFIX         <symbolic*ref>
                                       SYMBOLIC       <mode>
Macro:              Compound                          <trace*reference>
DEFINE       DIR    Commands:                         <unlimited*match*cond>
DISABLE      ENABLE COUNT                             <user*symbols>
INCLUDE      PUT    IF
<MACRO*DISPLAY>      REPEAT
<MACRO*INVOCATION>
*
*
*HELP IF
IF - The conditional command allows conditional execution of one or more commands
based on the values of boolean conditions.
     IF <expr> 'THEN  <cr>              <true*list>::='<command> <cr> @
      <true*list>                       <false*list>::='<command> <cr> @
      'ORIF <expr> <cr>                 <command>::=An ICE-42 command.
      <true*list> @
      'ELSE <cr>
      <false*list>
      END
The <expr>s are evaluated in order as 16-bit unsigned integers. If one is
reached whose value has low-order bit 1 (TRUE), all commands in the <true*list>
following that <expr> are then executed and all commands in the other <true*-
list>s and in the <false*list> are skipped. If all <expr>s have value with low-
order bit 0 (FALSE), then all commands in all <true*list>s are skipped and, if
ELSE is present, all commands in the <false*list> are executed.
     (EX:  IF .LOOP=5 THEN
             STEP
             ELSE
             GO
             END)
*
*
*
*
*EXIT
```

may be used in the instruction operand field. The emulator supplies the absolute address or data values as stored in the emulator symbol table. These features eliminate user time spent translating to and from machine code and searching for absolute addresses, with a corresponding reduction in transcription errors.

HELP

The HELP file allows display of ICE-42 command syntax information at the Intellec console. By typing "HELP", a listing of all items for which help messages are available is displayed. Typing "HELP <Item>" then displays relevant information about the item requested, including typical usage examples. Table 4 shows some sample HELP messages.

EMULATION ACCURACY

The speed and interface demands of a high-performance single-chip microcomputer require extremely accurate emulation, including full-speed, real-time operation with the full function of the microcomputer. The ICE-42 module achieves accurate emulation with an 8042 emulator chip, a special configuration of the 8042 microcomputer family, as its emulation processor.

Each of the 40 pins on the user plug is connected directly to the corresponding 8042 pin on the emulator chip. Thus the user system sees the emulator as an 8042 microcomputer at the 8042 socket. The resulting characteristics provide extremely accurate emulation of the 8042 including speed, timing characteristics, load and drive values, and crystal operation. However, the emulator may draw more power from the user system than a standard 8042 family device.

Additional emulator processor pins provide signals such as internal address, data, clock, and control lines to the emulator buffer box. These signals let static RAM in the buffer box substitute for on-chip program ROM or EPROM. The emulator chip also gives the ICE module "back-door" access to internal chip operation, allowing the emulator to break and trace execution without interfering with the values on the user-system pins.

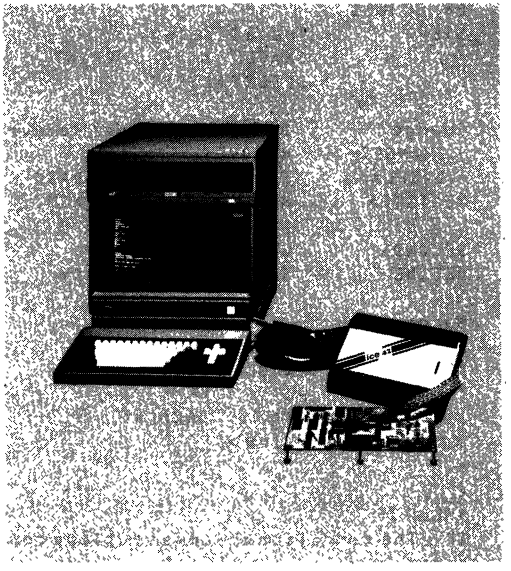


Figure 1 A Typical 8042 Development Configuration. The host system is an Intellec Model 225, plus 1 megabyte dual double-density flexible disk storage. The ICE-42 module is connected to a user prototype system.

SPECIFICATIONS

ICE™-42 Operating Requirements

Intellec Microcomputer Development System (64K RAM required)

System console

Intellec Diskette Operating System (single or double density) ISIS-II (Version 3.4 or later).

Equipment Supplied

- Printed circuit boards (2)
- Emulation buffer box, Intellec interface cables, and user-interface cable with 8042 emulation processor
- Crystal power accessory
- Operating instructions manuals
- Diskette-based ICE-42 software (single and double density)

Emulation Clock

User's system clock (up to 12MHz) or ICE-42 crystal power accessory (12 MHz)

Environmental Characteristics

Operating Temperature — 0° to 40°C

Operating Humidity — Up to 95% relative humidity without condensation.

Physical Characteristics

Printed Circuit Boards

Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)

Buffer Box

Width: 8.00 in. (20.32 cm)
Length: 12.00 in. (30.48 cm)
Depth: 1.75 in. (4.44 cm)
Weight: 4.0 lb. (1.81 kg)

Electrical Characteristics

DC Power Requirements (from Intellec® system)

$V_{CC} = +5V, \pm 5\%$
$I_{CC} = 13.2A$ max; 11.0A typical
$V_{DD} = +12V, \pm 5\%$
$I_{DD} = 0.1A$ max; 0.05A typical
$V_{BB} = -10V, \pm 5\%$
$I_{BB} = 0.05A$ max; 0.01A typical

User plug characteristics at 8042 socket — Same as 8042 or 8742 except that the user system sees an added load of 25 pF capacitance and 50μA leakage from the ICE-42 emulator user plug at ports 1, 2, T0, and T1.

ORDERING INFORMATION

Part Number	Description
ICE-42	8042 Microcontroller In-Circuit Emulator, cable assembly and interactive diskette software

MCS®-48 DISKETTE-BASED SOFTWARE SUPPORT PACKAGE

- **Extends Intellec microcomputer development system to support MCS-48 development**
- **MCS-48 assembler provides conditional assembly and macro capability**
- **Takes advantage of powerful ISIS-II file handling and storage capabilities**
- **Provides assembler output in standard Intel hex format**

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes, and provides both conditional and macroassembler programming. Output may be loaded either to an ICE-49 module for debugging or into the iUP Universal PROM Programmer for 8748 PROM programming. The MCS-48 assembler operates under the ISIS-II operating system on Intel Development systems.

FUNCTIONAL DESCRIPTION

The MCS-48 assembler translates symbolic 8048 assembly language instructions into the appropriate machine operation codes. The ability to refer to program addresses with symbolic names eliminates the errors of hand translation and makes it easier to modify programs when adding or deleting instructions. Conditional assembly permits the programmer to specify which portions of the master source document should be included or deleted in variations on a basic system design, such as the code required to handle optional external devices. Macro capability allows the programmer use of a single label to define a routine. The MCS-48 assembler will assemble the code required by the reserved routine whenever the macro label is inserted in the text. Output from the assembler is in standard Intel hex format. It may be either loaded directly to an in-circuit emulator (ICE-49) module for integrated hardware/software debugging, or loaded into the iUP Universal PROM Programmer for 8748 PROM programming. A sample assembly listing is shown in Table 1.

Table 1. Sample MCS-48 Diskette-Based

```
ISIS II 8048 MACROASSEMBLER V1.0                                    PAGE 1

LOC  OBJ                    SEQ          SOURCE STATEMENT

                             1    ;DECIMAL ADDITION ROUTINE  ADD BCD NUMBER
                             2    ; AT LOCATION 'BETA' TO BCD NUMBER AT 'ALPHA' WITH
                             3    ;RESULT IN 'ALPHA'. LENGTH OF NUMBER IS 'COUNT' DIGIT
                             4    ; PAIRS (ASSUME BOTH BETA AND ALPHA ARE SAME LENGTH
                             5    ; AND HAVE EVEN NUMBER OF DIGITS OR MSD IS 0 IF
                             6    ; ODD)
                             7    INIT    MACRO   AUGND,ADDND,CNT
                             8            MOV     R0 #AUGND
                             9    L1      MOV     R1 #ADDND
                            10            MOV     R2 #CNT
                            11            ENDM
                            12
0001E                       13    ALPHA   EQU     30
0028                        14    BETA    EQU     40
0032                        15    COUNT   EQU     5
0100                        16            ORG     100H
                            17            INIT    ALPHA BETA COUNT
0100 B81E                   18+           MOV     R0 #ALPHA
0102 B928                   19+L1         MOV     R1 #BETA
0104 BA32                   20+           MOV     R2 #COUNT
0106 97                     21            CLR     C
0107 F0                     22    LP      MOV     A @R0
0108 71                     23            ADDC    A @R1
0109 57                     24            DA      A
010A A1                     25            MOV     @R0 A
010B 18                     26            INC     R0
010C 19                     27            INC     R1
010D EA07                   28            DJNZ    R2 LP
                                          END

USER SYMBOLS
ALPHA  0001E    BETA 0028    COUNT 0005    LP 0107
L1     0102

ASSEMBLY COMPLETE  NO ERRORS


ISIS II ASSEMBLER SYMBOL CROSS REFERENCE V1.0                       PAGE 1

SYMBOL CROSS REFERENCE

ALPHA  13#  17
BETA   14#  17
COUNT  15#  17
INIT    7#  17
L1     19#
LP     22#  28
```

SPECIFICATIONS

Operating Environment

(All) Intel Microcomputer Development Systems
(Series II, Series III/Intel equivalent)
Intel Personal Development System

Documentation Package

Titles of: User Guides
Operating Instructions
Reference Manuals

Ordering Information

Part Number	Description
MDS-D48	MCS-48 Disk Based Assembler

Requires Software License

SUPPORT:

Hotline Telephone Support, Software Performance Reports (SPR), Software Updates, Technical Reports, Monthly Newsletters are available.

iUP-200/iUP-201
UNIVERSAL PROM PROGRAMMERS

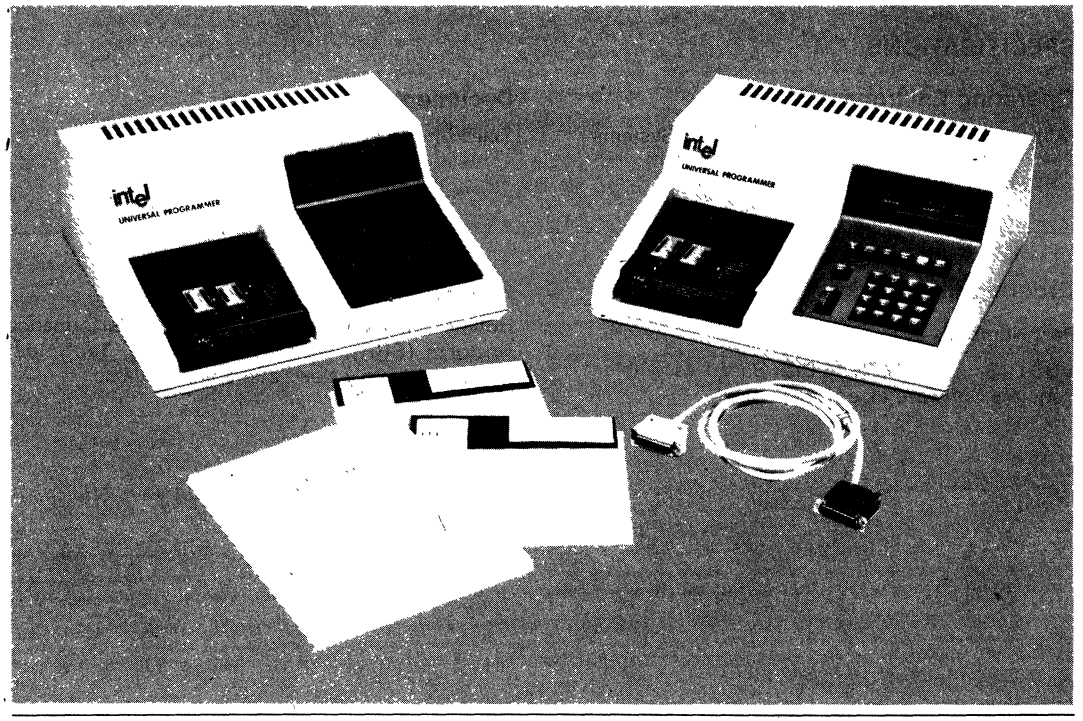
MAJOR iUP-200/iUP-201 FEATURES:

- **Serial interface to all INTELLEC® - Development Systems**
- **Powerful PROM Programming Software utility (iPPS)**
- **Support for all Intel PROM families through multiple device Personality Modules**
- **iUP system self-tests plus device integrity checks**

ADDITIONAL iUP-201 FEATURES:

- **24-character alpha-numeric display**
- **Full hexadecimal plus 11-function keypads**
- **Off-line editing and device duplication**
- **16K bytes RAM expandable to 32K bytes**

The iUP-200 and iUP-201 Universal Prom Programmers provide programming and verification of data in all the Intel programmable ROMs (PROMs). They can also be used for programming the PROM memory portions of Intel's single-chip microcomputer and peripheral devices. When used with any INTELLEC Development System, the iUP-200 and iUP-201 provide on-line programming and verification with the aid of the Intel PROM Programming Software utility (iPPS). In addition, the iUP-201 supports off-line, stand-alone, program editing and PROM duplication. The iUP-200 is completely expandable to the iUP-201.



FUNCTIONAL DESCRIPTION

On-Line System

Hardware Components—The basic iUP-200 and iUP-201 consist of a free-standing unit that, when interfaced directly to any Intel Development System equipped with at least 64K bytes of user memory, provides "on-line" PROM programming and verification of Intel programmable devices. In addition, the units can read the contents of the ROM versions of these devices. Communication with the host is accomplished through a standard RS232C serial data link. A serial converter is needed when using the MDS-800 as a host system. These converters are available from other manufacturers. Each unit contains an 8085 CPU, selectable power supply, 2.3K bytes of static RAM, 8K bytes of pre-programmed EPROM, a programmable timer, and circuitry for interfacing to a Personality Module, keyboard, display, and host system. The pre-programmed EPROM contains the firmware needed for all iUP edit and control functions.

The interface between the iUP and the target PROM is accomplished using a family or single-device Personality Module. No additional sockets or adaptors are necessary. These Personality Modules are iUP front panel inserted units containing all the hardware and firmware necessary for programming either a family of Intel PROMs or a single Intel device. Figure 1 diagrams the on-line system data flow.

The iUP-201 will also accept Intel hexadecimal programs developed on a non-Intel Development System. Only a few keystrokes are required to download the program into iUP RAM for editing and loading into a PROM.

Software Components—The Intel PROM Programming Software utility (iPPS) is included with both the iUP-200 and iUP-201. Created to run on any INTELLEC Development System, iPPS provides user control of all reading, programming, and verification functions through an easy to use language driven interface. All iPPS commands, as well as program address and data information, are entered through the development system ASCII keyboard and displayed on the system CRT. These plain English commands allow the user to read and write data to or from any of three logical devices: the target PROM, the INTELLEC system memory, or a disk file system. Additional commands control iPPS program execution, display information and status, allow rearrangement of data from any of the three logical devices, and provide user assistance information in the form of a HELP command. Figure 2 summarizes these commands.

Loading programs into a PROM from INTELLEC system memory or directly from a disk file is accomplished under iPPS control. Access to the disk allows the user to create and manipulate data in a virtual buffer with an address range up to 16M. This large block of data can be formatted into different PROM word sizes for program storage into several different PROM types. In addition, a program from any of the three logical devices can be "interleaved" with a second program and entered into a specific target PROM or PROMs.

iPPS supports data manipulation in any Intel format: 8080 hexadecimal ASCII, 8080 absolute object, 8086 hexadecimal ASCII, 8086 absolute object, and 286 absolute object. Addresses and data can be displayed in one of several number bases including binary, octal, decimal, and hexadecimal. The user can easily change defaulted data formats as well as number bases.

iPPS requires that version 3.4 or later of Intel's ISIS-II Operating System be resident in INTELLEC Development System memory at the time of execution. The software is designed to run under control of ISIS "Submit Files" thereby freeing the user from repetitious command entry.

System Expansion—The iUP-200 can be easily expanded, by the user, for off-line operation. The Keyboard/Expansion Kit (iUP-PAK) is available from Intel or your local Intel Distributor.

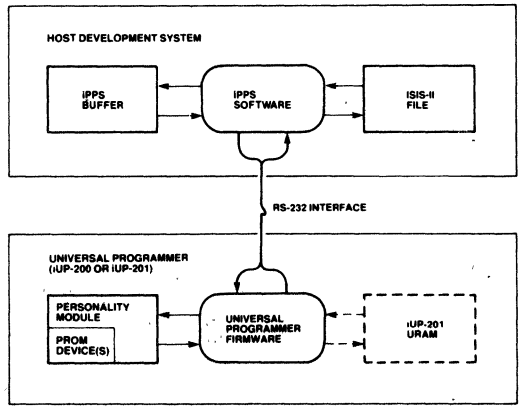


Figure 1. On-Line System Data Flow

Program Control Group—Controls the program execution of iPPS.

EXIT	Exits iPPS and returns control to ISIS-II
<ESC>	Terminates current command
REPEAT	Repeats full execution of previous command
ALTER	Allows edit and re-execution of previous command

Utility Group—Displays user information, status, and sets default values.

DISPLAY	Displays PROM, Buffer, or File data on the console
PRINT	Prints PROM, Buffer or File data on a printer
HELP	Selectively displays user assistance information
MAP	Displays Buffer structure and status
BLANKCHECK	Checks for unprogrammed PROM
OVERLAY	Checks if non-blank PROM can be programmed
TYPE	Selects PROM type
INIT	Initializes the default number base and file type
WORKFILES	Specifies drive device for temporary work files

Buffer Group—Edits, modifies, and verifies data in the Buffer.

SUBSTITUTE	Examines and modifies Buffer data
LOADDATA	Loads a section of the buffer with a constant
VERIFY	Verifies data in PROM with Buffer data

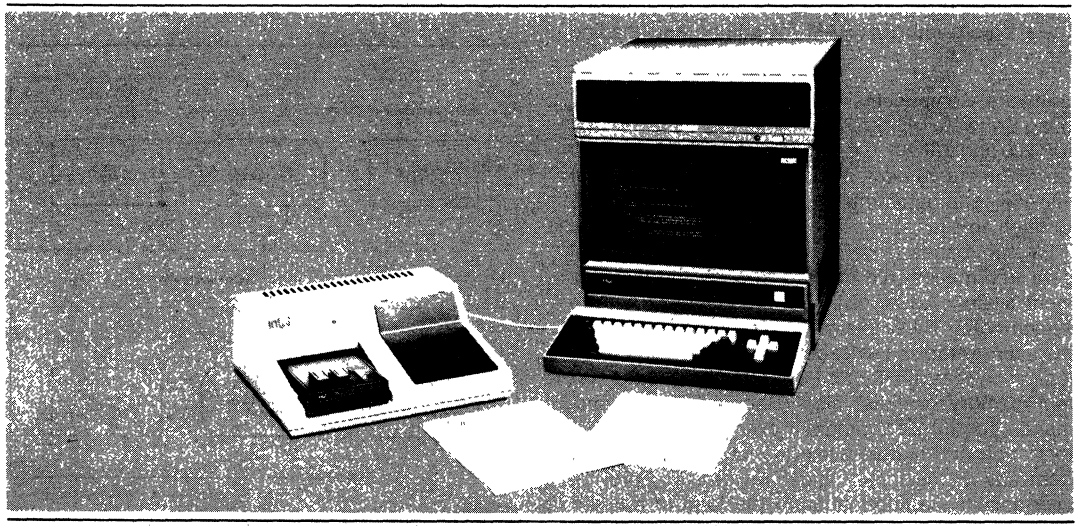
Formatting Group—Permits rearrangement of data from PROM, Buffer, or File.

FORMAT	Interactively formats the Buffer, PROM, or File data and places the result in a workfile

Copy Group—Provides for variations of the general purpose COPY command.

COPY (File to PROM)	Programs PROM with data in a file on disk	
COPY (PROM to File)	Saves PROM data in file on disk	
COPY (Buffer to PROM)	Programs PROM device from Buffer	
COPY (PROM to Buffer)	Loads Buffer with data in PROM	
COPY (Buffer to File)	Saves Buffer in file on disk	
COPY (File to Buffer)	Loads Buffer from file on disk	
COPY (File to URAM)	Loads file data into iUP URAM	(iUP-201 only)
COPY (URAM to File)	Save iUP URAM data in a file	(iUP-201 only)
COPY (Buffer to URAM)	Loads Buffer into iUP URAM	(iUP-201 only)
COPY (URAM to Buffer)	Loads iUP URAM data into the Buffer	(iUP-201 only)

Figure 2. iPPS Command Summary



iUP-200 On-Line System Configuration

Off-Line System

While capable of performing all the on-line functions, the iUP-201 allows program editing, PROM duplication, and program verification independent of the host system. In addition to the hardware components included as part of the iUP-200, the iUP-201 contains a 24-character alphanumeric display, full HEX and 11-function keypads, and 16K bytes of user RAM (URAM) expandable to 32K bytes. This expansion provides memory needed to store data for PROMs exceeding 16K bytes (128K bits) in size. Figure 3 illustrates the iUP-201 keyboard and display.

The two logical devices accessible during off-line operation are the PROM device and iUP-201 RAM. Typical operation would entail copying the data from a PROM (or ROM) into iUP RAM, modifying this data in RAM, and programming the modified data back into a PROM device. The address range of the needed RAM is automatically determined by the iUP when PROM type selection is made.

Figure 4 summarizes the off-line commands.

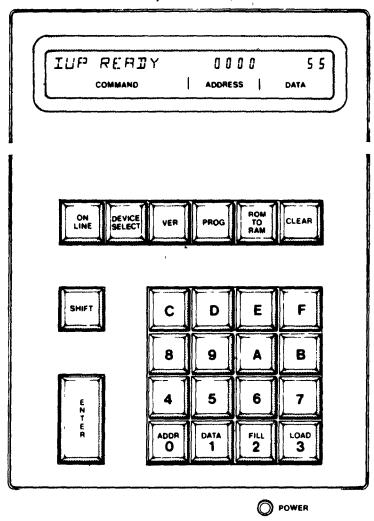


Figure 3. iUP-201 Keyboard and Display

ON LINE	Selects either the on-line or the off line operation. When on-line, all other function keys are disabled.
DEVICE SELECT	Selects the PROM type when a Personality Module capable of programming multiple devices is used. The selected device is indicated by an adjacent LED on the installed module.
VER	Verifies the contents of the installed PROM device with that of the iUP RAM. The iUP display indicates address and the 2's complement of any expected vs. actual mismatch.
PROG	Performs a device Blank Check and then programs the target PROM with data from iUP RAM. If Blank Check fails, pressing PROG again will perform a stuck bit check to further verify PROM/Program compatibility.
ROM TO RAM	Loads the iUP RAM with the data from the PROM device installed in the Personality Module.
CLEAR	Terminates the current off-line function, clears a user entry, or restores the display after an error condition.
ENTER	Pressing the ENTER key transfers information from the iUP display (addresses or data) into URAM.
SHIFT + ADDR 0	Pressing the shift key and ADDR/0 key selects the address field for keypad entry.
SHIFT + DATA 1	Pressing the shift key and DATA/1 key selects the data field for keypad editing and entry.
SHIFT + FILL 2	Pressing the shift key and FILL/2 key selects the fill function, which allows a contiguous section of RAM locations to be loaded with a constant.
SHIFT + LOAD 3	Pressing the shift key and LOAD/3 initiates a download of Intel hexadecimal data from any development system with an RS-232C port.

Figure 4. Off-Line Command Summary

SYSTEM DIAGNOSTICS

Both the iUP-200 and iUP-201 include self-contained system diagnostics that provide verification of system operation and aid the user in fault isolation. Diagnostics are performed on the power supply, CPU, internal firmware ROM, internal RAM, timer, and on the iUP-201 keyboard and URAM. In addition, tests are made on any Personality Module installed in the programmer the first time the module is accessed. They include tests on the power select circuitry and the 2K of module firmware. Easy to read status messages are provided on the development system display in the on-line mode and the iUP-201 display in the off-line mode.

PERSONALITY MODULES

The iUP-200 and iUP-201 interface with a selected PROM (or ROM) through an associated Personality Module. These modules contain all of the hardware and firmware needed to read and program a family of Intel devices. Each module is a single molded unit, front panel inserted on either programmer. No additional adapters or sockets are needed. Figure 5 lists the available modules.

iUP-F27/128 - E^2/EPROM Personality Module capable of reading and programming the 2716, 2732, 2732A, 2764, 27128, 2815, and 2816.

iUP-F87/51 - MICROCONTROLLER Personality Module capable of reading and programming the 8748, 8748H, 8048, 8749, 8049, 8750, 8050, 8751, and 8051.

iUP-F87/44 - PERIPHERAL Personality Module capable of reading and programming the 8741A, 8041A, 8742, 8042, 8744, 8044, and 8755A.

iUP-F36/32 - BIPOLAR Personality Module capable of reading and programming the 3628, 3632, 3632A, 3636, 3636B, and 3624.

Figure 5. iUP Personality Modules

Interfaces

Each personality module, an example is shown in Figure 6, interfaces with the programmer through a 41-pin connector. Module firmware is uploaded into iUP RAM and executed by the onboard 8085A processor. This firmware contains routines needed to Read and Program a number of PROMs. In addition, the personality module sends specific information regarding the selected PROM to the iUP to aid in performing PROM device integrity checks.

Operational status is indicated through individual LEDs on each module. A column of device selection LEDs indicate which PROM device type the user has selected. After device selection, an LED below each socket (on modules containing more than one socket) indicates the socket to be used. A red indicator light (Hot Socket) warns the user when power is being supplied to the selected device.

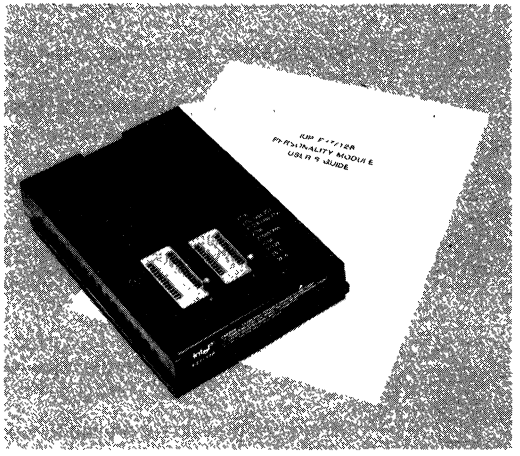


Figure 6. iUP-F27/128

Device Integrity Checks

In addition to the iUP system self-tests, each Personality Module contains diagnostics in firmware that perform selected PROM tests and indicate status. These tests are performed in both the on-line and off-line modes. A PROM installation test is performed to insure the device is installed in the module correctly and the ZIF socket is closed. A PROM Blank Check is

performed to determine whether a device is in its erased state. The iUP automatically determines whether this erased state is all zeros or all ones. A stuck bit check is performed when a PROM is found to be not blank. This test determines which bits are pre-programmed, compares those bits against the program to be loaded, and allows programming to continue if they match. As with the system self-tests, easy to read status messages are provided. All of the PROM device integrity checks, with the exception of the installation test which occurs automatically any time an operation is selected, can be invoked by the user.

Figure 7 illustrates a typical on-line and off-line programming sequence.

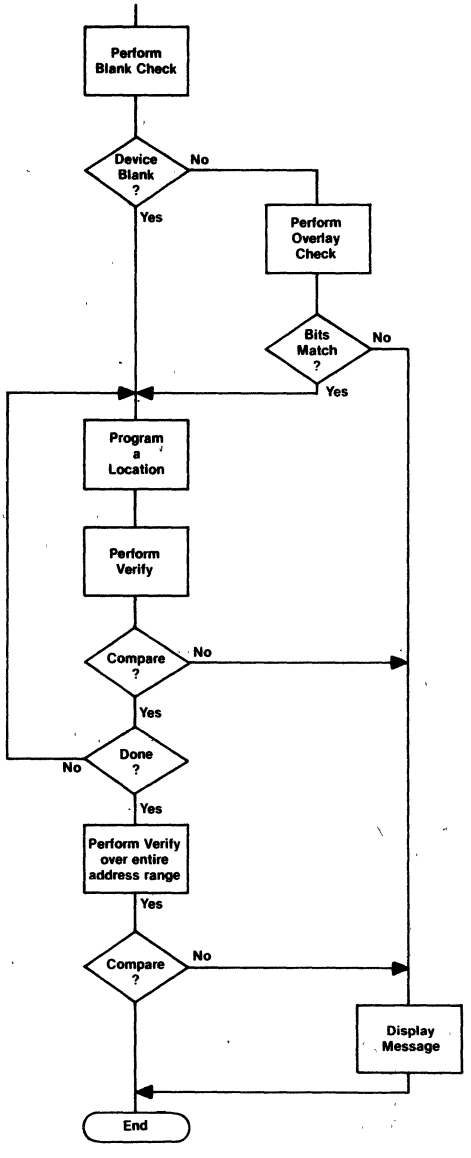


Figure 7. iUP Programming Sequence

iUP-200/201 SPECIFICATIONS

Control Processor

Intel 8085A Microprocessor
6.144 MHz Clock Rate

Memory

RAM—2.3K bytes Static
ROM—8K bytes EPROM

Interfaces

Keyboard—16 character Hexadecimal and 11-function keypad (iUP-201 only)
Display—24 Character Alphanumeric (iUP-201 only)

Software

Monitor—System Controller in pre-programmed EPROM
iPPS—Intel PROM Programming Software utility on supplied diskette

Physical Characteristics

Depth—15 inches (38.1 cm)
Width—15 inches (38.1 cm)
Height—6 inches (15.2 cm)
Weight—15 lbs. (6.8 kg)

Electrical Characteristics

Selectable 100, 120, 200, or 240 Vac ± 10%; 50 - 60 Hz
Maximum power consumption—80 watts

Environmental Characteristics

Operating Temperature—10°C to 40°C
Operating Humidity—0% to 95% Relative Humidity

Reference Material

iUP-200/201 Universal Programmer User's Guide
iUP-200/201 Pocket Reference Card

PERSONALITY MODULE SPECIFICATIONS

Memory

EPROM — 2K bytes

Physical Characteristics

Width — 5.5 inches (14.0 cm)
Height — 1.6 inches (4.1 cm)
Depth — 7.0 inches (17.8 cm)
Weight — 1 lb. (.45 kg)

Electrical Characteristics

Maximum power consumption (module)—5 watts
Maximum power consumption (device)—2.5 watts
Maximum power consumption (total from iUP)—7.5 watts

Environmental Characteristics

Operating Temperature—10°C to 40°C
Operating Humidity—0% to 95% relative humidity

Reference Material

Selected Personality Module User's Guide

ORDERING INFORMATION

Part Number	Description
iUP-200	Intel On-Line Universal Programmer
iUP-201	Intel On-Line/Off-Line Universal Programmer
iUP-F27/128	E^2EPROM Personality Module
iUP-F87/51	MICROCONTROLLER Personality Module
iUP-F87/44	PERIPHERAL Personality Module
iUP-F36/32	BIPOLAR Personality Module

Data Communications 7

Peripherals Section

INTEL DATA COMMUNICATIONS FAMILY OVERVIEW

Data Communications has become an increasingly important factor in computer system design with the evolution of distributed processing and remote, networked peripherals. Intel's data communications product line provides a range of components to satisfy the broad spectrum of speed, protocol support and protocol flexibility needs (Figure 1).

GLOBAL DATA COMMUNICATIONS: ASYCHRONOUS AND SYNCHRONOUS PROTOCOLS

Dedicated data communications controllers

For low-to-medium speed (up to 19.2 Kbps), the 8251A USART (Universal Synchronous Asynchronous Receiver/Transmitter) is the industry standard for asynchronous communications. It can be used in such applications as personal computers, workstations, word processors, CRT terminals point-of-sale terminals, banking terminals, printers, communications processors, data concentrators, industrial control networks, etc.

The 8256 MUART (Multi-function Universal Asynchronous Receiver/Transmitter) is an highly competent asynchronous communications controller. It considerably minimizes the number of LSI required in a system with an asynchronous interface. The 8256 integrates the four more common peripheral functions of a microprocessor based system as well as a full-duplex, double buffered serial asynchronous receiver/transmitter with an on-chip baud rate generator.

The 8273 is a dedicated high level peripheral controller for SDLC/HDLC protocol support. It provides an high level of Data Link Control support for IBM-SNA or CCITT X.25 compatible microcomputer systems. This device minimizes CPU overhead by supporting a comprehensive frame level operation. The 8273 is compatible with every telephone network-based communication system due to its speed (up to 64 Kbps) and flexible modem interface.

Multiprotocol controllers

Multi-protocol controllers bridge the gap between byte oriented and bit oriented protocols (HDLC/SDLC). They provide an easy migration path for the user through a single software reconfiguration. Design of high-level protocols like X.25 are considerably simplified when they are coupled with the power of high performance processors such as the iAPX 86/88/186, or 188. They are also used to implement custom high-level protocols on top of standard bit-synchronous protocols.

The dual-channel 8274 MPSC (Multi-Protocol Serial

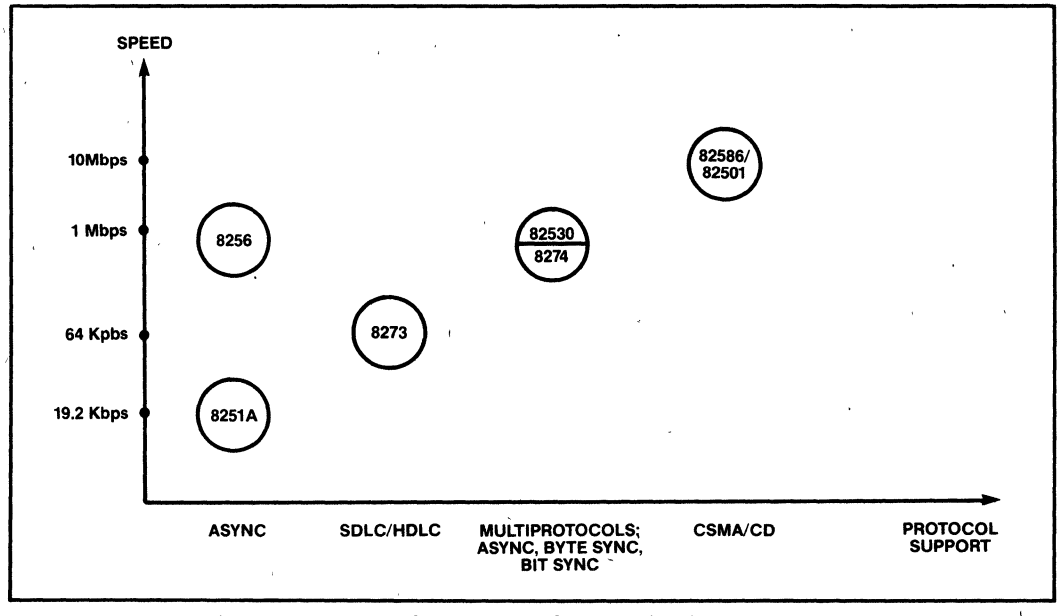


FIGURE 1: A Spectrum of Data Communications Solutions

Controller) provide a solution for Asynchronous, Byte Synchronous (IBM Bisync) and Bit Synchronous (HDLC/SDLC) protocols support. It is optimized for high-speed applications requiring the flexibility of the protocol support and the integration of multiple communications channels.

The 82530 SCC (Serial Communications Controller) is another dual channel multiprotocol controller. It contains new functions including on-chip baud rate generators, digital phase locked loops, various data encoding/-decoding schemes and extensive diagnostic capabilities. All these added features reduce the need for external logic and greatly improve the reliability and maintainability of the system.

Distributed Intelligence Systems

The 8044/8744 is a microcontroller with an on chip serial communication processor. It simplifies control of remote subsystems (subsystems that are physically separated from the host CPU and communicate over a serial link).

The 8044 and 8051 CPUs are identical. The serial communication is handled by an additional processor called the Serial Interface Unit (SIU). The SIU operates concurrently with the CPU and offers a high level of intelligence and performance for HDLC/SDLC based communications. The SIU can handle 2.4 Mbps in Half-Duplex mode.

In addition to controlling communications with the host CPU, the 8044 provides significant peripheral control. Examples include local keyboard, CRT and printer control as well as design of network for Distributed Intelligence Systems (Medical instrumentation, CATV, PABX, etc. . . .)

Detailed 8044/8744 information is contained in the Intel Microcontroller Handbook.

Instrumentation

The 8291A, 8292, and 8293 family of components provide complete, high-performance support for IEEE-488 (GPIB) standard interface. GPIB is used in instrumentation applications.

The 8291A implements the Talker/Listener functions of the GPIB.

The 8292 provides the controller functions. Operating in tandem with the 8291A, it complements its interface functions to provide a full-capability GPIB interface.

The 8293 is a low-power, high-current, HMOS 8-line transceiver. It provides the electrical interface to the GPIB.

Local Area Networks

Intel has developed the first complete VLSI solution for Local Area Networks (LANs) and Ethernet in particular: the 82586 Local Area Network Coporcessor and the 82501 ESI (Ethernet Serial Interface).

Four on chip DMA channels allow the 82586 to operate as a bus master. The 82586 manages the entire process of transmitting and receiving frames, thereby relieving the host processor of the tasks of managing the communication interface to the network.

An extensive set of diagnostic capabilities, implemented in silicon, simplifies the design of more reliable local networks and facilitates their maintenance. In order to take full advantage of the LAN concept and CSMA/CD access method, the 82586 architecture is software configurable. This allows the 82586 to be "customized" for other applications including serial backplanes (serial peripheral interconnection), low cost short distance LANs, broadband networks and medium speed (1–2 Mbps) LANs.

The 82501 is designed to work directly with the 82586 in Ethernet applications. The major functions of the ESI are to generate the 10 MHz transmit clock for the 82586, to perform Manchester encoding/decoding of transmitted/-received frames, and to provide the electrical interface to the Ethernet transceiver cable

The Intel Data Communications product family provides a wide range of solutions for the needs of data communications systems.

intel®

APPLICATION NOTE

AP-16

Using The 8251 Universal Synchronous/Asynchronous Receiver/ Transmitter

Lionel Smith
Microcomputer Applications

PRICE $1.00

INTRODUCTION

The Intel 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) which is capable of operating with a wide variety of serial communication formats. Since many peripheral devices are available with serial interfaces, the 8251 can be used to interface a microcomputer to a broad spectrum of peripherals, as well as to a serial communications channel. The 8251 is part of the MCS-80™ Microprocessor Family, and as such it is capable of interfacing to the 8080 system with a minimum of external hardware.

This application note describes the 8251 as a component and then explains its use in sample applications via several examples. A specific use of the 8251 to facilitate communication between two MCS-80 systems is discussed in detail from both the hardware and software viewpoints. The first two sections of this application note describe the 8251 first from a functional standpoint and then on a detailed level. The function of each input and output pin is fully defined. The next section describes the various operating modes and how they can be selected, and finally, a sample design is discussed using the 8251 as a data link between the MCS-80 systems.

COMMUNICATION FORMATS

Serial communications, either on a data link or with a local peripheral, occurs in one of two basic formats; asynchronous or synchronous. These formats are similar in that they both require framing information to be added to the data to enable proper detection of the character at the receiving end. The major difference between the two formats is that the asynchronous format requires framing information to be added to each character, while the synchronous format adds framing information to blocks of data, or messages. Since the synchronous format is more efficient than the asynchronous format but requires more complex decoding, it is typically found on high-speed data links, while the asynchronous format is used on lower speed lines.

The asynchronous format starts with the basic data bits to be transmitted and adds a "START" bit to the front of them and one or more "STOP" bits behind them as they are transmitted. The START bit is a logical zero, or SPACE, and is defined as the positive voltage level by RS-232-C. The STOP bit is a logical one, or MARK, and is defined as the negative voltage level by RS-232-C. In current loop applications current flow normally indicates a MARK and lack of current a SPACE. The START bit tells the receiver to start assembling a character and allows the receiver to synchronize itself with the transmitter. Since this synchronization only has to last for the duration of the character (the next character will contain a new START bit), this method works quite well assuming a properly designed receiver. One or more STOP bits are added to the end of the character to ensure that the START bit of the next character will cause a transition on the communication line and to give the receiver time to "catch up" with the transmitter if its basic clock happens to be running slightly slower than that of the transmitter. If, on the other hand, the receiver clock happens to be running slightly faster than the transmitter clock, the receiver will perceive gaps between characters but will still correctly decode the data. Because of this tolerance to minor frequency deviations, it is not necessary that the transmitter and receiver clocks be locked to the identical frequency for successful asynchronous communication.

The synchronous format, instead of adding bits to each character, groups characters into records and adds framing characters to the record. The framing characters are generally known as SYN characters and are used by the receiver to determine where the character boundaries are in a string of bits. Since synchronization must be held over a fairly long stream of data, bit synchronization is normally either extracted from the communication channel by the modem or supplied from an external source.

An example of the synchronous and asynchronous formats is shown in Figure 1. The synchronous format shown is fairly typical in that it requires two SYN characters at the start of the message. The asynchronous format, also typical, requires a START bit preceding each character and a single STOP bit following it. In both cases, two 8-bit characters are to be transmitted. In the asynchronous mode 10*n bits are used to transmit n characters and in the synchronous mode 8N + 16 bits are used. For the example shown the asynchronous mode is actually more efficient, using 20 bits versus 32. To transmit a thousand characters in the asynchronous mode, however, takes 10,000 bits versus 8,016 for the synchronous format mode. For long messages the synchronous format becomes much more efficient than the asynchronous format; the crossover point for the examples shown in Figure 1 is eight characters, for which both formats require 80 bits.

In addition to the differences in format between synchronous and asynchronous communication, there are differences with regards to the type of modems that can be used. Asynchronous modems typically employ FSK (Frequency Shift Keying) techniques which simply generate one audio tone for a MARK and another for a SPACE. The receiving modem detects these tones on the telephone

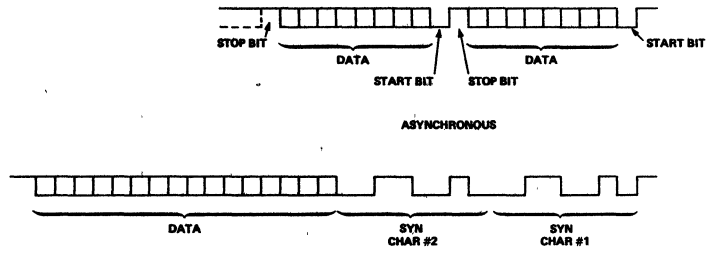


Figure 1. Transmission Formats

line, converts them to logical signals, and presents them to the receiving terminal. Since the modem itself is not concerned with the transmission speed, it can handle baud rates from zero to its maximum speed. Synchronous modems, in contrast to asynchronous modems, supply timing information to the terminal and require data to be presented to them in synchronism with this timing information. Synchronous modems, because of this extra clocking, are only capable of operating at certain preset baud rates. The receiving modem, which has an oscillator running at the same frequency as the transmitting modem, phase locks its clock to that of the transmitter and interprets changes of phase as data.

In some cases it is desirable to operate in a hybrid mode which involves transmitting data with the asynchronous format using a synchronous modem. This occurs when an increase in operating speed is required without a change in the basic protocol of the system. This hybrid technique is known as isosynchronous and involves the generation of the start and stop bits associated with the asynchronous format, while still using the modem clock for bit synchronization.

The 8251 USART has been designed to meet a broad spectrum of requirements in the synchronous, asynchronous, and isosynchronous modes. In the synchronous mode the 8251 operates with 5, 6, 7, or 8-bit characters. Even or odd parity can be optionally appended and checked. Synchronization can be achieved either externally via added hardware or internally via SYN character detection. SYN detection can be based on one or two characters which may or may not be the same. The single or double SYN characters are inserted into the data stream automatically if the software fails to supply data in time. The automatic generation of SYN characters is required to prevent the loss of synchronization. In the asynchronous mode the 8251 operates with the same data and parity structures as it does in the synchronous mode. In addition to appending a START bit to this data, the 8251 appends 1, 1½, or 2 STOP bits. Proper framing is checked by the receiver and a status flag set if an error occurs. In the asynchronous mode the USART can be programmed to accept clock rates of 16 or 64 times the required baud rate. Isosynchronous operation is a special case of asynchronous with the multiplier rate programmed as one instead of 16 or 64. Note that X1 operation is only valid if the clocks of the receiver and transmitter are synchronized.

The 8251 USART can transmit the three formats in half or full duplex mode and is double-buffered internally (i.e., the software has a complete character time to respond to a service request). Although the 8251 supports basic data set control signals (e.g., DTR and RTS), it does not fully support the signaling described in EIA-RS-232-C. Examples of unsupported signals are Carrier Detect (CF), Ring Indicator (CE), and the secondary channel signals. In some cases an additional port will be required to implement these signals. The 8251 also does not interface to the voltage levels required by EIA-RS-232-C; drivers and receivers must be added to accomplish this interface.

BLOCK DIAGRAM

A block diagram of the 8251 is shown in Figure 2. As can be seen in the figure, the 8251 consists of five major sections which communicate with each other on an internal data bus. The five sections are the receiver, transmitter, modem control, read/write control, and I/O Buffer. In order to facilitate discussion, the I/O Buffer has been shown broken down into its three major subsections: the status buffer, the transmit data/command buffer, and the receive data buffer.

Receiver

The receiver accepts serial data on the RxD pin and converts it to parallel data according to the appropriate format. When the 8251 is in the asynchronous mode and it is ready to accept a character

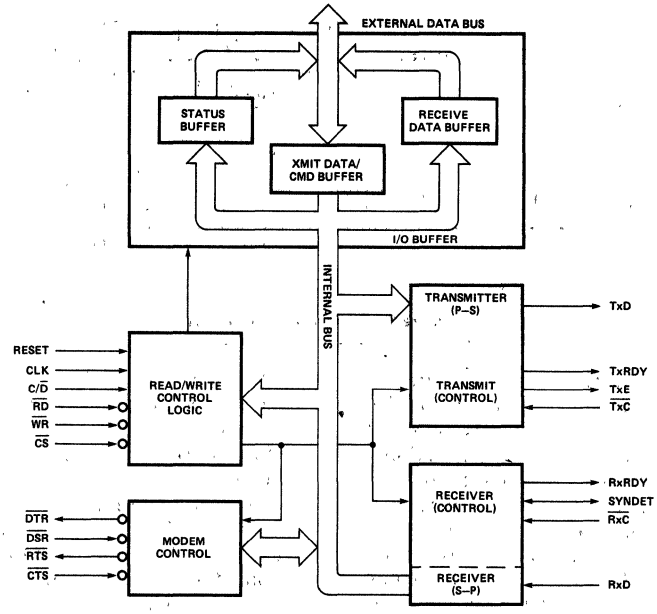


Figure 2. 8251 Block Diagram

(i.e., it is not in the process of receiving a character), it looks for a low level on the RxD line. When it sees the low level, it assumes that it is a START bit and enables an internal counter. At a count equivalent to one-half of a bit time, the RxD line is sampled again. If the line is still low, a valid START bit has probably been received and the 8251 proceeds to assemble the character. If the RxD line is high when it is sampled, then either a noise pulse has occurred on the line or the receiver has become enabled in the middle of the transmission of a character. In either case the receiver aborts its operation and prepares itself to accept a new character. After the successful reception of a START bit the 8251 clocks in the data, parity, and STOP bits, and then transfers the data on the internal data bus to the receive data register. When operating with less than 8 bits, the characters are right-justified. The RxRDY signal is asserted to indicate that a character is available.

In the synchronous mode the receiver simply clocks in the specified number of data bits and transfers them to the receiver buffer register, setting RxRDY. Since the receiver blindly groups data bits into characters, there must be a means of synchronizing the receiver to the transmitter so that the proper character boundaries are maintained in the serial data stream. This synchronization is achieved in the HUNT mode.

In the HUNT mode the 8251 shifts in data on the RxD line one bit at a time. After each bit is received, the receiver register is compared to a register holding the SYN character (program loaded). If the two registers are not equal, the 8251 shifts in another bit and repeats the comparison. When the registers compare as equal, the 8251 ends the HUNT mode and raises the SYNDET line to indicate that it has achieved synchronization. If the USART has been programmed to operate with two SYN characters the process is as described above, except that two contiguous characters from the line must compare to the two stored SYN characters before synchronization is declared. Parity is not checked. If the USART has been programmed to accept external synchronization, the SYNDET pin is used as an input to synchronize the receiver. The timing necessary to do this is discussed in the SIGNALS section of this note. The USART enters the HUNT mode when it is initialized into the synchronous mode or when it is commanded to do so by the command instruction. Before the receiver is operated, it must be enabled by the RxE bit (D_2) of the command instructions. If this bit is not set the receiver will not assert the RxRDY bit.

Transmitter

The transmitter accepts parallel data from the processor, adds the appropriate framing information, serializes it, and transmits it on the TxD pin. In the asynchronous mode the transmitter always

adds a START bit; depending on how the unit is programmed, it also adds an optional even or odd parity bit, and either 1, 1½, or 2 STOP bits. In the synchronous mode no extra bits (other than parity, if enable) are generated by the transmitter unless the computer fails to send a character to the USART. If the USART is ready to transmit a character and a new character has not been supplied by the computer, the USART will transmit a SYN character. This is necessary since synchronous communications, unlike asynchronous communications, does not allow gaps between characters. If the USART is operating in the dual SYN mode, both SYN characters will be transmitted before the message can be resumed. The USART will not generate SYN characters until the software has supplied at least one character; i.e., the USART will fill 'holes' in the transmission but will not initiate transmission itself. The SYN characters which are to be transmitted by the USART are specified by the software during the initialization procedure. In either the synchronous or asynchronous modes, transmission is inhibited until TxEnable and the $\overline{CTS}$ input are asserted.

An additional feature of the transmitter is the ability to transmit a BREAK. A BREAK is a period of continuous SPACE on the communication line and is used in full duplex communication to interrupt the transmitting terminal. The 8251 USART will transmit a BREAK condition as long as bit 3 (SBRK) of the command register is set.

Modem Control

The modem control section provides for the generation of $\overline{RTS}$ and the reception of $\overline{CTS}$. In addition, a general purpose output and a general purpose input are provided. The output is labeled $\overline{DTR}$ and the input is labeled $\overline{DSR}$. $\overline{DTR}$ can be asserted by setting bit 2 of the command instruction; $\overline{DSR}$ can be sensed as bit 7 of the status register. Although the USART itself attaches no special significance to these signals, DTR (Data Terminal Ready) is normally assigned to the modem, indicating that the terminal is ready to communicate and DSR (Data Set Ready) is a signal from the modem indicating that it is ready for communications.

I/O Control

The Read/Write Control Logic decodes control signals on the 8080 control bus into signals which gate data on and off the USART's internal bus and controls the external I/O bus (DB_0–DB_7). The truth table for these operations is as follows:

If neither $\overline{READ}$ or $\overline{WRITE}$ is a zero, then the USART will not perform an I/O function. $\overline{READ}$ and $\overline{WRITE}$ being a zero at the same time is an illegal state with undefined results. The Read/Write Control Logic contains synchronization circuits so that the $\overline{READ}$ and $\overline{WRITE}$ pulses can occur at any time with respect to the clock inputs to the USART.

CE	C/$\overline{D}$	$\overline{READ}$	$\overline{WRITE}$	Function
0	0	0	1	CPU Reads Data from USART
0	1	0	1	CPU Reads Status from USART
0	0	1	0	CPU Writes Data to USART
0	1	1	0	CPU Writes Command to USART
1	X	X	X	USART Bus Floating (NO-OP)

The I/O buffer contains the STATUS buffer, the RECEIVE DATA buffer and the XMIT DATA/CMD buffer as shown in Figure 2. Note that although there are two registers which store data for transfer to the CPU (STATUS and RECEIVE DATA), there is only one register which stores data being transferred to the USART. The sharing of the input register for both transmit data and commands makes it important to ensure that the USART does not have data stored in this register before sending a command to the device. The TxRDY signal can be monitored to accomplish this. Neither data nor commands should be transferred to the USART if TxRDY is low. Failure to perform this check can result in erroneous data being transmitted.

INTERFACE SIGNALS

The interface signals of the 8251 USART can be broken down into two groups – a CPU-related group and a device-related group. The CPU-related signals have been designed to optimize the attachment of the 8251 to a MCS-80™ system. The device-related signals are intended to interface a modem or like device. Since many peripherals (TTY, CRT, etc.) can be obtained with a modem-like interface, the USART has a broad range of applications which do not include a modem. Note that although the USART provides a logical interface to an EIA-RS-232 device, it does not provide EIA compatible drive, and this must be added via circuitry external to the 8251. As an example of a peripheral interface application and to aid in understanding the signal descriptions which follow, Figure 3 shows a system configured to interface with a TTY or CRT.

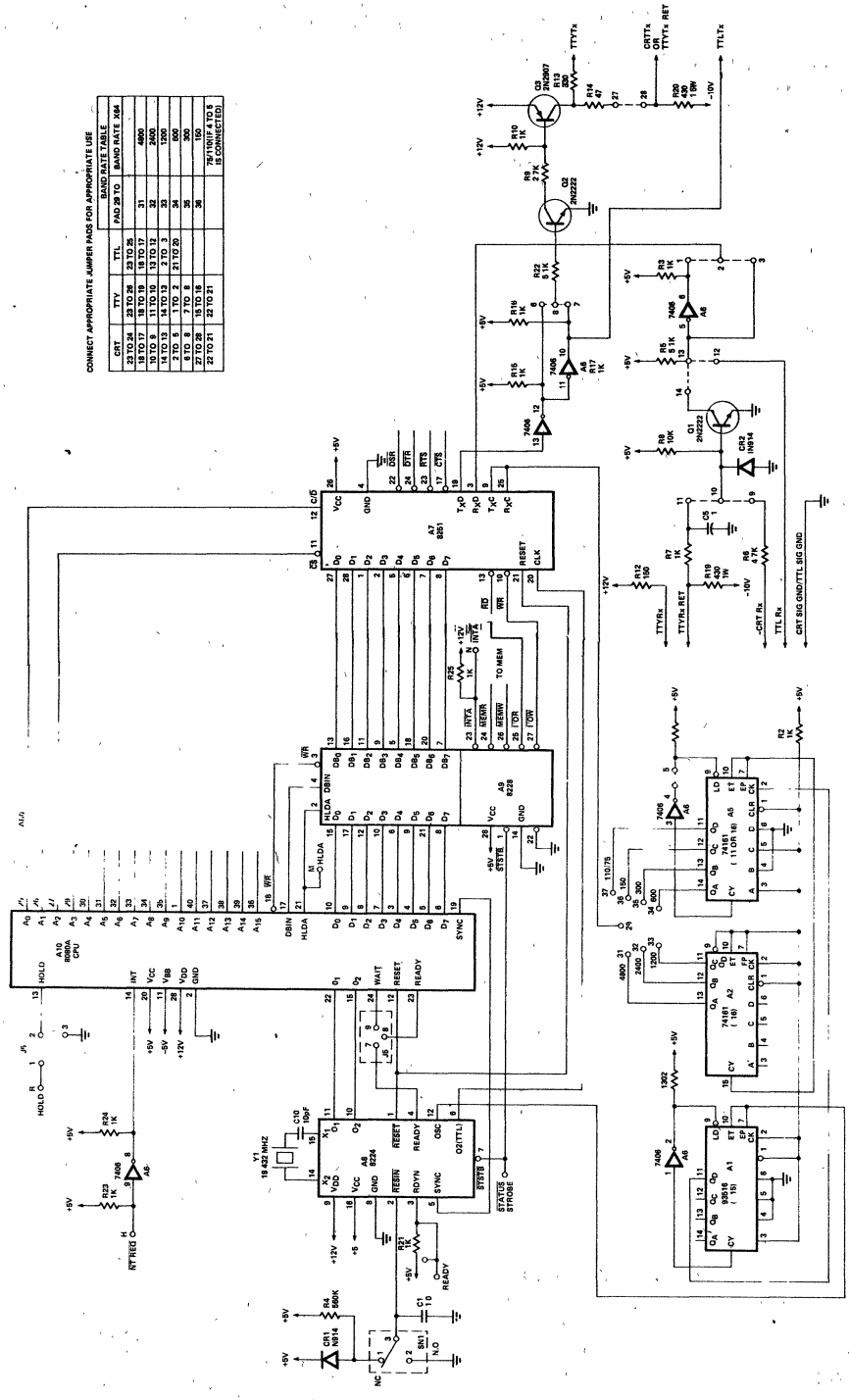


Figure 3. Terminal Interface

CPU-Related Signals

V_{CC} (26)	I	+5 Volt Supply
GND (4)	I	+5 Volt Common
CLK (20)	I	The CLK input generates internal device timing. No external inputs or outputs are referenced to CLK, but the frequency of CLK must be greater than 30 times the Receiver or Transmitter clock inputs for synchronous mode or 4.5 times the clock inputs for an asynchronous mode. An additional constraint is imposed by the electrical specifications (ref. Appendix B) which require the period of CLK be between 0.42 μsec and 1.35 μsec. The CLK input can generally be connected to the Phase 2 (TTL) output of the 8224 clock generator.
RESET (21)	I	A high on this input performs a master reset on the 8251. The device returns to the idle mode and will remain there until reinitialized with the appropriate control words.
DB_7–DB_0 (8,7,6,5,2,1, 28,27)	I/O	The DB signals form a three-state bus which can be connected to the CPU data bus. Control, status, and data are transferred on this bus. Note that the CPU always remains in control of the bus and all transfers are initiated by it.
$\overline{CS}$ (11)	I	*Chip Select.* A low on this input enables communication between the USART and the CPU. Chip Select should go low when the USART is being addressed by the CPU.
C/$\overline{D}$ (12)	I	*Control/Data.* During a read operation this pin selects either status or data to be input to the CPU (high=status, low=data). During a write operation this pin causes the USART to interpret the data on the bus as a command if it is high or as data if it is low.
$\overline{RD}$ (13)	I	A low on this input causes the USART to gate either status or data onto the data bus.
$\overline{WR}$ (10)	I	A low on this input causes the USART to accept data on the data bus as either a command or as a data character.
TxRDY (15)	O	*Transmitter Ready.* This output signals the CPU that the USART is ready to accept a data character or command. It can be used as an interrupt to the system or, for polled operation, the CPU can check TxRDY using the status read operation. Note, however, that while the TxRDY status bit will be asserted whenever the XMIT DATA/CMD buffer is empty, the TxRDY output will be asserted only if the buffer is empty and the USART is enabled to transmit (i.e., $\overline{CTS}$ is low and TxEN is high). TxRDY will be reset when the USART receives a character from the program.
TxE (18)	O	*Transmitter Empty.* A high output on this line indicates that the parallel to serial converter in the transmitter is empty. In the synchronous mode, if the CPU has failed to load a new character in time, TxE will go high momentarily as SYN characters are loaded into the transmitter to fill the gap in transmission.
RxRDY (14)	O	*Transmitter Ready.* This output goes high to indicate that the 8251 has received a character on its serial input and is ready to transfer it to the CPU. Although the receiver runs continuously, RxRDY will only be asserted if the RxE (Receive Enable) bit in the command register has been set. RxRDY can be connected to the interrupt structure or, for polled operation, the CPU can check the condition of RxRDY using a status read operation. RxRDY will be reset when the character is read by the CPU.

SYNDET (16) I/O *Synch Detect.* This line is used in the synchronous mode only. It can be either an input or output, depending on whether the initialization program sets the USART for external or internal synchronization. SYNDET is reset to a zero by RESET. When in the internal synchronization mode, the USART uses SYNDET as an output to indicate that the device has detected the required SYN character(s). A high output indicates synchronization has been achieved. If the USART is programmed to operate with double SYN characters, SYNDET will go high in the middle of the last bit of the second SYN character. SYNDET will be reset by a status read operation. When in the external synchronization mode a positive-going input on the SYNDET line will cause the 8251 to start assembling characters on the next falling edge of $\overline{RxC}$. The high input should be maintained at least for one RxC cycle following this edge.

Device-Related Signals

$\overline{DTR}$ (24) O *Data Terminal Ready.* This is a general purpose output signal which can be set low by programming a '1' in command instruction bit 1. This signal allows additional device control.

$\overline{DSR}$ (22) I *Data Set Ready.* This is a general purpose input signal. The status of this signal can be tested by the CPU through a status read. This pin can be used to test device status and is read as bit 7 of the status register.

$\overline{RTS}$ (23) O *Request to Send.* This is a general purpose output signal equivalent to $\overline{DTR}$. RTS is normally used to request that the modem prepare itself to transmit (i.e., establish carrier). $\overline{RTS}$ can be asserted (brought low) by setting bit 5 in the command instruction.

$\overline{CTS}$ (17) I *Clear to Send.* A low on this input enables the USART to transmit data. CTS is normally generated by the modem in response to a $\overline{RTS}$.

$\overline{RxC}$ (25) I *Receiver Clock.* This clock controls the data rate of characters to be received by the USART. In the synchronous mode RxC is equivalent to the baud rate, and is supplied by the modem. In asynchronous mode $\overline{RxC}$ is 1, 16, or 64 times the baud rate. The clock division is preselected by the mode control instruction. Data is sampled by the USART on the rising edge of $\overline{RxC}$.

RxD (3) I *Receiver Data.* Characters are received serially on this pin and assembled into parallel characters. RxD is high true (i.e., High = MARK or ONE).

$\overline{TxC}$ (9) I *Transmitter Clock.* This clock controls the rate at which characters are transmitted by the USART. The relationship between clock rate and baud rate is the same as for RxC. Data is shifted out of the USART on the falling edge of TxC.

TxD (19) O *Transmit Data.* Parallel characters sent by the CPU are transmitted serially by the USART on this line. TxD is high true (i.e., High = MARK or ONE).

MODE SELECTION

The 8251 USART is capable of operating in a number of modes (e.g., synchronous or asynchronous). In order to keep the hardware as flexible as possible (both at the chip and end product level), these operating modes are selected via a series of control outputs to the USART. These mode control outputs must occur between the time the USART is reset and the time it is utilized for data transfer. Since the USART needs this information to structure its internal logic it is essential to complete the initialization before any attempts are made at data transfer (including reading status).

A flowchart of the initialization process appears in Figure 4. The first operation which must occur following a reset is the loading of the mode control

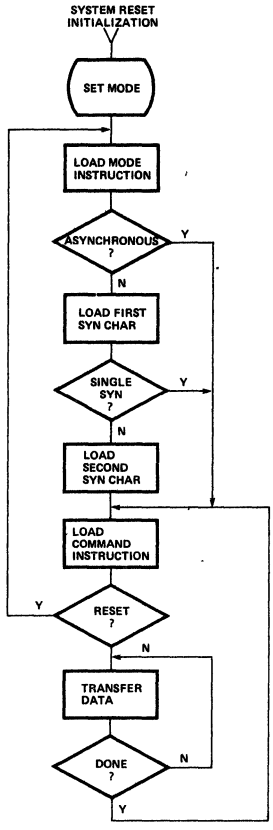


Figure 4. Initialization Flowchart

register. The mode control register is loaded by the first control output (C/$\overline{D}$=1, $\overline{RD}$=1, $\overline{WR}$=0, $\overline{CS}$=0) following a reset. The format of the mode control instruction is shown in Figure 5. The instruction can be considered as four 2-bit fields. The first 2-bit field (D_1 D_0) determines whether the USART is to operate in the synchronous (00) or asynchronous mode. In the asynchronous mode this field also controls the clock scaling factor. As an example, if D_1 and D_0 are both ones, the $\overline{RxC}$ and $\overline{TxC}$ will be divided by 64 to establish the baud rate. The second field, D_3–D_2, determines the number of data bits in the character and the third, D_5–D_4, controls parity generation. Note that the parity bit (if enabled) is added to the data bits and is not considered as part of them when setting up the character length. As an example, standard ASCII transmission, which is seven data bits plus even parity, would be specified as:

X X 1 1 1 0 X X

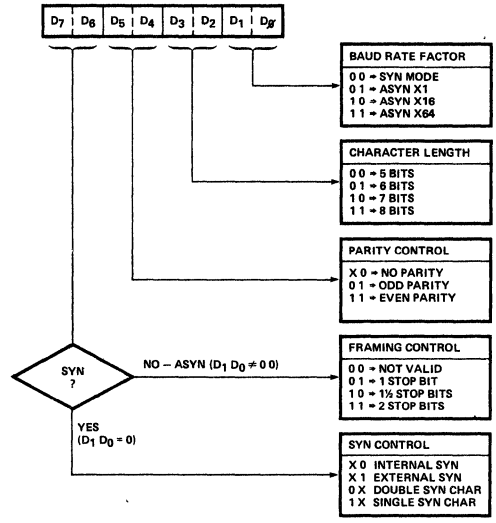


Figure 5. Mode Instruction Format

The last field, D_7–D_6, has two meanings, depending on whether operation is to be in the synchronous or asynchronous mode. For the asynchronous mode (i.e., D_1 $D_0 \neq 00$), it controls the number of STOP bits to be transmitted with the character. Since the receiver will always operate with only one STOP bit, D_7 and D_6 only control the transmitter. In the synchronous mode (D_1 $D_0 = 00$), this field controls the synchronizing process. Note that the choice of single or double SYN characters is independent of the choice of internal or external synchronization. This is because even though the receiver may operate with external synchronization logic, the transmitter must still know whether to send one or two SYN characters should the CPU fail to supply a character in time.

Following the loading of the mode instruction the appropriate SYN character (or characters) must be loaded if synchronous mode has been specified. The SYN character(s) are loaded by the same control output instruction used to load the mode instruction. The USART determines from the mode instruction whether no, one, or two SYN characters are required and uses the control output to load SYN characters until the required number are loaded.

At completion of the load of SYN characters (or after the mode instruction in the asynchronous mode), a command character is issued to the USART. The command instruction controls the operation of the USART within the basic framework established by the mode instruction. The format of the command instruction is shown in

Figure 6. Note that if, as an example, the USART is waiting for a SYN character load and instead is issued an internal reset command, it will accept the command as a SYN character instead of resetting. This situation, which should only occur if two independent programs control the USART, can be avoided by outputting three all zero characters as commands before issuing the internal reset command. The USART indicates its state in a status register which can be read under program control. The format of the status register read is shown in Figure 7.

When operating the receiver it is important to realize that RxE (bit 2 of the command instruction) only inhibits the assertion of RxRDY; it does not inhibit the actual reception of characters. Because the receiver is constantly running, it is possible for it to contain extraneous data when it is enabled. To avoid problems this data should be read from the USART and discarded. The read should be done immediately following the setting of Receive Enable in the asynchronous mode, and following the setting of Enter Hunt in the synchronous mode. It is not necessary to wait for RxRDY before executing the dummy read.

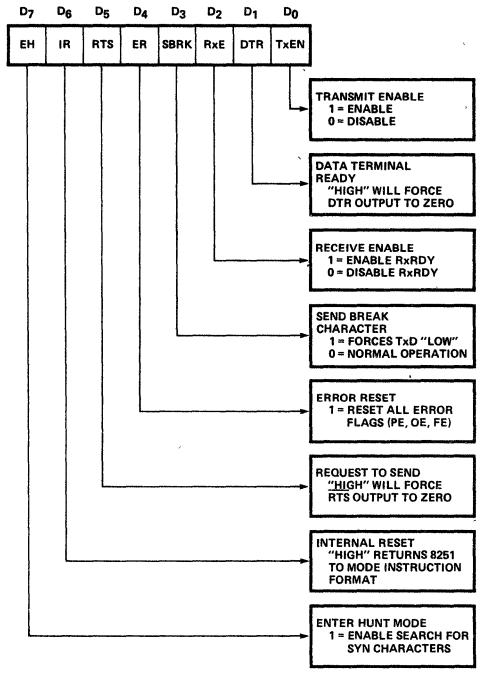


Figure 6. Command Instruction Format

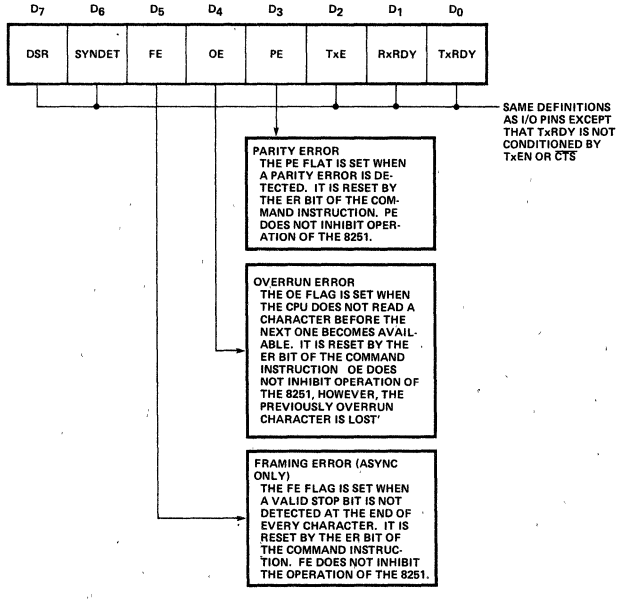


Figure 7. Status Register Format

PROCESSOR DATA LINK

The ability to change the operating mode of the USART by software makes the 8251 an ideal device to use to implement a serial communication link. A terminal initially configured with a simple asynchronous protocol can be upgraded to a synchronous protocol such as IBM Binary Synchronous Communication by a software only upgrade. In order to demonstrate the use of the 8251 USART, the remainder of this document will describe the implementation of an interrupt-driven, full duplex communication link on the Intel MDS™ system. With minor modifications, the program developed could be used on the Intel SBC-80/10™ OEM card, thus implementing a data link between the two systems. Such a facility can be used to down-load programs, run diagnostics, and maintain common data bases in multiprocessor systems.

The factors which must be considered in the design of such a link include the desired transmission rate and format, the error checking requirements, the desirability of full duplex operation, and the physical implementation of the link. The basic requirement of the system described here is that it allow an Intel SBC-80/10 OEM card to be loaded from an MDS development system, either locally or on the switched telephone network. An additional constraint is that the modem used on the switched network be readily available and inexpensive. These requirements led to the choice of a modem such as the Bell 103A to implement the link. These modems, which support full duplex communication at up to 300 baud, are readily available from a number of sources at reasonable cost. These modems are also available in acoustically coupled versions which do not require permanent installation on the telephone network. Interface to the 103A modem is accomplished with nine wires: Protective Ground, Signal Ground, Transmitted Data, Received Data, Clear to Send, Data Set Ready, Data Terminal Ready, Carrier Detector, and Ringing Indicator.

The utilization of the interface signals to the modem is as follows:

Protective Ground	Protective Ground is used to bond the chassis ground of the modem to that of the terminal.
Signal Ground	Signal Ground provides a common ground reference between the modem and the terminal.
Transmitted Data	Transmitted Data is used to transfer serial data from the terminal to the modem.
Received Data	Received Data is used to transfer serial data from the modem to the terminal.
Clear to Send	Clear to Send indicates that the modem has established a connection with a remote modem and is ready to transmit data.
Data Set Ready	Data Set Ready indicates that the modem is connected to the telephone line and is in the data mode.
Data Terminal Ready	Data Terminal Ready is a signal from the terminal which permits the modem to enter the data mode.
Carrier Detector	Carrier Detector is identical to Clear to Send in the 103 modem and will not be used in this interface.
Ringing Indicator	Ringing Indicator indicates that the modem is receiving a ringing signal from the telephone system. This signal will not be used in the interface, since it is possible for the terminal to assert Data Terminal Ready whenever it is ready for the modem to "answer the telephone". The modem uses Data Set Ready to indicate that it has answered the call.

A block diagram showing the connections between the MDS and the SBC-80/10 through the modems is shown in Figure 8. Figure 9 shows the portion of the MDS monitor board devoted to the USARTs and Figure 10 shows the equivalent section of the SBC-80/10 board. Note that several signals on the MDS to not have the proper EIA defined voltage levels, and for this reason the adapter shown in Figure 11 was added to the MDS. The 390 pF capacitor was added to the 1488 driver to bring the rise time within EIA imposed limits of 30 volts/ μsec. In Figure 7 the signal labels within the MDS and SBC-80/10 blocks correspond to the labels on the schematics, the signal labels within the modem blocks correspond to EIA conventions, and the signal labels on the wires between the blocks are abbreviations for the English language names of the signals.

As an example of how the USART clocks can be generated, circuits A27, A16, and A15 of Figure 9 form a divider of the OSC signal. The OSC signal has a frequency of 18.432 MHz and is generated by the 8224 which generates system timing for the 8080A. The 18.432 MHz signal results in a state time of 488 ns versus the normal 500 ns for the 8080A. (This does not violate 8080A specifications.) The 18.432 MHz signal can be divided by

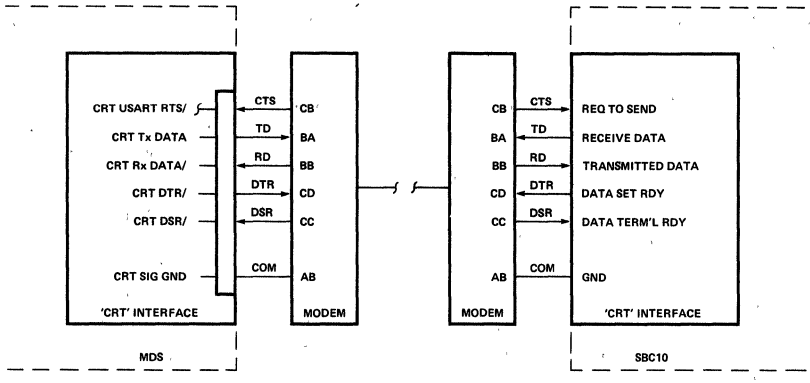


Figure 8. System Block Diagram

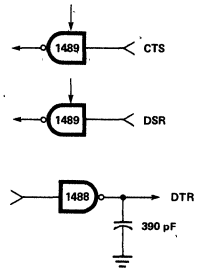


Figure 9. EIA Adapter

30 and then 64 to give a 9600 baud communication standard. The 9600 baud signal can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud standard signal rate. Note that because of constraints on the CLK input 9600 baud operation is not possible in the X64 mode. The divide by 64 can be accomplished by dividing by 4 with a counter and then 16 within the USART.

In order to keep the system as general purpose as possible, it was decided to transmit 8-bit data characters with an appended odd parity bit. Having a full 8-bit byte available for data enables the transmission of codes such as ASCII (which is 7-level with an additional parity bit) to be transmitted and received transparently in the system. Also, of course, it allows 8-bit bytes from the 8080A memory to be transferred in one transmission character. If error checking beyond the parity check is required, it could be added to the data record to be transmitted in the form of redundant check characters.

Before the software design of the system could be undertaken, it was necessary to decide whether service requests from the USART would be handled on a polled or interrupt driven mode. Polled operation normally results in more compact code but it requires that whatever programs are running concurrently with a transmission or reception must periodically either check the status of the USART or call a routine that does. Since it was not possible to determine what program might be running during a receive or transmit operation, it was decided to operate in an interrupt driven mode.

The program which operates the 8251 must be instructed as to what data it should transmit or receive from some other program resident in the 8080 system. To facilitate the discussion of the operation of the software, the following definitions will be made:

> USRUN is the program which controls the operation of the 8251.
>
> USER is a program which utilizes USRUN in order to effect a data transmission.

USER passes commands and parameters to USRUN by means of the control block shown in Figure 12. The first byte of the block contains the command which USER wants USRUN to execute. Valid contents of this byte are "C" which causes USRUN to initialize itself and the 8251, "R" which causes the execution of the data input (or READ) operation, and "W" which causes a data output (WRITE) operation. The second byte of the control block is used by USRUN to inform USER of the status of the requested operation. The third and fourth bytes specify the starting address of a buffer set up by USER which contains the data for a transmit operation or which will be used by USRUN to store received data. The fifth and sixth bytes are concatenated to form a positive binary

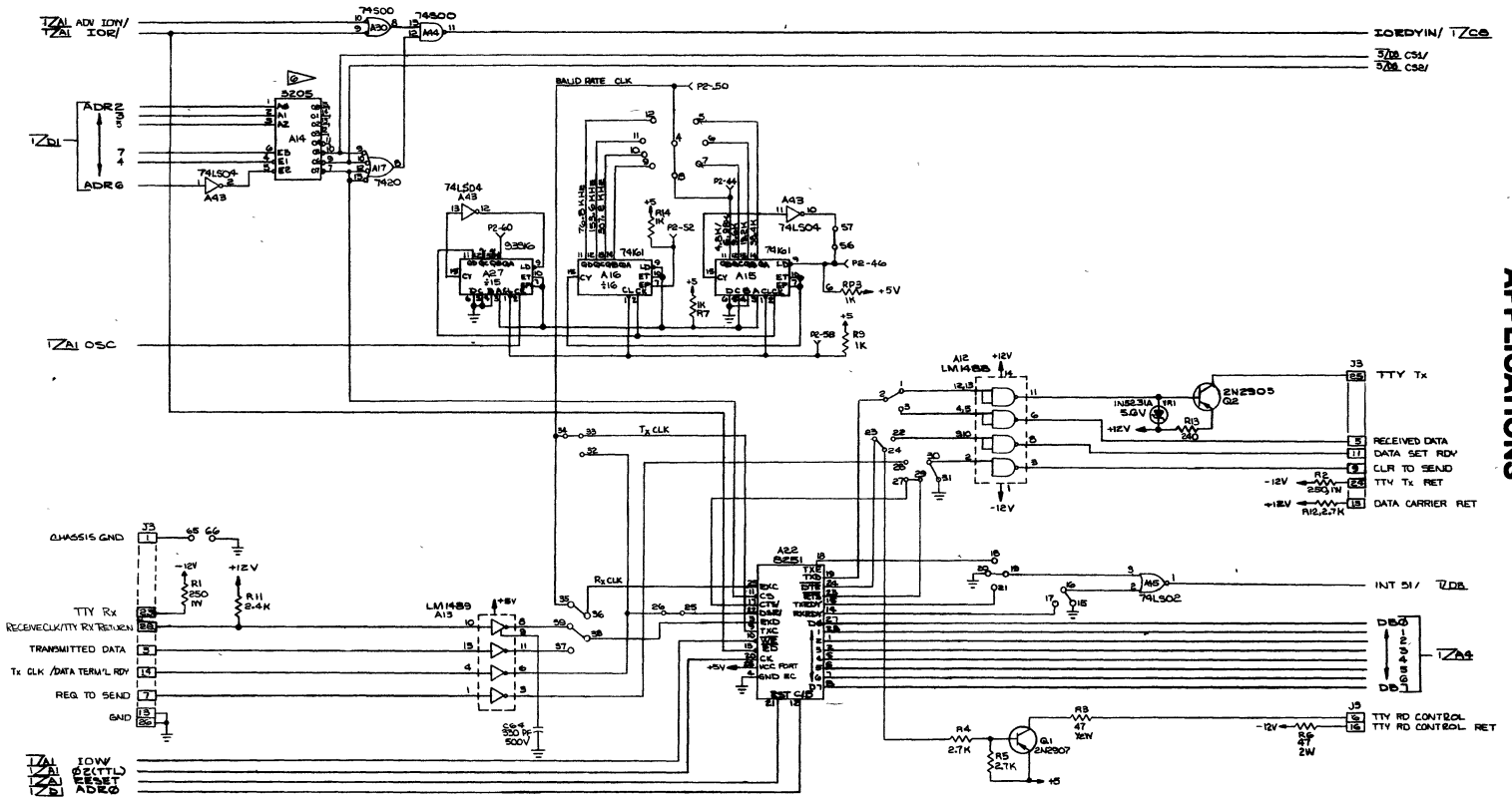


Figure 10. SBC 80/10 Serial I/O

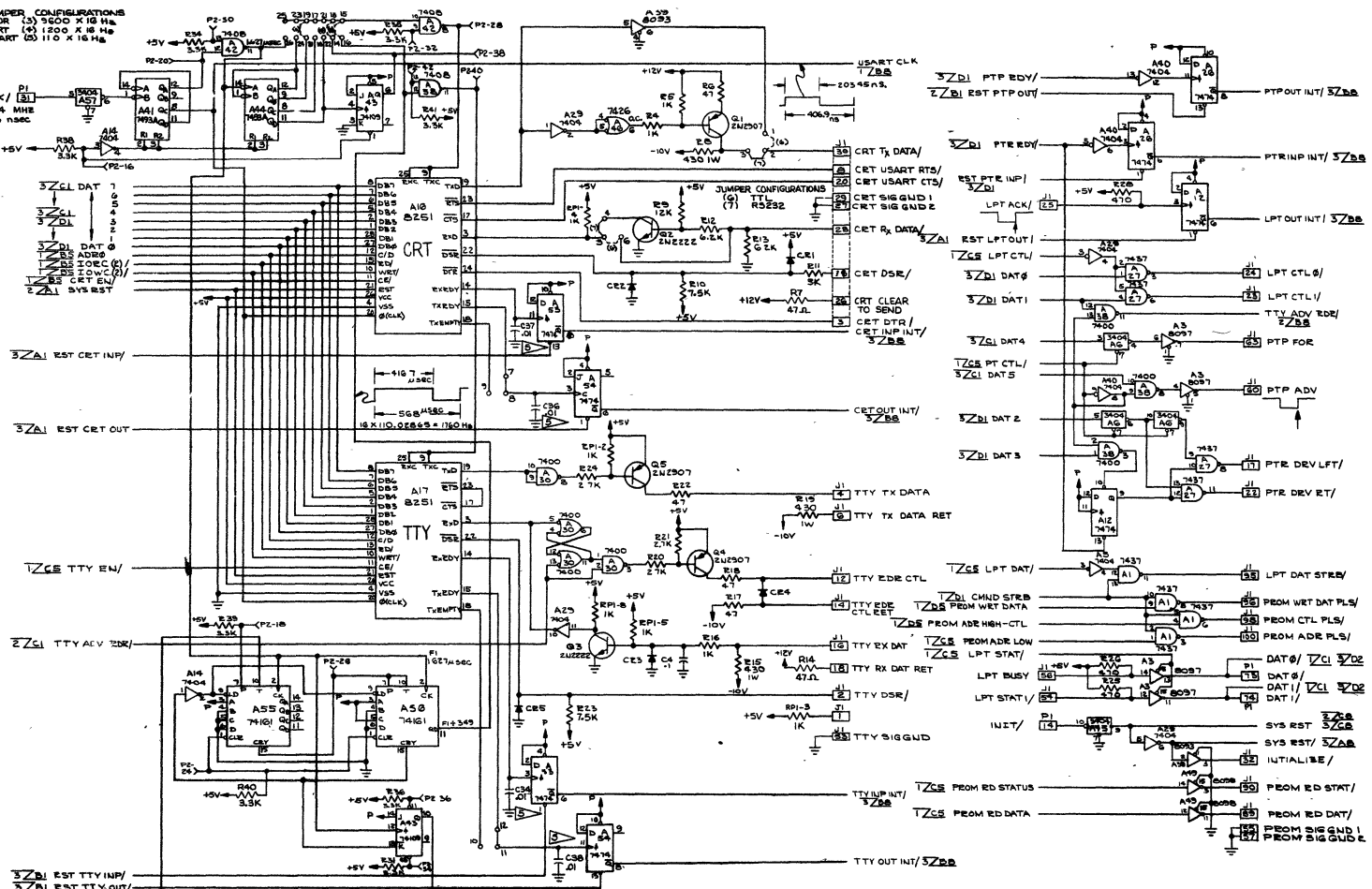


Figure 11. MDS Monitor Module

number which specifies how many bytes of data USER wants transferred. The seventh and eighth bytes are concatenated and used by USRUN to count the number of bytes that have been transferred. When the required number of characters have been transferred, or if USRUN terminates a READ or WRITE due to an abnormal condition, then USRUN calls a subroutine at an address defined by the ninth and tenth bytes of the command block. This subroutine, which is provided by USER, must determine the state of the process and then take appropriate action.

Since USRUN must be capable of operation in a full duplex mode (i.e., be able to receive and transmit simultaneously), it keeps the address of two control blocks; one for a READ operation and one for a WRITE. The address of the controlling command block is kept in RAM locations labeled RCBA for the READ operation and TCBA for the WRITE operation. If RCBA (Receive Control Block Address) or TCBA (Transmit Control Block Address) is zero, it indicates that the corresponding operation is in an idle status.

Flowcharts of USRUN appear in Figure 13 and the listings appear in Figure 14. The first section of the flowcharts (Figures 13.1 and 13.2) consists of two subroutines which are used as convenient tools for operating on the control blocks. These routines are labeled LOADA and CLEAN. LOADA is entered with the address of a control block in registers H and L. Upon return registers D and E have been set equal to the address in the buffer which is the target of the next data transfer (i.e., D,E = BAD+ CCT); and CCT (transferred byte count) has then been incremented. In addition, the B register is set to zero if the number of bytes that have been transferred is equal to the number requested (i.e., CCT = RCT). CLEAN, the second routine, is also entered with the address of a command block in the H and L registers. In addition, the Accumulator holds the status which will be placed in the STATUS byte of the command block. On exit the STATUS byte has been updated and the address of the completion routine has been placed in H and L.

Upon interrupt, control of the MCS-80 system is transferred to VECTOR (Figure 13.3). Vector is a program which saves the state of the system, gets the status of the USART and jumps to the RISR (Receive Interrupt Service Routine) or the TISR (Transmit Interrupt Service Routine), depending on which of the two ready flags is active. If neither ready flag is active, VECTOR restores the status of the running program, enables interrupts, and returns. (Interrupts are automatically disabled by the hardware upon an interrupt.) This exit from VECTOR, which is labeled VOUT, is used from other

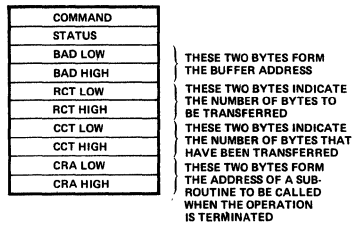


Figure 12. Control Block

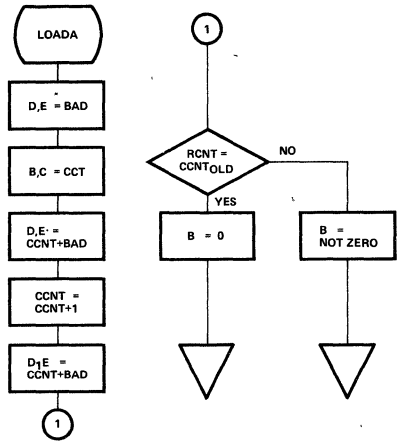


Figure 13.1. LOADA Subroutine

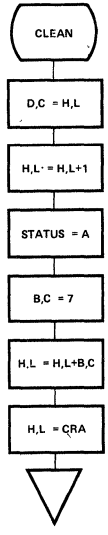


Figure 13.2. CLEAN Subroutine

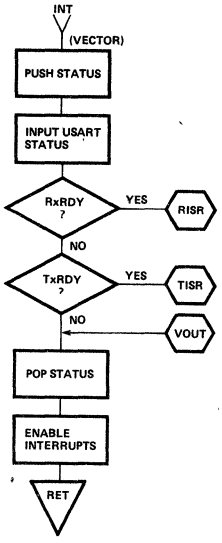


Figure 13.3. Interrupt Entry

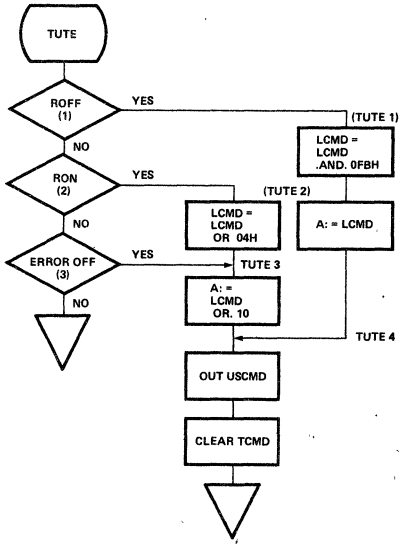


Figure 13.4. Transmit Interrupt Service Routine

portions of USRUN if return from the interrupt mode is required.

In addition to handling normal data transfers, TISR (Figure 13.4) checks a location in memory named TCMD in order to determine if the receive program wishes to send a command to the USART. Since the transmit data and command must share a buffer within the USART, any command output must occur when TxRDY is asserted. If TCMD is zero, TISR proceeds with the data transfer. If TCMD is non-zero, TISR calls TUTE (Transmit Utility, Figure 13.5) which, depending on the value

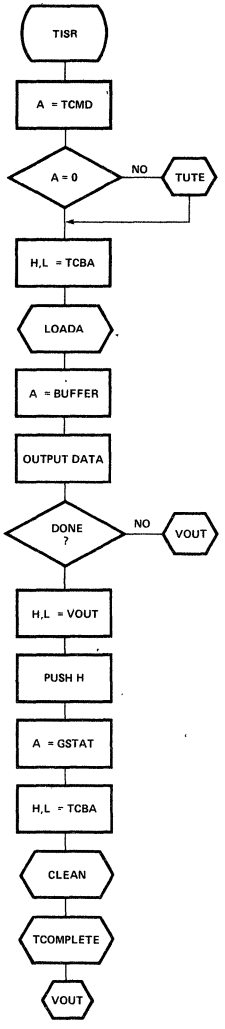


Figure 13.5. Transmit Utility Routine

in TCMD, turns off the receiver, turns on the receiver, or clears error conditions. Note that the error flags (parity, framing, and overrun) are always cleared by the software when the receiver is first enabled.

The flowchart of the RISR is shown in Figure 13.6. Note that in addition to terminating whenever the required number of characters have been received, the RISR also terminates if one of the error flags becomes set or if the received character matches a character found in a table pointed to by the label ETAB. This table, which starts at ETAB and continues until an all "ones" entry is found, can be used by USER to define special characters, such as EOT (End Of Transmission), which will terminate a READ operation. The remainder of Figure 13 (13.7) shows the decoding of the commands to USRUN. The listings also include a test USER which exercises USRUN. This program sets up a 256-byte transmit buffer and transfers it to a similar input buffer by means of a local loop. When both the READ and WRITE operations are complete, the test USER checks to insure that the two buffers are identical. If the buffers differ, the MDS monitor is called; if the data is correct, the test is repeated.

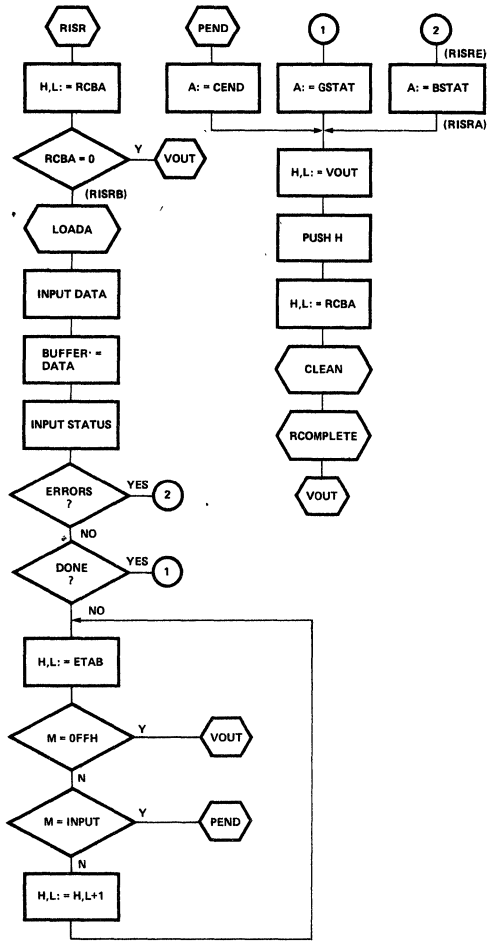


Figure 13.6. Receive Interrupt Service Routine

CONCLUSION

The 8251 USART has been described both as a device and as a component in a system. Since not only modems but also many peripheral devices have a serial interface, the 8251 is an extremely useful component in a microcomputer system. A particular advantage of the device is that it is capable of operating in various modes without requiring hardware modifications to the system of which it is a part. As with any complex subsystem, however, the 8251 USART must be carefully applied so that it can be utilized to full advantage in the overall system. It is hoped that this application note will aid in the designer in the application of the 8251 USART. As a further aid to the application of the 8251, the appendix of this document includes a list of design hints based on past experience with the 8251.

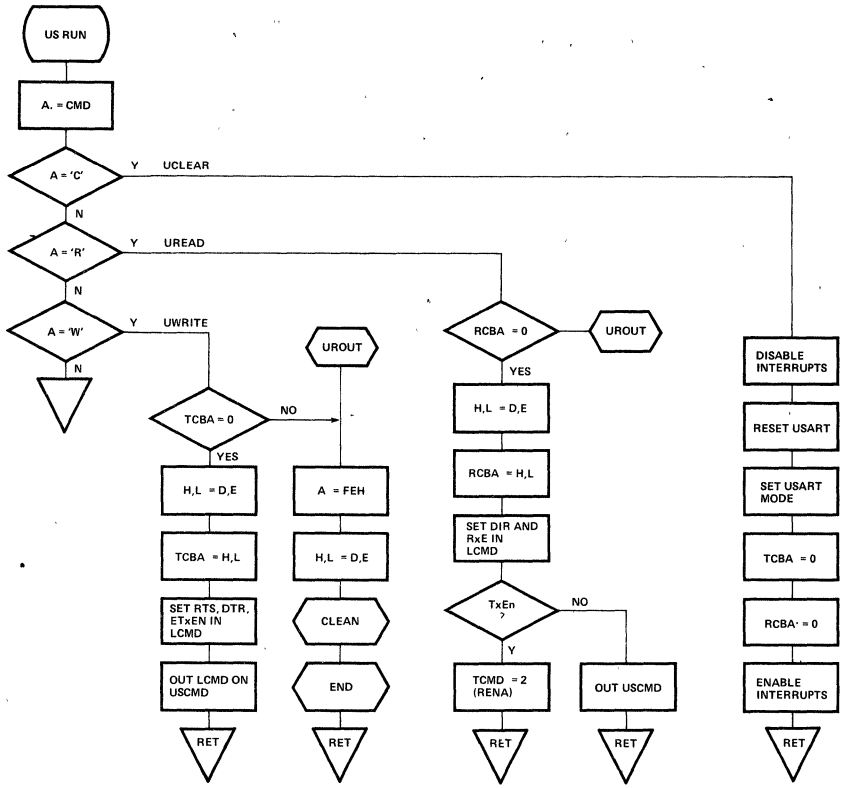


Figure 13.7. URUN Command Decode

Figure 14. Program Listing

```
                        ;*****
                        ;
                        ;       SYSTEM ORIGIN STATEMENT
                        ;
                        ;*****

4000                            ORG     4000H


                        ;*****
                        ;
                        ;       DATA STORAGE FOR TEST USER
                        ;
                        ;*****

4000            BUFIN:  DS      100H    ;INPUT BUFFER
4100            BUFOUT: DS      100H    ;OUTPUT BUFFER
4200 5200       RBLOCK: DB      'R',00H ;RECEIVE CONTROL BLOCK
4202 0040       RBAD:   DW      BUFIN
4204 FF00       RRCT:   DW      0FFH
4206 0000       RCCT:   DW      00H
4208 1742       RCRA:   DW      RCR
420A 5700       TBLOCK: DB      'W',00H ;TRANSMIT CONTROL BLOCK
420C 0041       TBAD:   DW      BUFOUT
420E FF00       TRCT:   DW      0FFH
4210 0000       TCCT:   DW      00H
4212 2742       TCRA:   DW      TCR
4214 4300       GBLOCK: DB      'C',00H
4216 00         FLAG:   DB      00H


                        ;*****
                        ;
                        ;       COMPLETION ROUTINES
                        ;
                        ;*****

4217 AF         RCR:    XRA     A       ;CLEAR A
4218 323B42             STA     RCBA    ;TURN OFF RECEIVE
421B 323C42             STA     RCBA+1
421E 3A1642             LDA     FLAG    ;GET FLAG
4221 E60F               ANI     0FH     ;CLEAR UPPER FOUR BITS
4223 321642             STA     FLAG    ;RESTORE FLAG
4226 C9                 RET
4227 AF         TCR:    XRA     A       ;CLEAR A
4228 323942             STA     TCBA    ;TURN OFF TRANSMIT
422B 323A42             STA     TCBA+1
422E 3A1642             LDA     FLAG    ;GET FLAG
4231 E6F0               ANI     0F0H    ;CLEAR LOWER FOUR BITS
4233 321642             STA     FLAG    ;RESTORE FLAG
4236 C9                 RET             ;THEN RETURN
```

```
                ;*****
                ;
                ;       SYSTEM EQUATES
                ;
                ;*****

00F5            USTAT   EQU     0F5H    ;USART STATUS ADDRESS
00F5            USCMD   EQU     0F5H    ;USART CMD ADDRESS
00F4            USDAI   EQU     0F4H    ;USART DATA INPUT ADDRESS
00F4            USDAO   EQU     0F4H    ;USART DATA OUTPUT ADDRESS
0000            GSTAT   EQU     00H     ;GOOD STATUS
00FF            BSTAT   EQU     0FFH    ;BAD STATUS
0001            CEND    EQU     01H


                ;*****
                ;
                ;       SYSTEM DATA TABLE
                ;
                ;*****

4237 00         LCMD:   DB      00H     ;CURRENT OPERATING COMMAND
4238 00         TCMD:   DB      00H     ;IF NON ZERO A COMMAND TO BE SENT
4239 0000       TCBA:   DW      00H     ;ADDRESS OF XMIT CBLOCK
423B 0000       RCBA:   DW      00H     ;ADDRESS OF RECEIVE CBLOCK
423D FF         MTAB:   DB      0FFH    ;END CHARACTER TABLE
```

```
                ;*****
                ;
                ;       LOAD ADDRESS ROUTINE
                ;       LOADA IS ENTERED WITH THE ADDRESS OF A CONTROL
                ;       BLOCK IN H,L. ON EXIT D,E CONTAINS THE ADDRESS
                ;       WHICH IS THE TARGET OF THE NEXT DATA TRANSFER (BAD+CCNT)
                ;       AND B HAS BEEN SET TO ZERO IF THE REQUESTED NUMBER OF
                ;       TRANSFERS HAS BEEN ACCOMPLISHED. CCNT IS INCREMENTED
                ;       AFTER THE TARGET ADDRESS HAS BEEN CALCULATED.
                ;
                ;*****

423E 23         LOADA:  INX     H       ;D,E GETS BUFFER ADDRESS
423F 23                 INX     H
4240 5E                 MOV     E,M
4241 23                 INX     H
4242 56                 MOV     D,M     ;DONE
4243 23                 INX     H       ;B,C GETS COMPLETED COUNT (CCNT)
4244 23                 INX     H
4245 23                 INX     H
4246 4E                 MOV     C,M
4247 23                 INX     H
4248 46                 MOV     B,M     ;DONE
4249 EB                 XCHG            ;D,E GETS BAD+CCNT
424A 09                 DAD     B
424B EB                 XCHG            ;DONE
424C 03                 INX     B       ;CCNT GETS INCREMENTED
424D 70                 MOV     M,B
424E 2B                 DCX     H
424F 71                 MOV     M,C     ;DONE
4250 0B                 DCX     B       ;DOES OLD CCNT=RCNT?
4251 2B                 DCX     H
4252 7E                 MOV     A,M
4253 90                 SUB     B
4254 47                 MOV     B,A
4255 C0                 RNZ             ;NO-RETURN WITH B NOT ZERO
4256 2B                 DCX     H
4257 7E                 MOV     A,M
4258 91                 SUB     C
4259 47                 MOV     B,A
425A C9                 RET             ;RETURN WITH B=0 IF RCNT=CCNT
```

```
                ;*****
                ;
                ;       CLEAN-UP ROUTINE
                ;       CLEAN IS ENTERED WITH THE ADDRESS OF A CONTROL
                ;       BLOCK IN H,L AND A NEW STATUS TO BE
                ;       ENTERED INTO IT IN A. ON EXIT THE ADDRESS OF THE
                ;       CONTROL BLOCK IS IN D,E; THE STATUS OF THE BLOCK
                ;       HAS BEEN UPDATED; AND THE ADDRESS OF THE COMPLETION
                ;       ROUTINE IS IN H,L.
                ;
                ;*****


425B 5D         CLEAN:  MOV     E,L     ;SAVE THE ADRESS OF THE COMMAND BLOCK
425C 54                 MOV     D,H
425D 23                 INX     H       ;POINT AT STATUS
425E 77                 MOV     M,A     ;SET STATUS EQUAL TO A
425F 010700             LXI     B,7     ;SET INDEX TO SEVEN
4262 09                 DAD     B       ;POINT AT COMPLETION ADDRESS
4263 7E                 MOV     A,M     ;GET LOWER ADDRESS
4264 23                 INX     H       ;POINT AT UPPER ADDRESS
4265 66                 MOV     H,M     ;H GETS HIGH ADDRESS BYTE
4266 6F                 MOV     L,A     ;L GETS LOW ADDRESS BYTE
4267 C9                 RET


                ;*****
                ;
                ;       INTERUPT VECTOR ROUTINE
                ;       VECTOR SAVES THE STATUS OF THE RUNNING PROGRAM
                ;       THEN READS THE STATUS OF THE USART TO DETERMINE
                ;       IF A RECEIVE OR TRANSMIT INTERUPT OCCURRED.
                ;       VECTOR THEN CALLS THE APPROPRIATE SERVICE ROUTINE.
                ;       IF NEITHER INTERUPTS OCCURRED THEN VECTOR RESTORES
                ;       THE STATUS OF THE RUNNING PROGAM. THE SERVICE
                ;       ROUTINES USE THE EXIT CODE, LABLED VOUT, TO EFFECT
                ;       THEIR EXIT FROM INTERUPT MODE.
                ;
                ;*****

4268 F5         VECTOR: PUSH    PSW     ;PUSH STATUS INTO THE STACK
4269 C5                 PUSH    B
426A D5                 PUSH    D
426B E5                 PUSH    H
426C DBF5               IN      USTAT   ;GET USART ADDRESS
426E DBFA               IN      0FAH    ;MDS-GET MONITOR CARD INT. STATUS
4270 0F                 RRC             ;ROTATE TWO PLACES
4271 0F                 RRC             ;SO THAT CARRY=RXRDY
4272 DA8842             JC      RISR    ;IF RXRDY GO TO SERVICE ROUTINE
4275 07                 RLC             ;IF NOT ROTATE BACK
4276 07                 RLC             ;LEAVING TXRDY IN CARRY
4277 DAD442             JC      TISR    ;IF TXRDY THEN GO TO SERVICE ROUTINE
427A 3EFC               MVI     A,0FCH  ;MDS-CLEAR OTHER LEVEL THREE INTERUPTS
427C D3F3               OUT     0F3H    ;MDS
427E E1         VOUT:   POP     H       ;ELSE EXIT FROM INTERUPT MODE
427F D1                 POP     D
4280 C1                 POP     B
4281 3E20               MVI     A,20H   ;MDS-RESTORE CURRENT LEVEL
4283 D3FD               OUT     0FDH    ;MDS
4286 FB                 EI              ;ENABLE INTERUPTS
4287 C9                 RET
```

```
                ;*****
                ;
                ;       RECEIVE INTERUPT SERVICE ROUTINE;
                ;               RISR PROCESSES A RECEIVE INTERUPT
                ;               AT THE END OF RECEIVE THE USER SUPPLIED
                ;               COMPLETION ROUTINE IS CALLED AND THEN AN
                ;               EXIT IS TAKEN THROUGH VOUT OF THE
                ;               VECTOR
                ;*****

4288 2A3B42     RISR:   LHLD    RCBA
428B 3E82               MVI     A,82H   ;MDS-CLEAR RECEIVE INTERUPT
428D D3F3               OUT     0F3H    ;MDS
428F 2C                 INR     L
4290 2D                 DCR     L
4291 C29942             JNZ     RISRB
4294 24                 INR     H
4295 25                 DCR     H
4296 CA7E42             JZ      VOUT
4299 CD3E42     RISRB:  CALL    LOADA   ;READY-SET UP ADDRESS
429C DBF4               IN      USDAI   ;GET INPUT DATA
429E 12                 STAX    D       ;AND PUT IN THE BUFFER
429F 4F                 MOV     C,A     ;SAVE INPUT DATA IN C
42A0 DBF5               IN      USTAT   ;GET STATUS AGAIN
42A2 E638               ANI     38H     ;MASK FOR ERROR FIELD
42A4 C2B942             JNZ     RISRE   ;NOT ZERO-TAKE ERROR EXIT
42A7 04                 INR     B       ;B WAS 00 IF DONE
42A8 05                 DCR     B
42A9 C2BE42             JNZ     EXCHAR  ;NOT DONE-EXIT
42AC 3E00               MVI     A,GSTAT ;A GETS GOOD STATUS
42AE 217E42     RISRA:  LXI     H,VOUT  ;GET RETURN ADDRESS
42B1 E5                 PUSH    H       ;AND PUSH IT INTO THE STACK
42B2 2A3B42             LHLD    RCBA    ;POINT H,L AT THE CMD BLOCK
42B5 CD5B42             CALL    CLEAN   ;CALL CLEANUP ROUTINE
42B8 E9                 PCHL            ;EFFECTIVELY CALLS COMPLETION ROUTINE
                                        ;RETURN IS TO VOUT BECAUSE OF PUSH H
42B9 3EFF       RISRE:  MVI     A,BSTAT ;A GETS BAD STATUS
42BB C3AE42             JMP     RISRA   ;OTHERWISE EXIT IS NORMAL
42BE 213D42     EXCHAR: LXI     H,MTAB  ;TEST CHARACTER AGAINST EXIT TABLE
42C1 7E         EXA:    MOV     A,M
42C2 FEFF               CPI     0FFH    ;END OF TABLE
42C4 CA7E42             JZ      VOUT
42C7 B9                 CMP     C
42C8 CACF42             JZ      PEND    ;MATCH-TERMINATE READ
42CB 23                 INX     H
42CC C3C142             JMP     EXA
42CF 3E01       PEND:   MVI     A,CEND
42D1 C3AE42             JMP     RISRA
```

```
                ;*****
                ;
                ;       TRANSMIT INTERUPT SERVICE ROUTINE
                ;       TISR PROCCESSES TRANSMITTER INTERUPTS
                ;       WHEN THE END OF A TRANSMISSION IS
                ;       DETECTED THE USER SUPPLIED COMPLETION
                ;       ROUTINE IS CALLED AND THEN AN EXIT IS
                ;       TAKEN THROUGH VOUT OF VECTOR
                ;
                ;*****

42D4 3A3842     TISR:   LDA     TCMD    ;GET POTENTIAL COMMAND
42D7 B7                 ORA     A       ;DESIGNATE ON IT
42D8 C40443             CNZ     TUTE    ;DO UTILITY COMMAND
42DB 3E81               MVI     A,081H  ;MDS-CLEAR XMIT INTERUPTS
42DD D3F3               OUT     0F3H    ;MDS
42DF 2A3942             LHLD    TCBA
42E2 2C                 INR     L       ;MAKE SURE HAVE VALID CONTROL BLOCK
42E3 2D                 DCR     L
42E4 C2EC42             JNZ     TISRA   ;GOOD
42E7 24                 INR     H
42E8 25                 DCR     H
42E9 CA7E42             JZ      VOUT    ;NON VALID BLOCK (H,L=0)
42EC CD3E42     TISRA:  CALL    LOADA   ;SET UP ADDRESS
42EF 1A                 LDAX    D       ;GET DATA FROM BUFFER
42F0 D3F4               OUT     USDAO   ;AND OUTPUT IT
42F2 04                 INR     B       ;B WAS 00 IF DONE
42F3 05                 DCR     B
42F4 C27E42             JNZ     VOUT    ;NOT DONE-EXIT FROM SERVICE ROUTINE
42F7 217E42             LXI     H,VOUT  ;SET UP RETURN ADDRESS
42FA E5                 PUSH    H       ;AND PUSH IT INTO THE STACK
42FB 3E00               MVI     A,GSTAT ;A GETS GOOD STATUS
42FD 2A3942             LHLD    TCBA    ;POINT H,L AT COMMAND BLOCK
4300 CD5B42             CALL    CLEAN   ;CALL CLEANUP ROUTINE
4303 E9                 PCHL            ;CALL COMPLETION ROUTINE
                                        ;RETURN WILL BE TO VOUT
4304 FE01       TUTE:   CPI     01      ;RECEIVER OFF
4306 CA2443             JZ      TUTE1
4309 FE02               CPI     02      ;RECEIVER ON
430B CA1443             JZ      TUTE2
430E FE03               CPI     03      ;CLEAR ERRORS
4310 CA1C43             JZ      TUTE3
4313 C9                 RET
4314 3A3742     TUTE2:  LDA     LCMD
4317 F604               ORI     04
4319 323742             STA     LCMD
431C 3A3742     TUTE3:  LDA     LCMD
431F F610               ORI     10H
4321 D3F5       TUTE4:  OUT     USCMD
4323 C9                 RET
4324 3A3742     TUTE1:  LDA     LCMD
4327 E6FB               ANI     0FBH
4329 323742             STA     LCMD
432C C32143             JMP     TUTE4
```

```
                ;*****
                ;
                ;       USART COMMAND BLOCK INTERPRETER
                ;       USRUN IS CALLED BY USER WITH THE ADDRESS
                ;       OF THE COMMAND BLOCK IN H,L. USRUN EXAMINES
                ;       THE BLOCK AND INTIALIZES THE REQUESTED OPERATION
                ;
                ;*****

432F 1A         USRUN:  LDAX    D       ;GET THE CMD FROM THE BLOCK
4330 FE43               CPI     'C'     ;IS IT A CLEAR COMMAND?
4332 CA4043             JZ      UCLEAR  ;YES GO TO CLEAR ROUTINE
4335 FE52               CPI     'R'     ;IS IT A READ COMMAND?
4337 CA5D43             JZ      UREAD   ;YES-GO TO READ ROUTINE
433A FE57               CPI     'W'     ;IS IT A WRITE COMMAND?
433C CA9D43             JZ      UWRITE  ;GO TO WRITE ROUTINE
433F C9                 RET             ;NOT A GOOD COMMAND-RETURN
4340 F3         UCLEAR: DI              ;DISABLE INTERUPTS
4341 AF                 XRA     A       ;CLEAR A
4342 D3F5               OUT     USCMD   ;OUTPUT THREE TIMES TO ENSURE
4344 D3F5               OUT     USCMD   ;THAT THE USART IS IN A KNOWN STATE
4346 D3F5               OUT     USCMD
4348 3E40               MVI     A,40H   ;CODE TO RESET USART
434A D3F5               OUT     USCMD   ;OUTPUT ON CMD CHANNEL
434C 3E5E               MVI     A,05EH  ;CE IMPLIES ASYN MODE (X16)
                                        ;           8 DATA BITS
                                        ;           ODD PARITY
                                        ;           1 STOP BIT
434E D3F5               OUT     USCMD   ;OUTPUT ON CMD CHANNEL
4350 AF                 XRA     A       ;CLEAR A, SET ZERO
4351 213942             LXI     H,TCBA  ;CLEAR TCBA AND RCBA
4354 77                 MOV     M,A
4355 23                 INX     H
4356 77                 MOV     M,A
4357 23                 INX     H
4358 77                 MOV     M,A
4359 23                 INX     H
435A 77                 MOV     M,A
435B FB                 EI              ;ENABLE INTERUPTS
435C C9                 RET             ;AND RETURN TO USER
                                        ;
                                        ;
435D 213B42     UREAD:  LXI     H,RCBA  ;CHECK READ IDLE
4360 7E                 MOV     A,M
4361 B7                 ORA     A
4362 C26B43             JNZ     UROUT
4365 23                 INX     H
4366 7E                 MOV     A,M
4367 B7                 ORA     A
4368 CA7743             JZ      URDA    ;READ IS IDLE-PROCEDE
436B 3EFE       UROUT:  MVI     A,0FEH  ;ALREADY RUNNING-ERROR STATUS
436D 217643             LXI     H,URDB  ;SET UP RETURN ADDRESS
4370 E5                 PUSH    H       ;PUSH IT INTO STACK
4371 EB                 XCHG            ;H GETS COMMAND BLOCK ADDRESS
4372 CD5B42             CALL    CLEAN   ;CALL CLEANUP ROUTINE
4375 E9                 PCHL            ;EFFECTIVELY CALLS END ROUTINE
4376 C9         URDB:   RET             ;RETURN TO USER

4377 EB         URDA:   XCHG            ;H GETS COMMAND BLOCK ADDRESS
4378 223B42             SHLD    RCBA    ;RCBA GETS COMMAND BLOCK ADDRESS
437B 3A3742             LDA     LCMD    ;GET LAST COMMAND
437E F616               ORI     16H     ;SET RXE AND DTR AND RESET ERRORS
4380 323742             STA     LCMD    ;AND RETURN TO MEMORY
4383 0F                 RRC             ;SET CARRY EQUAL TO TXE
```

```
4384 D28C43            JNC     URDC
4387 3E02              MVI     A,2
4389 323842            STA     TCMD
438C 07        URDC:   RLC
438D D3F5              OUT     USCMD   ;OUTPUT CMD
438F DBF4              IN      USDAI   ;CLEAR USART OF LEFT OVER CHARACTERS
4391 DBF4              IN      USDAI
4393 3E82              MVI     A,82H   ;MDS-CLEAR RECEIVE INTERUPT
4395 D3F3              OUT     0F3H    ;MDS
4397 3EF6              MVI     A,0F6H  ;MDS-ENABLE LEVEL THREE
4399 D3FC              OUT     0FCH    ;MDS
439B FB                EI              ;ENABLE INTERUPTS
439C C9                RET             ;RETURN TO USER


439D 213942    UWRITE: LXI     H,TCBA  ;CHECK WRITE IDLE
43A0 7E                MOV     A,M
43A1 B7                ORA     A
43A2 C26B43            JNZ     UROUT   ;BUSY-EXIT
43A5 23                INX     H
43A6 7E                MOV     A,M
43A7 C26B43            JNZ     UROUT   ;BUSY-EXIT
43AA EB                XCHG            ;OK-H GETS COMMAND BLOCK ADDRESS
43AB 223942            SHLD    TCBA    ;TCBA GETS COMMAND BLOCK ADDRESS
43AE 3A3742            LDA     LCMD    ;GET LAST COMMAND
43B1 F623              ORI     023H    ;SET RTS,DTR, AND TXEN
43B3 323742            STA     LCMD
43B6 D3F5              OUT     USCMD
43B8 3EF6              MVI     A,0F6H  ;MDS-ENABLE LEVEL THREE INTERUPTS
43BA D3FC              OUT     0FCH    ;MDS
43BC FB                EI              ;ENABLE SYSTEM INTERUPTS
43BD C9                RET             ;AND RETURN
```

```
                ;*****
                ;
                ;       USER IS A TEST PROGRAM WHICH EXERCISES USRUN
                ;
                ;*****

43BE 3EC3       USER:   MVI     A,0C3H  ;MDS-SET INTERUPT VECTOR
43C0 321800             STA     018H
43C3 216842             LXI     H,VECTOR
43C6 221900             SHLD    019H
43C9 3E43               MVI     A,'C'   ;SET GENERAL BLOCK TO A 'C'
43CB 111442             LXI     D,GBLOCK
43CE 12                 STAX    D
43CF CD2F43             CALL    USRUN
43D2 210040             LXI     H,BUFIN ;CLEAR INPUT BUFFER
43D5 AF                 XRA     A
43D6 77                 MOV     M,A
43D7 2C                 INR     L
43D8 C2D643             JNZ     $-2
43DB 210041             LXI     H,BUFOUT ;INITIALIZE OUTPUT BUFFER
43DE 75                 MOV     M,L
43DF 2C                 INR     L
43E0 C2DE43             JNZ     $-2
43E3 65                 MOV     H,L     ;REINTIALIZE CONTROL BLOCKS
43E4 2E52               MVI     L,'R'
43E6 220042             SHLD    RBLOCK
43E9 2E57               MVI     L,'W'
43EB 220A42             SHLD    TBLOCK
43EE 6C                 MOV     L,H
43EF 220642             SHLD    RCCT
43F2 221042             SHLD    TCCT
43F5 110042             LXI     D,RBLOCK ;START READ
43F8 CD2F43             CALL    USRUN
43FB 110A42             LXI     D,TBLOCK ;START WRITE
43FE CD2F43             CALL    USRUN
4401 3EFF               MVI     A,0FFH  ;LOOP WAITING COMPLETION
4403 321642             STA     FLAG    ;FLAG WILL BE SET BY COMPLETION ROUTINES
4406 3A1642             LDA     FLAG
4409 B7                 ORA     A
440A C20644             JNZ     $-4
440D 210040             LXI     H,BUFIN ;TEST INPUT BUFFER=OUTPUT BUFFER
4410 7E         COMLP:  MOV     A,M
4411 24                 INR     H
4412 BE                 CMP     M
4413 C21E44             JNZ     COMER
4416 25                 DCR     H
4417 2C                 INR     L
4418 C21044             JNZ     COMLP
441B C3BE43             JMP     USER    ;GOOD COMPARE-REPEAT TEST
441E C7         COMER:  RST     0       ;ERROR-RETURN TO MONITOR

0000                    END
```

```
BSTAT 00FF      BUFIN 4000      BUFOU 410Q      CEND  0001
CLEAN 425B      COMER 441E      COMLP 4410      EXA   42C1
EXCHA 42BE      FLAG  4216      GBLOC 4214      GSTAT 0000
LCMD  4237      LOADA 423E      MTAB  423D      PEND  42CF
RBAD  4202      RBLOC 4200      RCBA  423B      RCCT  4206
RCR   4217      RCRA  4208      RISR  4288      RISRA 42AE
RISRB 4299      RISRE 42B9      RRCT  4204      TBAD  420C
TBLOC 420A      TCBA  4239      TCCT  4210      TCMD  4238
TCR   4227      TCRA  4212      TISR  42D4      TISRA 42EC
TRCT  420E      TUTE  4304      TUTE1 4324      TUTE2 4314
TUTE3 431C      TUTE4 4321      UCLEA 4340      URDA  4377
URDB  4376      URDC  438C      UREAD 435D      UROUT 436B
USCMD 00F5      USDAI 00F4      USDAO 00F4      USER  43BE
USRUN 432F      USTAT 00F5      UWRIT 439D      VECTO 4268
VOUT  427E
```

APPENDIX A
8251 DESIGN HINTS

1. Output of a command to the USART destroys the integrity of a transmission in progress if timed incorrectly.

   Sending a command into the USART will overwrite any character which is stored in the buffer waiting for transfer to the parallel-to-serial converter in the device. This can be avoided by waiting for TxRDY to be asserted before sending a command if transmission is taking place. Due to the internal structure of the USART, it is also possible to disturb the transmission if a command is sent while a SYN character is being generated by the device. (The USART generates a SYN if the software fails to respond to TxRDY.) If this occurrence is possible in a system, commands should be transferred only when a positive-going edge is detected on the TxRDY line.

2. RxE only acts as a mask to RxRDY; it does not control the operation of the receiver.

   When the receiver is enabled, it is possible for it to already contain one or two characters. These characters should be read and discarded when the RxE bit is first set. Because of these extraneous characters the proper sequence for gaining synchronization is as follows:

   1. Disable interrupts
   2. Issue a command to enter hunt mode, clear errors, and enable the receiver (EH,ER,RxE= 1)
   3. Read USART data (it is not necessary to check status)
   4. Enable interrupts

   The first RxRDY that occurs after the above sequence will indicate that the SYN character or characters have been detected and the next character has been assembled and is ready to be read.

3. Loss of CTS or dropping TxEnable will immediately clamp the serial output line.

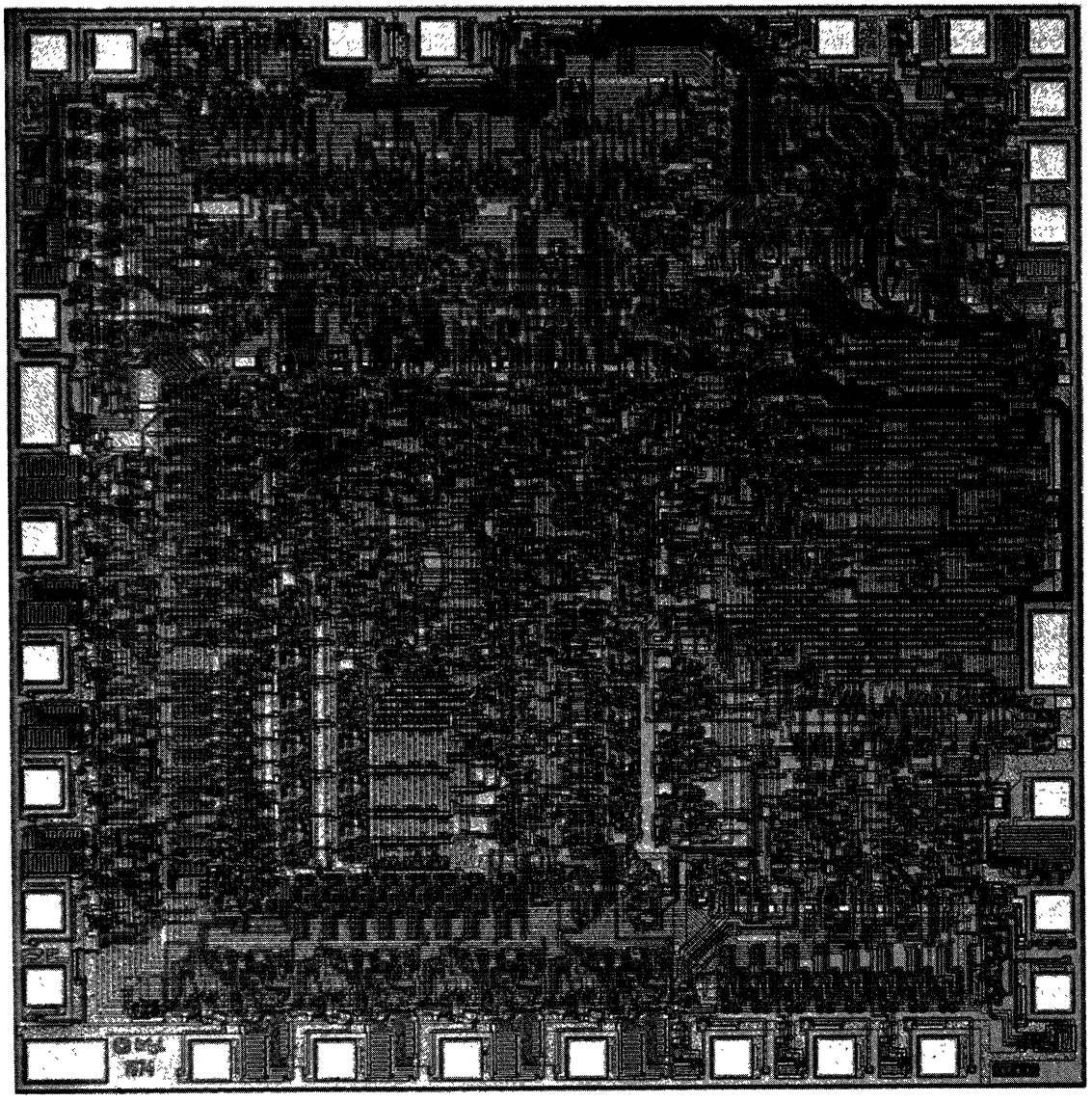
   TxEnable and RTS should remain asserted until the transmission is complete. Note that this implies that not only has the USART completed the transfer of all bits of the last character, but also that they have cleared the modem. A delay of 1 msec following a proper occurrence of TxEmpty is usually sufficient (see item 4). An additional problem can occur in the synchronous mode because the loss of TxEnable clamps the data in at a SPACE instead of the normal MARK. This problem, which does not occur in the asynchronous mode, can be corrected by an external gate combining RTS and the serial output data.

4. Extraneous transitions can occur on TxEmpty while data (including USART generated SYNs) is transferred to the parallel-to-serial converter.

   This situation can be avoided by ensuring that TxEmpty occurs during several consecutive status reads before assuming that the transmitter is truly in the empty state.

5. A BREAK (i.e., long space) detected by the receiver results in a string of characters which have framing errors.

   If reception is to be continued after a BREAK, care must be taken to ensure that valid data is being received; special care must be taken with the last character perceived during a BREAK, since its value, including any framing error associated with it, is indeterminate.



intel®

APPLICATION NOTE

AP-36

March 1978

Using the 8273 SDLC/HDLC Protocol Controller

John Beaston
Microcomputer Applications

INTRODUCTION

The Intel 8273 is a Data Communications Protocol Controller designed for use in systems utilizing either SDLC or HDLC (Synchronous or High-Level Data Link Control) protocols. In addition to the usual features such as full duplex operation, automatic Frame Check Sequence generation and checking, automatic zero bit insertion and deletion, and TTL compatibility found on other single component SDLC controllers; the 8273 features a frame level command structure, a digital phase locked loop, SDLC loop operation, and diagnostics.

The frame level command structure is made possible by the 8273's unique internal dual processor architecture. A high-speed bit processor handles the serial data manipulations and character recognition. A byte processor implements the frame level commands. These dual processors allow the 8273 to control the necessary byte-by-byte operation of the data channel with a minimum of CPU (Central Processing Unit) intervention. For the user this means the CPU has time to take on additional tasks. The digital phase locked loop (DPLL) provides a means of clock recovery from the received data stream on-chip. This feature, along with the frame level commands, makes SDLC loop operation extremely simple and flexible. Diagnostics in the form of both data and clock loopback are available to simplify board debug and link testing. The 8273 is a dedicated function peripheral in the MCS-80/85 Microcomputer family and as such, it interfaces to the 8080/8085 system with a minimum of external hardware.

This application note explains the 8273 as a component and shows its use in a generalized loop configuration and a typical 8085 system. The 8085 system was used to verify the SDLC operation of the 8273 on an actual IBM SDLC data communications link.

The first section of this application note presents an overview of the SDLC/HDLC protocols. It is fairly tutorial in nature and may be skipped by the more knowledgeable reader. The second section describes the 8273 from a functional standpoint with explanation of the block diagram. The software aspects of the 8273, including command examples, are discussed in the third section. The fourth and fifth sections discuss a loop SDLC configuration and the 8085 system respectively.

SDLC/HDLC OVERVIEW

SDLC is a protocol for managing the flow of information on a data communications link. In other words, SDLC can be thought of as an envelope — addressed, stamped, and containing an s.a.s.e. — in which information is transferred from location to location on a data communications link. (Please note that while SDLC is discussed specifically, all comments also apply to HDLC except where noted.) The link may be either point-to-point or multi-point, with the point-to-point configuration being either switched or nonswitched. The information flow may use either full or half duplex exchanges. With this many configurations supported, it is difficult to find a synchronous data communications application where SDLC would not be appropriate.

Aside from supporting a large number of configurations, SDLC offers the potential of a 2× increase in throughput over the presently most prevalent protocol: Bi-Sync. This performance increase is primarily due to two characteristics of SDLC: full duplex operation and the implied acknowledgement of transferred information. The performance increase due to full duplex operation is fairly obvious since, in SDLC, both stations can communicate simultaneously. Bi-Sync supports only half-duplex (two-way alternate) communication. The increase from implied acknowledgement arises from the fact that a station using SDLC may acknowledge previously received information while transmitting different information. Up to 7 messages may be outstanding before an acknowledgement is required. These messages may be acknowledged as a block rather than singly. In Bi-Sync, acknowledgements are unique messages that may not be included with messages containing information and each information message requires a separate acknowledgement. Thus the line efficiency of SDLC is superior to Bi-Sync. On a higher level, the potential of a 2× increase in performance means lower cost per unit of information transferred. Notice that the increase is not due to higher data link speeds (SDLC is actually speed independent), but simply through better line utilization.

Getting down to the more salient characteristics of SDLC; the basic unit of information on an SDLC link is that of the frame. The frame format is shown in Figure 1. Five fields comprise each frame: flag, address, control, information, and frame check sequence. The flag fields (F) form the boundary of the frame and all other fields are positionally related to one of the two flags. All frames start with an opening flag and end with a closing flag. Flags are used for frame synchronization. They also may serve as time-fill characters between frames. (There are no intraframe time-fill characters in SDLC as there are in Bi-Sync.) The opening flag serves as a reference point for the address (A) and control (C) fields. The frame check sequence (FCS) is referenced from the closing flag. All flags have the binary configuration 01111110 (7EH).

SDLC is a bit-oriented protocol, that is, the receiving station must be able to recognize a flag (or any other special character) at any time, not just on an 8-bit boundary. This, of course, implies that a frame may be N-bits in length. (The vast majority of applications tend to use frames which are multiples of 8 bits long, however.)

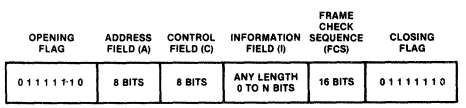


Figure 1. SDLC Frame Format

The fact that the flag has a unique binary pattern would seem to limit the contents of the frame since a flag pattern might inadvertently occur within the frame. This would cause the receiver to think the closing flag was received, invalidating the frame. SDLC handles this situation through a technique called zero bit insertion. This techniques specifies that within a frame a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1s. Thus, no pattern of 01111110 is ever transmitted by chance. On the receiving end, after the opening flag is detected, the receiver removes any 0 following 5 consecutive 1s. The inserted and deleted 0s are not counted for error determination.

Before discussing the address field, an explanation of the roles of an SDLC station is in order. SDLC specifies two types of stations: primary and secondary. The primary is the control station for the data link and thus has responsibility of the overall network. There is only one predetermined primary station, all other stations on the link assume the secondary station role. In general, a secondary station speaks only when spoken to. In other words, the primary polls the secondaries for responses. In order to specify a specific secondary, each secondary is assigned a unique 8-bit address. It is this address that is used in the frame's address field.

When the primary transmits a frame to a specific secondary, the address field contains the secondary's address. When responding, the secondary uses its own address in the address field. The primary is never identified. This ensures that the primary knows which of many secondaries is responding since the primary may have many messages outstanding at various secondary stations. In addition to the specific secondary address, an address common to all secondaries may be used for various purposes. (An all 1s address field is usually used for this "All Parties" address.) Even though the primary may use this common address, the secondaries are expected to respond with their unique address. The address field is always the first 8 bits following the opening flag.

The 8 bits following the address field form the control field. The control field embodies the link-level control of SDLC. A detailed explanation of the commands and responses contained in this field is beyond the scope of this application note. Suffice it to say that it is in the control field that the implied acknowledgement is carried out through the use of frame sequence numbers. None of the currently available SDLC single chip controllers utilize the control field. They simply pass it to the processor for analysis. Readers wishing a more detailed explanation of the control field, or of SDLC in general, should consult the IBM documents referenced on the front page overleaf.

In some types of frames, an information field follows the control field. Frames used strictly for link management may or may not contain one. When an information field is used, it is unrestricted in both content and length. This code transparency is made possible because of the zero bit insertion mentioned earlier and the bit-oriented nature of SDLC. Even main memory core dumps may be transmitted because of this capability. This feature is unique to bit-oriented protocols. Like the control field, the information field is not interpreted by the SDLC device; it is merely transferred to and from memory to be operated on and interpreted by the processor.

The final field is the frame check sequence (FCS). The FCS is the 16 bits immediately preceding the closing flag. This 16-bit field is used for error detection through a Cyclic Redundancy Checkword (CRC). The 16-bit transmitted CRC is the complement of the remainder obtained when the A, C, and I fields are "divided" by a generating polynomial. The receiver accumulates the A, C, and I fields and also the FCS into its internal CRC register. At the closing flag, this register contains one particular number for an error-free reception. If this number is not obtained, the frame was received in error and should be discarded. Discarding the frame causes the station to not update its frame sequence numbering. This results in a retransmission after the station sends an acknowledgement from previous frames. [Unlike all other fields, the FCS is transmitted MSB (Most Significant Bit) first. The A, C, and I fields are transmitted LSB (Least Significant Bit) first.] The details of how the FCS is generated and checked is beyond the scope of this application note and since all single component SDLC controllers handle this function automatically, it is usually sufficient to know only that an error has or has not occurred. The IBM documents contain more detailed information for those readers desiring it.

The closing flag terminates the frame. When the closing flag is received, the receiver knows that the preceding 16 bits constitute the FCS and that any bits between the control field and the FCS constitute the information field.

SDLC does not support an interframe time-fill character such as the SYN character in Bi-Sync. If an unusual condition occurs while transmitting, such as data is not available in time from memory or CTS (Clear-to-Send) is lost from the modem, the transmitter aborts the frame by sending an Abort character to notify the receiver to invalidate the frame. The Abort character consists of eight contiguous 1s sent without zero bit insertion. Intraframe time-fill consists of either flags, Abort characters, or any combination of the two.

While the Abort character protects the receiver from transmitted errors, errors introduced by the transmission medium are discovered at the receiver through the FCS check and a check for invalid frames. Invalid frames are those which are not bounded by flags or are too short, that is, less than 32 bits between flags. All invalid frames are ignored by the receiver.

Although SDLC is a synchronous protocol, it provides an optional feature that allows its use on basically asynchronous data links — NRZI (Non-Return-to-Zero-Inverted) coding. NRZI coding specifies that the signal condition does not change for transmitting a binary 1, while a binary 0 causes a change of state. Figure 2 illustrates NRZI coding compared to the normal NRZ. NRZI coding guarantees that an active line will have a transition at least every 5-bit times; long strings of zeroes cause a transition every bit time, while long strings of 1s are broken up by zero bit insertion. Since asynchronous

operation requires that the receiver sampling clock be derived from the received data, NRZI encoding plus zero bit insertion make the design of clock recovery circuitry easier.

All of the previous discussion has applied to SDLC on either point-to-point or multi-point data networks, SDLC (but not HDLC) also includes specification for a loop configuration. Figure 3 compares these three configurations. IBM uses this loop configuration in its 3650 Retail Store System. It consists of a single loop controller station with one or more down-loop secondary stations. Communications on a loop rely on the secondary stations repeating a received message down loop with a delay of one bit time. The reason for the one bit delay will be evident shortly.

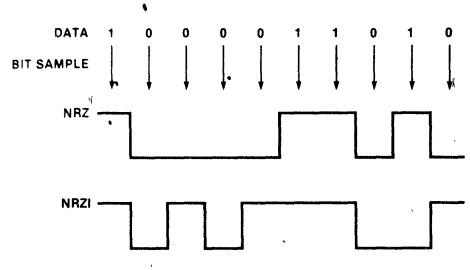


Figure 2. NRZI vs NRZ Encoding

Loop operation defines a new special character: the EOP (End-of-Poll) character which consists of a 0 followed by 7 contiguous, non-zero bit inserted, ones. After the loop controller transmits a message, it idles the line (sends all 1s). The final zero of the closing flag plus the first 7 1s of the idle form an EOP character. While repeating, the secondaries monitor their incoming line for an EOP character. When an EOP is detected, the secondary checks to see if it has a message to transmit. If it does, it changes the seventh 1 to a 0 (the one bit delay allows time for this) and repeats the modified EOP (now alias flag). After this flag is transmitted, the secondary terminates its repeater function and inserts its message (with multiple preceding flags if necessary). After the closing flag, the secondary resumes its one bit delay repeater function. Notice that the final zero of the secondary's closing flag plus the repeated 1s from the controller form an EOP for the next down-loop secondary, allowing it to insert a message if it desires.

One might wonder if the secondary missed any messages from the controller while it was inserting its own message. It does not. Loop operation is basically half-duplex. The controller waits until it receives an EOP before it transmits its next message. The controller's reception of the EOP signifies that the original message has propagated around the loop followed by any messages inserted by the secondaries. Notice that secondaries cannot communicate with one another directly, all secondary-to-secondary communication takes place by way of the controller.

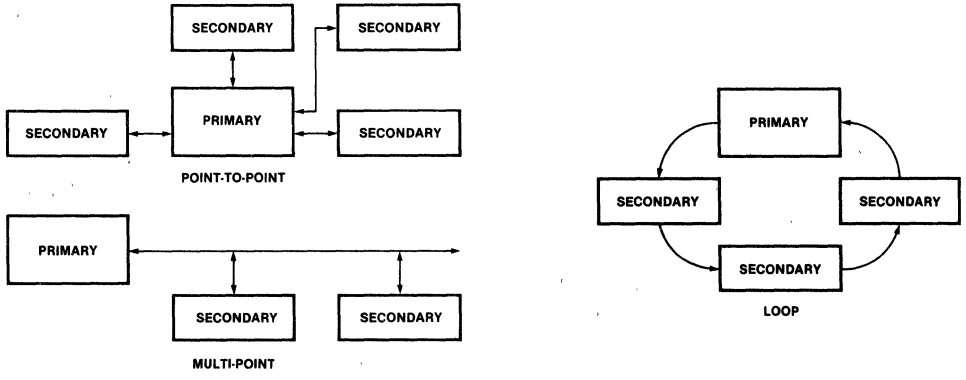


Figure 3 Network Configurations

Loop protocol does not utilize the normal Abort character. Instead, an abort is accomplished by simply transmitting a flag character. Down loop, the receiver sees the abort as a frame which is either too short (if the abort occurred early in the frame) or one with an FCS error. Either results in a discarded frame. For more details on loop operation, please refer to the IBM documents referenced earlier.

Another protocol very similar to SDLC which the 8273 supports is HDLC (High-Level Data Link Control). There are only three basic differences between the two: HDLC offers extended address and control fields, and the HLDC Abort character is 7 contiguous 1s as opposed to SDLC's 8 contiguous 1s.

Extended addressing, beyond the 256 unique addresses possible with SDLC, is provided by using the address field's least significant bit as the extended address modifier. The receiver examines this bit to determine if the octet should be interpreted as the final address octet. As long as the bit is 0, the octet that contains it is considered an extended address. The first time the bit is a 1, the receiver interprets that octet as the final address octet. Thus the address field may be extended to any number of octets. Extended addressing is illustrated in Figure 4a.

A similar technique is used to extend the control field although the extension is limited to only one extra control octet. Figure 4b illustrates control field extension.

Those readers not yet asleep may have noticed the similarity between the SDLC loop EOP character (a 0 followed by 7 1s) and the HDLC Abort (7 1s). This possible incompatibility is neatly handled by the HDLC protocol not specifying a loop configuration.

This completes our brief discussion of the SDLC/HDLC protocols. Now let us turn to the 8273 in particular and discuss its hardware aspects through an explanation of the block diagram and generalized system schematics.

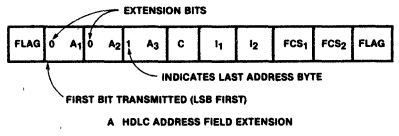


Figure 4a

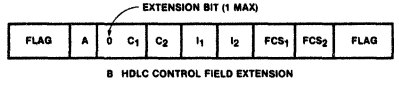


Figure 4b

BASIC 8273 OPERATION

It will be helpful for the following discussions to have some idea of the basic operation of the 8273. Each operation, whether it is a frame transmission, reception or port read, etc., is comprised of three phases: the Command, Execution, and Result phases. Figure 5 shows the sequence of these phases. As an illustration of this sequence, let us look at the transmit operation.

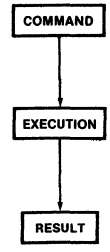


Figure 5. 8273 Operational Phases

When the CPU decides it is time to transmit a frame, the Command phase is entered by the CPU issuing a Transmit Frame command to the 8273. It is not sufficient to just instruct the 8273 to transmit. The frame level command structure sometimes requires more information such as frame length and address and control field content. Once this additional information is supplied, the Command phase is complete and the Execution phase is entered. It is during the Execution phase that the actual operation, in this case a frame transmission, takes place. The 8273 transmits the opening flag, A and C fields, the specified number of I field bytes, inserts the FCS, and closes with the closing flag. Once the closing flag is transmitted, the 8273 leaves the Execution phase and begins the Result phase. During the Result phase the 8273 notifies the CPU of the outcome of the command by supplying interrupt results. In this case, the results would be either that the frame is complete or that some error condition causes the transmission to be aborted. Once the CPU reads all of the results (there is only one for the Transmit Frame command), the Result phase and consequently the operation, is complete. Now that we have a general feeling for the operation of the 8273, let us discuss the 8273 in detail.

HARDWARE ASPECTS OF THE 8273

The 8273 block diagram is shown in Figure 6. It consists of two major interfaces: the CPU module interface and the modem interface. Let's discuss each interface separately.

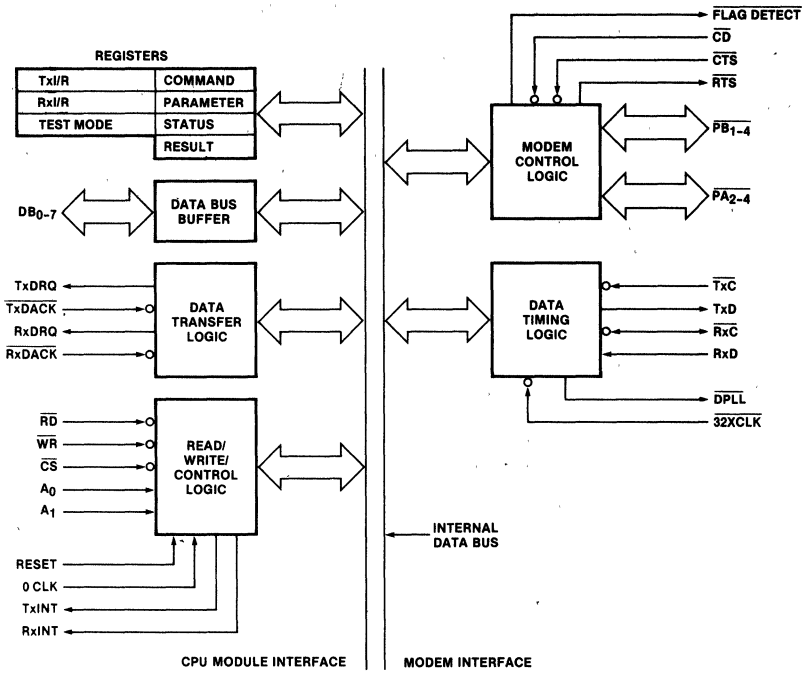


Figure 6. 8273 Block Diagram

CPU Interface

The CPU interface consists of four major blocks: Control/Read/Write logic (C/R/W), internal registers, data transfer logic, and data bus buffers.

The CPU module utilizes the C/R/W logic to issue commands to the 8273. Once the 8273 receives a command and executes it, it returns the results (good/bad completion) of the command by way of the C/R/W logic. The C/R/W logic is supported by seven registers which are addressed via the A_0, A_1, $\overline{RD}$, and $\overline{WR}$ signals, in addition to $\overline{CS}$. The A_0 and A_1 signals are generally derived from the two low order bits of the CPU module address bus while $\overline{RD}$ and $\overline{WR}$ are the normal I/O Read and Write signals found on the system control bus. Figure 7 shows the address of each register using the C/R/W logic. The function of each register is defined as follows:

ADDRESS INPUTS		CONTROL INPUTS	
A_1	A_0	$\overline{CS} \cdot \overline{RD}$	$\overline{CS} \cdot \overline{WR}$
0	0	STATUS	COMMAND
0	1	RESULT	PARAMETER
1	0	TxI/R	TEST MODE
1	1	RxI/R	—

Figure 7. 8273 Register Selection

Command — 8273 operations are initiated by writing the appropriate command byte into this register.

Parameter — Many commands require more information than found in the command itself. This additional information is provided by way of the parameter register.

Immediate Result (Result) — The completion information (results) for commands which execute immediately are provided in this register.

Transmit Interrupt Result (TxI/R) — Results of transmit operations are passed to the CPU in this register.

Receiver Interrupt Result (RxI/R) — Receive operation results are passed to the CPU via this register.

Status — The general status of the 8273 is provided in this register. The Status register supplies the handshaking necessary during various phases of the 8273 operation.

Test Mode — This register provides a software reset function for the 8273.

The commands, parameters, and bit definition of these registers are discussed in the following software section. Notice that there are not specific transmit or receive data registers. This feature is explained in the data transfer logic discussion.

The final elements of the C/R/W logic are the interrupt lines (RxINT and TxINT). These lines notify the CPU module that either the transmitter or the receiver requires service; i.e., results should be read from the appropriate interrupt result register or a data transfer is required. The interrupt request remains active until all the associated interrupt results have been read or the data transfer is performed. Though using the interrupt lines relieves the CPU module of the task of polling the 8273 to check if service is needed, the state of each interrupt line is reflected by a bit in the Status register and non-interrupt driven operation is possible by examing the contents of these bits periodically.

The 8273 supports two independent data interfaces through the data transfer logic; receive data and transmit data. These interfaces are programmable for either DMA or non-DMA data transfers. While the choice of the configuration is up to the system designer, it is based on the intended maximum data rate of the communications channel. Figure 8 illustrates the transfer rate of data bytes that are acquired by the 8273 based on link data rate. Full-duplex data rates above 9600 baud usually require DMA. Slower speeds may or may not require DMA depending on the task load and interrupt response time of the processor.

Figure 9 shows the 8273 in a typical DMA environment. Notice that a separate DMA controller, in this case the Intel 8257, is required. The DMA controller supplies the timing and addresses for the data transfers while the 8273 manages the requesting of transfers and the actual counting of the data block lengths. In this case, elements of the data transfer interface are:

TxDRQ: Transmit DMA Request — Asserted by the 8273, this line requests a DMA transfer from memory to the 8273 for transmit.

TxDACK: Transmit DMA Acknowledge — Returned by the 8257 in response to TxDRQ, this line notifies the 8273 that a request has been granted, and provides access to the transmitter data register.

RxDRQ: Receiver DMA Request — Asserted by the 8273, it requests a DMA transfer from the 8273 to memory for a receive operation.

TxDACK: Receiver DMA Acknowledge — Returned by the 8257, it notifies the 8273 that a receive DMA cycle has been granted, and provides access to the receiver data register.

RD: Read — Supplied by the 8257 to indicate data is to be read from the 8273 and placed in memory.

WR: Write — Supplied by the 8257 to indicate data is to be written to the 8273 from memory.

To request a DMA transfer the 8273 raises the appropriate DMA request line; let us assume it is a transmitter request (TxDRQ). Once the 8257 obtains control of the system bus by way of its HOLD and HLDA (hold acknowledge) lines, it notifies the 8273 that TxDRQ has been granted by returning $\overline{\text{TxDACK}}$ and $\overline{\text{WR}}$. The $\overline{\text{TxDACK}}$ and $\overline{\text{WR}}$ signals transfer data to the 8273 for a transmit, independent of the 8273 chip select pin ($\overline{\text{CS}}$). A similar sequence of events occurs for receiver requests. This "hard select" of data into the transmitter or out of the receiver alleviates the need for the normal transmit and receive data registers addressed by a combination of address lines, CS, and WR or RD. Competitive devices that do not have this "hard select" feature require the use of an external multiplexer to supply the correct inputs for register selection during DMA. (Do not forget that the SDLC controller sees both the addresses and control signals supplied by the DMA controller during DMA cycles.) Let us look at typical frame transmit and frame receive sequences to better see how the 8273 truly manages the DMA data transfer.

Before a frame can be transmitted, the DMA controller is supplied, by the CPU, the starting address for the desired information field. The 8273 is then commanded to transmit a frame. (Just how this is done is covered later during our software discussion.) After the command, but before transmission begins, the 8273 needs a little more information (parameters). Four parameters are required for the transmit frame command: the address field byte, the control field byte, and two bytes which are the least significant and most significant bytes of the information field byte length. Once all four parameters are loaded, the 8273 makes RTS (Request-to-Send) active and waits for CTS (Clear-to-Send) to go active. Once CTS is active, the 8273 starts the frame transmission. While the 8273 is transmitting the opening flag, address field, and control field; it starts making transmitter DMA requests. These requests continue at character (byte) boundaries until the pre-loaded number of bytes of information field have been transmitted. At this point the requests stop, the FCS and closing flag are transmitted, and the TxINT line is raised, signaling the CPU that the frame transmission is complete. Notice that after the initial command and parameter loading, absolutely no CPU intervention was required (since DMA is used for data transfers) until the entire frame was transmitted. Now let's look at a frame reception.

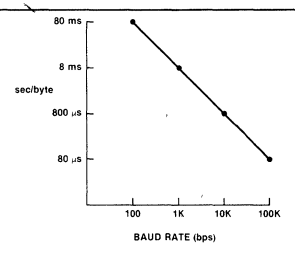


Figure 8. Byte Transfer Rate vs Baud Rate

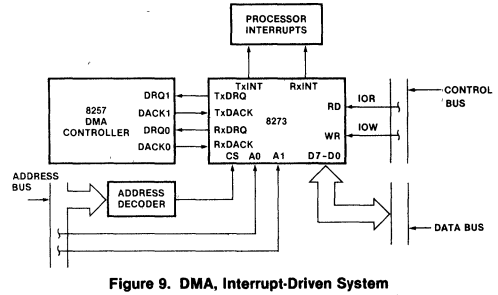


Figure 9. DMA, Interrupt-Driven System

The receiver operation is very similar. Like the initial transmit sequence, the DMA controller is loaded with a starting address for a receiver data buffer and the 8273 is commanded to receive. Unlike the transmitter, there are two different receive commands: General Receive, where all received frames are transferred to memory, and Selective Receive, where only frames having an address field matching one of two preprogrammed 8273 address fields are transferred to memory. Let's assume for now that we want to general receive. After the receive command, two parameters are required before the receiver becomes active: the least significant and most significant bytes of the receiver buffer length. Once these bytes are loaded, the receiver is active and the CPU may return to other tasks. The next frame appearing at the receiver input is transferred to memory using receiver DMA requests. When the closing flag is received, the 8273 checks the FCS and raises its RxINT line. The CPU can then read the results which indicate if the frame was error-free or not. (If the received frame had been longer than the pre-loaded buffer length, the CPU would have been notified of that occurrence earlier with a receiver error interrupt. The command description section contains a complete list of error conditions.) Like the transmit example, after the initial command, the CPU is free for other tasks until a frame is completely received. These examples have illustrated the 8273's management of both the receiver and transmitter DMA channels.

It is possible to use the DMA data transfer interface in a non-DMA interrupt-driven environment. In this case, 4 interrupt levels are used: one each for TxINT and RxINT, and one each for TxDRQ and RxDRQ. This configuration is shown in Figure 10. This configuration offers the advantages that no DMA controller is required and data requests are still separated from result (completion) requests. The disadvantages of the configuration are that 4 interrupt levels are required and that the CPU must actually supply the data transfers. This, of course, reduces the maximum data rate compared to the configuration based strictly on DMA. This system could use an Intel 8259 8-level Priority Interrupt Controller to supply a vectored CALL (subroutine) address based on requests on its inputs. The 8273 transmitter and receiver make data requests by raising the respective DRQ line. The CPU is interrupted by the 8259 and vectored to a data transfer routine. This routine either writes (for transmit) or reads (for receive) the 8273 using the respective TxDACK or RxDACK line. As in the case above, the DACK lines serve as "hard" chip selects into and out of the 8273. $\overline{\text{TxDACK}}$ + $\overline{\text{WR}}$ writes data into the 8273 for transmit. $\overline{\text{RxDACK}}$ + $\overline{\text{RD}}$ reads data from the 8273 for receive.) The CPU is notified of operation completion and results by way of TxINT and RxINT lines. Using the 8273, and the 8259, in this way, provides a very effective, yet simple, interrupt-driven interface.

Figure 11 illustrates a system very similar to that described above. This system utilizes the 8273 in a non-DMA data transfer mode as opposed to the two DMA approaches shown in Figures 9 and 10. In the non-DMA case, data transfer requests are made on the TxINT and RxINT lines. The DRQ lines are not used. Data transfer requests are separated from result requests by a bit in the Status register. Thus, in response to an interrupt, the CPU reads the Status register and branches to either a result or a data transfer routine based on the status of one bit. As before, data transfers are made via using the DACK lines as chip selects to the transmitter and receiver data registers.

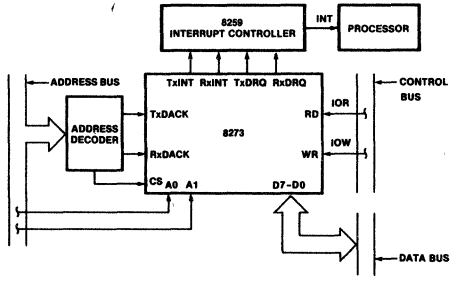


Figure 10. Interrupt-Based DMA System

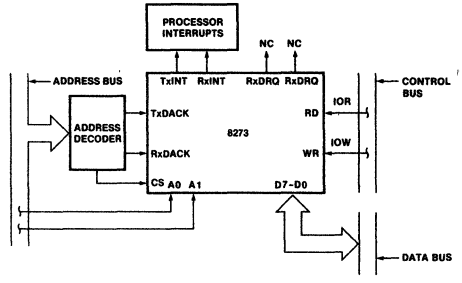


Figure 11. Non-DMA Interrupt-Driven System

Figure 12 illustrates the simplest system of all. This system utilizes polling for all data transfers and results. Since the interrupt pins are reflected in bits in the Status register, the software can read the Status register periodically looking for one of these to be set. If it finds an INT bit set, the appropriate Result Available bit is examined to determine if the "interrupt" is a data transfer or completion result. If a data transfer is called for, the DACK line is used to enter or read the data from the 8273. If the interrupt is a completion result, the appropriate result register is read to determine the good/bad completion of the operation.

The actual selection of either DMA or non-DMA modes is controlled by a command issued during initialization. This command is covered in detail during the software discussion.

The final block of the CPU module interface is the Data Bus Buffer. This block supplies the tri-state, bidirectional data bus interface to allow communication to and from the 8273.

Modem Interface

As the name implies, the modem interface is the modem side of the 8273. It consists of two major blocks: the modem control block and the serial data timing block.

The modem control block provides both dedicated and user-defined modem control functions. All signals supported by this interface are active low so that EIA inverting drivers (MC1488) and inverting receivers (MC1489) may be used to interface to standard modems.

Port A is a modem control input port. Its representation on the data bus is shown in Figure 13. Bits D_0 and D_1 have dedicated functions. D_0 reflects the logical state of the $\overline{CTS}$ (Clear-to-Send) pin. [If $\overline{CTS}$ is active (low), D_0 is a 1.] This signal is used to condition the start of a transmission. The 8273 waits until $\overline{CTS}$ is active before it starts transmitting a frame. While transmitting, if $\overline{CTS}$ goes inactive, the frame is aborted and the CPU is interrupted. When the CPU reads the interrupt result, a $\overline{CTS}$ failure is indicated.

D_1 reflects the logical state of the $\overline{CD}$ (Carrier Detect) pin. $\overline{CD}$ is used to condition the start of a frame reception. $\overline{CD}$ must be active in time for a frame's address field. If $\overline{CD}$ is lost (goes inactive) while receiving a frame, an interrupt is generated with a $\overline{CD}$ failure result. $\overline{CD}$ may go inactive between frames.

Bits D_2 thru D_4 reflect the logical state of the $\overline{PA_2}$ thru $\overline{PA_4}$ pins respectively. These inputs are user defined. The 8273 does not interrogate or manipulate these bits. Bits D_5, D_6,and D_7 are not used and each is read as a 1 for a Read Port A command.

Port B is a modem control output port. Its data bus representation is shown in Figure 14. As in Port A, the bit values represent the logical condition of the pins. D_0 and D_5 are dedicated function outputs. D_0 represents the $\overline{RTS}$ (Request-to-Send) pin. $\overline{RTS}$ is normally used to notify the modem that the 8273 wishes to transmit. This function is handled automatically by the 8273. If $\overline{RTS}$ is inactive (pin is high) when the 8273 is commanded to transmit, the 8273 makes it active and then waits for $\overline{CTS}$ before transmitting the frame. One byte time after the end of the frame, the 8273 returns $\overline{RTS}$ to its inactive state. However, if $\overline{RTS}$ was active when a transmit command is issued, the 8273 leaves it active when the frame is complete.

Bit D_5 reflects the state of the $\overline{Flag\ Detect}$ pin. This pin is activated whenever an active receiver sees a flag character. This function is useful to activate a timer for line activity timeout purposes.

Bits D_1 thru D_4 provide four user-defined outputs. Pins $\overline{PB_1}$ thru $\overline{PB_4}$ reflect the logical state of these bits. The 8273 does not interrogate or manipulate these bits. D_6 and D_7 are not used. In addition to being able to output to Port B, Port B may be read using a Read Port B command. All Modem control output pins are forced high on reset. (All commands mentioned in this section are covered in detail later.).

The final block to be covered is the serial data timing block. This block contains two sections: the serial data logic and the digital phase locked loop (DPLL).

Elements of the serial data logic section are the data pins, TxD (transmit data output) and RxD (receive data input), and the respective data clocks, $\overline{TxC}$ and $\overline{RxC}$. The transmit and receive data is synchronized by the $\overline{TxC}$ and $\overline{RxC}$ clocks. Figure 15 shows the timing for these signals. The leading edge (negative transition) of $\overline{TxC}$ generates new transmit data and the trailing edge (positive transition) of $\overline{RxC}$ is used to capture the receive data.

It is possible to reconfigure this section under program control to perform diagnostic functions; both data and clock loopback are available. In data loopback mode, the TxD pin is internally routed to the RxD pin. This allows simple board checkout since the CPU can send an SDLC message to itself. (Note that transmitted data will still appear on the TxD pin.)

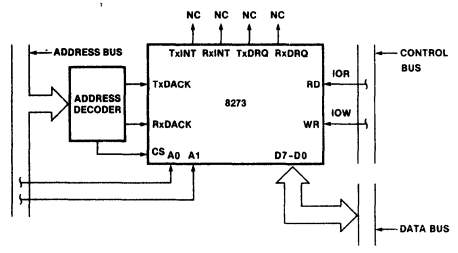


Figure 12. Polled System

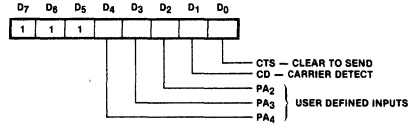


Figure 13. Port A (Input) Bit Definition

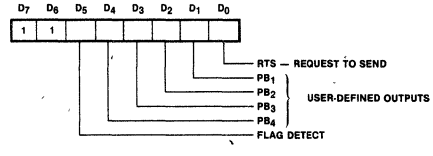


Figure 14. Port B (Output) Bit Definition

When data loopback is utilized, the receiver may be presented incorrect sample timing ($\overline{RxC}$) by the external circuitry. Clock loopback overcomes this problem by allowing the internal routing of $\overline{TxC}$ and $\overline{RxC}$. Thus the same clock used to transmit the data is used to receive it. Examination of Figure 15 shows that this method ensures bit synchronism. The final element of the serial data logic is the Digital Phase Locked Loop.

The DPLL provides a means of clock recovery from the received data stream. This feature allows the 8273 to interface without external synchronizing logic to low cost asynchronous modems (modems which do not supply clocks). It also makes the problem of clock timing in loop configurations trivial.

To use the DPLL, a clock at 32 times the required baud rate must be supplied to the $\overline{32\times CLK}$ pin. This clock provides the interval that the DPLL samples the received data. The DPLL uses the 32× clock and the received data to generate a pulse at the $\overline{DPLL}$ output pin. This $\overline{DPLL}$ pulse is positioned at the nominal center of the received data bit cell. Thus the $\overline{DPLL}$ output may be wired to $\overline{RxC}$ and/or $\overline{TxC}$ to supply the data timing. The exact position of the pulse is varied depending on the line noise and bit distortion of the received data. The adjustment of the $\overline{DPLL}$ position is determined according to the rules outlined in Figure 16.

Adjustments to the sample phase of $\overline{DPLL}$ with respect to the received data is made in discrete increments. Referring to Figure 16, following the occurrence of $\overline{DPLL}$ pulse A, the DPLL counts $\overline{32\times CLK}$ pulses and examines the received data for a data edge. Should no edge be detected in 32 pulses, the $\overline{DPLL}$ positions the next $\overline{DPLL}$ pulse (B) at 32 clock pulses from pulse A. Since no new phase information is contained in the data stream, the sample phase is assumed to be at nominal 1× baud rate. Now assume a data edge occurs after $\overline{DPLL}$ pulse B. The distance from B to the next pulse C is influenced according to which quadrant (A_1, B_1, B_2, or A_2) the data edge falls in. (Each quadrant represents 8 $\overline{32\times CLK}$ times.) For example, if the edge is detected in quadrant A_1, it is apparent that pulse B was too close to the data edge and the time to the next pulse must be shortened. The adjustment for quadrant A_1 is specified as −2. Thus, the next $\overline{DPLL}$ pulse, pulse C, is positioned 32−2 or 30 $\overline{32\times CLK}$ pulses following $\overline{DPLL}$ pulse B. This adjustment moves pulse C closer to the nominal bit center of the next received data cell. A data edge occurring in quadrant B_2 would have caused the adjustment to be small, namely 32+1 or 33 $\overline{32\times CLK}$ pulses. Using this technique, the $\overline{DPLL}$ pulse converges to the nominal bit center within 12 data transitions, worse case — 4-bit times adjusting through quadrant A_1 or A_2 and 8-bit-times adjusting through B_1 or B_2.

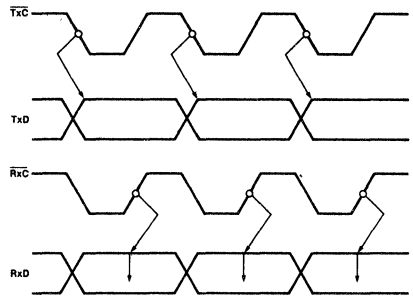


Figure 15. Transmit/Receive Timing

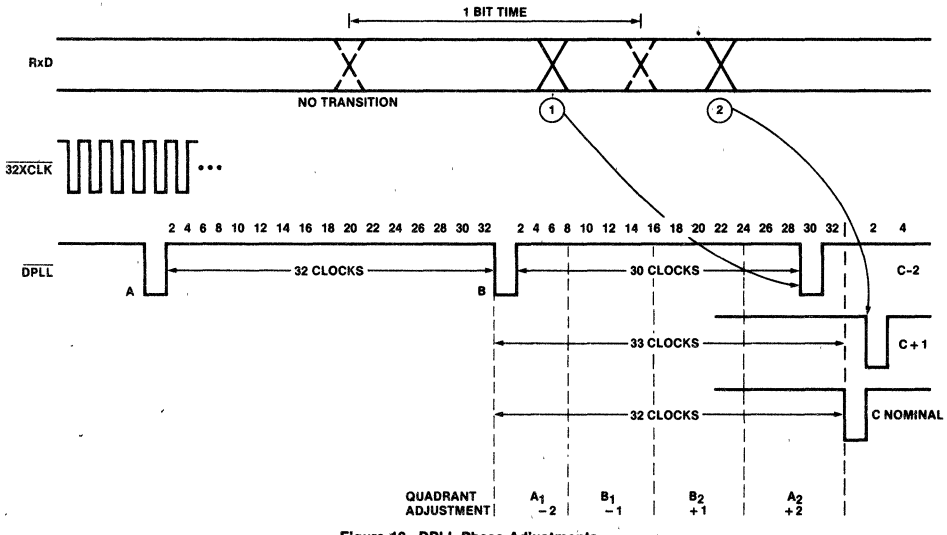


Figure 16. DPLL Phase Adjustments

When the receive data stream goes idle after 15 ones, DPLL pulses are generated at 32 pulse intervals of the 32x CLK. This feature allows the DPLL pulses to be used as both transmitter and receiver clocks.

In order to guarantee sufficient transitions of the received data to enable the $\overline{\text{DPLL}}$ to lock, NRZI encoding of the data is recommended. This ensures that, within a frame, data transitions occur at least every five bit times — the longest sequence of 1s which may be transmitted with zero bit insertion. It is also recommended that frames following a line idle be transmitted with preframe sync characters which provide a minimum of 12 transitions. This ensures that the $\overline{\text{DPLL}}$ is generating $\overline{\text{DPLL}}$ pulses at the nominal bit centers in time for the opening flag. (Two 00H characters meet this requirement by supplying 16 transitions with NRZI encoding. The 8273 contains a mode which supplies such a preframe sync.)

Figure 17 illustrates 8273 clock configurations using either synchronous or asynchronous modems. Notice how the $\overline{\text{DPLL}}$ output is used for both $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$ in the asynchronous case. This feature eliminates the need for external clock generation logic where low cost) asynchronous modems are used and also allows direct connection of 8273s for the ultimate in low cost data links. The configuration for loop applications is discussed in a following section.

This completes our discussion of the hardware aspects of the 8273. Its software aspects are now discussed.

SOFTWARE ASPECTS OF THE 8273

The software aspects of the 8273 involve the communication of both commands from the CPU to the 8273 and the return of results of those commands from the 8273 to the CPU. Due to the internal processor architecture of the 8273, this CPU-8273 communication is basically a form of interprocessor communication. Such communication usually requires a form of protocol of its own. This protocol is implemented through use of handshaking supplied in the 8273 Status register. The bit definition of this register is shown in Figure 18.

CBSY: Command Busy — CBSY indicates when the 8273 is in the command phase. CBSY is set when the CPU writes a command into the Command register, starting the Command phase. It is reset when the last parameter is deposited in the Parameter register and accepted by the 8273, completing the Command phase.

CBF: Command Buffer Full — When set, this bit indicates that a byte is present in the Command register. This bit is normally not used.

CPBF: Command Parameter Buffer Full — This bit indicates that the Parameter register contains a parameter. It is set when the CPU deposits a parameter in the Parameter register. It is reset when the 8273 accepts the parameter.

CRBF: Command Result Buffer Full — This bit is set when the 8273 places a result from an immediate type command in the Result register. It is reset when the CPU reads the result from the Result register.

RxINT: Receiver Interrupt — The state of the RxINT pin is reflected by this bit. RxINT is set by the 8273 whenever the receiver needs servicing. RxINT is reset when the CPU reads the results or performs the data transfer.

TxINT: Transmitter Interrupt — This bit is identical to RxINT except action is initiated based on transmitter interrupt sources.

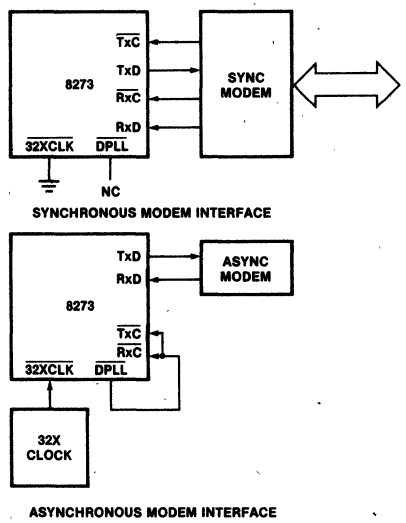


Figure 17. Serial Data Timing Configuration

RxIRA: Receiver Interrupt Result Available — RxIRA is set when the 8273 places an interrupt result byte into the RxI/R register. RxIRA is reset when the CPU reads the RxI/R register.

TxIRA: Transmitter Interrupt Result Available — TxIRA is the corresponding Result Available bit for the transmitter. It is set when the 8273 places an interrupt result byte in the TxI/R register and reset when the CPU reads the register.

The significance of each of these bits will be evident shortly. Since the software requirements of each 8273 phase are essentially independent, each phase is covered separately.

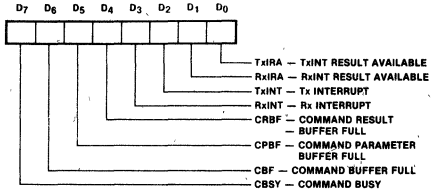


Figure 18. Status Register Format

Command Phase Software

Recalling the Command phase description in an earlier section, the CPU starts the Command phase by writing a command byte into the 8273 Command register. If further information about the command is required by the 8273, the CPU writes this information into the Parameter register. Figure 19 is a flowchart of the Command phase. Notice that the CBSY and CPBF bits of the Status register are used to handshake the command and parameter bytes. Also note that the chart shows that a command may not be issued if the Status register indicates the 8273 is busy (CBSY = 1). If a command is issued while CBSY = 1, the original command is overwritten and lost. (Remember that CBSY signifies the command phase is in progress and not the actual execution of the command.) The flowchart also includes a Parameter buffer full check. The CPU must wait until CPBF = 0 before writing a parameter to the Parameter register. If a parameter is issued while CPBF = 1, the previous parameter is overwritten and lost. An example of command output assembly language software is provided in Figure 20a. This software assumes that a command buffer exists in memory. The buffer is pointed at by the HL register. Figure 20b shows the command buffer structure.

The 8273 is a full duplex device, i.e., both the transmitter and receiver may be executing commands or passing interrupt results at any given time. (Separate Rx and Tx interrupt pins and result registers are provided for this reason.) However, there is only one Command register. Thus, the Command register must be used for only one command sequence at a time and the transmitter and receiver may never be simultaneously in a command phase. A detailed description of the commands and their parameters is presented in a following section.

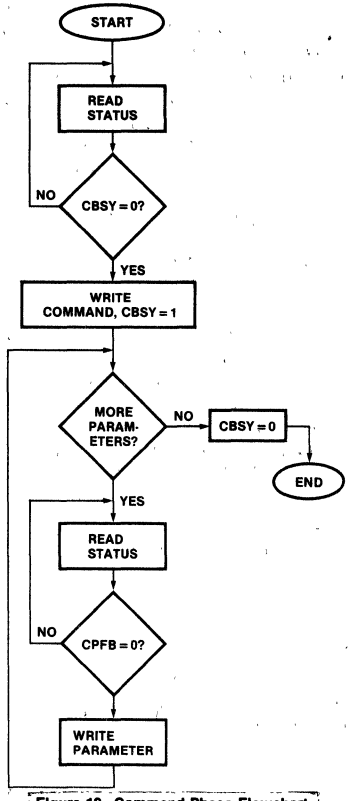


Figure 19. Command Phase Flowchart

```
;FUNCTION: COMMAND DISPATCHER
;INPUTS: HL - COMMAND BUFFER ADDRESS
;OUTPUTS: NONE
;CALLS: NONE
;DESTROYS: A,B,H,L,F/F'S
;DESCRIPTION: CMDOUT ISSUES THE COMMAND + PARAMETERS
;IN THE COMMAND BUFFER POINTED AT BY HL
;
CMDOUT: LXI     H,CMDBUF;POINT HL AT BUFFER
        MOV     B,M     ;1ST ENTRY IS PAR. COUNT
        INX     H       ;POINT AT COMMAND BYTE
CMD1:   IN      STAT73  ;READ 8273 STATUS
        RLC             ;ROTATE CBSY INTO CARRY
        JC      CMD1    ;WAIT UNTIL CBSY=0
        MOV     A,M     ;MOVE COMMAND BYTE TO A
        OUT     COMM73  ;PUT COMMAND IN COMMAND REG
CMD2:   MOV     A,B     ;GET PARAMETER COUNT
        ANA     A       ;TEST IF ZERO
        RZ              ;IF 0 THEN DONE
        INX     H       ;NOT DONE, SO POINT AT NEXT PAR
        DCR     B       ;DEC PARAMETER COUNT
CMD3:   IN      STAT73  ;READ 8273 STATUS
        ANI     CPBF    ;TEST CPBF BIT
        JNZ     CMD3    ;WAIT UNTIL CPBF IS 0
        MOV     A,M     ;GET PARAMETER FROM BUFFER
        OUT     PARM73  ;OUTPUT PAR TO PARAMETER REG
        JMP     CMD2    ;CHECK IF MORE PARAMETERS
```

Figure 20A. Command Phase Software

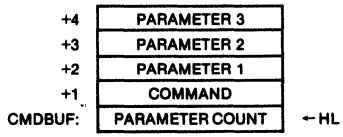


Figure 20B. Command Buffer Format

Execution Phase Software

During the Execution phase, the operation specified by the Command phase is performed. If the system utilizes DMA for data transfers, there is no CPU involvement during this phase, so no software is required. If non-DMA data transfers are used, either interrupts or polling is used to signal a data transfer request.

For interrupt-driven transfers the 8273 raises the appropriate INT pin. When responding to the interrupt, the CPU must determine whether it is a data transfer request or an interrupt signaling that an operation is complete and results are available. The CPU determines the cause by reading the Status register and interrogating the associated IRA (Interrupt Result Available) bit (TxIRA for TxINT and RxIRA for RxINT). If the IRA = 0, the interrupt is a data transfer request. If the IRA = 1, an operation is complete and the associated Interrupt Result register must be read to determine the completion status (good/bad/etc.). A software interrupt handler implementing the above sequence is presented as part of the Result phase software.

When polling is used to determine when data transfers are required, the polling routine reads the Status register looking for one of the INT bits to be set. When a set INT bit is found, the corresponding IRA bit is examined. Like in the interrupt-driven case, if the IRA = 0, a data transfer is required. If IRA = 1, an operation is complete and the Interrupt Result register needs to be read. Again, example polling software is presented in the next section.

Result Phase Software

During the Result phase the 8273 notifies the CPU of the outcome of a command. The Result phase is initiated by either a successful completion of an operation or an error detected during execution. Some commands such as reading or writing the I/O ports provide immediate results, that is, there is essentially no delay from the issuing of the command and when the result is available. Other commands such as frame transmit, take time to complete so their result is not available immediately. Separate result registers are provided to distinguish these two types of commands and to avoid interrupt handling for simple results.

Immediate results are provided in the Result register. Validity of information in this register is indicated to the CPU by way of the CRBF bit in the Status register. When the CPU completes the Command phase of an immediate command, it polls the Status register waiting until CRBF = 1. When this occurs, the CPU may read the Result register to obtain the immediate result. The Result register provides only the results from immediate commands.

Example software for handling immediate results is shown in Figure 21. The routine returns with the result in the accumulator. The CPU then uses the result as is appropriate.

All non-immediate commands deal with either the transmitter or receiver. Results from these commands are provided in the TxI/R (Transmit Interrupt Result) and RxI/R (Receive Interrupt Result) registers respectively. Results in these registers are conveyed to the CPU by the TxIRA and RxIRA bits of the Status register. Results of non-immediate commands consist of one byte result interrupt code indicating the condition for the interrupt and, if required, one or more bytes supplying additional information. The interrupt codes and the meaning of the additional results are covered following the detailed command description.

Non-immediate results are passed to the CPU in response to either interrupts or polling of the Status register. Figure 22 illustrates an interrupt-driven result handler. (Please note that all of the software presented in this application note is not optimized for either speed or code efficiency. They are provided as a guide and to illustrate concepts.) This handler provides for interrupt-driven data transfers as was promised in the last section. Users employing DMA-based transfers do not need the lines where the IRA bit is tested for zero. (These lines are denoted by an asterisk in the comments column.) Note that the INT bit is used to determine when all results have been read. All results must be read. Otherwise, the INT bit (and pin) will remain high and further interrupts may be missed. These routines place the results in a result buffer pointed at by RCRBUF and TxRBUF.

A typical result handler for systems utilizing polling is shown in Figure 23. Data transfers are also handled by this routine. This routine utilizes the routines of Figure 22 to handle the results.

At this point, the reader should have a good conceptual feel about how the 8273 operates. It is now time for the particulars of each command to be discussed.

```
;FUNCTION: IMDRLT
;INPUTS: NONE
;OUTPUTS: RESULT REGISTER IN A
;CALLS: NONE
;DESTROYS: A,F/F'S
;DESCRIPTION: IMDRLT IS CALLED AFTER A CMDOUT FOR AN
;IMMEDIATE COMMAND TO READ THE RESULT REGISTER
;
IMDRLT: IN      STAT73  ;READ 8273 STATUS
        ANI     CRBF    ;TEST IF RESULT REG READY
        JZ      IMDRLT  ;WAIT IF CRBF=0
        IN      RESL73  ;READ RESULT REGISTER
        RET     ;RETURN
```

Figure 21. Immediate Result Handler

```
;FUNCTION: RXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: RCRBUF, RCVPNT
;CALLS: NONE
;OUTPUTS: RCRBUF, RCVPNT
;DESTROYS: NOTHING
;DESCRIPTION: RXI IS ENTERED AT A RECEIVER INTERRUPT.
;THE INTERRUPT IS TESTED FOR DATA TRANSFER (IRA=0)
;OR RESULT (IRA=1). FOR DATA TRANSFER, THE DATA IS
;PLACED IN A BUFFER AT RCVPNT. RESULTS ARE PLACED IN
;A BUFFER AT RCRBUF.
;A FLAG(RXFLAG) IS SET IF THE INTERRUPT WAS A RESULT.
;(DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*) AND
;MAYBE ELIMINATED BY USERS USING DMA.
;
RXI:    PUSH    H       ;SAVE HL
        PUSH    PSW     ;SAVE PSW
        PUSH    B       ;SAVE B
        IN      STAT73  ; (*) READ 8273 STATUS
        ANI     RXIRA   ; (*) TEST IRA BIT
        JZ      RXI2    ; (*) IF 0, DATA TRANSFER NEEDED
RXI1:   LHLD    RCRBUF  ;GET RESULT BUFFER POINTER
        IN      STAT73  ;READ 8273 STATUS AGAIN
        ANI     RXINT   ;TEST INT BIT
        JZ      RXI4    ;IF 0, THEN DONE
        IN      STAT73  ;READ 8273 STATUS AGAIN
        ANI     RXIRA   ;TEST IRA AGAIN
        JZ      RXI1    ;LOOP UNTIL RESULT IS READY
        IN      RXIR73  ;READY, READ RXI/R
        MOV     M,A     ;STORE RESULT IN BUFFER
        INX     H       ;BUMP RESULT POINTER
        SHLD    RCRBUF  ;RESTORE BUFFER POINTER
        JMP     RXI1    ;GO BACK TO SEE IF MORE
RXI2:   SHLD    RCVPNT  ; (*) GET DATA BUFFER POINTER
        IN      RCVDAT  ; (*) READ DATA VIA RXDACK
        MOV     M,A     ; (*) STORE DATA IN BUFFER
        INX     H       ; (*) BUMP DATA POINTER
        JMP     RXI3    ; (*) DONE
RXI4:   MVI     A,01H   ;SET RX FLAG TO SHOW COMPLETION
        STA     RXFLAG  ;COMPLETION
RXI3:   POP     B       ;RESTORE BC
        POP     PSW     ;RESTORE PSW
        POP     H       ;RESTORE HL
        EI      ;ENABLE INTERRUPTS
        RET     ;DONE


;FUNCTION: TXI - INTERRUPT DRIVEN RESULT/DATA HANDLER
;INPUTS: TXRBUF, TXPNT, TXFLAG
;OUTPUTS: TXRBUF, TXPNT, TXFLAG
;CALLS: NONE
;DESTROYS: NOTHING
;DESCRIPTION: TXI IS ENTERED AT A TRANSMITTER INTERRUPT.
;THE INTERRUPT IS TESTED BY WAY OF THE IRA BIT TO SEE
;IF A DATA TRANSFER OR RESULT COMPLETION HAS OCCURED.
;FOR DATA TRANSFERS (IRA=0), THE DATA IS OBTAINED FROM
;A BUFFER LOCATION POINTED AT BY TXPNT. FOR COMPLETION,
;(IRA=1), THE RESULTS ARE READ AND PLACED AT A RESULT
;BUFFER POINTED AT BY TXRBUF, AND THE TXFLAG IS SET
;TO INDICATE TO THE MAIN PROGRAM THAT A OPERATION IS
;COMPLETE. TX OPERATIONS HAVE ONLY ONE RESULT.
;DATA TRANSFER INSTRUCTIONS ARE DENOTED BY (*). THESE
;MAYBE REMOVED BY USERS USING DMA.
;
TXI:    PUSH    H       ;SAVE HL
        PUSH    PSW     ;SAVE PSW
        IN      STAT73  ; (*) READ 8273 STATUS
        ANI     TXIRA   ; (*) TEST TXIRA BIT
        JZ      TXI2    ; (*) IF 0, DATA TRANSFER
        IN      TXIR73  ;1, THEN READ TXIR
        LHLD    TXRBUF  ;GET RESULT BUFFER POINTER
        MOV     M,A     ;STORE RESULT IN BUFFER
        INX     H       ;BUMP RESULT POINTER
        SHLD    TXRBUF  ;RESTORE RESULT POINTER
        MVI     A,01H   ;SET TXFLAG TO SHOW COMPLETION
        STA     TXFLAG  ;SET FLAG
TXI1:   POP     PSW     ;RESTORE PSW
        POP     H       ;RESTORE HL
        EI      ;ENABLE INTERRUPTS
        RET     ;DONE
TXI2:   LHLD    TXPNT   ; (*) GET DATA POINTER
        MOV     A,M     ; (*) GET DATA FROM BUFFER
        OUT     TXDATA  ; (*) OUTPUT TO 8273 VIA TXDACK
        INX     H       ; (*) BUMP DATA POINTER
        SHLD    TXPNT   ; (*) RESTORE POINTER
        JMP     TXI1    ; (*) RETURN AFTER RESTORE
```

Figure 22. Interrupt-Driven Result Handlers with Non-DMA Data Transfers

```
;FUNCTION: POLOP
;INPUTS: NONE
;OUTPUTS: C=0 (NO STATUS), =1 (RX COMPLETION),
;   =2 (TX COMPLETION), =3 (BOTH)
;CALLS: TXI, RXI
;DESTROYS: B,C
;DESCRIPTION: POLOP IS CALLED TO POLL THE 8273 FOR
;DATA TRANSFERS AND COMPLETION RESULTS. THE
;ROUTINES TXI AND RXI ARE USED FOR THE ACTUAL
;TRANSFERS AND BUFFER WORK. POLOP RETURNS
;THE STATUS OF THEIR ACTION.
;
POLOP:  PUSH    PSW     ;SAVE PSW
        MVI     C,00H   ;CLEAR C
POLOP1: IN      STAT73  ;READ 8273 STATUS
        ANI     INT     ;ARE TXINT OR RXINT SET?
        JZ      PEXIT   ;NO, EXIT
        IN      STAT73  ;READ 8273 STATUS
        ANI     RXINT   ;TEST RX INT
        JNZ     RXIC    ;YES, GO SERVICE RX
        CALL    TXI     ;MUST BE TX, GO SERVICE IT
        LDA     TXFLAG  ;GET TX FLAG
        CPI     01H     ;WAS IT A COMPLETION? (01)
        JNZ     PEXIT   ;NO, SO JUST EXIT
        INR     C       ;YES, UPDATE C
        INR     C
        JMP     POLOP1  ;TRY AGAIN
;
RXIC:   CALL    RXI     ;GO SERVICE RX
        LDA     RXFLAG  ;GET RX FLAG
        CPI     01H     ;WAS IT A COMPLETION? (01)
        JNZ     PEXIT   ;NO, SO JUST EXIT
        INR     C       ;YES, UPDATE C
        JMP     POLOP1  ;TRY AGAIN
;
PEXIT:  POP     PSW     ;RESTORE PSW
        RET     ;RETURN WITH COMP. STATUS IN C
```

Figure 23. Polling Result Handler

8273 COMMAND DESCRIPTION

In this section, each command is discussed in detail. In order to shorten the notation, please refer to the command key in Table 1. The 8273 utilizes five different command types: Initialization/Configuration, Receive, Transmit, Reset, and Modem Control.

Initialization/Configuration Commands

The Initialization/Configuration commands manipulate registers internal to the 8273 that define the various operating modes. These commands either set or reset specified bits in the registers depending on the type of command. One parameter is required. Set commands perform a logical OR operation of the parameter (mask) and the internal register. This mask contains 1s where register bits are to be set. A 0 in the mask causes no change in the corresponding register bit. Reset commands perform a logical AND operation of the parameter (mask) and the internal register, i.e., the mask is 0 to reset a register bit and a 1 to cause no change. Before presenting the commands, the register bit definitions are discussed.

TABLE 1. COMMAND SUMMARY KEY

B_0, B_1 — LSB AND MSB OF RECEIVE BUFFER LENGTH
R_0, R_1 — LSB AND MSB OF RECEIVED FRAME LENGTH
L_0, L_1 — LSB AND MSB OF TRANSMIT FRAME LENGTH
A_1, A_2 — MATCH ADDRESSES FOR SELECTIVE RECEIVE
RIC — RECEIVER INTERRUPT RESULT CODE
TIC — TRANSMITTER INTERRUPT RESULT CODE
A — ADDRESS FIELD OF RECEIVED FRAME
C — CONTROL FIELD OF RECEIVED FRAME

Operating Mode Register (Figure 24)

D_7-D_6: *Not Used* — These bits must not be manipulated by any command; i.e., D_7-D_6 must be 0 for the Set command and 1 for the Reset command.

D_5: *HDLC Abort* — When this bit is set, the 8273 will interrupt when 7 1s (HDLC Abort) are received by an active receiver. When reset, an SDLC Abort (8 1s) will cause an interrupt.

D_4: *EOP Interrupt* — Reception of an EOP character (0 followed by 7 1s) will cause the 8273 to interrupt the CPU when this bit is set. Loop controller stations use this mode as a signal that a polling frame has completed the loop. No EOP interrupt is generated when this bit is reset.

D_3: *Early Tx Interrupt* — This bit specifies when the transmitter should generate an end of frame interrupt. If this bit is set, an interrupt is generated when the last data character has been passed to the 8273. If the user software issues another transmit command within two byte times, the final flag interrupt does not occur and the new frame is transmitted with only one flag of separation. If this restriction is not met, more than one flag will separate the frames and a frame complete interrupt is generated after the closing flag. If the bit is reset, only the frame complete interrupt occurs. This bit, when set, allows a single flag to separate consecutive frames.

D_2: *Buffered Address and Control* — When set, the address and control fields of received frames are buffered in the 8273 and passed to the CPU as results after a received frame interrupt (they are not transferred to memory with the information field). On transmit, the A and C fields are passed to the 8273 as parameters. This mode simplifies buffer management. When this bit is reset, the A and C fields are passed to and from memory as the first two data transfers.

D_1: *Preframe Sync* — When set, the 8273 prefaces each transmitted frame with two characters before the opening flag. These two characters provide 16 transitions to allow synchronization of the opposing receiver. To guarantee 16 transitions, the two characters are 55H-55H for non-NRZI mode (see Serial I/O Register description) or 00H-00H for NRZI mode. When reset, no preframe characters are transmitted.

D_0: *Flag Stream* — When set, the transmitter will start sending flag characters as soon as it is idle; i.e., immediately if idle when the command is issued or after a transmission if the transmitter is active when this bit is set. When reset, the transmitter starts sending Idle characters on the next character boundary if idle already, or at the end of a transmission if active.

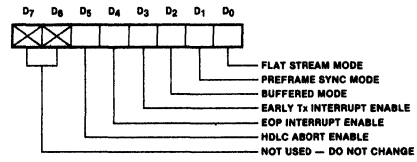


Figure 24. Operating Mode Register

Serial I/O Mode Register (Figure 25)

D_7-D_3: *Not Used* — These bits must be 0 for the Set command and 1 for the Reset command.

D_2: *Data Loopback*— When set, transmitted data (TxD) is internally routed to the receive data circuitry. When reset, TxD and RxD are independent.

D_1: *Clock Loopback* — When set, $\overline{TxC}$ is internally routed to $\overline{RxC}$. When reset, the clocks are independent.

D_0: *NRZI (Non-Return to Zero Inverted)* — When set, the 8273 assumes the received data is NRZI encoded, and NRZI encodes the transmitted data. When reset, the received and transmitted data are treated as a normal positive logic bit stream.

Data Transfer Mode Register (Figure 26)

D_7-D_1: *Not Used* — These bits must be 0 for the Set command and 1 for the Reset command.

D_0: *Interrupt Data Transfer* — When set, the 8273 will interrupt the CPU when data transfers are required (the corresponding IRA Status register bit will be 0 to signify a data transfer interrupt rather than a Result phase interrupt). When reset, 8273 data transfers are performed through DMA requests on the DRQ pins without interrupting the CPU.

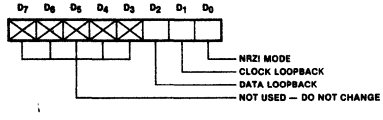


Figure 25. Serial I/O Mode Register

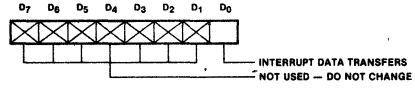


Figure 26. Data Transfer Mode Register

One Bit Delay Register (Figure 27)

D_7: *One Bit Delay* — When set, the 8273 retransmits the received data stream one bit delayed. This mode is entered and exited at a received character boundary. When reset, the transmitted and received data are independent. This mode is utilized for loop operation and is discussed in a later section.

D_6-D_0: *Not Used* — These bits must be 0 for the Set command and 1 for the Reset command.

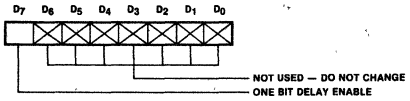


Figure 27. One Bit Delay Mode Register

Figure 28 shows the Set and Reset commands associated with the above registers. The mask which sets or resets the desired bits is treated as a single parameter. These commands do not interrupt nor provide results during the Result phase. After reset, the 8273 defaults to all of these bits reset.

REGISTER	COMMAND	HEX CODE	PARAMETER
ONE BIT DELAY MODE	SET	A4	SET MASK
	RESET	64	RESET MASK
DATA TRANSFER MODE	SET	97	SET MASK
	RESET	57	RESET MASK
OPERATING MODE	SET	91	SET MASK
	RESET	51	RESET MASK
SERIAL I/O MODE	SET	A0	SET MASK
	RESET	60	RESET MASK

Figure 28. Initialization/Configuration Command Summary

Receive Commands

The 8273 supports three receive commands plus a receiver disable function.

General Receive

When commanded to General Receive, the 8273 passes all frames either to memory (DMA mode) or to the CPU (non-DMA mode) regardless of the contents of the frame's address field. This command is used for primary and loop controller stations. Two parameters are required: B_0 and B_1. These parameters are the LSB and MSB of the receiver buffer size. Giving the 8273 this extra information alleviates the CPU of the burden of checking for buffer overflow. The 8273 will interrupt the CPU if the received frame attempts to overfill the allotted buffer space.

Selective Receive

In Selective Receive, two additional parameters besides B_0 and B_1 are required: A_1 and A_2. These parameters are two address match bytes. When commanded to Selective Receive, the 8273 passes to memory or the CPU only those frames having an address field matching either A_1 or A_2. This command is usually used for secondary stations with A_1 being the secondary address and A_2 is the "All Parties" address. If only one match byte is needed, A_1 and A_2 should be equal. As in General Receive, the 8273 counts the incoming data bytes and interrupts the CPU if B_0, B_1 is exceeded.

Selective Loop Receive

This command is very similar in operation to Selective Receive except that One Bit Delay mode must be set and that the loop is captured by placing transmitter in Flag Stream mode automatically after an EOP character is detected following a selectively received frame. The details of using the 8273 in loop configurations is discussed in a later section so please hold questions until then.

The handling of interrupt results is common among the three commands. When a frame is received without error, i.e., the FCS is correct and $\overline{CD}$ (Carrier Detect) was active throughout the frame or no attempt was made to overfill the buffer; the 8273 interrupts the CPU following the closing flag to pass the completion results. These results, in order, are the receiver interrupt result code (RIC), and the byte length of the information field of the received frame (R_0, R_1). If Buffered mode is selected, the address and control fields are passed as two additional results. If Buffered mode is not selected, the address and control fields are passed as the first two data transfers and R_0, R_1 reflect the information field length plus two.

Receive Disable

The receiver may also be disabled using the Receive Disable command. This command terminates any receive operation immediately. No parameters are required and no results are returned.

The details for the Receive command are shown in Figure 29. The interrupt result code key is shown in Figure 30. Some explanation of these result codes is appropriate.

The interrupt result code is the first byte passed to the CPU in the RxI/R register during the Result phase. Bits D_4-D_0 define the cause of the receiver interrupt. Since each result code has specific implications, they are discussed separately below.

COMMAND	HEX CODE	PARAMETERS	RESULTS* RxI/R
GENERAL RECEIVE	C0	B_0, B_1	RIC, R_0, R_1, A, C
SELECTIVE RECEIVE	C1	B_0, B_1, A_1, A_2	RIC, R_0, R_1, A, C
SELECTIVE LOOP RECEIVE	C2	B_0, B_1, A_1, A_2	RIC, R_0, R_1, A, C
DISABLE RECEIVER	C5	NONE	NONE

*A AND C ARE PASSED AS RESULTS ONLY IN BUFFERED MODE.

Figure 29. Receiver Command Summary

RIC D_7-D_0	RECEIVER INTERRUPT RESULT CODE	Rx STATUS AFTER INT
* 00000	A_1 MATCH OR GENERAL RECEIVE	ACTIVE
* 00001	A_2 MATCH	ACTIVE
000 00011	CRC ERROR	ACTIVE
000 00100	ABORT DETECTED	ACTIVE
000 00101	IDLE DETECTED	DISABLED
000 00110	EOP DETECTED	DISABLED
000 00111	FRAME < 32 BITS	ACTIVE
000 01000	DMA OVERRUN	DISABLED
000 01001	MEMORY BUFFER OVERFLOW	DISABLED
000 01010	CARRIER DETECT FAILURE	DISABLED
000 01011	RECEIVER INTERRUPT OVERRUN	DISABLED

*D_7-D_5	PARTIAL BYTE RECEIVED
111	ALL 8 BITS OF LAST BYTE
000	D_0
100	D_1-D_0
010	D_2-D_0
110	D_3-D_0
001	D_4-D_0
101	D_5-D_0
011	D_6-D_0

Figure 30. Receiver Interrupt Result Codes(RIC)

The first two result codes result from the error-free reception of a frame. If the frame is received correctly after a General Receive command, the first result is returned. If either Selective Receive command was used (normal or loop), a match with A_1 generates the first result code and a match with A_2 generates the second. In either case, the receiver remains active after the interrupt; however, the internal buffer size counters are not reset. That is, if the receive command indicated 100 bytes were allocated to the receive buffer (B_0, B_1) and an 80-byte frame was received correctly, the maximum next frame size that could be received without recommanding the receiver (resetting B_0 and B_1) is 20 bytes. Thus, it is common practice to recommand the receiver after each frame reception. DMA and/or memory pointers are usually updated at this time. (Note that users who do not wish to take advantage of the 8273's buffer management features may simply use B_0, B_1 = 0FFH for each receive command. Then frames of 65K bytes may be received without buffer overflow errors.)

The third result code is a CRC error. This indicates that a frame was received in the correct format (flags, etc.); however, the received FCS did not check with the internally generated FCS. The frame should be discarded. The receiver remains active. (Do not forget that even though an error condition has been detected, all frame information up until that error has either been transferred to memory or passed to the CPU. This information should be invalidated. This applies to all receiver error conditions.) Note that the FCS, either transmitted or received, is never available to the CPU.

The Abort Detect result occurs whenever the receiver sees either an SDLC (8 1s) or an HDLC (7 1s), depending on the Operating Mode register. However, the intervening Abort character between a closing flag and an Idle does not generate an interrupt. If an Abort character (seen by an active receiver within a frame) is not preceded by a flag and is followed by an Idle, an interrupt will be generated for the Abort, followed by an Idle interrupt one character time later. The Idle Detect result occurs whenever 15 consecutive 1s are received. After the Abort Detect interrupt, the receiver remains active. After the Idle Detect interrupt, the receiver is disabled and must be recommanded before further frames may be received.

If the EOP Interrupt bit is set in the Operating Mode register, the EOP Detect result is returned whenever an EOP character is received. The receiver is disabled, so the Idle following the EOP does not generate an Idle Detect interrupt.

The minimum number of bits in a valid frame between the flags is 32. Fewer than 32 bits indicates an error. If Buffered mode is selected, such frames are ignored, i.e., no data transfers or interrupts are generated. In non-Buffered mode, a < 32-bit frame generates an interrupt with the < 32-bit Frame result since data transfers may already have disturbed the 8257 or interrupt handler. The receiver remains active.

The DMA Overrun result results from the $\overline{\text{DMA}}$ controller being too slow in extracting data from the 8273, i.e., the $\overline{\text{RxDACK}}$ signal is not returned before the next received byte is ready for transfer. The receiver is disabled if this error condition occurs.

The Memory Buffer Overflow result occurs when the number of received bytes exceeds the receiver buffer length supplied by the B_0 and B_1 parameters in the receive command. The receiver is disabled.

The Carrier Detect Failure result occurs when the $\overline{\text{CD}}$ pin goes high (inactive) during reception of a frame. The $\overline{\text{CD}}$ pin is used to qualify reception and must be active by the time the address field starts to be received. If $\overline{\text{CD}}$ is lost during the frame, a $\overline{\text{CD}}$ Failure interrupt is generated and the receiver is disabled. No interrupt is generated if $\overline{\text{CD}}$ goes inactive between frames.

If a condition occurs requiring an interrupt be generated before the CPU has finished reading the previous interrupt results, the second interrupt is generated after the current Result phase is complete (the RxINT pin and status bit go low then high). However, the interrupt result for this second interrupt will be a Receive Interrupt Overrun. The actual cause of the second interrupt is lost. One case where this may occur is at the end of a received frame where the line goes idle. The 8273 generates a received frame interrupt after the closing flag and then 15-bit times later, generates an Idle Detect interrupt. If the interrupt service routine is slow in reading the first interrupt's results, the internal RxI/R register still contains result information when the Idle Detect interrupt occurs. Rather than wiping out the previous results, the 8273 adds a Receive Interrupt Overrun result as an extra result. If the system's interrupt structure is such that the second interrupt is not acknowledged (interrupts are still disabled from the first interrupt), the Receive Interrupt Overrun result is read as an extra result, after those from the first interrupt. If the second interrupt is serviced, the Receive Interrupt Overrun is returned as a single result. (Note that the INT pins supply the necessary transitions to support a Program-

mable Interrupt Controller such as the Intel 8259. Each interrupt generates a positive-going edge on the appropriate INT pin and the high level is held until the interrupt is completely serviced.) In general, it is possible to have interrupts occurring at one character time intervals. Thus the interrupt handling software must have at least that much response and service time.

The occurrence of Receive Interrupt Overruns is an indication of marginal software design; the system's interrupt response and servicing time is not sufficient for the data rates being attempted. It is advisable to configure the interrupt handling software to simply read the interrupt results, place them into a buffer, and clear the interrupt as quickly as possible. The software can then examine the buffer for new results at its leisure, and take appropriate action. This can easily be accomplished by using a result buffer flag that indicates when new results are available. The interrupt handler sets the flag and the main program resets it once the results are retrieved.

Both SDLC and HDLC allow frames which are of arbitrary length (>32 bits). The 8273 handles this N-bit reception through the high order bits (D_7–D_5) of the result code. These bits code the number of valid received bits in the last received information field byte. This coding is shown in Figure 30. The high order bits of the received partial btye are indeterminate. [The address, control, and information fields are transmitted least significant bit (A_0) first. The FCS is complemented and transmitted most significant bit first.]

Transmit Commands

The 8273 transmitter is supported by three Transmit commands and three corresponding Abort commands.

Transmit Frame

The Transmit Frame command simply transmits a frame. Four parameters are required when Buffered mode is selected and two when it is not. In either case, the first two parameters are the least and the most significant bytes of the desired frame length (L_0, L_1). In Buffered mode, L_0 and L_1 equal the length in bytes of the desired information field, while in the non-Buffered mode, L_0 and L_1 must be specified as the information field length plus two. (L_0 and L_1 specify the number of data transfers to be performed.) In Buffered mode, the address and control fields are presented to the transmitter as the third and fourth parameters respectively. In non-Buffered mode, the A and C fields must be passed as the first two data transfers.

When the Transmit Frame command is issued, the 8273 makes $\overline{RTS}$ (Request-to-Send) active (pin low) if it was not already. It then waits until $\overline{CTS}$ (Clear-to-Send) goes active (pin low) before starting the frame. If the Preframe Sync bit in the Operting Mode register is set, the transmitter prefaces two characters (16 transitions) before the opening flag. If the Flag Stream bit is set in the Operating Mode register, the frame (including Preframe Sync if selected) is started on a flag boundary. Otherwise the frame starts on a character boundary.

At the end of the frame, the transmitter interrupts the CPU (the interrupt results are discussed shortly) and returns to either Idle or Flag Stream, depending on the Flag Stream bit of the Operating Mode register. If $\overline{RTS}$ was active before the transmit command, the 8273 does not change it. If it was inactive, the 8273 will deactivate it within one character time.

Loop Transmit

Loop Transmit is similar to Frame Transmit (the parameter definition is the same). But since it deals with loop configurations, One Bit Delay mode must be selected.

If the transmitter is not in Flag Stream mode when this command is issued, the transmitter waits until after a received EOP character has been converted to a flag (this is done automatically) before transmitting. (The one bit delay is, of course, suspended during transmit.) If the transmitter is already in Flag Stream mode as a result of a selectively received frame during a Selective Loop Receive command, transmission will begin at the next flag boundary for Buffered mode or at the third flag boundary for non-Buffered mode. This discrepancy is to allow time for enough data transfers to occur to fill up the internal transmit buffer. At the end of a Loop Transmit, the One Bit Delay mode is re-entered and the flag stream mode is reset. More detailed loop operation is covered later.

Transmit Transparent

The Transmit Transparent command enables the 8273 to transmit a block of raw data. This data is without SDLC protocol, i.e., no zero bit insertion, flags, or FCS. Thus it is possible to construct and transmit a Bi-Sync message for front-end processor switching or to construct and transmit an SDLC message with incorrect FCS for diagnostic purposes. Only the L_0 and L_1 parameters are used since there are not fields in this mode. (the 8273 does not support a Receive Transparent command.)

Abort Commands

Each of the above transmit commands has an associated Abort command. The Abort Frame Transmit command causes the transmitter to send eight contiguous ones (no zero bit insertion) immediately and then revert to either idle or flag streaming based on the Flag Stream bit. (The 8 1s as an Abort character is compatible with both SDLC and HDLC.)

For Loop Transmit, the Abort Loop Transmit command causes the transmitter to send one flag and then revert to one bit delay. Loop protocol depends upon FCS errors to detect aborted frames.

The Abort Transmit Transparent simply causes the transmitter to revert to either idles or flags as a function of the Flag Stream mode specified.

The Abort commands require no parameters, however, they do generate an interrupt and return a result when complete.

A summary of the Transmit commands is shown in Figure 31. Figure 32 shows the various transmit interrupt result codes. As in the receiver operation, the transmitter generates interrupts based on either good

completion of an operation or an error condition to start the Result phase.

The Early Transmit Interrupt result occurs after the last data transfer to the 8273 if the Early Transmit Interrupt bit is set in the Operating Mode register. If the 8273 is commanded to transmit again within two character times, a single flag will separate the frames. (Buffered mode must be used for a single flag to separate the frames. If non-Buffered mode is selected, three flags will separate the frames.) If this time constraint is not met, another interrupt is generated and multiple flags or idles will separate the frames. The second interrupt is the normal Frame Transmit Complete interrupt. The Frame Transmit Complete result occurs at the closing flag to signify a good completion.

The DMA Underrun result is analogous to the DMA Overrun result in the receiver. Since SDLC does not support intraframe time fill, if the DMA controller or CPU does not supply the data in time, the frame must be aborted. The action taken by the transmitter on this error is automatic. It aborts the frame just as if an Abort command had been issued.

Clear-to-Send Error result is generated if $\overline{CTS}$ goes inactive during a frame transmission. The frame is aborted as above.

The Abort Complete result is self-explanatory. Please note however that no Abort Complete interrupt is generated when an automatic abort occurs. The next command type consists of only one command.

COMMAND	HEX CODE	PARAMETERS*	RESULTS TxI/R
TRANSMIT FRAME	C8	L_0, L_1, A, C	TIC
ABORT	CC	NONE	TIC
LOOP TRANSMIT	CA	L_0, L_1, A, C	TIC
ABORT	CE	NONE	TIC
TRANSMIT TRANSPARENT	C0	L_0, L_1	TIC
ABORT	CD	NONE	TIC

*A AND C ARE PASSED AS PARAMETERS IN BUFFERED MODE ONLY.

Figure 31. Transmitter Command Summary

TIC D_7-D_0	TRANSMITTER INTERRUPT RESULT CODE	Tx STATUS AFTER INT
000 01100	EARLY Tx INTERRUPT	ACTIVE
000 01101	FRAME Tx COMPLETE	IDLE OR FLAGS
000 01110	DMA UNDERRUN	ABORT
000 01111	CLEAR TO SEND ERROR	ABORT
000 10000	ABORT COMPLETE	IDLE OR FLAGS

Figure 32. Transmitter Interrupt Result Codes

Reset Command

The Reset command provides a software reset function for the 8273. It is a special case and does not utilize the normal command interface. The reset facility is provided in the Test Mode register. The 8273 is reset by simply outputting a 01H followed by a 00H to the Test Mode register. Writing the 01 followed by the 00 mimicks the action required by the hardware reset. Since the 8273 requires time to process the reset internally, at least 10 cycles of the øCLK clock must occur between the writing of the 01 and the 00. The action taken is the same as if a hardware reset is performed, namely:

1. The modem control outputs are forced high inactive).
2. The 8273 Status register is cleared.
3. Any commands in progress cease.
4. The 8273 enters an idle state until the next command is issued.

Modem Control Commands

The modem control ports were discussed earlier in the Hardware section. The commands used to manipulate these ports are shown in Figure 33. The Read Port A and Read Port B commands are immediate. The bit definition for the returned byte is shown in Figures 13 and 14. Do not forget that the returned value represents the logical condition of the pin, i.e., pin active (low) = bit set.

PORT	COMMAND	HEX CODE	PARAMETER	REG RESULT
A INPUT	READ	22	NONE	PORT VALUE
B OUTPUT	READ	23	NONE	PORT VALUE
	SET	A3	SET MASK	NONE
	RESET	63	RESET MASK	NONE

Figure 33. Modem Control Command Summary

The Set and Reset Port B commands are similar to the Initialization commands in that they use a mask parameter which defines the bits to be changed. Set Port B utilizes a logical OR mask and Reset Port B uses a logical AND mask. Setting a bit makes the pin active (low). Resetting the bit deactivates the pin (high).

To help clarify the numerous timing relationships that occur and their consequences, Figures 34 and 35 are provided as an illustration of several typical sequences. It is suggested that the reader go over these diagrams and re-read the appropriate part of the previous sections if necessary.

HLDC CONSIDERATIONS

The 8273 supports HDLC as well as SDLC. Let's discuss how the 8273 handles the three basic HDLC/SDLC differences: extended addressing, extended control, and the 7 1s Abort character.

Recalling Figure 4A, HDLC supports an address field of indefinite length. The actual amount of extension used is determined by the least significant bit of the characters immediately following the opening flag. If the LSB is 0, more address field bytes follow. If the LSB is 1, this byte is the final address field byte. Software must be used to determine this extension.

If non-Buffered mode is used, the A, C, and I fields are in memory. The software must examine the initial characters to find the extent of the address field. If Buffered mode is used, the characters corresponding to the SDLC A and C fields are transferred to the CPU as interrupt results. Buffered mode assumes the two characters following the opening flag are to be transferred as interrupt results regardless of content or meaning. (The 8273

does not know whether it is being used in an SDLC or an HDLC environment.) In SDLC, these characters are necessarily the A and C field bytes, however in HDLC, their meaning may change depending on the amount of extension used. The software must recognize this and examine the transferred results as possible address field extensions.

Frames may still be selectively received as is needed for secondary stations. The Selective Receive command is still used. This command qualifies a frame reception on the first byte following the opening flag matching either of the A_1 or A_2 match byte parameters. While this does not allow qualification over the complete range of HDLC addresses, it does perform a qualification on the first address byte. The remaining address field bytes, if any, are then examined via software to completely qualify the frame.

Once the extent of the address field is found, the following bytes form the control field. The same LSB test used for the address field is applied to these bytes to determine the control field extension, up to two bytes maximum. The remaining frame bytes in memory represent the information field.

The Abort character difference is handled in the Operating Mode register. If the HDLC Abort Enable bit is set, the reception of seven contiguous ones by an active receiver will generate an Abort Detect interrupt rather than eight ones. (Note that both the HDLC Abort Enable bit and the EOP Interrupt bit must not be set simultaneously.)

Now let's move on to the SDLC loop configuration discussion.

LOOP CONFIGURATION

Aside from use in the normal data link applications, the 8273 is extremely attractive in loop configuration due to the special frame-level loop commands and the Digital Phase Locked Loop. Toward this end, this section details the hardware and software considerations when using the 8273 in a loop application.

The loop configuration offers a simple, low-cost solution for systems with multiple stations within a small physical location, i.e., retail stores and banks. There are two primary reasons to consider a loop configuration. The interconnect cost is lower for a loop over a multi-point configuration since only one twisted pair or fiber optic cable is used. (The loop configuration does not support the passing of distinct clock signals from station to station.) In addition, loop stations do not need the intelligence of a multi-point station since the loop protocal is simpler. The most difficult aspects of loop station design are clock recovery and implementation of one bit delay (both are handled neatly by the 8273).

Figure 36 illustrates a typical loop configuration with one controller and two down-loop secondaries. Each station must derive its own data timing from the received data stream. Recalling our earlier discussion of the DPLL, notice that $\overline{TxC}$ and $\overline{RxC}$ clocks are provided by the $\overline{DPLL}$ output. The only clock required in the secondaries is a simple, non-synchronized clock at 32 times the desired baud rate. The controller requires both 32× and 1× clocks. (The 1× is usually implemented by dividing the 32× clock with a 5-bit divider. However, there is no synchronism requirement between these clocks so any convenient implementation may be used.)

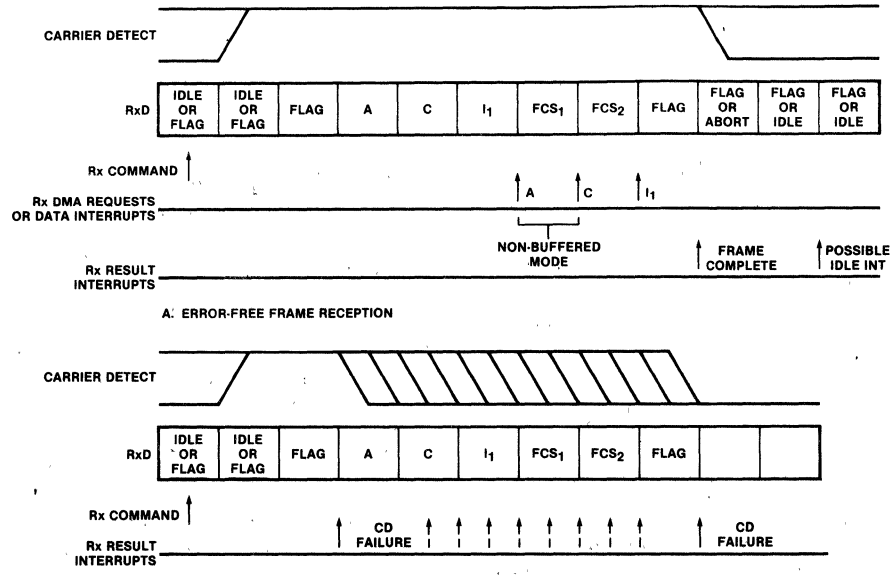


Figure 34. Sample Receiver Timing Diagrams

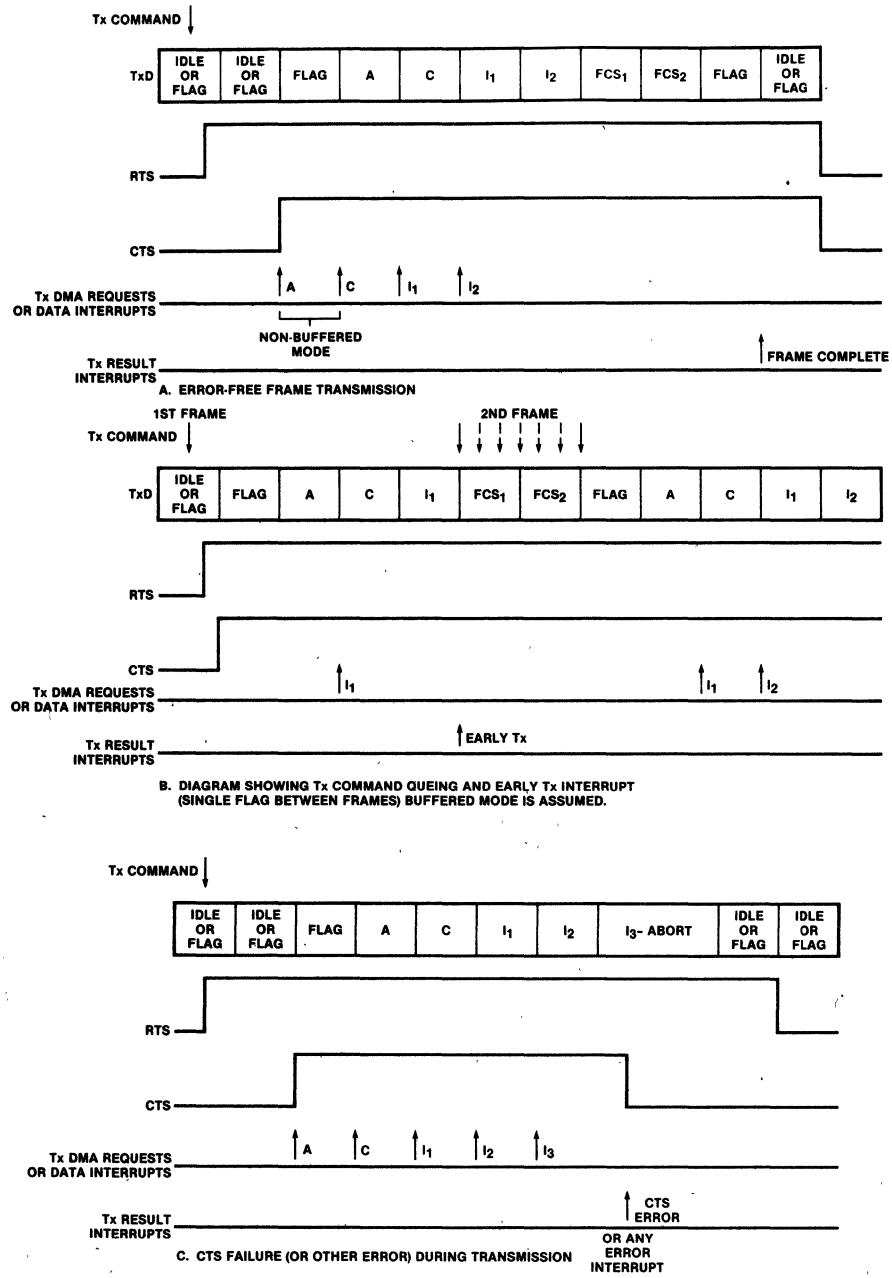


Figure 35. Sample Transmitter Timing Diagrams

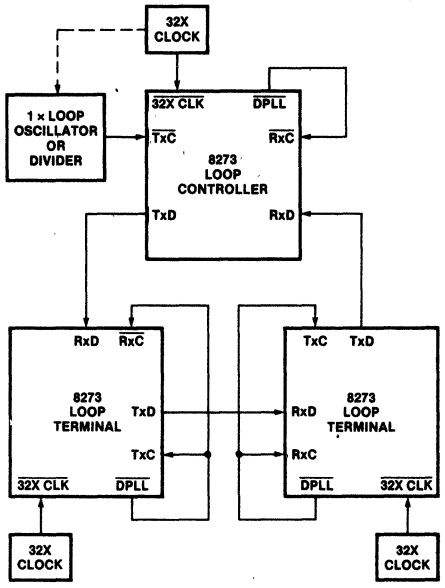


Figure 36. SDLC Loop Application

A quick review of loop protocol is appropriate. All communication on the loop is controlled by the loop controller. When the controller wishes to allow the secondaries to transmit, it sends a polling frame (the control field contains a poll code) followed by an EOP (End-of-Poll) character. The secondaries use the EOP character to capture the loop and insert a response frame as will be discussed shortly.

The secondaries normally operate in the repeater mode, retransmitting received data with one bit time of delay. All received frames are repeated. The secondary uses the one bit time of delay to capture the loop.

When the loop is idle (no frames), the controller transmits continuous flag characters. This keeps transitions on the loop for the sake of down-loop phase locked loops. When the controller has a non-polling frame to transmit, it simply transmits the frame and continues to send flags. The non-polling frame is then repeated around the loop and the controller receives it to signify a complete traversal of the loop. At the particular secondary addressed by the frame, the data is transferred to memory while being repeated. Other secondaries simply repeat it.

If the controller wants to poll the secondaries, it transmits a polling frame followed by all 1s (no zero bit insertion). The final zero of the closing frame plus the first seven 1s form an EOP. While repeating, the secondaries monitor their incoming line for an EOP. When an EOP is received, the secondary checks if it has any response for the controller. If not, it simply continues repeating. If the secondary has a response, it changes the seventh EOP one into a zero (the one bit time of delay allows time for this) and repeats it, forming a flag for the down-loop stations. After this flag is transmitted, the secondary terminates its repeater function and inserts its response frame (with multiple preceding flags if necessary). After the closing flag of the response, the secondary re-enters its repeater function, repeating the up-loop controller 1s. Notice that the final zero of the response's closing flag plus the repeated 1s from the controller form a new EOP for the next down-loop secondary. This new EOP allows the next secondary to insert a response if it desires. This gives each secondary a chance to respond.

Back at the controller, after the polling frame has been transmitted and the continuous 1s started, the controller waits until it receives an EOP. Receiving an EOP signifies to the controller that the original frame has propagated around the loop followed by any responses inserted by the secondaries. At this point, the controller may either send flags to idle the loop or transmit the next frame. Let's assume that the loop is implemented completely with the 8273s and describe the command flows for a typical controller and secondary.

The loop controller is initialized with commands which specify that the NRZI, Preframe Sync, Flag Stream, and EOP Interrupt modes are set. Thus, the controller encodes and decodes all data using NRZI format. Preframe Sync mode specifies that all transmitted frames be prefaced with 16 line transitions. This ensures that the minimum of 12 transitions needed by the DPLLs to lock after an all 1s line have occurred by the time the secondary sees a frame's opening flag. Setting the Flag Stream mode starts the transmitter sending flags which idles the loop. And the EOP Interrupt mode specifies that the controller processor will be interrupted whenever the active receiver sees an EOP, indicating the completion of a poll cycle.

When the controller wishes to transmit a non-polling frame, it simply executes a Frame Transmit command. Since the Flag Stream mode is set, no EOP is formed after the closing flag. When a polling frame is to be transmitted, a General Receive command is executed first. This enables the receiver and allows reception of all incoming frames; namely, the original polling frame plus any response frames inserted by the secondaries. After the General Receive command, the frame is transmitted with a Frame Transmit command. When the frame is complete, a transmitter interrupt is generated. The loop controller processor uses this interrupt to reset Flag Stream mode. This causes the transmitter to start sending all 1s. An EOP is formed by the last flag and the first 7 1s. This completes the loop controller transmit sequence.

At any time following the start of the polling frame transmission the loop controller receiver will start receiving frames. (The exact time difference depends, of course, on the number of down-loop secondaries due to each inserting one bit time of delay.) The first received frame is simply the original polling frame. However, any additional frames are those inserted by the secondaries. The loop controller processor knows all frames have been received when it sees an EOP Interrupt. This interrupt is generated by the 8273 since the EOP Interrupt mode was set during initialization. At this point, the transmitter may be commanded either to enter Flag

Stream mode, idling the loop, or to transmit the next frame. A flowchart of the above sequence is shown in Figure 37.

The secondaries are initialized with the NRZI and One Bit Delay modes set. This puts the 8273 into the repeater mode with the transmitter repeating the received data with one bit time of delay. Since a loop station cannot transmit until it sees and EOP character, any transmit command is queued until an EOP is received. Thus whenever the secondary wishes to transmit a response, a Loop Transmit command is issued. The 8273 then waits until it receives an EOP. At this point, the receiver changes the EOP into a flag, repeats it, resets One Bit Delay mode stopping the repeater function, and sets the transmitter into Flag Stream mode. This captures the loop. The transmitter now inserts its message. At the closing flag, Flag Stream mode is reset, and One Bit Delay mode is set, returning the 8273 to repeater function and forming an EOP for the next down-loop station. These actions happen automatically after a Loop Transmit command is issued.

When the secondary wants its receiver enabled, a Selective Loop Receive command is issued. The receiver then looks for a frame having a match in the Address field. Once such a frame is received, repeated, and transferred to memory, the secondary's processor is interrupted with the appropriate Match interrupt result and the 8273 continues with the repeater function until an EOP is received, at which point the loop is captured as above. The processor should use the interrupt to determine if it has a message for the controller. If it does, it simply issues a Loop Transmit command and things progress as above. If the processor has no message, the software must reset the Flag Stream mode bit in the Operating Mode register. This will inhibit the 8273 from capturing the loop at the EOP. (The match frame and the EOP may be separated in time by several frames depending on how many up-loop stations inserted messages of their own.) If the timing is such that the receiver has already captured the loop when the Flag Stream mode bit is reset, the mode is exited on a flag boundary and the frame just appears to have extra closing flags before the EOP. Notice that the 8273 handles the queuing of the transmit commands and the setting and resetting of the mode bits automatically. Figure 38 illustrates the major points of the secondary command sequence.

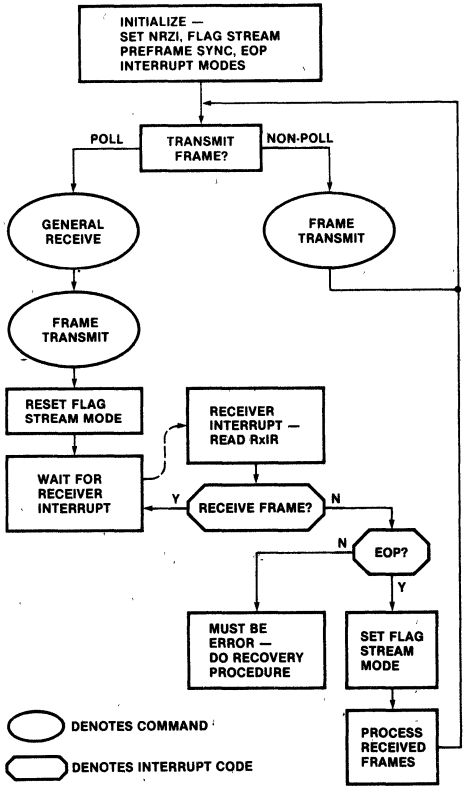


Figure 37. Loop Controller Flowchart

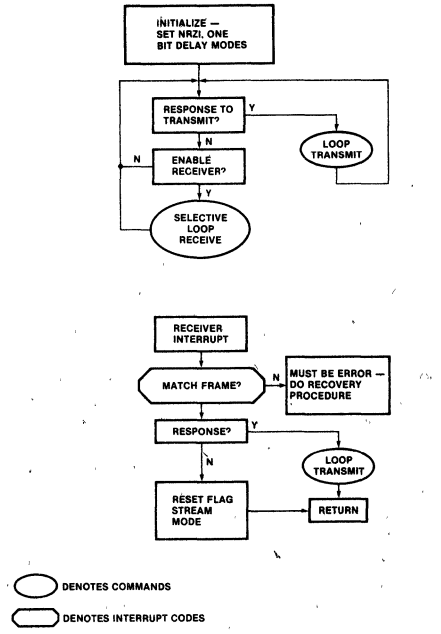


Figure 38. Loop Secondary Flowchart

When an off-line secondary wishes to come on-line, it must do so in a manner which does not disturb data on the loop. Figure 39 shows a typical hardware interface. The line labeled Port could be one of the 8273 Port B outputs and is assumed to be high (1) initially. Thus up-loop data is simply passed down-loop with no delay; however, the receiver may still monitor data on the loop. To come on-line, the secondary is initialized with only the EOP interrupt mode set. The up-loop data is then monitored until an EOP occurs. At this point, the secondary's CPU is interrupted with an EOP interrupt. This signals the CPU to set One Bit Delay mode in the 8273 and then to set Port low (active). These actions switch the secondary's one bit delay into the loop. Since after the EOP only 1s are traversing the loop, no loop disturbance occurs. The secondary now waits for the next EOP, captures the loop, and inserts a "new on-line" message. This signals the controller that a new secondary exists and must be acknowledged. After the secondary receives its acknowledgement, the normal command flow is used.

It is hopefully evident from the above discussion that the 8273 offers a very simple and easy to implement solution for designing loop stations whether they are controllers or down-loop secondaries.

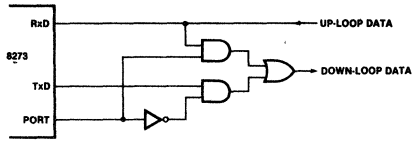


Figure 39. Loop Interface

APPLICATION EXAMPLE

This section describes the hardware and software of the 8273/8085 system used to verify the 8273 implementation of SDLC on an actual IBM SDLC Link. This IBM link was gratefully volunteered by Raytheon Data Systems in Norwood, Mass. and I wish to thank them for their generous cooperation. The IBM system consisted of a 370 Mainframe, a 3705 Communications Processor, and a 3271 Terminal Controller. A Comlink II Modem supplied the modem interface and all communications took place at 4800 baud. In addition to observing correct responses, a Spectron D601B Datascope was used to verify the data exchanges. A block diagram of the system is shown in Figure 40. The actual verification was accomplished by the 8273 system receiving and responding to polls from the 3705. This method was used on both point-to-point and multi-point configurations. No attempt was made to implement any higher protocol software over that of the poll and poll responses since such software would not affect the verification of the 8273 implementation. As testimony to the ease of use of the 8273, the system worked on the first try.

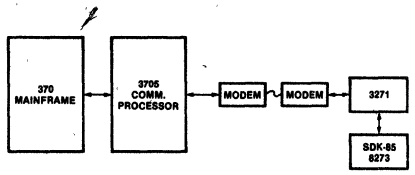


Figure 40. Raytheon Block Diagram

An SDK-85 (System Design Kit) was used as the core 8085 system. This system provides up to 4K bytes of ROM/EPROM, 512 bytes of RAM, 76 I/O pins, plus two timers as provided in two 8755 Combination EPROM/I/O devices and two 8155 Combination RAM/I/O/Timer devices. In addition, 5 interrupt inputs are supplied on the 8085. The address, data, and control buses are buffered by the 8212 and 8216 latches and bidirectional bus drivers. Although it was not used in this application, an 8279 Display Driver/Keyboard Encoder is included to interface the on-board display and keyboard. A block diagram of the SDK-85 is shown in Figure 41. The 8273 and associated circuitry was constructed on the ample wire-wrap area provided for the user.

The example 8273/8085 system is interrupt-driven and uses DMA for all data transfers supervised by an 8257 DMA Controller. A 2400 baud asynchronous line, implemented with an 8251A USART, provides communication between the software and the user. 8253 Programmable Interval Timer is used to supply the baud rate clocks for the 8251A and 8273. (The 8273 baud rate clocks were used only during initial system debug. In actual operation, the modem supplied these clocks via the RS-232 interface.) Two 2142 1K×4 RAMs provided 512 bytes of transmitter and 512 bytes of receiver buffer memory. (Command and result buffers, plus miscellaneous variables are stored in the 8155s.) The RS-232 interface utilized MC1488 and MC1489 RS-232 drivers and receivers. The schematic of the system is shown in Figure 42.

One detail to note is the DMA and interrupt structure of the transmit and receive channels. In both cases, the receiver is always given the higher priority (8257 DMA channel 0 has priority over the remaining channels and the 8085 RST 7.5 interrupt input has priority over the RST 6.5 input.) Although the choice is arbitrary, this technique minimizes the chance that received data could be lost due to other processor or DMA commitments.

Also note that only one 8205 Decoder is used for both the peripherals' and the memorys' Chip Selects. This was done to eliminate separate memory and I/O decoders since it was known beforehand that neither address space would be completely filled.

The 4 MHz crystal and 8224 Clock Generator were used only to verify that the 8273 operates correctly at that maximum spec. speed. In a normal system, the 3.072 MHz clock from the 8085 would be sufficient. (This fact was verified during initial checkout.)

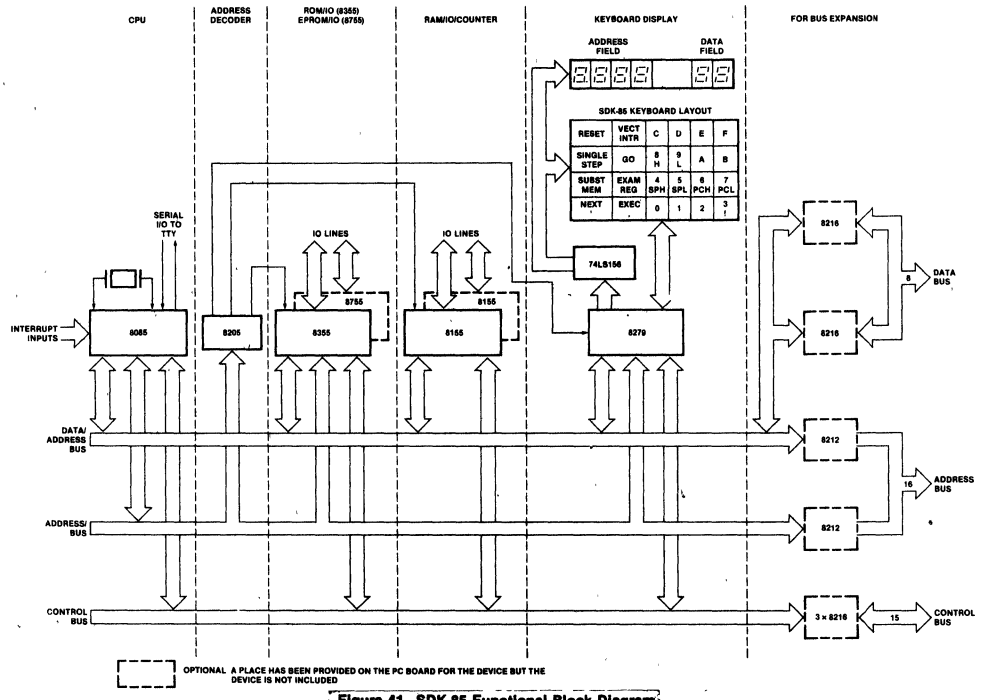


Figure 41. SDK-85 Functional Block Diagram

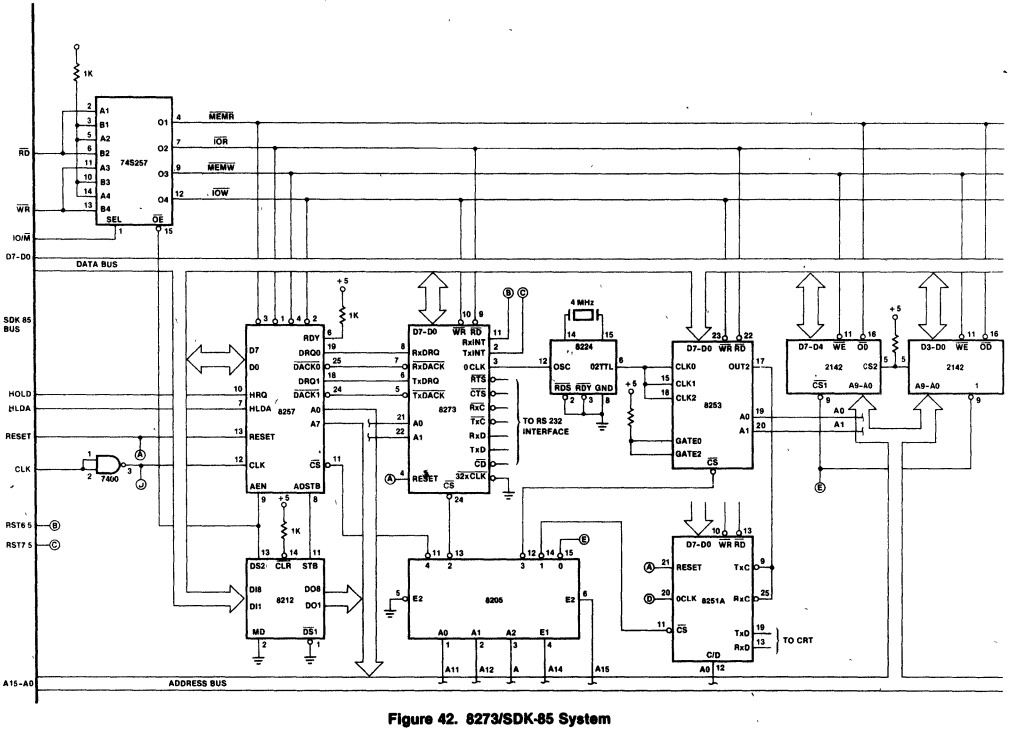


Figure 42. 8273/SDK-85 System

The software consists of the normal monitor program supplied with the SDK-85 and a program to input commands to the 8273 and to display results. The SDK-85 monitor allows the user to read and write on-board RAM, start execution at any memory location, to single-step through a program, and to examine any of the 8085's internal registers. The monitor drives either the on-board keyboard/LED display or a serial TTY interface. This monitor was modified slightly in order to use the 8251A with a 2400 baud CRT as opposed to the 110 baud normally used. The 8273 program implements monitor-like user interface. 8273 commands are entered by a two-character code followed by any parameters required by that command. When 8273 interrupts occur, the source of the interrupt is displayed along with any results associated with it. To gain a flavor of how the user/program interface operates, a sample output is shown in Figure 43. The 8273 program prompt character is a "–" and user inputs are underlined.

The "SO 05" implements the Set Operating Mode command with a parameter of 05H. This sets the Buffer and Flag Stream modes. "SS 01" sets the 8273 in NRZI mode using the Set Serial I/O Mode command. The next command specifies General Receiver with a receiver buffer size of 0100H bytes ($B_0 = 00$, $B_1 = 01$). The "TF" command causes the 8273 to transmit a frame containing an address field of C2H and control field of 11H. The information field is 001122. The "TF" command has a special format. The L_0 and L_1 parameters are computed from the number of information field bytes entered.

After the TF command is entered, the 8273 transmits the frame (assuming that the modem protocol is observed). After the closing flag, the 8273 interrupts the 8085. The 8085 reads the interrupt results and places them in a buffer. The software examines this buffer for new results and if new results exist, the source of the interrupt is displayed along with the results.

In this example, the 0DH result indicates a Frame Complete interrupt. There is only one result for a transmitter interrupt, the interrupt's trailing zero results were included to simplify programming.

The next event is a frame reception. The interrupt results are displayed in the order read from the 8273. The E0H indicates a General Receive interrupt with the last byte of the information field received on an 8-bit boundary. The 03 00 (R_0, R_1) results show that there are 3H bytes of information field received. The remaining two results indicate that the received frame had a C2H address field and a 34H control field. The 3 bytes of information field are displayed on the next line.

```
8273 MONITOR V1.2

– SO 05
– SS 01
– GR 00 01
– TF C2 11 00 11 22
–
TxINT – 0D 00 00 00 00
–
RxINT – E0 03 00 C2 34
FF EE DD
–
```

Figure 43. Sample 8273 Monitor I/O

Figures 44 through 51 show the flowcharts used for the 8273 program development. The actual program listing is included as Appendix A. Figure 44 is the main status poll loop. After all devices are initialized and a prompt character displayed, a loop is entered at LOOPIT. This loop checks for a change of status in the result buffer or if a keyboard character has been received by the 8251 or if a poll frame has been received. If any of these conditions are met, the program branches to the appropriate routine. Otherwise, the loop is traversed again.

The result buffer is implemented as a 255-byte circular buffer with two pointers: CNADR and LDADR. CNADR is the console pointer. It points to the next result to be displayed LDADR is the load pointer. It points to the next empty position in the buffer into which the interrupt handler places the next result. The same buffer is used for both transmitter and receiver results. LOOPIT examines these pointers to detect when CNADR is not equal to LDADR indicating that the buffer contains results which have not been displayed. When this occurs, the program branches to the DISPLY routine.

DISPLY determines the source of the undisplayed results by testing the first result. This first result is necessarily the interrupt result code. If this result is 0CH or greater, the result is from a transmitter interrupt. Otherwise it is from a receiver source. The source of the result code is then displayed on the console along with the next four results from the buffer. If the source was a transmitter interrupt, the routine merely repoints the pointer CNADR and returns to LOOPIT. For a receiver source, the receiver data buffer is displayed in addition to the receiver interrupt results before returning to LOOPIT.

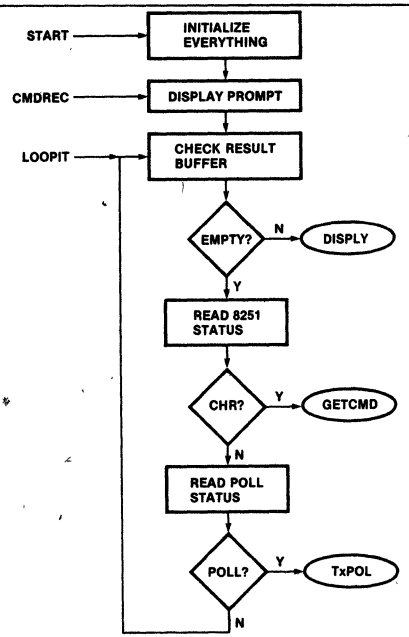


Figure 44. Main Status Poll Loop

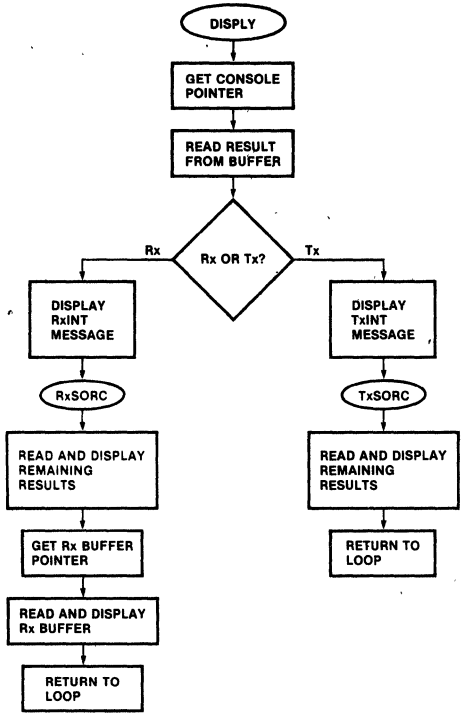


Figure 45. DISPLY Subroutine

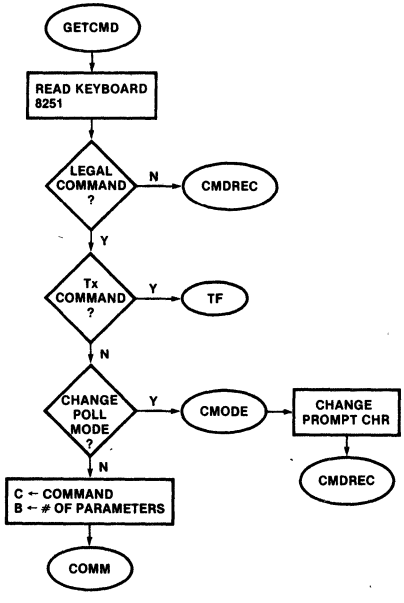


Figure 46. GETCMD Subroutine

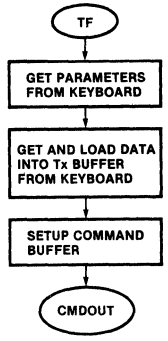


Figure 47. TF Subroutine

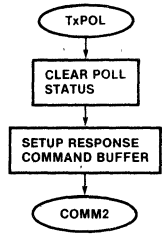


Figure 48. TxPOL Subroutine

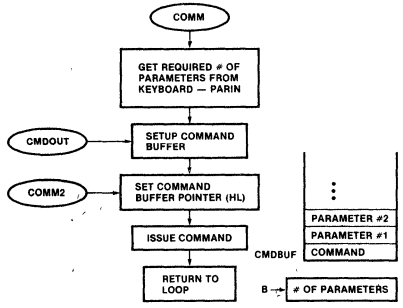


Figure 49. COMM Subroutine with Command Buffer Format

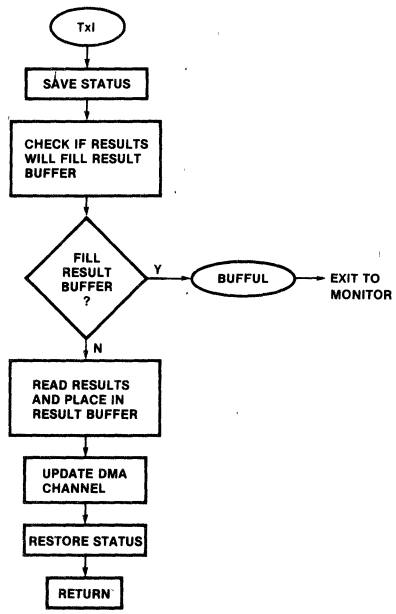


Figure 50. TxI (Transmitter Interrupt) Routine

If the result buffer pointers indicate an empty buffer, the 8251A is polled for a keyboard character. If the 8251 has a character, GETCMD is called. There the character is read and checked if legal. Illegal characters simply cause a reprompt. Legal characters indicate the start of a command input. Most commands are organized as two characters signifying the command action; i.e., GR — General Receive. The software recognizes the two character command code and takes the appropriate action. For non-Transmit type commands, the hex equivalent of the command is placed in the C register and the number of parameters associated with that command is placed in the B register. The program then branches to the COMM routine.

The COMM routine builds the command buffer by reading the required number of parameters from the keyboard and placing them at the buffer pointed at by CMDBUF. The routine at COMM2 then issues this command buffer to the 8273.

If a Transmit type command is specified, the command buffer is set up similarly to the the COMM routine; however, since the information field data is entered from the keyboard, an intermediate routine, TF, is called. TF loads the transmit data buffer pointed at by TxBUF. It counts the number of data bytes entered and loads this number into the command buffer as L_0, L_1. The command is then issued to the 8273 by jumping to CMDOUT.

One command does not directly result in a command being issued to the 8273. This command, Z, operates a software flip-flop which selects whether the software will respond automatically to received polling frames. If the Poll-Response mode is selected, the prompt character is changed to a '+'. If a frame is received which contains a prearranged poll control field, the memory location POLIN is made nonzero by the receiver interrupt handler. LOOPIT examines this location and if it is nonzero, causes a branch to the TxPOL routine. The TxPOL routine clears POLIN, sets a pointer to a special command buffer at CMDBUF1, and issues the command by way of the COMM2 entry in the COMM routine. The special command buffer contains the appropriate response frame for the poll frame received. These actions only occur when the Z command has changed the prompt to a '+'. If the prompt is normal '−', polling frames are displayed as normal frames and no response is transmitted. The Poll-Response mode was used during the IBM tests.

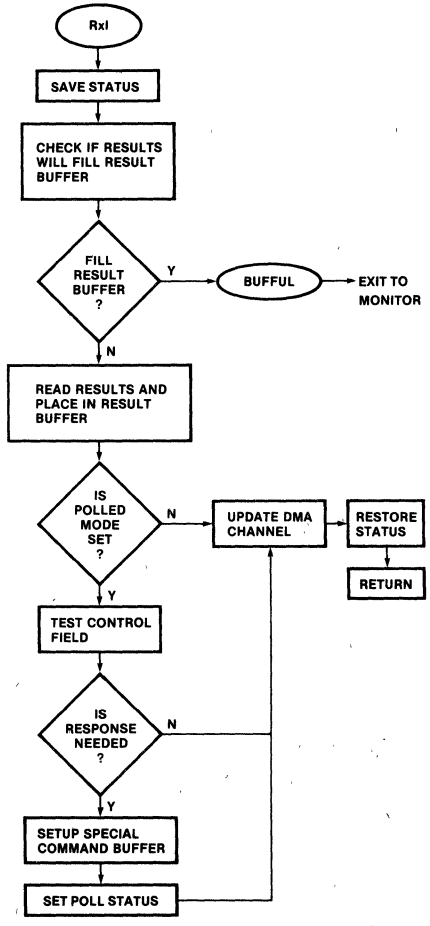


Figure 51. RxI (Recevier Interrupt) Routine

The final two software routines are the transmitter and receiver interrupt handlers. The transmit interrupt handler, TxI, simply saves the registers on the stack and checks if loading the result buffer will fill it. If the result buffer will overfill, the program is exited and control is passed to the SDK-85 monitor. If not, the results are read from the TxI/R register and placed in the result buffer at LDADR. The DMA pointers are then reset, the registers restored, and interrupts enabled. Execution then returns to the pre-interrupt location.

The receiver interrupt handler, RxI, is only slightly more complex. As in TxI, the registers are saved and the possibility of overfilling the result buffer is examined. If the result buffer is not full, the results are read from RxI/R and placed in the buffer. At this point the prompt character is examined to see if the Poll-Response mode is selected. If so, the control field is compared with two possible polling control fields. If there is a match, the special command buffer is loaded and the poll indicator, POLIN, is made nonzero. If no match occurred, no action is taken. Finally, the receiver DMA buffer pointers are reset, the processor status restored, and interrupts are enabled. The RET instruction returns execution to the pre-interrupt location.

This completes the discussion of the 8273/8085 system design.

CONCLUSION

This application note has covered the 8273 in some detail. The simple and low cost loop configuration was explored. And an 8273/8085 system was presented as a sample design illustrating the DMA/interrupt-driven interface. It is hoped that the major features of the 8273, namely the frame-level command structure and the Digital Phase Locked Loop, have been shown to be a valuable asset in an SDLC system design.

APPENDIX A

APPENDIX A

```
ASM80 :F1:RAYT73.SRC


ISIS-II 8080/8085 MACRO ASSEMBLER, X108        MODULE   PAGE   1


 LOC  OBJ        SEQ        SOURCE STATEMENT

                   1 $NOPAGING MOD85  NOCOND
 0000              2 TRUE    EQU     00H             ;00 FOR RAYTHEON
                   3 ;                               ;FF FOR SELF-TEST
 0000              4 TRUE1   EQU     00H             ;00 FOR NORMAL RESPONSE
                   5 ;                               ;FF FOR LOOP RESPONSE
 0000              6 DEM     EQU     00H             ;00 FOR NO DEMO
                   7 ;                               ;FF FOR DEMO
                   8 ;
                   9 ;
                  10 ;GENERAL 8273 MONITOR WITH RAYTHEON POLL MODE ADDED
                  11 ;
                  17 ;
                  18 ;
                  19 ;COMMAND SUPPORTED ARE: RS - RESET SERIAL I/O MODE
                  20 ;                       SS - SET SERIAL I/O MODE
                  21 ;                       PO - RESET OPERATING MODE
                  22 ;                       SO - SET OPERATING MODE
                  23 ;                       PD - RECEIVER DISABLE
                  24 ;                       GP - GENERAL RECEIVE
                  25 ;                       SP - SELECTIVE RECEIVE
                  26 ;                       TF - TRANSMIT FRAME
                  27 ;                       AF - ABORT FRAME
                  28 ;                       SP - SET PORT B
                  29 ;                       RP - RESET PORT B
                  30 ;                       RB - RESET ONE BIT DELAY (PAR = 7F)
                  31 ;                       SB - SET ONE BIT DELAY (PAR = 80)
                  32 ;                       SL - SELECTIVE LOOP RECEIVE
                  33 ;                       TL - TRANSMIT LOOP
                  34 ;                       Z  - CHANGE MODES FLIP/FLOP
                  38 ;
                  39 ;*****************************************************************************
                  40 ;
                  41 ;NOTE.  'SET' COMMANDS IMPLEMENT LOGICAL 'OR' FUNCTIONS
                  42 ;       'RESET' COMMANDS IMPLEMENT LOGICAL 'AND' FUNCTIONS
                  43 ;
                  44 ;*****************************************************************************
                  45 ;
                  46 ;BUFFERED MODE MUST BE SELECTED WHEN SELECTIVE RECEIVE IS USED.
                  47 ;
                  48 ;COMMAND FORMAT IS  'COMMAND (2 LTRS)' 'PAR.#1' 'PAR.#2' ETC.
                  49 ;
                  50 ;THE TRANSMIT FRAME COMMAND FORMAT IS: 'TF' 'A' 'C' 'BUFFER CONTENTS'.
                  51 ;       NO LENGTH COUNT IS NEEDED.  BUFFER CONTENTS IS ENDED WITH A CR.
                  52 ;
                  53 ;*****************************************************************************
                  54 ;
                  55 ;POLLED MODE    WHEN POLLED MODE IS SELECTED (DENOTED BY A '+' PROMPT), IF
```

```
                56 ;                A SNRM-P OR RR(0)-P IS RECEIVED, A RESPONSE FRAME OF NSA-F
                57 ;                OR RR(0)-F IS TRANSMITTED.  OTHER COMMANDS OPERATE NORMALLY.
                62 ;
                63 ;*******************************************************************************
                64 ;
                65 ;8273 EQUATES
                66 ;
0090            67 STAT73  EQU     90H             ;STATUS REGISTER
0090            68 COMM73  EQU     90H             ;COMMAND REGISTER
0091            69 PARM73  EQU     91H             ;PARAMETER REGISTER
0091            70 RESL73  EQU     91H             ;RESULT REGISTER
0092            71 TXIR73  EQU     92H             ;TX INTERRUPT RESULT REGISTER
0093            72 RXIR73  EQU     93H             ;RX INTERRUPT RESULT REGISTER
0092            73 TEST73  EQU     92H             ;TEST MODE REGISTER
0020            74 CPBF    EQU     20H             ;PARAMETER BUFFER FULL BIT
0004            75 TXINT   EQU     04H             ;TX INTERRUPT BIT IN STATUS REGISTER
0008            76 RXINT   EQU     08H             ;RX INTERRUPT BIT IN STATUS REGISTER
0001            77 TXIRA   EQU     01H             ;TX INT RESULT AVAILABLE BIT
0002            78 RXIRA   EQU     02H             ;RX INT RESULT AVAILABLE BIT
                79 ;
                80 ;8253 EQUATES
                81 ;
009B            82 MODE53  EQU     9BH             ;8253 MODE WORD REGISTER
009C            83 CNT053  EQU     9CH             ;COUNTER 0 REGISTER
009D            84 CNT153  EQU     9DH             ;COUNTER 1 REGISTER
009E            85 CNT253  EQU     9EH             ;COUNTER 2 REGISTER
000C            86 COBP    EQU     000CH           ;CONSOLE BAUD RATE (2400)
0036            87 MDCNT0  EQU     36H             ;MODE FOR COUNTER 0
00B6            88 MDCNT2  EQU     0B6H            ;MODE FOR COUNTER 2
2017            89 LKBR1   EQU     2017H           ;8273 BAUD RATE LSB ADR
2018            90 LKBR2   EQU     2018H           ;8273 BAUD RATE MSB ADR
                91 ;
                92 ;BAUD RATE TABLE.        BAUD RATE       LKBR1   LKBR2
                93 ;                        *********       *****   *****
                94 ;                          9600           2E      00
                95 ;                          4800           5C      00
                96 ;                          2400           B9      00
                97 ;                          1200           72      01
                98 ;                           600           E5      02
                99 ;                           300           C9      05
               100 ;
               101 ;
               102 ;8257 EQUATES
               103 ;
00A8           104 MODE57  EQU     0A8H            ;8257 MODE PORT
00A0           105 CH0ADR  EQU     0A0H            ;CH0 (RX) ADR REGISTER
00A1           106 CH0TC   EQU     0A1H            ;CH0 TERMINAL COUNT REGISTER
00A2           107 CH1ADR  EQU     0A2H            ;CH1 (TX) ADR REGISTER
00A3           108 CH1TC   EQU     0A3H            ;CH1 TERMINAL COUNT REGISTER
00A8           109 STAT57  EQU     0A8H            ;STATUS REGISTER
8200           110 RXBUF   EQU     8200H           ;RX BUFFER START ADDRESS
8000           111 TXBUF   EQU     8000H           ;TX BUFFER START ADDRESS
0062           112 DRDMA   EQU     62H             ;DISABLE RX DMA CHANNEL, TX STILL ON
41FF           113 RXTC    EQU     41FFH           ;TERMINAL COUNT AND MODE FOR RX CHANNEL
0063           114 ENDMA   EQU     63H             ;ENABLE BOTH TX AND RX CHANNELS-EXT. WR, TX STOP
0061           115 DTDMA   EQU     61H             ;DISABLE TX DMA CHANNEL, RX STILL ON
81FF           116 TXTC    EQU     81FFH           ;TERMINAL COUNT AND MODE FOR TX CHANNEL
               117 ;
```

```
                    118 ;8251A EQUATES
                    119 ;
0089                120 CNTL51  EQU     89H             ;CONTROL WORD REGISTER
0089                121 STAT51  EQU     89H             ;STATUS REGISTER
0088                122 TXD51   EQU     88H             ;TX DATA REGISTER
0088                123 RXD51   EQU     88H             ;RX DATA REGISTER
00CE                124 MDE51   EQU     0CEH            ;MODE 16X,2 STOP,NO PARITY
0027                125 CMD51   EQU     27H             ;COMMAND, ENABLE TX&RX
0002                126 RDY     EQU     02H             ;RXRDY BIT
                    127 ;
                    128 ;MONITOR SUBROUTINE EQUATES
                    129 ;
061F                130 GETCH   EQU     061FH           ;GET CHR FROM KEYBOARD, ASCII IN CH
05F8                131 ECHO    EQU     05F8H           ;ECHO CHR TO DISPLAY
075E                132 VALDG   EQU     075EH           ;CHECK IF VALID DIGIT, CARRY SET IF VALID
05BB                133 CNVBN   EQU     05BBH           ;CONVERTS ASCII TO HEX
05EB                134 CRLF    EQU     05EBH           ;DISPLAY CR, HENCE LF TOO
06C7                135 NMOUT   EQU     06C7H           ;CONVERT BYTE TO 2 ASCII CHR AND DISPLAY
                    136 ;
                    137 ;MISC EQUATES
                    138 ;
20C0                139 STKSRT  EQU     20C0H           ;STACK START
0003                140 CNTLC   EQU     03H             ;CNTL-C EQUIVALENT
0008                141 MONTOR  EQU     0008H           ;MONITOR
2000                142 CMDBUF  EQU     2000H           ;START OF COMMAND BUFFER
2020                143 CMDBF1  EQU     2020H           ;POLL MODE SPECIAL TX COMMAND BUFFER
000D                144 CR      EQU     0DH             ;ASCII CR
000A                145 LF      EQU     0AH             ;ASCII LF
20D4                146 RST75   EQU     20D4H           ;RST7.5 JUMP ADDRESS
20CE                147 RST65   EQU     20CEH           ;RST6.5 JUMP ADDRESS
2010                148 LDADR   EQU     2010H           ;RESULT BUFFER LOAD POINTER STORAGE
2013                149 CNADR   EQU     2013H           ;RESULT BUFFER CONSOLE POINTER STORAGE
2800                150 RESBUF  EQU     2800H           ;RESULT BUFFER START - 255 BYTES
0093                151 SNRMP   EQU     93H             ;SNRM-P CONTROL CODE
0011                152 RR0P    EQU     11H             ;RR(0)-P CONTROL CODE
0073                153 NSAF    EQU     73H             ;NSA-F CONTROL CODE
0011                154 RR0F    EQU     11H             ;RR(0)-F CONTROL CODE
2015                155 PRMPT   EQU     2015H           ;PRMPT STORAGE
2016                156 POLIN   EQU     2016H           ;POLL MODE SELECTION INDICATOR
2027                157 DEMODE  EQU     2027H           ;DEMO MODE INDICATOR
                    161 ;
                    162 ;*****************************************************************************
                    163 ;
                    164 ;RAM STORAGE DEFINITIONS:
                    165 ;       LOC             DEF
                    166 ;       ---             ---
                    167 ;       2000-200F       COMMAND BUFFER
                    168 ;       2010-2011       RESULT BUFFER LOAD POINTER
                    169 ;       2013-2014       RESULT BUFFER CONSOLE POINTER
                    170 ;       2015            PROMPT CHARACTER STORAGE
                    171 ;       2016            POLL MODE INDICATOR
                    172 ;       2017            BAUD RATE LSB FOR SELF-TEST
                    173 ;       2018            BAUD RATE MSB FOR SELF-TEST
                    177 ;       2019            SPARE
                    179 ;       2020-2026       RESPONSE COMMAND BUFFER FOR POLL MODE
                    180 ;       2800-28FF       RESULT BUFFER
                    181 ;
                    182 ;*****************************************************************************
```

```
                        183 ;
                        184 ;PROGRAM START
                        185 ;
                        186 ;INITIALIZE 8253, 8257, 8251A, AND RESET 8273.
                        187 ;ALSO SET NORMAL MODE, AND PRINT SIGNON MESSAGE
                        188 ;
0800                    189         ORG     800H
                        190
0800 31C020             191 START:  LXI     SP,STKSRT       ;INITIALIZE SP
0803 3E36               192         MVI     A,MDCNT0        ;8253 MODE SET
0805 D39B               193         OUT     MODE53          ;8253 MODE PORT
0807 3A1720             194         LDA     LKBR1           ;GET 8273 BAUD RATE LSB
080A D39C               195         OUT     CNT053          ;USING COUNTER 0 AS BAUD RATE GEN
080C 3A1820             196         LDA     LKBR2           ;GET 8273 BUAD RATE MSB
080F D39C               197         OUT     CNT053          ;COUNTER 0
0811 CD1A0B             198         CALL    RXDMA           ;INITIALIZE 8257 RX DMA CHANNEL
0814 CD350B             199         CALL    TXDMA           ;INITIALIZE 8257 TX DMA CHANNEL
0817 3E01               200         MVI     A,01H           ;OUTPUT 1 FOLLOWED BY A 0
0819 D392               201         OUT     TEST73          ;TO TEST MODE REGISTER
081B 3E00               202         MVI     A,00H           ;TO RESET THE 8273
081D D392               203         OUT     TEST73
081F 3E2D               204         MVI     A,'-'           ;NORMAL MODE PROMPT CHR
0821 321520             205         STA     PRMPT           ;PUT IN STORAGE
0824 3E00               206         MVI     A,00H           ;TX POLL RESPONSE INDICATOR
0826 321620             207         STA     POLIN           ;0 MEANS NO SPECIAL TX
0829 322720             208         STA     DEMODE          ;CLEAR DEMO MODE
082C 21A30C             212         LXI     H,SIGNON                ;SIGNON MESSAGE ADR
082F CD920C             213         CALL    TYMSG           ;DISPLAY SIGNON
                        214 ;
                        215 ;MONITOR USES JUMPS IN RAM TO DIRECT INTERRUPTS
                        216 ;
0832 21D420             217         LXI     H,RST75         ;RST7.5 JUMP LOCATION USED BY MONITOR
0835 01000C             218         LXI     B,RXI           ;ADDRESS OF RX INT ROUTINE
0838 36C3               219         MVI     M,0C3H          ;LOAD 'JMP' OPCODE
083A 23                 220         INX     H               ;INC POINTER
083B 71                 221         MOV     M,C             ;LOAD RXI LSB
083C 23                 222         INX     H               ;INC POINTER
083D 70                 223         MOV     M,B             ;LOAD RXI MSB
083E 21CE20             224         LXI     H,RST65         ;RST6.5 JUMP LOCATION USED BY MONITOR
0841 01CE0C             225         LXI     B,TXI           ;ADDRESS OF TX INT ROUTINE
0844 36C3               226         MVI     M,0C3H          ;LOAD 'JMP' OPCODE
0846 23                 227         INX     H               ;INC POINTER
0847 71                 228         MOV     M,C             ;LOAD TXI LSB
0848 23                 229         INX     H               ;INC POINTER
0849 70                 230         MOV     M,B             ;LOAD TXI MSB
084A 3E18               231         MVI     A,18H           ;GET SET TO RESET INTERRUPTS
084C 30                 232         SIM                     ;RESET INTERRUPTS
084D FB                 233         EI                      ;ENABLE INTERRUPTS
                        234 ;
                        235 ;INITIALIZE BUFFER POINTER
                        236 ;
                        237 ;
084E 210028             238         LXI     H,RESBUF        ;SET RESULT BUFFER POINTERS
0851 221320             239         SHLD    CNADR           ;RESULT CONSOLE POINTER
0854 221020             240         SHLD    LDADR           ;RESULT LOAD POINTER
                        241 ;
                        242 ;MAIN PROGRAM LOOP - CHECKS FOR CHANGE IN RESULT POINTERS, USART STATUS,
                        243 ;       OR POLL STATUS
```

```
                        244 ;
0857 CDEB05             245 CMDREC: CALL    CRLF            ;DISPLAY CR
085A 3A1520             246         LDA     PRMPT           ;GET CURRENT PROMPT CHR
085D 4F                 247         MOV     C,A             ;MOVE TO C
085E CDF805             248         CALL    ECHO            ;DISPLAY IT
0861 2A1320             249 LOOPIT: LHLD    CNADR           ;GET CONSOLE POINTER
0864 7D                 250         MOV     A,L             ;SAVE POINTER LSB
0865 2A1020             251         LHLD    LDADR           ;GET LOAD POINTER
0868 BD                 252         CMP     L               ;SAME LSB?
0869 C2790A             253         JNZ     DISPY           ;NO, RESULTS NEED DISPLAYING
086C DB89               259         IN      STAT51          ;YES, CHECK KEYBOARD
086E E602               260         ANI     RDY             ;CHR RECEIVED?
0870 C27D08             261         JNZ     GETCMD          ;MUST BE CHR SO GO GET IT
0873 3A1620             262         LDA     POLIN           ;GET POLL MODE STATUS
0876 A7                 263         ANA     A               ;IS IT 0?
0877 C24C09             264         JNZ     TXPOL           ;NO, THEN POLL OCCURRED
087A C36108             265         JMP     LOOPIT          ;YES, TRY AGAIN
                        266 ;
                        267 ;
                        268 ;COMMAND RECOGNIZER ROUTINE
                        269 ;
                        270 ;
087D CD1F06             271 GETCMD. CALL    GETCH           ;GET CHR
0880 CDF805             272         CALL    ECHO            ;ECHO IT
0883 79                 273         MOV     A,C             ;SETUP FOR COMPARE
0884 FE52               274         CPI     'R'             ;R?
0886 CAAF08             275         JZ      RDWN            ;GET MORE
0889 FE53               276         CPI     'S'             ;S?
088B CAD708             277         JZ      SDWN            ;GET MORE
088E FE47               278         CPI     'G'             ;G?
0890 CAFF08             279         JZ      GDWN            ;GET MORE
0893 FE54               280         CPI     'T'             ;T?
0895 CA0E09             281         JZ      TDWN            ;GET MORE
0898 FE41               282         CPI     'A'             ;A?
089A CA2209             283         JZ      ADWN            ;GET MORE
089D FE5A               284         CPI     'Z'             ;Z?
089F CA3109             285         JZ      CMODE           ;YES, GO CHANGE MODE
08A2 FE03               290         CPI     CNTLC           ;CNTL-C?
08A4 CA0800             291         JZ      MONTOR          ;EXIT TO MONITOR
08A7 0E3F               292 ILLEG   MVI     C,'?'           ;PRINT ?
08A9 CDF805             293         CALL    ECHO            ;DISPLAY IT
08AC C37D08             294         JMP     CMDREC          ;LOOP FOR COMMAND
                        295
08AF CD1F06             296 RDWN.   CALL    GETCH           ;GET NEXT CHR
08B2 CDF805             297         CALL    ECHO            ;ECHO IT
08B5 79                 298         MOV     A,C             ;SETUP FOR COMPARE
08B6 FE4F               299         CPI     'O'             ;O?
08B8 CA5D09             300         JZ      ROCMD           ;RO COMMAND
08BB FE53               301         CPI     'S'             ;S?
08BD CA6709             302         JZ      RSCMD           ;RS COMMAND
08C0 FE44               303         CPI     'D'             ;D?
08C2 CA7109             304         JZ      RDCMD           ;RD COMMAND
08C5 FE50               305         CPI     'P'             ;P?
08C7 CAD809             306         JZ      RPCMD           ;RP COMMAND
08CA FE52               307         CPI     'R'             ;R?
08CC CA0008             308         JZ      START           ;START OVER
08CF FE42               309         CPI     'B'             ;B?
08D1 CA7B09             310         JZ      RBCMD           ;RB COMMAND
```

```
08D4 C3A708      311          JMP     ILLEG          ;ILLEGAL, TRY AGAIN
                 312
08D7 CD1F06      313 SDWN:    CALL    GETCH          ;GET NEXT CHR
08DA CDF805      314          CALL    ECHO           ;ECHO IT.
08DD 78          315          MOV     A,B            ;SETUP FOR COMPARE
08DE FE4F        316          CPI     'O'            ;O?
08E0 CAA609      317          JZ      SOCMD          ;SO COMMAND
08E3 FE53        318          CPI     'S'            ;S?
08E5 CAB009      319          JZ      SSCMD          ;SS COMMAND
08E8 FE52        320          CPI     'R'            ;R?
08EA CABA09      321          JZ      SRCMD          ;SR COMMAND
08ED FE50        322          CPI     'P'            ;P?
08EF CAE209      323          JZ      SPCMD          ;SP COMMAND
08F2 FE42        324          CPI     'B'            ;B?
08F4 CA8509      325          JZ      SBCMD          ;SB COMMAND
08F7 FE4C        326          CPI     'L'            ;L?
08F9 CA8F09      327          JZ      SLCMD          ;SL COMMAND
08FC C3A708      328          JMP     ILLEG          ;ILLEGAL, TRY AGAIN
                 329
08FF CD1F06      330 GDWN.    CALL    GETCH          ;GET NEXT CHR
0902 CDF805      331          CALL    ECHO           ;ECHO IT
0905 78          332          MOV     A,B            ;SETUP FOR COMPARE
0906 FE52        333          CPI     'R'            ;R?
0908 CAC409      334          JZ      GRCMD          ;GR COMMAND
090B C3A708      335          JMP     ILLEG          ;ILLEGAL, TRY AGAIN
                 336
090E CD1F06      337 TDWN.    CALL    GETCH          ;GET NEXT CHR
0911 CDF805      338          CALL    ECHO           ;ECHO IT
0914 78          339          MOV     A,B            ;SETUP FOR COMPARE
0915 FE46        340          CPI     'F'            ;F?
0917 CAEC09      341          JZ      TFCMD          ;TF COMMAND
091A FE4C        342          CPI     'L'            ;L?
091C CA9909      343          JZ      TLCMD          ;TL COMMAND
091F C3A708      344          JMP     ILLEG          ;ILLEGAL, TRY AGAIN
                 345
0922 CD1F06      346 ADWN.    CALL    GETCH          ;GET NEXT CHR
0925 CDF805      347          CALL    ECHO           ;ECHO IT
0928 78          348          MOV     A,B            ;SETUP FOR COMPARE
0929 FE46        349          CPI     'F'            ;F?
092B CACE09      350          JZ      AFCMD          ;AF COMMAND
092E C3A708      351          JMP     ILLEG          ;ILLEGAL. TRY AGAIN.
                 352 ;
                 353 ;RESET POLL MODE RESPONSE - CHANGE PROMPT CHR AS INDICATOR
                 354 ;
0931 F3          355 CMODE    DI                     ;DISABLE INTERRUPTS
0932 3A1520      356          LDA     PRMPT          ;GET CURRENT PROMPT
0935 FE2D        357          CPI     '-'            ;NORMAL MODE?
0937 C24309      358          JNZ     SW             ;NO, CHANGE IT
093A 3E2B        359          MVI     A,'+'          ;NEW PROMPT
093C 321520      360          STA     PRMPT          ;STORE NEW PROMPT
093F FB          365          EI                     ;ENABLE INTERRUPTS
0940 C35708      366          JMP     CMDREC         ;RETURN TO LOOP
0943 3E2D        367 SW.      MVI     A,'-'          ;NEW PROMPT CHR
0945 321520      368          STA     PRMPT          ;STORE IT
0948 FB          369          EI                     ;ENABLE INTERRUPTS
0949 C35708      370          JMP     CMDREC         ;RETURN TO LOOP
                 371 ;
                 372 ;
```

```
                373 ;TRANSMIT ANSWER TO POLL SETUP
                374 ;
094C 3E00       382 TXPOL:  MVI     A,00H           ;CLEAR POLL INDICATOR
094E 321620     384         STA     POLIN           ;INDICATOR ADR
0951 216108     385         LXI     H,LOOPIT        ;SETUP STACK FOR COMMAND OUTPUT
0954 E5         386         PUSH    H               ;PUT RETURN TO CMDREC ON STACK
0955 0604       387         MVI     B,04H           ;GET # OF PARAMETERS READY
0957 212020     388         LXI     H,CMDBF1        ;POINT TO SPECIAL BUFFER
095A C3FF0A     389         JMP     COMM2           ;JUMP TO COMMAND OUTPUTER
                390 ;
                391 ;
                392 ;
                393 ;COMMAND IMPLEMENTING ROUTINES
                394 ;
                395 ;
                396 ;RO - RESET OPERATING MODE
                397 ;
095D 0601       398 ROCMD:  MVI     B,01H           ;# OF PARAMETERS
095F 0E51       399         MVI     C,51H           ;COMMAND
0961 CDE50A     400         CALL    COMM            ;GET PARAMETERS AND ISSUE COMMAND
0964 C35708     401         JMP     CMDREC          ;GET NEXT COMMAND
                402 ;
                403 ;RS - RESET SERIAL I/O MODE COMMAND
                404 ;
0967 0601       405 RSCMD:  MVI     B,01H           ;# OF PARAMETERS
0969 0E60       406         MVI     C,60H           ;COMMAND
096B CDE50A     407         CALL    COMM            ;GET PARAMETERS AND ISSUE COMMAND
096E C35708     408         JMP     CMDREC          ;GET NEXT COMMAND
                409 ;
                410 ;RD - RECEIVER DISABLE COMMAND
                411 ;
0971 0600       412 RDCMD:  MVI     B,00H           ;# OF PARAMETERS
0973 0EC5       413         MVI     C,0C5H          ;COMMAND
0975 CDE50A     414         CALL    COMM            ;ISSUE COMMAND
0978 C35708     415         JMP     CMDREC          ;GET NEXT COMMAND
                416 ;
                417 ;RB - RESET ONE BIT DELAY COMMAND
                418 ;
097B 0601       419 RBCMD:  MVI     B,01H           ;# OF PARAMETERS
097D 0E64       420         MVI     C,64H           ;COMMAND
097F CDE50A     421         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
0982 C35708     422         JMP     CMDREC          ;GET NEXT COMMAND
                423 ;
                424 ;SB - SET ONE BIT DELAY COMMAND
                425 ;
0985 0601       426 SBCMD:  MVI     B,01H           ;# OF PARAMETERS
0987 0EA4       427         MVI     C,0A4H          ;COMMAND
0989 CDE50A     428         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
098C C35708     429         JMP     CMDREC          ;GET NEXT COMMAND
                430 ;
                431 ;SL - SELECTIVE LOOP RECEIVE COMMAND
                432 ;
098F 0604       433 SLCMD:  MVI     B,04H           ;# OF PARAMETERES
0991 0EC2       434         MVI     C,0C2H          ;COMMAND
0993 CDE50A     435         CALL    COMM            ;GET PARAMETERS AND ISSUE COMMAND
0996 C35708     436         JMP     CMDREC          ;GET NEXT COMMAND
                437 ;
                438 ;TL - TRANSMIT LOOP COMMAND
```

```
                        439 ;
0999 210020             440 TLCMD:  LXI     H,CMDBUF        ;SET COMMAND BUFFER POINTER
099C 0602               441         MVI     B,02H           ;LOAD PARAMETER COUNTER
099E 36CA               442         MVI     M,0CAH          ;LOAD COMMAND INTO BUFFER
09A0 210220             443         LXI     H,CMDBUF+2      ;POINT AT ADR AND CNTL POSITIONS
09A3 C3F609             444         JMP     TFCMD1          ;FINISH OFF COMMAND IN TF ROUTINE
                        445 ;
                        446 ;SO - SET OPERATING MODE COMMAND
                        447 ;
09A6 0601               448 SOCMD:  MVI     B,01H           ;# OF PARAMETERS
09A8 0E91               449         MVI     C,91H           ;COMMAND
09AA CDE50A             450         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09AD C35708             451         JMP     CMDREC          ;GET NEXT COMMAND
                        452 ;
                        453 ;SS - SET SERIAL I/O COMMAND
                        454 ;
09B0 0601               455 SSCMD:  MVI     B,01H           ;# OF PARAMETERS
09B2 0EA0               456         MVI     C,0A0H          ;COMMAND
09B4 CDE50A             457         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09B7 C35708             458         JMP     CMDREC          ;GET NEXT COMMAND
                        459 ;
                        460 ;SR - SELECTIVE RECEIVE COMMAND
                        461 ;
09BA 0604               462 SRCMD:  MVI     B,04H           ;# OF PARAMETERS
09BC 0EC1               463         MVI     C,0C1H          ;COMMAND
09BE CDE50A             464         CALL    COMM            ;GET PARAMETERS AND ISSUE COMMAND
09C1 C35708             465         JMP     CMDREC          ;GET NEXT COMMAND
                        466 ;
                        467 ;GR - GENERAL RECEIVE COMMAND
                        468 ;
09C4 0602               469 GRCMD:  MVI     B,02H           ;NO PARAMETERS
09C6 0EC0               470         MVI     C,0C0H          ;COMMAND
09C8 CDE50A             471         CALL    COMM            ;ISSUE COMMAND
09CB C35708             472         JMP     CMDREC          ;GET NEXT COMMAND
                        473 ;
                        474 ;AF - ABORT FRAME COMMAND
                        475 ;
09CE 0600               476 AFCMD:  MVI     B,00H           ;NO PARAMETERS
09D0 0ECC               477         MVI     C,0CCH          ;COMMAND
09D2 CDE50A             478         CALL    COMM            ;ISSUE COMMAND
09D5 C35708             479         JMP     CMDREC          ;GET NEXT COMMAND
                        480 ;
                        481 ;RP - RESET PORT COMMAND
                        482 ;
09D8 0601               483 RPCMD:  MVI     B,01H           ;# OF PARAMETERS
09DA 0E63               484         MVI     C,63H           ;COMMAND
09DC CDE50A             485         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09DF C35708             486         JMP     CMDREC          ;GET NEXT COMMAND
                        487 ;
                        488 ;SP - SET PORT COMMAND
                        489 ;
09E2 0601               490 SPCMD:  MVI     B,01H           ;# OF PARAMETERS
09E4 0EA3               491         MVI     C,0A3H          ;COMMAND
09E6 CDE50A             492         CALL    COMM            ;GET PARAMETER AND ISSUE COMMAND
09E9 C35708             493         JMP     CMDREC          ;GET NEX COMMAND
                        494 ;
                        495 ;TF - TRANSMIT FRAME COMMAND
                        496 ;
```

```
09EC 210020      497 TFCMD:  LXI    H,CMDBUF       ;SET COMMAND BUFFER POINTER
09EF 0602        498         MVI    B,02H          ;LOAD PARAMETER COUNTER
09F1 36C8        499         MVI    M,0C8H         ;LOAD COMMAND INTO BUFFER
09F3 210220      500         LXI    H,CMDBUF+2     ;POINT AT ADR AND CNTL POSITIONS
09F6 78          501 TFCMD1: MOV    A,B            ;TEST PARAMETER COUNT
09F7 A7          502         ANA    A              ;IS IT 0?
09F8 CA070A      503         JZ     TBUFL          ;YES, LOAD TX DATA BUFFER
09FB CDAD0A      504         CALL   PARIN          ;GET PARAMETER
09FE DA8708      505         JC     ILLEG          ;ILLEGAL CHR RETURNED
0A01 23          506         INX    H              ;INC COMMAND BUFFER POINTER
0A02 05          507         DCR    B              ;DEC PARAMETER COUNTER
0A03 77          508         MOV    M,A            ;LOAD PARAMETER INTO COMMAND BUFFER
0A04 C3F609      509         JMP    TFCMD1         ;GET NEXT PARAMETER
                 510
0A07 210080      511 TBUFL:  LXI    H,TXBUF        ;LOAD TX DATA BUFFER POINTER
0A0A 010000      512         LXI    B,0000H        ;CLEAR BC - BYTE COUNTER
0A0D C5          513 TBUFL1: PUSH   B              ;SAVE BYTE COUNTER
0A0E CDAD0A      514         CALL   PARIN          ;GET DATA, ALIAS PARAMETER
0A11 DA1B0A      515         JC     ENDCHK         ;MAYBE END IF ILLEGAL
0A14 77          516         MOV    M,A            ;LOAD DATA BYTE INTO BUFFER
0A15 23          517         INX    H              ;INC BUFFER POINTER
0A16 C1          518         POP    B              ;RESTORE BYTE COUNTER
0A17 03          519         INX    B              ;INC BYTE COUNTER
0A18 C30D0A      520         JMP    TBUFL1         ;GET NEXT DATA
0A1B FE0D        521 ENDCHK: CPI    CR             ;RETURNED ILLEGAL CHR CR?
0A1D CA240A      522         JZ     TBUFFL         ;YES, THEN TX BUFFER FULL
0A20 C1          523         POP    B              ;RESTORE B TO SAVE STACK
0A21 C3A708      524         JMP    ILLEG          ;ILLEGAL CHR
0A24 C1          525 TBUFFL: POP    B              ;RESTORE BYTE COUNTER
0A25 210120      526         LXI    H,CMDBUF+1     ;POINT INTO COMMAND BUFFER
0A28 71          527         MOV    M,C            ;STORE BYTE COUNT LSB
0A29 22          528         INX    H              ;INC POINTER
0A2A 70          529         MOV    M,B            ;STORE BYTE COUNT MSB
0A2B 0604        530         MVI    B,04H          ;LOAD PARAMETER COUNT INTO B
0A2D 21360A      531         LXI    H,TFRET        ;GET RETURN ADR FOR THIS ROUTINE
0A30 C5          532         PUSH   B              ;PUSH ONCE
0A31 E3          533         XTHL                  ;PUT RETURN ON STACK
0A32 C5          534         PUSH   B              ;PUSH IT SO CMDOUT CAN USE IT
0A33 C3FB0A      535         JMP    CMDOUT         ;ISSUE COMMAND
0A36 C35708      536 TFRET:  JMP    CMDREC         ;GET NEXT COMMAND
                 537 ;
                 538 ;
                 539 ;ROUTINE TO DISPLAY RESULT IN RESULT BUFFER WHEN LOAD AND CONSOLE
                 540 ;POINTERS ARE DIFFERENT.
                 541 ;
                 542 ;
0A39 1605        543 DISPY   MVI    D,05H          ;D IS RESULT COUNTER
0A3B 2A1320      544         LHLD   CNADR          ;GET CONSOLE POINTER
0A3E E5          545         PUSH   H              ;SAVE IT
0A3F 7E          546         MOV    A,M            ;GET RESULT IC
0A40 E61F        547         ANI    1FH            ;LIMIT TO RESULT CODE
0A42 FE0C        548         CPI    0CH            ;TEST IF RX OR TX SOURCE
0A44 DA620A      549         JC     RXSORC         ;CARRY, THEN RX SOURCE
0A47 21C30C      550 TXSORC  LXI    H,TXIMSG       ;TX INT MESSAGE
0A4A CD920C      551         CALL   TYMSG          ;DISPLAY IT
0A4D E1          552 DISPY2: POP    H              ;RESTORE CONSOLE POINTER
0A4E 7E          553 DISPY1: MOV    A,M            ;GET RESULT
0A4F CDC706      554         CALL   NMOUT          ;CONVERT AND DISPLAY
```

```
0A52 0E20       555         MVI     C,' '           ;SP CHR
0A54 CDF805     556         CALL    ECHO            ;DISPLAY IT
0A57 2C         557         INR     L               ;INC BUFFER POINTER
0A58 15         558         DCR     D               ;DEC RESULT COUNTER
0A59 C24E0A     559         JNZ     DISPY1          ;NOT DONE
0A5C 221320     560         SHLD    CNADR           ;UPDATE CONSOLE POINTER
0A5F C35708     561         JMP     CMDREC          ;RETURN TO LOOP
                562 ;
                563 ;
                564 ;RECEIVER SOURCE - DISPLAY RESULTS AND RECEVIE BUFFER CONTENTS
                565 ;
                566 ;
0A62 21880C     567 RXSORC: LXI     H,RXIMSG        ;RX INT MESSAGE ADR
0A65 CD920C     568         CALL    TYMSG           ;DISPLAY MESSAGE
0A68 E1         569         POP     H               ;RESTORE CONSOLE POINTER
0A69 7E         570 RXS1:   MOV     A,M             ;RETRIEVE RESULT FROM BUFFER
0A6A CDC706     571         CALL    NMOUT           ;CONVERT AND DISPLAY IT
0A6D 0E20       572         MVI     C,' '           ;ASCII SP
0A6F CDF805     573         CALL    ECHO            ;DISPLAY IT
0A72 2C         574         INR     L               ;INC CONSOLE POINTER
0A73 15         575         DCR     D               ;DEC RESULT COUNTER
0A74 7A         576         MOV     A,D             ;GET SET TO TEST COUNTER
0A75 FE04       577         CPI     04H             ;IS THE RESULT R0?
0A77 CAA20A     578         JZ      R0PT            ;YES, GO SAVE IT
0A7A FE03       579         CPI     03H             ;IS THE RESULT R1?
0A7C CAA70A     580         JZ      R1PT            ;YES, GO SAVE IT
0A7F A7         581 RXS2:   ANA     A               ;TEST RESULT COUNTER
0A80 C2690A     582         JNZ     RXS1            ;NOT DONE YET, GET NEXT RESULT
0A83 221320     583         SHLD    CNADR           ;DONE. SO UPDATE CONSOLE POINTER
0A86 CDEB05     584         CALL    CRLF            ;DISPLAY CR
0A89 210082     585         LXI     H,RXBUF         ;POINT AT RX BUFFER
0A8C C1         586         POP     B               ;RETRIEVE RECEIVED COUNT
0A8D 78         587 RXS3:   MOV     A,B             ;IS COUNT 0?
0A8E B1         588         ORA     C               ;
0A8F CA5708     589         JZ      CMDREC          ;YES, GO BACK TO LOOP
0A92 7E         590         MOV     A,M             ;NO, GET CHR
0A93 C5         591         PUSH    B               ;SAVE BC
0A94 CDC706     592         CALL    NMOUT           ;CONVERT AND DISPLAY CHR
0A97 0E20       593         MVI     C,' '           ;ASCII SP
0A99 CDF805     594         CALL    ECHO            ;DISPLAY IT TO SEPARATE DATA
0A9C C1         595         POP     B               ;RESTORE BC
0A9D 0B         596         DCX     B               ;DEC COUNT
0A9E 23         597         INX     H               ;INC POINTER
0A9F C38D0A     598         JMP     RXS3            ;GET NEXT CHR
                599
0AA2 4E         600 R0PT:   MOV     C,M             ;GET R0 FOR RESULT BUFFER
0AA3 C5         601         PUSH    B               ;SAVE IT
0AA4 C37F0A     602         JMP     RXS2            ;RETURN
                603
0AA7 C1         604 R1PT:   POP     B               ;GET R0
0AA8 46         605         MOV     B,M             ;GET R1 FOR RESULT BUFFER
0AA9 C5         606         PUSH    B               ;SAVE IT
0AAA C37F0A     607         JMP     RXS2
                608 ;
                609 ;
                610 ;
                611 ;PARAMETER INPUT - PARAMETER RETURNED IN E REGISTER
                612 ;
```

```
                    613 ;
0AAD C5             614 PARIN:  PUSH    B           ;SAVE BC
0AAE 1601           615         MVI     D,01H       ;SET CHR COUNTER
0AB0 CD1F06         616         CALL    GETCH       ;GET CHR
0AB3 CDF805         617         CALL    ECHO        ;ECHO IT
0AB6 79             618         MOV     A,C         ;PUT CHR IN A
0AB7 FE20           619         CPI     ' '         ;SP?
0AB9 C2E00A         620         JNZ     PARIN1      ;NO, ILLEGAL, TRY AGAIN
0ABC CD1F06         621 PARIN3: CALL    GETCH       ;GET CHR OF PARAMETER
0ABF CDF805         622         CALL    ECHO        ;ECHO IT
0AC2 CD5E07         623         CALL    VALDG       ;IS IT A VALID CHR?
0AC5 D2E00A         624         JNC     PARIN1      ;NO, TRY AGAIN
0AC8 CDBB05         625         CALL    CNVBN       ;CONVERT IT TO HEX
0ACB 4F             626         MOV     C,A         ;SAVE IT IN C
0ACC 7A             627         MOV     A,D         ;GET CHR COUNTER
0ACD A7             628         ANA     A           ;IS IT 0?
0ACE CADC0A         629         JZ      PARIN2      ;YES,DONE WITH THIS PARAMETER
0AD1 15             630         DCR     D           ;DEC CHR COUNTER
0AD2 AF             631         XRA     A           ;CLEAR CARRY
0AD3 79             632         MOV     A,C         ;RECOVER 1ST CHR
0AD4 17             633         RAL                 ;ROTATE LEFT 4 PLACES
0AD5 17             634         RAL
0AD6 17             635         RAL
0AD7 17             636         RAL
0AD8 5F             637         MOV     E,A         ;SAVE IT IN E
0AD9 C3BC0A         638         JMP     PARIN3      ;GET NEXT CHR
0ADC 79             639 PARIN2: MOV     A,C         ;2ND CHR IN A
0ADD B3             640         ORA     E           ;COMBINE BOTH CHRS
0ADE C1             641         POP     B           ;RESTORE BC
0ADF C9             642         RET                 ;RETURN TO CALLING PROGRAM
0AE0 79             643 PARIN1: MOV     A,C         ;PUT ILLEGAL CHR IN A
0AE1 37             644         STC                 ;SET CARRY AS ILLEGAL STATUS
0AE2 C1             645         POP     B           ;RESTORE BC
0AE3 C9             646         RET                 ;RETURN TO CALLING PROGRAM
                    647 ;
                    648 ;
                    649 ;JUMP HERE IF BUFFER FULL
                    650 ;
0AE4 CF             651 BUFFUL: DB      0CFH        ;EXIT TO MONITOR
                    652 ;
                    653 ;
                    654 ;COMMAND DISPATCHER
                    655 ;
                    656 ;
0AE5 210020         657 COMM:   LXI     H,CMDBUF    ;SET POINTER
0AE8 C5             658         PUSH    B           ;SAVE BC
0AE9 71             659         MOV     M,C         ;LOAD COMMAND INTO BUFFER
0AEA 78             660 COMM1:  MOV     A,B         ;CHECK PARAMETER COUNTER
0AEB A7             661         ANA     A           ;IS IT 0?
0AEC CAFB0A         662         JZ      CMDOUT      ;YES, GO ISSUE COMMAND
0AEF CDAD0A         663         CALL    PARIN       ;GET PARAMETER
0AF2 DAA708         664         JC      ILLEG       ;ILLEGAL CHR RETURNED
0AF5 23             665         INX     H           ;INC BUFFER POINTER
0AF6 05             666         DCR     B           ;DEC PARAMETER COUNTER
0AF7 77             667         MOV     M,A         ;PARAMETER TO BUFFER
0AF8 C3EA0A         668         JMP     COMM1       ;GET NEXT PARAMETER
0AFB 210020         669 CMDOUT: LXI     H,CMDBUF    ;REPOINT POINTER
0AFE C1             670         POP     B           ;RESTORE PARAMETER COUNT
```

```
0AFF DB90       671 COMM2:  IN      STAT73      ;READ 8273 STATUS
0B01 07         672         RLC                 ;ROTATE CBSY INTO CARRY
0B02 DAFF0A     673         JC      COMM2       ;WAIT FOR OK
0B05 7E         674         MOV     A,M         ;OK, MOVE COMMAND INTO A
0B06 D390       675         OUT     COMM73      ;OUTPUT COMMAND
0B08 78         676 PAR1:   MOV     A,B         ;GET PARAMETER COUNT
0B09 A7         677         ANA     A           ;IS IT 0?
0B0A C8         678         RZ                  ;YES, DONE, RETURN
0B0B 23         679         INX     H           ;INC COMMAND BUFFER POINTER
0B0C 05         680         DCR     B           ;DEC PARAMETER COUNT
0B0D DB90       681 PAR2:   IN      STAT73      ;READ STATUS
0B0F E620       682         ANI     CPBF        ;IS CPBF BIT SET?
0B11 C20D0B     683         JNZ     PAR2        ;WAIT TIL ITS 0
0B14 7E         684         MOV     A,M         ;OK, GET PARAMETER FROM BUFFER
0B15 D391       685         OUT     PARM73      ;OUTPUT PARAMETER
0B17 C3080B     686         JMP     PAR1        ;GET NEXT PARAMETER
                687 ;
                688 ;
                689 ;INITIALIZE AND ENABLE RX DMA CHANNEL
                690 ;
                691 ;
0B1A 3E62       692 RXDMA:  MVI     A,DRDMA     ;DISABLE RX DMA CHANNEL
0B1C D3A8       693         OUT     MODE57      ;8257 MODE PORT
0B1E 010082     694         LXI     B,RXBUF     ;RX BUFFER START ADDRESS
0B21 79         695         MOV     A,C         ;RX BUFFER LSB
0B22 D3A0       696         OUT     CH0ADR      ;CH0 ADR PORT
0B24 78         697         MOV     A,B         ;RX BUFFER MSB
0B25 D3A0       698         OUT     CH0ADR      ;CH0 ADR PORT
0B27 01FF41     699         LXI     B,RXTC      ;RX CH TEERMINAL COUNT
0B2A 79         700         MOV     A,C         ;RX TERMINAL COUNT LSB
0B2B D3A1       701         OUT     CH0TC       ;CH0 TC PORT
0B2D 78         702         MOV     A,B         ;RX TERMINAL COUNT MSB
0B2E D3A1       703         OUT     CH0TC       ;CH0 TC PORT
0B30 3E63       704         MVI     A,ENDMA     ;ENABLE DMA WORD
0B32 D3A8       705         OUT     MODE57      ;8257 MODE PORT
0B34 C9         706         RET                 ;RETURN
                707 ;
                708 ;
                709 ;INITIALIZE AND ENABLE TX DMA CHANNEL
                710 ;
                711 ;
0B35 3E61       712 TXDMA:  MVI     A,DTDMA     ;DISABLE TX DMA CHANNEL
0B37 D3A8       713         OUT     MODE57      ;8257 MODE PORT
0B39 010080     714         LXI     B,TXBUF     ;TX BUFFER START ADDRESS
0B3C 79         715         MOV     A,C         ;TX BUFFER LSB
0B3D D3A2       716         OUT     CH1ADR      ;CH1 ADR PORT
0B3F 78         717         MOV     A,B         ;TX BUFFER MSB
0B40 D3A2       718         OUT     CH1ADR      ;CH1 ADR PORT
0B42 01FF81     719 TXDMA1: LXI     B,TXTC      ;TX CH TERMINAL COUNT
0B45 79         720         MOV     A,C         ;TX TERMINAL COUNT LSB
0B46 D3A3       721         OUT     CH1TC       ;CH1 TC PORT
0B48 78         722         MOV     A,B         ;TX TERMINAL COUNT MSB
0B49 D3A3       723         OUT     CH1TC       ;CH1 TC PORT
0B4B 3E63       724         MVI     A,ENDMA     ;ENABLE DMA WORD
0B4D D3A8       725         OUT     MODE57      ;8257 MODE PORT
0B4F C9         726         RET                 ;RETURN
                727 ;
                728 ;
```

```
                    729 ;INERRUPT PROCESSING SECTION
                    730 ;
0C00                731         ORG     0C00H
                    732 ;
                    733 ;
                    734 ;RECEIVER INTERRUPT - RST 7.5 (LOC 3CH)
                    735 ;
0C00 E5             736 RXI:    PUSH    H               ;SAVE HL
0C01 F5             737         PUSH    PSW             ;SAVE PSW
0C02 C5             738         PUSH    B               ;SAVE BC
0C03 D5             739         PUSH    D               ;SAVE DE
0C04 3E62           740         MVI     A,DRDMA         ;DISABLE RX DMA
0C06 D3A8           741         OUT     MODE57          ;8257 MODE PORT
0C08 3E18           742         MVI     A,18H           ;RESET RST7.5 F/F
0C0A 30             743         SIM
0C0B 1604           744         MVI     D,04H           ;D IS RESULT COUNTER
0C0D 2A1020         745         LHLD    LDADR           ;GET LOAD POINTER
0C10 E5             746         PUSH    H               ;SAVE IT
0C11 E5             747         PUSH    H               ;SAVE IT AGAIN
0C12 45             748         MOV     B,L             ;SAVE LSB
0C13 2A1320         749         LHLD    CNADR           ;GET CONSOLE POINTER
0C16 04             750 RXI1:   INR     B               ;BUMP LOAD POINTER LSB
0C17 78             751         MOV     A,B             ;GET SET TO TEST
0C18 BD             752         CMP     L               ;LOAD=CONSOLE?
0C19 CAE40A         753         JZ      BUFFUL          ;YES, BUFFER FULL
0C1C 15             754         DCR     D               ;DEC COUNTER
0C1D C2160C         755         JNZ     RXI1            ;NOT DONE, TRY AGAIN
0C20 1605           756         MVI     D,05H           ;RESET COUNTER
0C22 E1             757         POP     H               ;RESTORE LOAD POINTER
0C23 DB90           758 RXI2:   IN      STAT73          ;READ STATUS
0C25 E608           759         ANI     RXINT           ;TEST RX INT BIT
0C27 CA390C         760         JZ      RXI3            ;DONE, GO FINISH UP
0C2A DB90           761         IN      STAT73          ;READ STATUS AGAIN
0C2C E602           762         ANI     RXIRA           ;IS RESULT READY?
0C2E CA230C         763         JZ      RXI2            ;NO, TEST AGAIN
0C31 DB93           764         IN      RXIR73          ;YES, READ RESULT
0C33 77             765         MOV     M,A             ;STORE IN BUFFER
0C34 2C             766         INR     L               ;INC BUFFER POINTER
0C35 15             767         DCR     D               ;DEC COUNTER
0C36 C3230C         768         JMP     RXI2            ;GET MORE RESULTS
0C39 7A             769 RXI3:   MOV     A,D             ;GET SET TO TEST
0C3A A7             770         ANA     A               ;ALL RESULTS?
0C3B CA450C         771         JZ      RXI4            ;YES, SO FINISH UP
0C3E 3600           772         MVI     M,00H           ;NO, LOAD 0 TIL DONE
0C40 2C             773         INR     L               ;BUMP POINTER
0C41 15             774         DCR     D               ;DEC COUNTER
0C42 C3390C         775         JMP     RXI3            ;GO AGAIN
0C45 221020         776 RXI4:   SHLD    LDADR           ;UPDATE LOAD POINTER
0C48 3A1520         777         LDA     PRMPT           ;GET MODE INDICATOR
0C4B FE2D           778         CPI     '-'             ;NORMAL MODE?
0C4D CA850C         779         JZ      RXI6            ;YES, CLEAN UP BEFORE RETURN
                    780 ;
                    781 ;       POLL MODE SO CHECK CONTROL BYTE
                    782 ;       IF CONTROL IS A POLL, SET UP SPECIAL TX COMMAND BUFFER
                    783 ;       AND RETURN WITH POLL INDICATOR NOT 0
                    784 ;
0C50 E1             785         POP     H               ;GET PREVIOUS LOAD ADR POINTER
0C51 7E             786         MOV     A,M             ;GET IC BYTE FROM BUFFER
```

```
0C52 E61E        787          ANI     1EH             ;LOOK AT GOOD FRAME BITS
0C54 C2890C      788          JNZ     RXI5            ;IF NOT 0, INTERRUPT WASN'T FROM A GOOD FRAME
0C57 2C          789          INR     L               ;BYPASS R0 AND R1 IN BUFFER
0C58 2C          790          INR     L
0C59 2C          791          INR     L
0C5A 56          792          MOV     D,M             ;GET ADR BYTE AND SAVE IT IN D
0C5B 2C          793          INR     L
0C5C 7E          794          MOV     A,M             ;GET CNTL BYTE FROM BUFFER
0C5D FE93        795          CPI     SNRMP           ;WAS IT SNRM-P?
0C5F CA6C0C      796          JZ      T1              ;YES, GO SET RESPONSE
0C62 FE11        797          CPI     RR0P            ;WAS IT RR(0)-P?
0C64 C2890C      798          JNZ     RXI5            ;YES, GO SET RESPONSE, OTHERWISE RETURN
0C67 1E11        799          MVI     E,RR0F          ;RR(0)-P SO SET RESPONSE TO RR(0)-F
0C69 C36E0C      800          JMP     TXRET           ;GO FINISH LOADING SPECIAL BUFFER
0C6C 1E73        801 T1:      MVI     E,NSAF          ;SNRM-P SO SET RESPONSE TO NSA-F
0C6E 212020      802 TXRET:   LXI     H,CMDBF1        ;SPECIAL BUFFER ADR
0C71 36C8        806          MVI     M,0C8H          ;LOAD TX FRAME COMMAND
0C73 23          808          INX     H               ;INC POINTER
0C74 3600        809          MVI     M,00H           ;L0=0
0C76 23          810          INX     H               ;INC POINTER
0C77 3600        811          MVI     M,00H           ;L1=0
0C79 23          812          INX     H               ;INC POINTER
0C7A 72          813          MOV     M,D             ;LOAD RCVD ADR BYTE
0C7B 23          814          INX     H               ;INC POINTER
0C7C 73          815          MOV     M,E             ;LOAD RESPONSE CNTL BYTE
0C7D 3E01        816          MVI     A,01H           ;SET POLL INDICATOR NOT 0
0C7F 321620      817          STA     POLIN           ;LOAD POLL INDICATOR
0C82 C3890C      818          JMP     RXI5            ;RETURN
                 819
0C85 E1          820 RXI6:    POP     H               ;CLEAN UP STACK IF NORMAL MODE
0C86 C3890C      821          JMP     RXI5            ;RETURN
                 822
0C89 CD1A0B      823 RXI5:    CALL    RXDMA           ;RESET DMA CHANNEL
0C8C D1          824          POP     D               ;RESTORE REGISTERS
0C8D C1          825          POP     B
0C8E F1          826          POP     PSW
0C8F E1          827          POP     H
0C90 FB          828          EI                      ;ENABLE INTERRUPTS
0C91 C9          829          RET                     ;RETURN
                 830 ;
                 831 ;
                 832 ;MESSAGE TYPER - ASSUMES MESSAGE STARTS AT HL
                 833 ;
                 834 ;
0C92 C5          835 TYMSG:   PUSH    B               ;SAVE BC
0C93 7E          836 TYMSG2:  MOV     A,M             ;GET ASCII CHR
0C94 23          837          INX     H               ;INC POINTER
0C95 FEFF        838          CPI     0FFH            ;STOP?
0C97 CAA10C      839          JZ      TYMSG1          ;YES, GET SET FOR EXIT
0C9A 4F          840          MOV     C,A             ;SET UP FOR DISPLAY
0C9B CDF805      841          CALL    ECHO            ;DISPLAY CHR
0C9E C3930C      842          JMP     TYMSG2          ;GET NEXT CHR
0CA1 C1          843 TYMSG1:  POP     B               ;RESTORE BC
0CA2 C9          844          RET                     ;RETURN
                 845 ;
                 846 ;
                 847 ;SIGNON MESSAGE
                 848 ;
```

```
0CA3 0D          849 SIGNON: DB      CR,'8273 MONITOR  V1.1',CR,0FFH
0CA4 38323733
0CA8 204D4F4E
0CAC 49544F52
0CB0 20205631
0CB4 2E31
0CB6 0D
0CB7 FF
                 850 ;
                 851 ;
                 852 ;
                 853 ;RECEIVER INTERRUPT MESSAGES
                 854 ;
                 855 ;
0CB8 0D          856 RXIMSG: DB      CR,'RX INT - ',0FFH
0CB9 52582049
0CBD 4E54202D
0CC1 20
0CC2 FF
                 857 ;
                 858 ;TRANSMITTER INTERRUPT MESSAGES
                 859 ;
0CC3 0D          860 TXIMSG: DB      CR,'TX INT - ',0FFH
0CC4 54582049
0CC8 4E54202D
0CCC 20
0CCD FF
                 861 ;
                 862 ;
                 863 ;TRANSMITTER INTERRUPT ROUTINE
                 864 ;
0CCE E5          865 TXI:    PUSH    H           ;SAVE HL
0CCF F5          866         PUSH    PSW         ;SAVE PSW
0CD0 C5          867         PUSH    B           ;SAVE BC
0CD1 D5          868         PUSH    D           ;SAVE DE
0CD2 3E61        869         MVI     A,DTDMA     ;DISABLE TX DMA
0CD4 D3A8        870         OUT     MODE57      ;8257 MODE PORT
0CD6 1604        871         MVI     D,04H       ;SET COUNTER
0CD8 2A1020      872         LHLD    LDADR       ;GET LOAD POINTER
0CDB E5          873         PUSH    H           ;SAVE IT
0CDC 45          874         MOV     B,L         ;SAVE LSB IN B
0CDD 2A1320      875         LHLD    CNADR       ;GET CONSOLE POINTER
0CE0 04          876 TXI1:   INR     B           ;INC POINTER
0CE1 78          877         MOV     A,B         ;GET SET TO TEST
0CE2 BD          878         CMP     L           ;LOAD=CONSOLE?
0CE3 CAE40A      879         JZ      BUFFUL      ;YES, BUFFER FULL
0CE6 15          880         DCR     D           ;NO, TEST NEXT LOCATION
0CE7 C2E00C      881         JNZ     TXI1        ;TRY AGAIN
0CEA E1          882         POP     H           ;RESTORE LOAD POINTER
0CEB DB92        883         IN      TXIR73      ;READ RESULT
0CED 77          884         MOV     M,A         ;STORE IN BUFFER
0CEE 2C          885         INR     L           ;INR POINTER
0CEF 3600        886         MVI     M,00H       ;EXTRA RESULT SPOTS 0
0CF1 2C          887         INR     L
0CF2 3600        888         MVI     M,00H
0CF4 2C          889         INR     L
0CF5 3600        890         MVI     M,00H
0CF7 2C          891         INR     L
```

```
0CF8 3600       892         MVI     M,00H
0CFA 2C         893         INR     L
0CFB 221020     894         SHLD    LDADR           ;UPDATE LOAD POINTER
0CFE CD350B     899         CALL    TXDMA           ;RESET DMA CHANNEL
0D01 D1         900         POP     D               ;RESTORE DE
0D02 C1         901         POP     B               ;RESTORE BC
0D03 F1         902         POP     PSW             ;RESTORE PSW
0D04 E1         903         POP     H               ;RESTORE HL
0D05 FB         904         EI                      ;ENABLE INTERRUPTS
0D06 C9         905         RET                     ;RETURN
                906 ;
                907 ;
                952 ;
                953 ;
                954         END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
ADWN   A 0922   AFCMD  A 09CE   BUFFUL A 0AE4   CH0ADR A 00A0   CH0TC  A 00A1   CH1ADR A 00A2   CH1TC  A 00A3
CMD51  A 0027   CMDBF1 A 2020   CMDBUF A 2000   CMDOUT A 0AFB   CMDREC A 0857   CMODE  A 0931   CNADR  A 2013
CNT053 A 009C   CNT153 A 009D   CNT253 A 009E   CNTL51 A 0089   CNTLC  A 0003   CNVBN  A 05BB   COBR   A 000C
COMM   A 0AE5   COMM1  A 0AEA   COMM2  A 0AFF   COMM73 A 0090   CPBF   A 0020   CR     A 000D   CRLF   A 05EB
DEM    A 0000   DEMODE A 2027   DISPV  A 0A39   DISPV1 A 0A4E   DISPV2 A 0A4D   DRDMA  A 0062   DTDMA  A 0061
ECHO   A 05F8   ENDCHK A 0A1B   ENDMA  A 0063   GDWN   A 08FF   GETCH  A 061F   GETCMD A 087D   GRCMD  A 09C4
ILLEG  A 08A7   LDADR  A 2010   LF     A 000A   LKBR1  A 2017   LKBR2  A 2018   LOOPIT A 0861   MDCNT0 A 0036
MDCNT2 A 00B6   MDE51  A 00CE   MODE53 A 009B   MODE57 A 00A8   MONTOR A 0008   NMOUT  A 06C7   NSAF   A 0073
PAR1   A 0B08   PAR2   A 0B0D   PARIN  A 0AAD   PARIN1 A 0AE0   PARIN2 A 0ADC   PARIN3 A 0ABC   PARM73 A 0091
POLIN  A 2016   PRMPT  A 2015   R0PT   A 0AA2   R1PT   A 0AA7   RBCMD  A 097B   RDCMD  A 0971   RDWN   A 08AF
RDY    A 0002   RESBUF A 2800   RESL73 A 0091   ROCMD  A 095D   RPCMD  A 09D8   RR0F   A 0011   RR0P   A 0011
RSCMD  A 0967   RST65  A 20CE   RST75  A 20D4   RXBUF  A 8200   RXD51  A 0088   RXDMA  A 0B1A   RXI    A 0C00
RXI1   A 0C16   RXI2   A 0C23   RXI3   A 0C39   RXI4   A 0C45   RXI5   A 0C89   RXI6   A 0C85   RXIMSG A 0CB8
RXINT  A 0008   RXIR73 A 0093   RXIRA  A 0002   RXS1   A 0A69   RXS2   A 0A7F   RXS3   A 0A8D   RXSORC A 0A62
RXTC   A 41FF   SBCMD  A 0985   SDWN   A 08D7   SIGNON A 0CA3   SLCMD  A 098F   SNRMP  A 0093   SOCMD  A 09A6
SPCMD  A 09E2   SRCMD  A 09BA   SSCMD  A 09B0   START  A 0800   STAT51 A 0089   STAT57 A 00A8   STAT73 A 0090
STKSRT A 20C0   SW     A 0943   T1     A 0C6C   TBUFFL A 0A24   TBUFL  A 0A07   TBUFL1 A 0A8D   TDWN   A 090E
TEST73 A 0092   TFCMD  A 09EC   TFCMD1 A 09F6   TFRET  A 0A36   TLCMD  A 0999   TRUE   A 0000   TRUE1  A 0000
TXBUF  A 8000   TXD51  A 0088   TXDMA  A 0B35   TXDMA1 A 0B42   TXI    A 0CCE   TXI1   A 0CE0   TXIMSG A 0CC3
TXINT  A 0004   TXIR73 A 0092   TXIRA  A 0001   TXPOL  A 094C   TXRET  A 0C6E   TXSORC A 0A47   TXTC   A 81FF
TYMSG  A 0C92   TYMSG1 A 0CA1   TYMSG2 A 0C93   VALDG  A 075E

ASSEMBLY COMPLETE,  NO ERRORS
```

intel

APPLICATION NOTE

AP-134

October 1981

Asynchronous Communication with the 8274 Multiple-Protocol Serial Controller

Order number: 210311-001

INTRODUCTION

The 8274 Multiprotocol serial controller (MPSC) is a sophisticated dual-channel communications controller that interfaces microprocessor systems to high-speed serial data links (at speeds to 880K bits per second) using synchronous or asynchronous protocols. The 8274 interfaces easily to most common microprocessors (e.g., 8048, 8051, 8085, 8086, and 8088), to DMA controllers such as the 8237 and 8257, and to the 8089 I/O processor. Both MPSC communication channels are completely independent and can operate in a full-duplex communication mode (simultaneous data transmission and reception).

Communication Functions

The 8274 performs many communications-oriented functions, including:

—Converting data bytes from a microprocessor system into a serial bit stream for transmission over the data link to a receiving system.

—Receiving serial bit streams and reconverting the data into parallel data bytes that can easily be processed by the microprocessor system.

—Performing error checking during data transfers. Error checking functions include computing/transmitting error codes (such as parity bits or CRC bytes) and using these codes to check the validity of received data.

—Operating independently of the system processor in a manner designed to reduce the system overhead involved in data transfers.

System Interface

The MPSC system interface is extremely flexible, supporting the following data transfer modes:

1. Polled Mode. The system processor periodically reads (polls) an 8274 status register to determine when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
2. Interrupt Mode. The MPSC interrupts the system processor when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
3. DMA Mode. The MPSC automatically requests data transfers from system memory for both transmit and receive functions by means of two DMA request signals per serial channel. These DMA request signals may be directly interfaced to an 8237 or 8257 DMA controller or to an 8089 I/O processor.
4. WAIT Mode. The MPSC ready signal is used to synchronize processor data transfers by forcing the processor to enter wait states until the 8274 is ready for another data byte. This feature enables the 8274 to interface directly to an 8086 or 8088 processor by means of string I/O instructions for very high-speed data links.

Scope

This application note describes the use of the 8274 in asynchronous communication modes. Asynchronous communication is typically used to transfer data to/from video display terminals, modems, printers, and other low-to-medium-speed peripheral devices. Use of the 8274 in both interrupt-driven and polled system environments is described. Use of the DMA and WAIT modes are not described since these modes are employed mainly in synchronous communication systems where extremely high data rates are common. Programming examples are written in PL/M-86 (Appendix B and Appendix C). PL/M-86 is executed by the iAPX-86 and iAPX-88 processor families. In addition, PL/M-86 is very similar to PL/M-80 (executed by the MCS-80 and MCS-85 processor families). In addition, Appendix D describes a simple application example using an SDK-86 in an iAPX-86/88 environment.

SERIAL-ASYNCHRONOUS DATA LINKS

A serial asynchronous interface is a method of data transmission in which the receiving and transmitting systems need not be synchronized. Instead of transmitting clocking information with the data, locally generated clocks (16, 32 or 64 times as fast as the data transmission rate) are used by the transmitting and receiving systems. When a character of information is sent by the transmitting system, the character data is framed (preceded and followed) by special START and STOP bits. This framing information permits the receiving system to temporarily synchronize with the data transmission. (Refer to Figure 1 during the following discussion of asynchronous data transmission.)

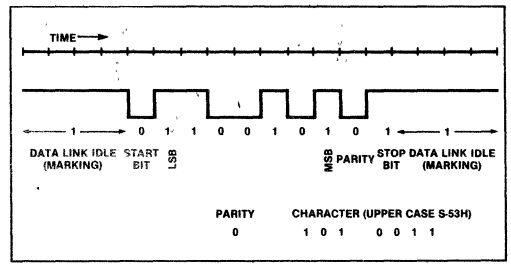


Figure 1. Transmission of a 7-Bit ASCII Character with Even Parity

Normally the data link is in an idle or marking state, continuously transmitting a "mark" (binary 1). When a character is to be sent, the character data bits are immediately preceded by a "space" (binary 0 START bit). The mark-to-space transition informs the receiving system that a character of information will immediately follow the start bit. Figure 1 illustrates the transmission of a 7-bit ASCII character (upper case S) with even parity. Note that the character is transmitted immediately following the start bit. Data bits within the character are transmitted from least-significant to most-significant. The parity bit is transmitted immediately following the character data bits and the STOP framing bit (binary 1) signifies the end of the character.

Asynchronous interfaces are often used with human interface devices such as CRT/keyboard units where the time between data transmissions is extremely variable.

Characters

In asynchronous mode, characters may vary in length from five to eight bits. The character length depends on the coding method used. For example, five-bit characters are used when transmitting Baudot Code, seven-bit characters are required for ASCII data, and eight-bit characters are needed for EBCDIC and binary data. To transmit messages composed of multiple characters, each character is framed and transmitted separately (Figure 2).

This framing method ensures that the receiving system can easily synchronize with the start and stop bits of each character, preventing receiver synchronization errors. In addition, this synchronization method makes both transmitting and receiving systems insensitive to possible time delays between character transmissions.

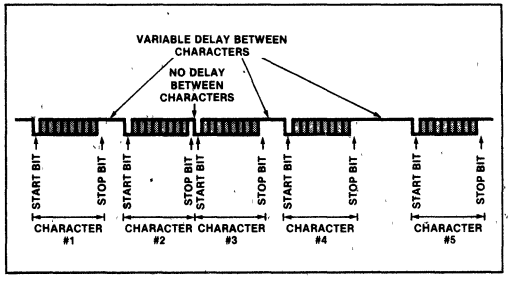


Figure 2. Multiple Character Transmission

Framing

Character framing is accomplished by the START and STOP bits described previously. When the START bit transition (mark-to-space) is detected, the receiving system assumes that a character of data will follow. In order to test this assumption (and isolate noise pulses on the data link), the receiving system waits one-half bit time and samples the data link again. If the link has returned to the marking state, noise is assumed, and the receiver waits for another START bit transition.

When a valid START bit is detected, the receiver samples the data link for each bit of the following character. Character data bits and the parity bit (if required) are sampled at their nominal centers until all required characters are received. Immediately following the data bits, the receiver samples the data link for the STOP bit, indicating the end of the character. Most systems permit specification of 1, 1½, or 2 stop bits.

Timing

The transmitter and receiver in an asynchronous data link arrangement are clocked independently. Normally, each clock is generated locally and the clocks are not synchronized. In fact, each clock may be a slightly different frequency. (In practice, the frequency difference should not exceed a few percent. If the transmitter and receiver clock rates vary substantially, errors will occur because data bits may be incorrectly identified as START or STOP framing bits.) These clocks are designed to operate at 16, 32, or 64 times the communications data rate. These clock speeds allow the receiving device to correctly sample the incoming bit stream.

Serial-interface data rates are measured in bits/second. The term "baud" is used to specify the number of times per second that the transmitted signal level can change states. In general, the baud is not equal to the bit rate. Only when the transmitted signal has two states (electrical levels) is the baud rate equal to the bit rate. Most point-to-point serial data links use RS-232-C, RS-422, or RS-423 electrical interfaces. These specifications call for two electrical signal levels (the baud is equal to the bit rate). Modem interfaces, however, may often have differing bit and baud rates.

While there are generally no limitations on the data transmission rates used in an asynchronous data link, a limited set of rates has been standardized to promote equipment interconnection. These rates vary from 75 bits per second to 38,400 bits per second. Table 1 illustrates typical asynchronous data rates and the associated clock frequencies required for the transmitter and receiver circuits.

Table 1. Communication Data Rates and Associated Transmitter/Receiver Clock Rates

Data Rate (bits/second)	Clock Rate (kHz)		
	X16	X32	X64
75	1.2	2.4	4.8
150	2.4	4.8	9.6
300	4.8	9.6	19.2
600	9.6	19.2	38.4
1200	19.2	38.4	76.8
2400	38.4	76.8	153.6
4800	76.8	153.6	307.2
9600	153.6	307.2	614.2
19200	307.2	614.4	—
38400	614.4	—	—

Parity

In order to detect transmission errors, a parity bit may be added to the character data as it is transferred over the data link. The parity bit is set or cleared to make the total number of "one" bits in the character even (even parity) or odd (odd parity). For example, the letter "A" is represented by the seven-bit ASCII code 1000001 (41H). The transmitted data code (with parity) for this character contains eight bits; 01000001 (41H) for even parity and 11000001 (OC1H) for odd parity. Note that a single bit error changes the parity of the received character and is therefore easily detected. The 8274 supports both odd and even parity checking as well as a parity disable mode to support binary data transfers.

Communication Modes

Serial data transmission between two devices can occur in one of three modes. In the simplex transmission mode, a data link can transmit data in one direction only. In the half-duplex mode, the data link can transmit data in both directions, but not simultaneously. In the full-duplex mode (the most common), the data link can transmit data in both directions simultaneously. The 8274 directly supports the full-duplex mode and will interface to simplex and half-duplex communication data links with appropriate software controls.

BREAK Condition

Asynchronous data links often include a special sequence known as a break condition. A break condition is initiated when the transmitting device forces the data link to a spacing state (binary 0) for an extended length of time (typically 150 milliseconds). Many terminals contain keys to initiate a break sequence. Under software control, the 8274 can initiate a break sequence when transmitting data and detect a break sequence when receiving data.

MPSC SYSTEM INTERFACE

Hardware Environment

The 8274 MPSC interfaces to the system processor over an 8-bit data bus. Each serial I/O channel responds to two I/O or memory addresses as shown in Table 2. In addition, the MPSC supports vectored and daisy-chained interrupts.

The 8274 may be configured for memory-mapped or I/O-mapped operation.

Table 2. 8274 Addressing

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch A Data Read	Ch A Data Write
0	1	0	Ch A Status Read	Ch A Command/Parameter
0	0	1	Ch B Data Read	Ch B Data Write
0	1	1	Ch B Status Read	Ch. B Command/Parameter
1	X	X	High Impedence	High Impedence

The 8274-processor hardware interface can be configured in a flexible manner, depending on the operating mode selected—polled, interrupt-driven, DMA, or WAIT. Figure 3 illustrates typical MPSC configurations for use with an 8088 microprocessor in the polled and interrupt-driven modes.

All serial-to-parallel conversion, parallel-to-serial conversion, and parity checking required during asynchronous serial I/O operation is automatically performed by the MPSC.

Operational Interface

Command, parameter, and status information is stored in 22 registers within the MPSC (8 writable registers and 3 readable registers for each channel). These registers are all accessed by means of the command/status ports for each channel. An internal pointer register selects which of the command or status registers will be written or read during a command/status access of an MPSC channel. Figure 4 diagrams the command/status register architecture for each serial channel. In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.

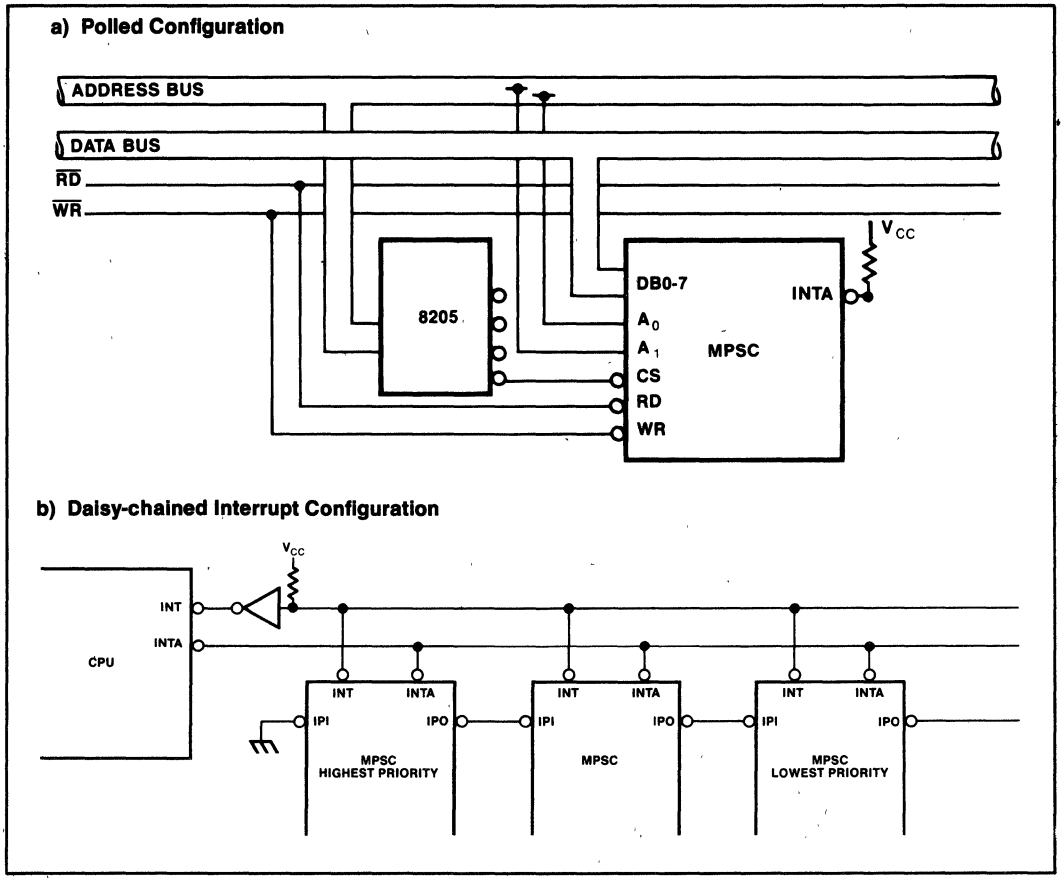


Figure 3. 8274 Hardware Interface for Polled and Interrupt-driven Environments

The least-significant three bits of WR0 are automatically loaded into the pointer register every time WR0 is written. After reset, WR0 is set to zero so that the first write to a command register causes the data to be loaded into WR0 (thereby setting the pointer register). After WR0 is written, the following read or write accesses the register selected by the pointer. The pointer is reset after the read or write operation is completed. In this manner, reading or writing an arbitrary MPSC channel register requires two I/O accesses. The first access is always a write command. This write command is used to set the pointer register. The second access is either a read or a write command; the pointer register (previously set) will ensure that the correct internal register is read or written. After this second access, the pointer register is automatically reset. Note that writing WR0 and reading RR0 does not require presetting of the pointer register.

During initialization and normal MPSC operation, various registers are read and/or written by the system processor. These actions are discussed in detail in the following paragraphs. Note that WR6 and WR7 are not used in the asynchronous communication modes.

RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals (RTS, DTR) are forced high. In addition, the pointer register is set to zero.

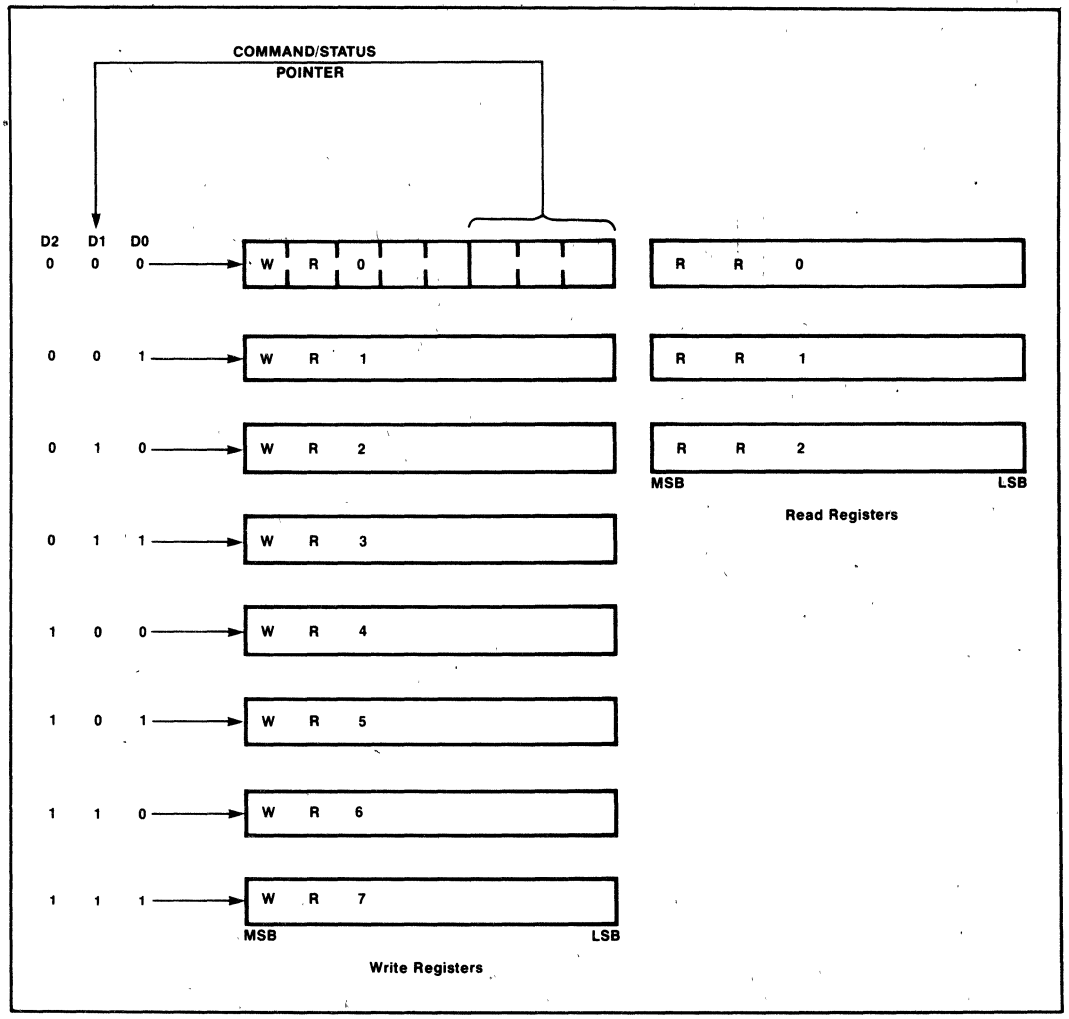


Figure 4. Command/Status Register Architecture (Each Serial Channel)

External/Status Latches

The MPSC continuously monitors the state of four external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).

A change of state in any of these monitored conditions will cause the associated status bit in RR0 (Appendix A) to be latched (and optionally cause an interrupt).

Error Reporting

Three error conditions may be encountered during data reception in the asynchronous mode:

1. Parity. If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4).
2. Framing. A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled).
3. Overrun. If an input character has been assembled but the receiver buffers are full (because the previously received characters have not been read by the system processor), an overrun error will occur. When an overrun error occurs, the input character that has just been received will overwrite the immediately preceding character.

Transmitter/Receiver Initialization

In order to operate in the asynchronous mode, each MPSC channel must be initialized with the following information:

1. Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 16, 32, or 64 times the data-link bit rate. (See Appendix A for WR4 details.)
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2. (See Appendix A for WR4 details.)
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4. (See Appendix A for WR4 details.)
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3. (See Appendix A for WR3 details.)
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3. (See Appendix A for WR3 details.)
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. (See Appendix A for WR5 details.) Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 1).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

D7	D6	D5	D4	D3	D2	D1	D0	Number of Bits Transmitted (Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5. (See Appendix A for WR5 details.)

For data transmissions via a modem or RS-232-C interface, the following information must also be specified:

1. Request-to-Send/Data-Terminal-Ready. Must be set to indicate status of data terminal equipment. Request-to-send is controlled by bit 1 of WR5 and data terminal ready is controlled by bit 7. (See Appendix A for WR5 details.)
2. Auto Enable. May be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. Auto Enable is controlled by bit 5 of WR3. (See Appendix A for WR3 details.)

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Appendix B. Figure 5 illustrates typical MPSC initialization parameters for use with this procedure.

```
call MPSC$RX$INIT(41,  1,1,0,1,  3,1,1,  3,1,1,0,1);

initializes the 8274 at address 41 as follows:

X16 clock rate                  Enable transmitter and receiver
1 stop bit                      Auto enable set
Odd parity                      DTR and RTS set
8-bit characters (Tx and Rx)    Break transmission disabled
```

Figure 5. Sample 8274 Initialization Procedure for Polled Operation

Polled Operation

In the polled mode, the processor must monitor the MPSC status by testing the appropriate bits in the read register. Data available, status, and error conditions are represented in RR0 and RR1 for channels A and B. An example of MPSC-polled transmitter/receiver routines are given in Appendix B. The following routines are detailed:

1. MPSC$POLL$RCV$CHARACTER—This procedure receives a character from the serial data link. The routine waits until the character-available flag in RR0 has been set. When this flag indicates that a character is available, RR1 is checked for errors (overrun, parity, or framing). If an error is detected, the character in the MPSC receive buffer must be read and discarded and the error routine (RECEIVE$ERROR) is called. If no receive errors have been detected, the character is input from the 8274 data port and returned to the calling program.

   MPSC$POLL$RCV$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the address of a byte variable in which to store the received character (character$ptr).
2. MPSC$POLL$TRAN$CHARACTER—This procedure transmits a character to the serial data link. The routine waits until the transmitter-buffer-empty flag has been set in RR0 before writing the character to the 8274.

   MPSC$POLL$TRAN$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the character of data that is to be transmitted (character).
3. RECEIVE$ERROR—This procedure processes receiver errors. First, an Error Reset command is written to the affected channel. All additional error processing is dependent on the specific application. For example, the receiving device may immediately request retransmission of the character or wait until a message has been completed.

   RECEIVE$ERROR requires two parameters—the address of the affected 8274 command port (cmd$port) and the error status (status) from 8274 register RR1.

Interrupt-driven Operation

In an interrupt-driven environment, all receiver operations are reported to the system processor by means of interrupts. Once a character has been received and assembled, the MPSC interrupts the system processor. The system processor must then read the character from the MPSC data buffer and clear the current interrupt. During transmission, the system processor starts serial I/O by writing the first character of a message to the MPSC. The MPSC interrupts the system processor whenever the next character is required (i.e., when the transmitter buffer is empty) and the processor responds by writing the next character of the message to the MPSC data port for the appropriate channel.

By using interrupt-driven I/O, the MPSC proceeds independently of the system processor, signalling the processor only when characters are required for transmission, when characters are received from the data link, or when errors occur. In this manner, the system processor may continue execution of other tasks while serial I/O is performed concurrently.

Interrupt Configurations

The 8274 is designed to interface to 8085- and 8086-type processors in much the same manner as the 8259A is designed. When operating in the 8085 mode, the 8274 causes a "call" to a prespecified, interrupt-service routine location. In the 8086 mode, the 8274 presents the processor with a one-byte interrupt-type number. This interrupt-type number is used to "vector" through the 8086 interrupt service table. In either case, the interrupt service address or interrupt-type number is specified during MPSC initialization.

To shorten interrupt latency, the 8274 can be programmed to modify the prespecified interrupt vector so that no software overhead is required to determine the cause of an interrupt. When this "status affects vector" mode is enabled, the following eight interrupts are differentiated automatically by the 8274 hardware:

1. Channel B Transmitter Buffer Empty.
2. Channel B External/Status Transition.
3. Channel B Character Available.
4. Channel B Receive Error.
5. Channel A Transmitter Buffer Empty.
6. Channel A External/Status Transition.
7. Channel A Character Available.
8. Channel A Receive Error.

Interrupt Sources/Priorities

The 8274 has three interrupt sources for each channel:

1. Receiver (RxA, RxB). An interrupt is initiated when a character is available in the receiver buffer or when a receiver error (parity, framing, or overrun) is detected.

2. Transmitter (TxA, TxB). An interrupt is initiated when the transmitter buffer is empty and the 8274 is ready to accept another character for transmission.
3. External/Status (ExTA, ExTB). An interrupt is initiated when one of the external/status conditions (CD, CTS, SYNDET, BREAK) changes state.

The 8274 supports two interrupt priority orderings (selectable during MPSC initialization) as detailed in Appendix A, WR2, CH-A.

Interrupt Initialization

In addition to the initialization parameters required for polled operation, the following parameters must be supplied to the 8274 to specify interrupt operation:

1. Transmit Interrupt Enable. Transmitter-buffer-empty interrupts are separately enabled by bit 1 of WR1. (See Appendix A for WR1 details.)
2. Receive Interrupt Enable. Receiver interrupts are separately enabled in one of three modes: a) interrupt on first received character only and on receive errors (used for message-oriented transmission systems), b) interrupt on all received characters and on receive errors, but do not interrupt on parity errors, and c) interrupt on all received characters and on receive errors (including parity errors). The ability to separately disable parity interrupts can be extremely useful when transmitting messages. Since the parity error bit in RR1 is latched, it will not be reset until an error reset operation is performed. Therefore, the parity error bit will be set if any parity errors were detected in a multicharacter message. If this mode is used, the serial I/O software must poll the parity error bit at the completion of a message and issue an error reset if appropriate. The receiver interrupt mode is controlled by bits 3 and 4 of WR1. (See Appendix A for WR1 details.)
3. External/Status Interrupts. External/Status interrupts can be separately enabled by bit 0 of WR1. (See Appendix A for WR1 details.)
4. Interrupt Vector. An eight-bit interrupt-service routine location (8085) or interrupt type (8086) is specified through WR2 of channel B. (See Appendix A for WR2 details). Table 3 lists interrupt vector addresses generated by the 8274 in the "status affects vector" mode.
5. "Status Affects Vector" Mode. The 8274 will automatically modify the interrupt vector if bit 3 of WR1 is set. (See Appendix A for WR1 details.)
6. System Configuration. Specifies the 8274 data transfer mode. Three configuration modes are available: a) interrupt-driven operation for both channels, b) DMA operation for both channels, and c) DMA operation for channel A, interrupt-driven operation for channel B. The system configuration is specified by means of bits 0 and 1 of WR2 (channel A). (See Appendix A for WR2 details.)
7. Interrupt Priorities. The 8274 permits software specification of receive/transmit priorities by means of bit 2 of WR2 (channel A). (See Appendix A of WR2 details.)
8. Interrupt Mode. Specifies whether the MPSC is to operate in a non-vectored mode (for use with an external interrupt controller), in an 8086-vectored mode, or in an 8085-vectored mode. This parameter is specified through bits 3 and 4 of WR2 (channel A). (See Appendix A for WR2 details.)

Table 3. MPSC-generated Interrupt Vectors in "Status Affects Vector" Mode

V7 V6 V5 V4 V3 V2 V1 V0	V7 V6 V5 V4 V3 V2 V1 V0	Original Vector (specified during initialization)
8086 Interrupt Type	8085 Interrupt Location	Interrupt Condition
V7 V6 V5 V4 V3 0 0 0	V7 V6 V5 0 0 0 V1 V0	Channel B Transmitter Buffer Empty
V7 V6 V5 V4 V3 0 0 1	V7 V6 V5 0 0 1 V1 V0	Channel B External/Status Change
V7 V6 V5 V4 V3 0 1 0	V7 V6 V5 0 1 0 V1 V0	Channel B Receiver Character Available
V7 V6 V5 V4 V3 0 1 1	V7 V6 V5 0 1 1 V1 V0	Channel B Receive Error
V7 V6 V5 V4 V3 1 0 0	V7 V6 V5 1 0 0 V1 V0	Channel A Transmitter Buffer Empty
V7 V6 V5 V4 V3 1 0 1	V7 V6 V5 1 0 1 V1 V0	Channel A External/Status Change
V7 V6 V5 V4 V3 1 1 0	V7 V6 V5 1 1 0 V1 V0	Channel A Receiver Character Available
V7 V6 V5 V4 V3 1 1 1	V7 V6 V5 1 1 1 V1 V0	Channel A Receive Error

An MPSC interrupt initialization procedure (MPSCINTINIT) is listed in Appendix C.

Interrupt Service Routines

Appendix C lists four interrupt service procedures, a buffer transmission procedure, and a buffer reception procedure that illustrate the use of the 8274 in interrupt-driven environments. Use of these procedures assumes that the 8086/8088 interrupt vector is set to 20H and that channel B is used with the "status affects vector" mode enabled.

1. TRANSMIT$BUFFER—This procedure begins serial transmission of a data buffer. Two parameters are required—a pointer to the buffer (buf$ptr) and the length of the buffer (buf$length). The procedure first sets the global buffer pointer, buffer length, and

initial index for the transmitter-interrupt service routine and initiates transmission by writing the first character of the buffer to the 8274. The procedure then enters a wait loop until the I/O completion status is set by the transmit-interrupt service routine (MPSC$TRANSMIT$CHARACTER$INT).

2. RECEIVE$BUFFER—This procedure inputs a line (terminated by a line feed) from a serial I/O port. Two parameters are required—a pointer to the input buffer (buf$ptr) and a pointer to the buffer length variable (buf$length$ptr). The buffer length will be set by this procedure when the complete line has been input. The procedure first sets the global buffer pointer and initial index for the receiver interrupt service routine. RECEIVE$BUFFER then enters a wait loop until the I/O completion status is set by the receive interrupt routine (MPSC$RECEIVE$-CHARACTER$INT).

3. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when the MPSC Tx-buffer-empty interrupt is acknowledged. If the current transmit buffer index is less than the buffer length, the next character in the buffer is written to the MPSC data port and the buffer pointer is updated. Otherwise, the transmission complete status is posted.

4. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when a character has been assembled by the MPSC and the MPSC has issued a character-available interrupt. If no input buffer has been set up by RECEIVE$BUFFER, the character is ignored. If a buffer has been set up, but it is full, a receive overrun error is posted. Otherwise, the received character is read from the MPSC data port and the buffer index is updated. Finally, if the received character is a line feed, the reception complete status is posted.

5. RECEIVE$ERROR$INT—This procedure is executed when a receive error is detected. First, the error conditions are read from RR1 and the character currently in the MPSC receive buffer is read and discarded. Next, an Error Reset command is written to the affected channel. All additional error procession is application dependent.

6. EXTERNAL$STATUS$CHANGE$INT—This procedure is executed when an external status condition change is detected. The status conditions are read from RR0 and a Reset External/Status Interrupt command is issued. Further error processing is application dependent.

DATA LINK INTERFACE

Serial Data Interface

Each serial I/O channel within the 8274 MPSC interfaces to two data link lines—one line for transmitting data and one for receiving data. During transmission, characters are converted from parallel data format (as supplied by the system processor or DMA device) into a serial bit stream (with START and STOP bits) and clocked out on the TxD pin. During reception, a serial bit stream is input on the RxD pin, framing bits are stripped out of the data stream, and the resulting character is converted to parallel data format and passed to the system processor or DMA device.

Data Clocking

As discussed previously, the frequency of data transmission/reception on the data link is controlled by the MPSC clock in conjunction with the programmed clock divider (in register WR4). The 8274 is designed to permit all four serial interface lines (TxD and RxD for each channel) to operate at different data rates. Four clock input pins (TxC and RxC for each channel) are available for this function. Note that the clock rate divider specified in WR4 is used for both RxC and TxC on the appropriate channel; clock rate dividers for each channel are independent.

Modem Control

The following four modem interface signals may be connected to the 8274:

1. Data Terminal Ready (DTR). This interface signal (output by the 8274) is software controlled through bit 7 of WR5. When active, DTR indicates that the data terminal/computer equipment is active and ready to interact with the data communications channel. In addition, this signal prepares the modem for connection to the communication channel and maintains connections previously established (e.g., manual call origination).
2. Request To Send (RTS). This interface signal (output by the 8274) is software controlled through bit 1 of WR5. When active, RTS indicates that the data terminal/computer equipment is ready to transmit data.
3. Clear To Send (CTS). This interface signal (input to the 8274) is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit

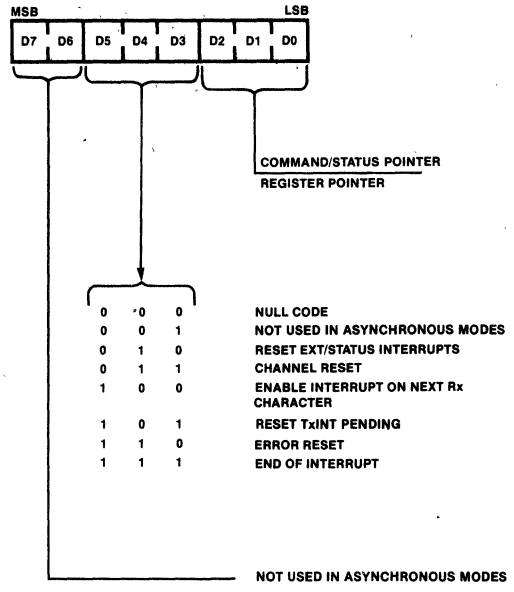
data. The state of CTS is available to the programmer as bit 5 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not transmit data bytes until RTS has been activated. If CTS becomes inactive during transmission of a character, the current character transmission is completed before the transmitter is disabled.

4. Carrier Detect (CD). This interface signal (input to the 8274) is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD line. The state of CD is available to the programmer as bit 3 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not enable the serial receiver until CD has been activated. If the CD signal becomes inactive during reception of a character, the receiver is disabled, and the partially received character is lost.

In addition to the above modem interface signals, the 8274 SYNDET input pin for channel A may be used as a general-purpose input in the asynchronous communication mode. The status of this signal is available to the programmer as bit 4 of status register RR0.

APPENDIX A
COMMAND/STATUS DETAILS FOR ASYNCHRONOUS COMMUNICATION

Write Register 0 (WR0):



D2,D1,D0 Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5,D4,D3 Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Not used in asynchronous modes.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and reenables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt-on-First-Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt Pending—if The Transmit Interrupt mode is selected, the MPSC automatically interrupts data when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

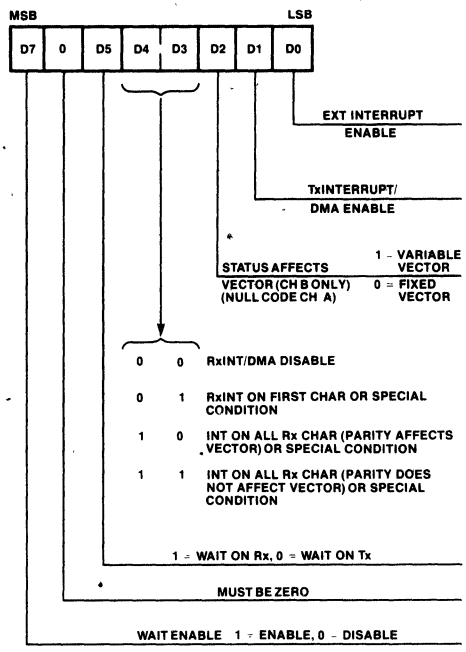
Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

D0 External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects Vector—(WR1, D2 active in channel B only.) If this bit is not set,

Write Register 1 (WR1):



then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set, then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4,D3 Receive Interrupt Mode.

0 0 Receive Interrupts/DMA Disabled.

0 1 Receive Interrupt on First Character Only or Special Condition.

1 0 Interrupt on All Receive Characters of Special Condition (Parity Error is a Special Receive Condition).

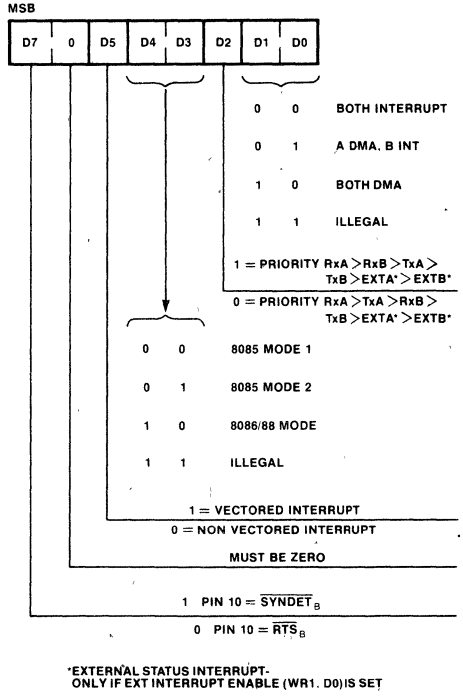
1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/$\overline{\text{Transmit}}$—when the following conditions are met, the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}=0$, A0=0/1, and A1=0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired *or* connected since only one signal is active at any one time while the other is in the High Z state.

D6 Must be Zero.

D7 Wait Enable—enables the wait function.

Write Register 2 (WR2): Channel A



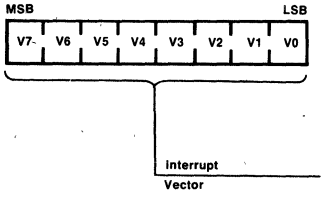
*EXTERNAL STATUS INTERRUPT ONLY IF EXT INTERRUPT ENABLE (WR1, D0) IS SET

D1,D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 Channel A and Channel B both use interrupts.

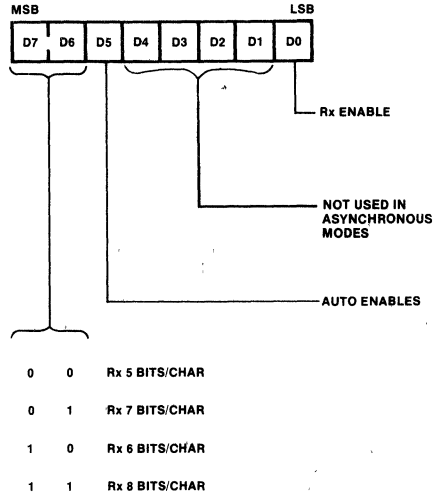
0 1	Channel A uses DMA, Channel Busses interrupt.
1 0	Channel A and Channel B both use DMA.
1 1	Illegal Code.
D2	Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.
0	(Highest) RxA, TxA, RxA, RxB, TxBExTA, ExTB (Lowest).
1	(Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).
D5,D4,D3	Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table).
0 X X	Non-vectored interrupts—intended for use with an external interrupt controller such as the 8259A.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy-chained priority structure.
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy-chained priority structure.
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy-chained priority structure.
D6	Must be Zero.
D7	
0	Pin 10 = $\overline{RTS}_B$.
1	Pin 10 = $\overline{SYNDET}_B$.

Write Register 2 (WR2): Channel B



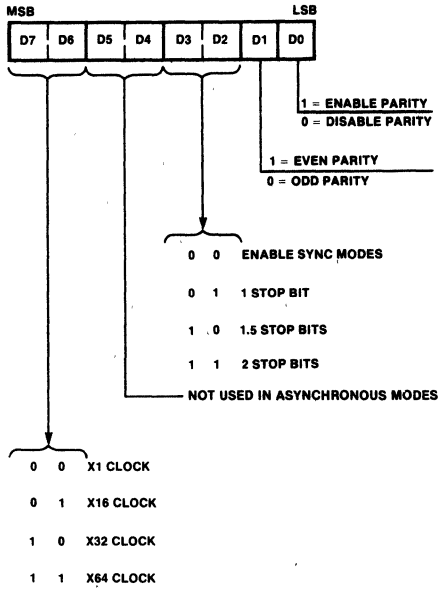
D7–D0	Interrupt vector—this register contains the value of the interrupt vector placed on the data bus during acknowledge sequences.

Write Register 3 (WR3):



D0	Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.
D5	Auto Enables—A one written to this bit causes $\overline{CD}$ to be an automatic enable signal for the receiver and $\overline{CTC}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).
D7,D6	Receiver Character length.
0 0	Receive 5 Data bits/character.
0 1	Receive 7 Data bits/character.
1 0	Receive 6 Data bits/character.
1 1	Receive 8 Data bits/character.

Write Register 4 (WR4):



D0 — Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

D1 — Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and zero causes it to send and expect odd parity.

D3,D2 — Stop Bits.

0 0 — Selects synchronous modes.

0 1 — Async mode, 1 stop bit/character.

1 0 — Async mode, 1½ stop bits/character.

1 1 — Async mode, 2 stop bits/character.

D7,D6 — Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. If the 1x mode is selected, bit synchronization must be done externally.

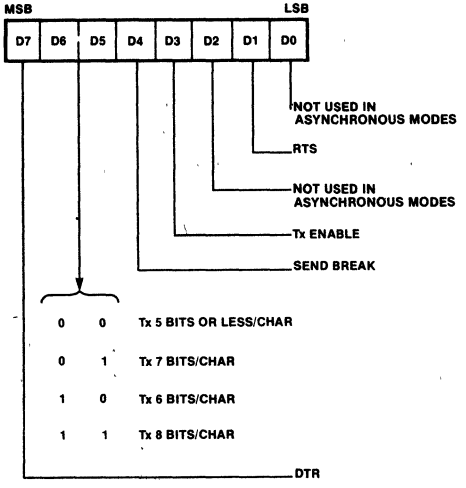
0 0 — Clock rate = Data rate x 1.

0 1 — Clock rate = Data rate x 16.

1 0 — Clock rate = Data rate x 32.

1 1 — Clock rate = Data rate x 64.

Write Register 5 (WR5):



D1 — Request to Send—a one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high).

D3 — Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

D4 — Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

D6,D5 — Transmit Character length.

0 0 — Transmit 5 or less bits/character.

0 1 — Transmit 7 bits/character.

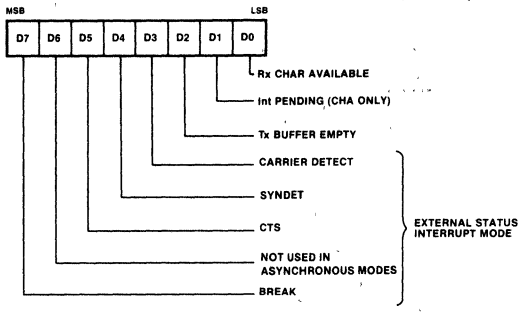
1 0 — Transmit 6 bits/character.

1 1 Transmit 8 bits/character.

Bits to be sent must be right justified, least-significant bit first, e.g.:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	B5	B4	B3	B2	B1	B0

Read Register 0 (RR0):



D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending—This Interrupt-Pending bit is reset when an E01 command is issued and there is no other interrupt request pending at that time. In vector mode, this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the CD pin at the time of the last change of any of the External/Status bits ($\overline{CD}$, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CD}$ pin causes the $\overline{CD}$ bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{CD}$ pin immediately following a Reset External/Status Interrupt command.

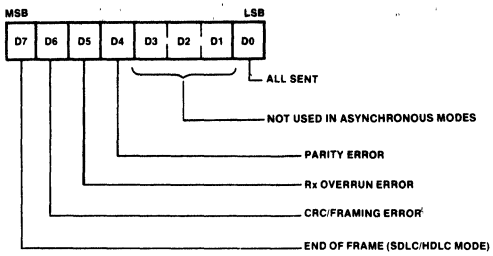
D4 SYNDET—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that it shows the state of the $\overline{SYNDET}$ input. Any High-to-Low transition on the $\overline{SYNDET}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{SYNDET}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNDET}$ input.

D5 Clear to Send—this bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, $\overline{CTS}$, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the $\overline{CTS}$ bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D7 Break—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

The Break bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single, extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

Read Register 1 (RR1)



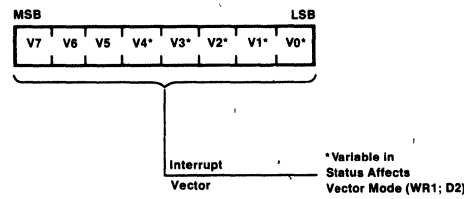
D0 All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

D4 Parity Error—if parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5 Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the "status affects vector" mode, the overrun causes a special Receive Error Vector.

D6 Framing Error—in async modes, a one in this bit indicates a receive framing error. It can be reset by issuing an Error Reset command.

Read Register 2 (RR2):



RR2 Channel B

D7–D0 Interrupt vector—contains the interrupt vector programmed into WR2. If the "status affects vector" mode is selected, it contains the modified vector. (See WR2.) RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

APPENDIX B
MPSC-POLLED TRANSMIT/RECEIVE CHARACTER ROUTINES

```
MPSC$RX$INIT: procedure (cmd$port,
                         clock$rate,stop$bits,parity$type,parity$enable,
                         rx$char$length,rx$enable,auto$enable,
                         tx$char$length,tx$enable,dtr,brk,rts);

  declare cmd$port       byte,
        clock$rate       byte,
        stop$bits        byte,
        parity$type      byte,
        parity$enable    byte,
        rx$char$length   byte,
        rx$enable        byte,
        auto$enable      byte,
        tx$char$length   byte,
        tx$enable        byte,
        dtr              byte,
        brk              byte,
        rts              byte;


  output(cmd$port)=30H;          /* channel reset */

  output(cmd$port)=14H;          /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                   or parity$enable;

  output(cmd$port)=13H;          /* point to WR3 */
  /* set up receiver parameters */
  output(cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(cmd$port)=15H;          /* point to WR5 */
  /* set up transmitter parameters */
  output(cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                   or shl(brk,4) or shl(rts,1);

  end MPSC$RX$INIT;
```

```
MPSC$POLL$RCV$CHARACTER: procedure(data$port,cmd$port,character$ptr) byte;

  declare data$port     byte,
          cmd$port      byte,
          character$ptr pointer,
          character     based character$ptr byte,
          status        byte;

  declare char$avail    literally '1',
          rcv$error     literally '70H';

  /* wait for input character ready */
  while (input(cmd$port) and char$avail) <> 0 do; end;

  /* check for errors in received character */
  output(cmd$port)=1;                          /* point to RR1 */
  if (status:=input(cmd$port) and rcv$error)
    then do;
      character=input(data$port);              /* read character to clear MPSC */
      call RECEIVE$ERROR(cmd$port,status);     /* clear receiver errors */
      return 0;                                /* error return - no character avail */
      end;
    else do;
      character=input(data$port);
      return 0FFH;                              /* good return - character avail */
      end;

end MPSC$POLL$RCV$CHARACTER;


MPSC$POLL$TRAN$CHARACTER: procedure(data$port,cmd$port,character);

  declare data$port      byte,
          cmd$port       byte,
          character      byte;

  declare tx$buffer$empty literally '4';

  /* wait for transmitter buffer empty */
  while not (input(cmd$port) and tx$buffer$empty) do; end;

  /* output character */
  output(data$port)=character;

end MPSC$POLL$TRAN$CHARACTER;


RECEIVE$ERROR: procedure(cmd$port,status);

  declare cmd$port      byte,
          status        byte;

  output(cmd$port)=30H;              /* error reset */

  /* *** other application dependent
         error processing should be placed here   *** */

end RECEIVE$ERROR;
```

```
TRANSMIT$BUFFER: procedure(buf$ptr,buf$length)

  declare
    buf$ptr       pointer,
    buf$length    byte;

  /* set up transmit buffer pointer and buffer length in global variables for
     interrupt service */
  tx$buffer$ptr=buf$ptr;
  transmit$length=buf$length;

  transmit$status=not$complete;              /* setup status for not complete */
  output(data$port)=transmit$buffer(0);      /* transmit first character */
  transmit$index=1;                          /* first character transmitted */

  /* wait until transmission complete or error detected */
  while transmit$status = not$complete do; end;
  if transmit$status <> complete
    then return false;
    else return true;

end TRANSMIT$BUFFER;


RECEIVE$BUFFER: procedure (buf$ptr,buf$length$ptr);

  declare
    buf$ptr         pointer,
    buf$length$ptr  pointer,
    buf$length      based buf$length$ptr byte;

  /* set up receive buffer pointer in global variable for interrupt service */
  rx$buffer$ptr=buf$ptr;
  receive$index=0;

  receive$status=not$complete;           /* set status to not complete */
  /* wait until buffer received */
  while receive$status = not$complete do; end;
  buf$length=receive$length;
  if receive$status = complete
    then return true;
    else return false;

end RECEIVE$BUFFER;
```

```
MPSC$RECEIVE$CHARACTER$INT: procedure interrupt 22H;

  /* ignore input if no open buffer */
  if receive$status <> not$complete then return;

  /* check for receive buffer overrun */
  if receive$index = 128
    then receive$status=overrun;
    else do;
      /* read character from MPSC and place in buffer - note that the
         parity of the character must be masked off during this step if
         the character is less than 8 bits (e.g., ASCII)  */
      receive$buffer(receive$index),character=input(data$port) and 7FH;
      receive$index=receive$index+1;      /* update receive buffer index */

      /* check for line feed to end line */
      if character = line$feed
        then do; receive$length=receive$index; receive$status=complete; end;
    end;

end MPSC$RECEIVE$CHARACTER$INT;


MPSC$TRANSMIT$CHARACTER$INT: procedure interrupt 20H;

  /* check for more characters to transfer */
  if transmit$index < transmit$length
    then do;
      /* write next character from buffer to MPSC */
      output(data$port)=transmit$buffer(transmit$index);
      transmit$index=transmit$index+1;     /* update transmit buffer index */
    end;
    else transmit$status=complete;

end MPSC$TRANSMIT$CHARACTER$INT;


RECEIVE$ERROR$INT: procedure interrupt 23H;

  declare
    temp                  byte;    /* temporary character storage */

  output(cmd$port)=1;              /* point to RR1 */
  receive$status=input(cmd$port);
  temp=input(data$port);           /* discard character */
  output(cmd$port)=error$reset;    /* send error reset */

  /* *** other application dependent
         error processing should be placed here   *** */

end RECEIVE$ERROR$INT;


EXTERNAL$STATUS$CHANGE$INT: procedure interrupt 21H;

  transmit$status=input(cmd$port)      /* input status change information */
  output(cmd$port)=reset$ext$status;

  /* *** other application dependent
         error processing should be placed here   *** */

end EXTERNAL$STATUS$CHANGE$INT;
```

APPENDIX C
INTERRUPT-DRIVEN TRANSMIT/RECEIVE SOFTWARE

```
declare
  /* global variables for buffer manipulation */

  rx$buffer$ptr          pointer,           /* pointer to receive buffer */
  receive$buffer based rx$buffer$ptr(128) byte,
  receive$status         byte initial(0),   /* indicates receive buffer status */
  receive$index          byte,              /* current index into receive buffer */
  receive$length         byte,              /* length of final receive buffer */

  tx$buffer$ptr          pointer,           /* pointer to transmit buffer */
  transmit$buffer based tx$buffer$ptr(128) byte,
  transmit$status        byte initial(0),   /* indicates transmit buffer status */
  transmit$index         byte,              /* current index into transmit buffer */
  transmit$length        byte,              /* length of buffer to be transmitted */

  cmd$port               literally '43H',
  data$port              literally '41H',
  a$cmd$port             literally '42H',
  b$cmd$port             literally '43H',
  line$feed              literally '0AH',
  not$complete           literally '0',
  complete               literally '0FFH',
  overrun                literally '1',

  channel$reset          literally '18H',
  error$reset            literally '30H',
  reset$ext$status       literally '10H';
```

```
MPSC$INT$INIT: procedure (clock$rate,stop$bits,parity$type,parity$enable,
                          rx$char$length,rx$enable,auto$enable,
                          tx$char$length,tx$enable,dtr,brk,rts,
                          ext$en,tx$en,rx$en,stat$affects$vector,
                          config,priority,vector$int$mode,int$vector);

  declare
        clock$rate        byte,         /* 2-bit code for clock rate divisor */
        stop$bits         byte,         /* 2-bit code for number of stop bits */
        parity$type       byte,         /* 1-bit parity type */
        parity$enable     byte,         /* 1-bit parity enable */
        rx$char$length    byte,         /* 2-bit receive character length */
        rx$enable         byte,         /* 1-bit receiver enable */
        auto$enable       byte,         /* 1-bit auto enable flag */
        tx$char$length    byte,         /* 2-bit transmit character length */
        tx$enable         byte,         /* 1-bit transmitter enable */
        dtr               byte,         /* 1-bit status of DTR pin */
        brk               byte,         /* 1-bit data link break enable */
        rts               byte,         /* 1-bit status of RTS pin */
        ext$en            byte,         /* 1-bit external/status enable */
        tx$en             byte,         /* 1-bit Tx interrupt enable */
        rx$en             byte,         /* 2-bit Rx interrupt enable/mode */
        stat$aff$vector   byte,         /* 1-bit status affects vector flag */
        config            byte,         /* 2-bit system config - int/DMA */
        priority          byte,         /* 1-bit priority flag */
        vector$int$mode   byte,         /* 3-bit interrupt mode code */
        int$vector        byte;         /* 8-bit interrupt type code */


  output(b$cmd$port)=channel$reset;   /* channel reset */

  output(b$cmd$port)=14H;             /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(b$cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                     or parity$enable;

  output(b$cmd$port)=13H;             /* point to WR3 */
  /* set up receiver parameters */
  output(b$cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(b$cmd$port)=15H;          /* point to WR5 */
  /* set up transmitter parameters */
  output(b$cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                    or shl(brk,4) or shl(rts,1);

  output(b$cmd$port)=12H;             /* point to WR2 */
  /* set up interrupt vector */
  output(b$cmd$port)=int$vector;

  output(a$cmd$port)=12H;             /* point to WR2, channel A */
  /* set up interrupt modes */
  output(a$cmd$port)=shl(vector$int$mode,3) or shl(priority,2) or config;

  output(b$cmd$port)=11H;             /* point to WR1 */
  /* set up interrupt enables */
  output(b$cmd$port)=shl(rx$en,3) or shl(stat$aff$vector,2) or shl(tx$en,1)
                     or ext$en;

  end MPSC$INT$INIT;
```

APPENDIX D
APPLICATION EXAMPLE USING SDK-86

This application example shows the 8274 in a simple iAPX-86/88 system. The 8274 controls two separate asynchronous channels using its internal interrupt controller to request all data transfers. The 8274 driver software is described which transmits and receives data buffers provided by the CPU. Also, status registers are maintained in system memory to allow the CPU to monitor progress of the buffers and error conditions.

THE HARDWARE INTERFACE

Nothing could be easier than the hardware design of an interrupt-driven 8274 system. Simply connect the data bus lines, a few bus control lines, supply a timing clock for baud rate and, voila, it's done! For this example, the ubiquitous SDK-86 is used as the host CPU system. The 8274 interface is constructed on the wire-wrap area provided. While discussing the hardware interface, please refer to Diagram 1.

Placing the 8274 on the lower 8 bits of the 8086 data bus allows byte-wide data transfers at even I/O addresses. For simplicity, the 8274's CS/ input is generated by combining the M-IO/ select line with address line A7 via a 7432. This places the 8274 address range in multiple spots within the 8086 I/O address space. (While fine for this example, a more complete address decoding is recommended for actual prototype systems.) The 8086's A1 and A2 address lines are connected to the A0 and A1 8274 register select inputs respectively. Although other port assignments are possible because of the overlapping address spaces, the following I/O port assignments are used in this example:

Port Function	I/O Address
Data channel A	0000H
Command/status A	0002H
Data channel B	0004H
Command/status B	0006H

To connect the 8274's interrupt controller into the system an inverter and pull-up resistor are needed to convert the 8274's active-low, interrupt-request output, IRQ, into the correct polarity for the 8086's INTR interrupt input. The 8274 recognizes interrupt-acknowledge bus cycles by connecting the INTA (INTerrupt Acknowledge) lines of the 8274 and 8086 together.

The 8274 ReaD and WRite lines directly connect to the respective 8086 lines. The RESET line requires an inverter. The system clock for the 8274 is provided by the PCLK (peripheral clock) output of the 8284A clock generator.

On the 8274's serial side, traditional 1488 and 1489 RS-232 drivers and receivers are used for the serial interface. The onboard baud rate generator supplies the channel baud rate timing. In this example, both sides of both channels operate at the same baud rate although this certainly is not a requirement. (On the SDK-86, the baud rate selection is hard-wired thru jumpers. A more flexible approach would be to incorporate an 8253 Programmable Interval Timer to allow software-configurable baud rate selection.)

That's all there is to it. This hardware interface is completely general-purpose and supports all of the 8274 features except the DMA data transfer mode which requires an external DMA controller. Now let's look at the software interface.

SOFTWARE INTERFACE

In this example, it is assumed that the 8086 has better things to do rather than continuously run a serial channel. Presenting the software as a group of callable procedures lets the designer include them in the main body of another program. The interrupt-driven data transfers give the effect that the serial channels are handled in the background while the main program is executing in the foreground. There are five basic procedures: a serial channel initialization routine and buffer handling routines for the transmit and receive data buffers of each channel. Appendix D-1 shows the entire software listing. Listing line numbers are referenced as each major routing is discussed.

The channel initialization routine (INITIAL 8274), starting with line #203, simply sets each channel into a particular operating mode by loading the command registers of the 8274. In normal operation, once these registers are loaded, they are rarely changed. (Although this example assumes a simple asynchronous operating mode, the concept is easily extended for the byte- and bit-synchronous modes.)

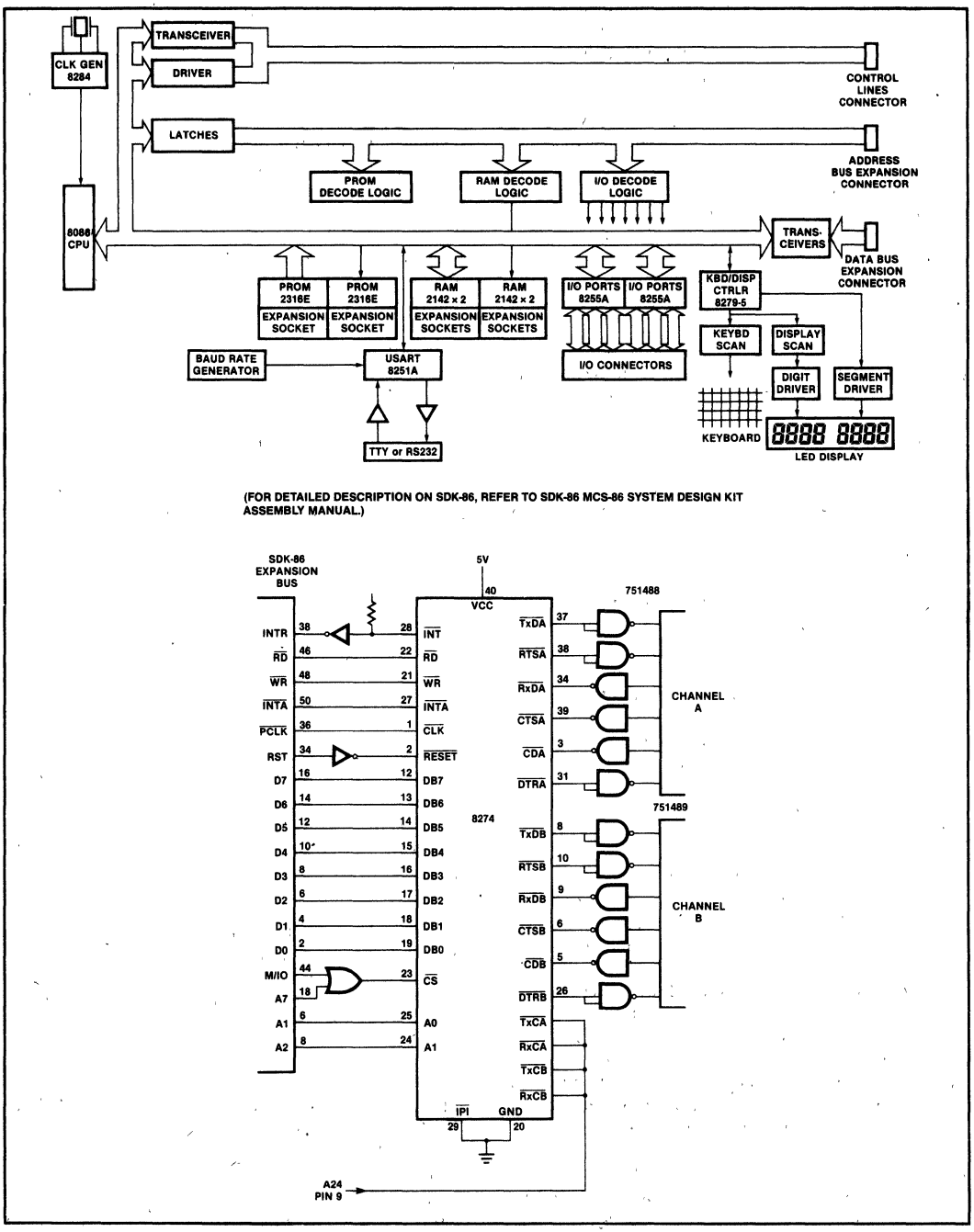


Figure D-1. 8274/SDK-86 Hardware Interface

The channel operating modes are contained in two tables starting with line #163. As the 8274 has only one command register per channel, the remaining seven registers are loaded indirectly through the WR0 (Write Register 0) register. The first byte of each table entry is the register pointer value which is loaded into WR0 and the second byte is the value for that particular register.

The indicated modes set the 8274 for asynchronous operation with data characters 8 bits long, no parity, and 2 stop bits. An X16 baud rate clock is assumed. Also selected is the "interrupt on all RX character" mode with a variable interrupt vector compatible with the 8086/8088. The transmitters are enabled and all model control lines are put in their active state.

In addition to initializing the 8274, this routine also sets up the appropriate interrupt vectors. The 8086 assumes the first 1K bytes of memory contain up to 256 separate interrupt vectors. On the SDK-86 the initial 2K bytes of memory is RAM and therefore must be initialized with the appropriate vectors. (In a prototype system, this initial memory is probably ROM, thus the vector set-up is not needed.) The 8274 supplies up to eight different interrupt vectors. These vectors are developed from internal conditions such as data requests, status changes, or error conditions for each channel. The initialization routine arbitrarily assumes that the initial 8274 vector corresponds to 8086 vector location 80H (memory location 200H). This choice is arbitrary since the 8274 initial vector location is programmable.

Finally, the initialization routine sets up the status and flag in RAM. The meaning and use of these locations are discussed later.

Following the initialization routine are those for the transmit commands (starting with line #268). These commands assume that the host CPU has initialized the publically declared variables for the transmit buffer pointer, TX_POINTER_CHx, and the buffer length, TX_LENGTH_CHx. The transmit command routines simply clear the transmitter empty flag, TX EMPTY CHx, and load the first character of the buffer into the transmitter. It is necessary to load the first character in this manner since transmitter interrupts are generated only when the 8274's transmit data buffer becomes empty. It is the act of becoming empty which generates the interrupt not simply the buffer being empty, thus the transmitter needs one character to start.

The host CPU can monitor the transmitter empty flag, TX_EMPTY_CHx, in order to determine when transmission of the buffer is complete. Obviously, the CPU should only call the command routine after first checking that the empty flag is set.

After returning to the main program, all transmitter data transfers are handled via the transmitter-interrupt service routines starting at lines #360 and #443. These routines start by issuing an End-Of-Interrupt command to the 8274. (This command resets the internal-interrupt controller logic of the 8274 for this particular vector and opens the logic for other internal interrupt requests. The routines next check the length count. If the buffer is completely transmitted, the transmitter empty flag, TX_EMPTY_CHx, is set and a command is issued to the 8274 to reset its interrupt line. Assuming that the buffer is not completely transmitted, the next character is output to the transmitter. In either case, an interrupt return is executed to return to the main CPU program.

The receiver commands start at line #314. Like the transmit commands, it is assumed that the CPU has initialized the receive-buffer-pointer public variable, RX_POINTER_CHx. This variable points to the first location in an empty receive buffer. The command routines clear the receiver ready flag, RX_READY-_CHx, and then set the receiver enable bit in the 8274 WR3 register. With the receiver now enabled, any received characters are placed in the receive buffer using interrupt-driven data transfers.

The received data service routines, starting at lines #402 and #485, simply place the received character in the buffer after first issuing the EOI command. The character is then compared to an ASCII CR. An ASCII CR causes the routine to set the receiver ready flag, RX_READY_CHx, and to disable the receiver. The CPU can interrogate this flag to determine when the buffer contains a new line of data. The receive buffer pointer, RX_POINTER_CHx, points to the last received character and the receive counter, RX-_COUNTER_CHx, contains the length.

That completes our discussion of the command routines and their associated interrupt service routines. Although not used by the commands, two additional service routines are included for completeness. These routines handle the error and status-change interrupt vectors.

The error service routines, starting at lines #427 and #510, are vectored to if a special receive condition is detected by the 8274. These special receive conditions include parity, receiver overrun, and framing errors. When this vector is generated, the error condition is indicated in RR1 (Read Register 1). The error service routine issues an EO1 command, reads RR1 and places it in the ERROR_MSG_CHx variable, and then issues

a reset error command to the 8274. The CPU can monitor the error message location to detect error conditions. The designer, of course, can supply his own error service routine.

Similarily, the status-change routines (starting lines #386 and #469) are initiated by a change in the modem-control status lines CTS/, CD/, or SYNDET/. (Note that WR2 bit 0 controls whether the 8274 generates interrupts based upon changes in these lines. Our WR2 parameter is such that the 8274 is programmed to ignore changes for these inputs.) The service routines simply read RR0, place its contents in the STATUS_MSG-_CHx variable and then issue a reset external status command. Read Register 0 contains the state of the modem inputs at the point of the last change.

Well, that's it. This application example has presented useful, albeit very simple, routines showing how the 8274 might be used to transmit and receive buffers using an asynchronous serial format. Extensions for byte- or bit-synchronous formats would require no hardware changes due to the highly programmable nature of the 8274's serial formats.

8274 APPLICATION BRIEF PROGRAM

```
MCS-86 MACRO ASSEMBLER    ASYNCB


ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ASYNCB
OBJECT MODULE PLACED IN :F1:ASYNCB.OBJ
ASSEMBLER INVOKED BY.  ASM86 .F1.ASYNCB.SRC


LOC  OBJ                  LINE     SOURCE

                             1     ;*************************************************************
                             2     ;*                                                           *
                             3     ;*              8274 APPLICATION BRIEF PROGRAM               *
                             4     ;*                                                           *
                             5     ;*                                                           *
                             6     ;*                                                           *
                             7     ;* THE 8274 IS INITIALIZED FOR SIMPLE ASYNCHRONOUS SERIAL    *
                             8     ;* FORMAT AND VECTORED INTERRUPT-DRIVEN DATA TRANSFERS.      *
                             9     ;* THE INITIALIZATION ROUTINE ALSO LOADS THE 8086'S INTERRUPT *
                            10     ;* VECTOR TABLE FROM THE CODE SEGMENT INTO LOW RAM ON THE    *
                            11     ;* SDK-86.  THE TRANSMITTER AND RECEIVER ARE LEFT ENABLED.   *
                            12     ;*                                                           *
                            13     ;* FOR TRANSMIT, THE CPU PASSES IN MEMORY THE POINTER OF A   *
                            14     ;* BUFFER TO TRANSMIT AND THE BYTE LENGTH OF THE BUFFER.     *
                            15     ;* THE DATA TRANSFER PROCEED USING INTERRUPT-DRIVEN TRANSFERS.*
                            16     ;* A STATUS BIT IN MEMORY IS SET WHEN IF BUFFERS IS EMPTY.   *
                            17     ;*                                                           *
                            18     ;* FOR RECEIVE, THE CPU PASSES THE POINTER OF A BUFFER TO FILL.*
                            19     ;* THE BUFFER IS FILLED UNTIL A 'CR_CHR' CHARACTER IS RECEIVED.*
                            20     ;* A STATUS BIT IS SET AND THE CPU MAY READ THE RX POINTER TO *
                            21     ;* DETERMINE THE LOCATION OF THE LAST CHARACTER.             *
                            22     ;*                                                           *
                            23     ;* ALL ROUTINES ARE ASSUMED TO EXIST IN THE SAME CODE SEGMENT.*
                            24     ;* CALL'S TO THE SERVICE ROUTINES ARE ASSUMED TO BE "SHORT" OR*
                            25     ;* INTRASEGMENT (ONLY THE RETURN ADDRESS IP IS ON THE STACK).*
                            26     ;*                                                           *
                            27     ;*                                                           *
                            28     ;*                                                           *
                            29     ;*                                                           *
                            30     ;*************************************************************
```

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ                 LINE    SOURCE

                           31
                           32            NAME    ASYNCB  ;MODULE NAME
                           33
                           34    ;PUBLIC DECLARATIONS FOR COMMAND ROUTINES
                           35
                           36            PUBLIC  INITIAL_8274    ;INITIALIZATION ROUTINE
                           37            PUBLIC  TX_COMMAND_CHB  ;TX BUFFER COMMAND CHANNEL B
                           38            PUBLIC  TX_COMMAND_CHA  ;TX BUFFER COMMAND CHANNEL A
                           39            PUBLIC  RX_COMMAND_CHB  ;RX BUFFER COMMAND CHANNEL B
                           40            PUBLIC  RX_COMMAND_CHA  ;RX BUFFER COMMAND CHANNEL A
                           41
                           42    ;PUBLIC DECLARATIONS FOR STATUS VARIABLES
                           43
                           44            PUBLIC  RX_READY_CHB    ;RX READY FLAG CHB
                           45            PUBLIC  RX_READY_CHA    ;RX READY FLAG CHA
                           46            PUBLIC  TX_EMPTY_CHB    ;TX EMPTY FLAG CHB
                           47            PUBLIC  TX_EMPTY_CHA    ;TX EMPTY FLAG CHA
                           48            PUBLIC  RX_COUNT_CHB    ;RX BUFFER COUNTER CHB
                           49            PUBLIC  RX_COUNT_CHA    ;RX BUFFER COUNTER CHA
                           50            PUBLIC  ERROR_MSG_CHB   ;ERROR FLAG CHB
                           51            PUBLIC  ERROR_MSG_CHA   ;ERROR FLAG CHA
                           52            PUBLIC  STATUS_MSG_CHB  ;STATUS FLAG CHB
                           53            PUBLIC  STATUS_MSG_CHA  ;STATUS FLAG CHA
                           54
                           55    ;PUBLIC DECLARATIONS FOR VARIABLES PASSED TO THE TRANSMIT
                           56    ;AND RECEIVE COMMANDS.
                           57
                           58            PUBLIC  TX_POINTER_CHB  ;TX BUFFER POINTER FOR CHB
                           59            PUBLIC  TX_LENGTH_CHB   ;TX LENGTH OF BUFFER FOR CHB
                           60            PUBLIC  TX_POINTER_CHA  ;TX BUFFER POINTER FOR CHA
                           61            PUBLIC  TX_LENGTH_CHA   ;TX LENGTH OF BUFFER FOR CHA
                           62            PUBLIC  RX_POINTER_CHB  ;RX BUFFER POINTER FOR CHB
                           63            PUBLIC  RX_POINTER_CHA  ;RX BUFFER POINTER FOR CHA
                           64
                           65    ;I/O PORT ASSIGNMENTS
                           66
                           67    ;CHANNEL A PORT ASSIGNMENTS
                           68
0000                       69    DATA_PORT_CHA           EQU     0                       ;DATA I/O PORT
0002                       70    COMMAND_PORT_CHA        EQU     2                       ;COMMAND PORT
0002                       71    STATUS_PORT_CHA         EQU     COMMAND_PORT_CHA        ;STATUS PORT
                           72
                           73    ;CHANNEL B PORT ASSIGNMENTS
                           74
0004                       75    DATA_PORT_CHB           EQU     4                       ;DATA I/O PORT
0006                       76    COMMAND_PORT_CHB        EQU     6                       ;COMMAND PORT
0006                       77    STATUS_PORT_CHB         EQU     COMMAND_PORT_CHB        ;STATUS PORT
                           78
                           79    ;MISC  SYSTEM EQUATES
                           80
000D                       81    CR_CHR          EQU     0DH     ;ASCII CR CHARACTER CODE
0200                       82    INT_TABLE_BASE  EQU     200H    ;INT  VECTOR BASE ADDRESS
0500                       83    CODE_START      EQU     500H    ;START LOCATION FOR CODE
                           84
                           85 +1 $EJECT
                           86
                           87    ;RAM ASSSIGNMENTS FOR DATA SEGMENT
                           88
----                       89            DATA    SEGMENT
                           90
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                  LINE     SOURCE

                            91     ;VECTOR INTERRUPT TABLE - ASSUME INITIAL 8274 INTERRUPT
                            92     ;VECTOR IS NUMBER 80 (0200H).  FOR EACH VECTOR, THE TABLE
                            93     ;CONTAINS START LOCATION AND CODE SEGMENT REGISTER VALUE.
                            94     ;THE TABLE IS LOADED FROM PROM
                            95
0200                        96             ORG     INT_TABLE_BASE
                            97
0200 0000                   98     TX_VECTOR_CHB   DW      0       ;TX INTERRUPT VECTOR FOR CHB
0202 0000                   99     TX_CS_CHB       DW      0
                           100
0204 0000                  101     STS_VECTOR_CHB  DW      0       ;STATUS INTERRUPT VECTOR FOR CHB
0206 0000                  102     STS_CS_CHB      DW      0
                           103
0208 0000                  104     RX_VECTOR_CHB   DW      0       ;RX INTERRUPT VECTOR FOR CHB
020A 0000                  105     RX_CS_CHB       DW      0
                           106
020C 0000                  107     ERR_VECTOR_CHB  DW      0       ;ERROR INTERRUPT VECTOR FOR CHB
020E 0000                  108     ERR_CS_CHB      DW      0
                           109
0210 0000                  110     TX_VECTOR_CHA   DW      0       ;TX INTERRUPT VECTOR FOR CHA
0212 0000                  111     TX_CS_CHA       DW      0
                           112
0214 0000                  113     STS_VECTOR_CHA  DW      0       ;STATUS INTERRUPT VECTOR FOR CHA
0216 0000                  114     STS_CS_CHA      DW      0
                           115
0218 0000                  116     RX_VECTOR_CHA   DW      0       ;RX INTERRUPT VECTOR FOR CHA
021A 0000                  117     RX_CS_CHA       DW      0
                           118
021C 0000                  119     ERR_VECTOR_CHA  DW      0       ;ERROR INTERRUPT VECTOR FOR CHA
021E 0000                  120     ERR_CS_CHA      DW      0
                           121
                           122     ;MISC RAM LOCATIONS FOR CHANNEL STATUS AND POINTERS
                           123
                           124     ;CHANNEL B POINTERS AND STATUS
                           125
0220 0000                  126     TX_POINTER_CHB  DW      0       ;TX BUFFER POINTER FOR CHB
0222 0000                  127     TX_LENGTH_CHB   DW      0       ;TX BUFFER LENGTH FOR CHB
0224 0000                  128     RX_POINTER_CHB  DW      0       ;RX BUFFER POINTER FOR CHB
0226 0000                  129     RX_COUNT_CHB    DW      0       ;RX LENGTH COUNTER FOR CHB
0228 00                    130     TX_EMPTY_CHB    DB      0       ;TX DONE FLAG
0229 00                    131     RX_READY_CHB    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
022A 00                    132     STATUS_MSG_CHB  DB      0       ;STATUS CHANGE MESSAGE
022B 00                    133     ERROR_MSG_CHB   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)
                           134
                           135     ;CHANNEL A POINTERS AND STATUS
                           136
022C 0000                  137     TX_POINTER_CHA  DW      0       ;TX BUFFER POINTER FOR CHA
022E 0000                  138     TX_LENGTH_CHA   DW      0       ;TX BUFFER LENGTH FOR CHA
0230 0000                  139     RX_POINTER_CHA  DW      0       ;RX BUFFER POINTER FOR CHA
0232 0000                  140     RX_COUNT_CHA    DW      0       ;RX LENGTH COUNTER FOR CHA
0234 00                    141     TX_EMPTY_CHA    DB      0       ;TX DONE FLAG
0235 00                    142     RX_READY_CHA    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
0236 00                    143     STATUS_MSG_CHA  DB      0       ;STATUS CHANGE MESSAGE
0237 00                    144     ERROR_MSG_CHA   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)
                           145
----                       146             DATA    ENDS
                           147
                           148 +1  $EJECT
```

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ                 LINE     SOURCE

                         149
----                     150            ABC     SEGMENT
                         151            ASSUME  CS:ABC,DS:DATA,SS:DATA
0500                     152            ORG     CODE_START
                         153
                         154    ;*************************************************************
                         155    ;*                                                           *
                         156    ;*          PARAMETERS FOR CHANNEL INITIALIZATION            *
                         157    ;*                                                           *
                         158    ;*************************************************************
                         159
                         160    ;CHANNEL B PARAMETERS
                         161
                         162            ;WR1 - INTERRUPT ON ALL RX CHR, VARIABLE INT VECTOR, TX INT ENABLE
0500 01                  163    CMDSTRB DB      1,16H
0501 16
                         164            ;WR2 - INTERRUPT VECTOR
0502 02                  165            DB      2,(INT_TABLE_BASE/4)
0503 80
                         166            ;WR3 - RX 8 BITS/CHR, RX DISABLE
0504 03                  167            DB      3,0C0H
0505 C0
                         168            ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0506 04                  169            DB      4,4CH
0507 4C
                         170            ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0508 05                  171            DB      5,0EAH
0509 EA
                         172            ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
050A 00                  173            DB      0,0
050B 00
                         174
                         175    ;CHANNEL A PARAMETERS
                         176
                         177            ;WR1 - INTERRUPT ON ALL RX CHR, TX INT ENABLE
050C 01                  178    CMDSTRA DB      1,12H
050D 12
                         179            ;WR2 - VECTORED INTERRUPT FOR 8086
050E 02                  180            DB      2,30H
050F 30
                         181            ;WR3 - RX 8 BITS/CHR, RX DISABLE
0510 03                  182            DB      3,0C0H
0511 C0
                         183            ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0512 04                  184            DB      4,4CH
0513 4C
                         185            ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0514 05                  186            DB      5,0EAH
0515 EA
                         187            ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
0516 00                  188            DB      0,0
0517 00
                         189
                         190 +1 $EJECT
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                  LINE    SOURCE

                          191
                          192     ;START OF COMMAND ROUTINES
                          193
                          194     ;*************************************************************
                          195     ;*                                                           *
                          196     ;*      INITIALIZATION COMMAND FOR THE 8274 - THE 8274       *
                          197     ;*      IS SETUP ACCORDING TO THE PARAMETERS STORED IN       *
                          198     ;*      PROM ABOVE STARTING AT CMSTRB FOR CHANNEL B AND      *
                          199     ;*      CMSTRA FOR CHANNEL A                                 *
                          200     ;*                                                           *
                          201     ;*************************************************************
                          202
0518                      203     INITIAL_8274
                          204             ;COPY INTERRUPT VECTOR IP AND CS VALUES FROM PROM TO RAM
0518 C70600020806         205             MOV     TX_VECTOR_CHB, OFFSET XMTINB    ;TX DATA VECTOR CHB
051E 8C0E0202             206             MOV     TX_CS_CHB, CS
0522 C70604023506         207             MOV     STS_VECTOR_CHB, OFFSET STAINB   ;STATUS VECTOR CHB
0528 8C0E0602             208             MOV     STS_CS_CHB, CS
052C C70608024906         209             MOV     RX_VECTOR_CHB, OFFSET RCVINB    ;RX DATA VECTOR CHB
0532 8C0E0A02             210             MOV     RX_CS_CHB, CS
0536 C7060C027706         211             MOV     ERR_VECTOR_CHB, OFFSET ERRINB   ;ERROR VECTOR CHB
053C 8C0E0A02             212             MOV     RX_CS_CHB, CS
0540 C70610028C06         213             MOV     TX_VECTOR_CHA, OFFSET XMTINA    ;TX DATA VECTOR CHA
0546 8C0E1202             214             MOV     TX_CS_CHA, CS
054A C706140289O6         215             MOV     STS_VECTOR_CHA, OFFSET STAINA   ;STATUS VECTOR CHA
0550 8C0E1602             216             MOV     STS_CS_CHA, CS
0554 C706182CD06          217             MOV     RX_VECTOR_CHA, OFFSET RCVINA    ;RX DATA VECTOR CHA
055A 8C0E1A02             218             MOV     RX_CS_CHA, CS
055E C7061C02FB06         219             MOV     ERR_VECTOR_CHA, OFFSET ERRINA   ;ERROR VECTOR CHA
0564 8C0E1E02             220             MOV     ERR_CS_CHA, CS
                          221
                          222     ;COPY SETUP TABLE PARAMETERS INTO 8274
                          223
0568 BF0005               224             MOV     DI, OFFSET CMDSTRB      ;INITIALIZE CHB
056B BA0600               225             MOV     DX, COMMAND_PORT_CHB
056E E82E00               226             CALL    SETUP                   ;COPY CHB PARAMETERS
0571 BF0C05               227             MOV     DI, OFFSET CMDSTRA      ;INITIALIZE CHA
0574 BA0200               228             MOV     DX, COMMAND_PORT_CHA
0577 E82500               229             CALL    SETUP                   ;COPY CHA PARAMETERS
                          230
                          231     ;INITIALIZE STATUS BYTES AND FLAGS
                          232
057A B80000               233             MOV     AX, 0
057D A22B02               234             MOV     ERROR_MSG_CHB, AL       ;CLEAR ERROR FLAG CHB
0580 A23702               235             MOV     ERROR_MSG_CHA, AL       ;CLEAR ERROR FLAG CHA
0583 A22A02               236             MOV     STATUS_MSG_CHB, AL      ;CLEAR STATUS FLAG CHB
0586 A23602               237             MOV     STATUS_MSG_CHA, AL      ;CLEAR STATUS FLAG CHA
0589 A32602               238             MOV     RX_COUNT_CHB, AX        ;CLEAR RX COUNTER CHB
058C A33202               239             MOV     RX_COUNT_CHA, AX        ;CLEAR RX COUNTER CHA
058F B001                 240             MOV     AL, 1
0591 A22902               241             MOV     RX_READY_CHB, AL        ;SET RX DONE FLAG CHB
0594 A23502               242             MOV     RX_READY_CHA, AL        ;SET RX DONE FLAG CHA
0597 A22802               243             MOV     TX_EMPTY_CHB, AL        ;SET TX DONE FLAG CHB
059A A23402               244             MOV     TX_EMPTY_CHA, AL        ;SET TX DONE FLAG CHA
059D FB                   245             STI                             ;ENABLE INTERRUPTS
059E C3                   246             RET                             ;RETURN - DONE WITH SETUP
                          247
059F 8A05                 248     SETUP:  MOV     AL, [DI]                ;PARAMETER COPYING ROUTINE
05A1 3C00                 249             CMP     AL, 0
05A3 7404                 250             JE      DONE
```

```
LOC  OBJ                  LINE    SOURCE

05A5 EE                    251            OUT     DX, AL                  ;OUTPUT PARAMETER
05A6 47                    252            INC     DI                      ;POINT AT NEXT PARAMETER
05A7 EBF6                  253            JMP     SETUP                   ;GO LOAD IT
05A9 C3                    254    DONE:   RET                             ;DONE - SO RETURN
                           255
                           256 +1 $EJECT
                           257
                           258    ;*************************************************************
                           259    ;*                                                           *
                           260    ;*    TX CHANNEL B COMMAND ROUTINE - ROUTINE IS CALLED TO    *
                           261    ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,       *
                           262    ;*    TX_POINTER_CHB, AND THE BUFFER LENGTH, TX_LENGTH_CHB,  *
                           263    ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.            *
                           264    ;*    BOTH ITEMS ARE WORD VARIABLES.                         *
                           265    ;*                                                           *
                           266    ;*************************************************************
                           267
05AA                       268    TX_COMMAND_CHB:
05AA 50                    269            PUSH    AX              ;SAVE REGISTERS
05AB 57                    270            PUSH    DI
05AC 52                    271            PUSH    DX
05AD C606280200            272            MOV     TX_EMPTY_CHB, 0 ;CLEAR EMPTY FLAG
05B2 BA0400                273            MOV     DX, DATA_PORT_CHB       ;SETUP PORT POINTER
05B5 8B3E2002              274            MOV     DI, TX_POINTER_CHB      ;GET TX BUFFER POINTER CHB
05B9 8A05                  275            MOV     AL, [DI]        ;GET FIRST CHARACTER TO TX
05BB EE                    276            OUT     DX, AL          ;OUTPUT IT TO 8274 TO GET IT STARTED
05BC 5A                    277            POP     DX
05BD 5F                    278            POP     DI
05BE 58                    279            POP     AX
05BF C3                    280            RET                     ;RETURN
                           281
                           282    ;*************************************************************
                           283    ;*                                                           *
                           284    ;*    TX CHANNEL A COMMAND ROUTINE - ROUTINE IS CALLED TO    *
                           285    ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,       *
                           286    ;*    TX_POINTER_CHA, AND THE BUFFER LENGTH, TX_LENGTH_CHA,  *
                           287    ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.            *
                           288    ;*    BOTH ITEMS ARE WORD VARIABLES.                         *
                           289    ;*                                                           *
                           290    ;*************************************************************
                           291
05C0                       292    TX_COMMAND_CHA:
05C0 50                    293            PUSH    AX              ;SAVE REGISTERS
05C1 57                    294            PUSH    DI
05C2 52                    295            PUSH    DX
05C3 C606340200            296            MOV     TX_EMPTY_CHA, 0 ;CLEAR EMPTY FLAG
05C8 BA0000                297            MOV     DX, DATA_PORT_CHA       ;SETUP PORT POINTER
05CB 8B3E2C02              298            MOV     DI, TX_POINTER_CHA      ;GET TX BUFFER POINTER CHA
05CF 8A05                  299            MOV     AL, [DI]        ;GET FIRST CHARACTER TO TX
05D1 EE                    300            OUT     DX, AL          ;OUTPUT IT TO 8274 TO GET IT STARTED
05D2 5A                    301            POP     DX
05D3 5F                    302            POP     DI
05D4 58                    303            POP     AX
05D5 C3                    304            RET                     ;RETURN
                           305
                           306    ;*************************************************************
                           307    ;*                                                           *
                           308    ;*    RX COMMAND FOR CHANNEL B - THE CALLING ROUTINE MUST    *
                           309    ;*    INITIALIZE RX_POINTER_CHB TO POINT AT THE RECEIVE      *
                           310    ;*    BUFFER BEFORE CALLING THIS ROUTINE.                    *
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                  LINE     SOURCE

                          311      ;*                                                    *
                          312      ;*****************************************************
                          313
05D6                      314      RX_COMMAND_CHB:
05D6 50                   315              PUSH    AX              ;SAVE REGISTERS
05D7 52                   316              PUSH    DX
05D8 C606290200           317              MOV     RX_READY_CHB, 0 ;CLEAR RX READY FLAG
05DD C70626020000         318              MOV     RX_COUNT_CHB, 0 ;CLEAR RX COUNTER
05E3 BA0600               319              MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
05E6 B003                 320              MOV     AL, 3           ;SET UP FOR WR3
05E8 EE                   321              OUT     DX, AL
05E9 B0C1                 322              MOV     AL, 0C1H        ;WR3 - 8 BITS/CHR, ENABLE RX
05EB EE                   323              OUT     DX, AL
05EC 5A                   324              POP     DX
05ED 58                   325              POP     AX
05EE C3                   326              RET                     ;RETURN
                          327
                          328      ;*****************************************************
                          329      ;*                                                   *
                          330      ;*    RX COMMAND FOR CHANNEL A - THE CALLING ROUTINE MUST      *
                          331      ;*    INITIALIZE RX_POINTER_CHA TO POINT AT THE RECEIVE        *
                          332      ;*    BUFFER BEFORE CALLING THIS ROUTINE                       *
                          333      ;*                                                   *
                          334      ;*****************************************************
                          335
05EF                      336      RX_COMMAND_CHA:
05EF 50                   337              PUSH    AX              ;SAVE REGISTERS
05F0 52                   338              PUSH    DX
05F1 C606350200           339              MOV     RX_READY_CHA, 0 ;CLEAR RX READY FLAG
05F6 C70632020000         340              MOV     RX_COUNT_CHA, 0 ;CLEAR RX COUNTER
05FC BA0200               341              MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
05FF B003                 342              MOV     AL, 3           ;SET UP FOR WR3
0601 EE                   343              OUT     DX, AL
0602 B0C1                 344              MOV     AL, 0C1H        ;WR3 - 8 BITS/CHR, ENABLE RX
0604 EE                   345              OUT     DX, AL
0605 5A                   346              POP     DX
0606 58                   347              POP     AX
0607 C3                   348              RET                     ;RETURN
                          349
                          350 +1   $EJECT
                          351
                          352      ;*****************************************************
                          353      ;*                                                   *
                          354      ;*          START OF INTERRUPT SERVICE ROUTINES      *
                          355      ;*                                                   *
                          356      ;*****************************************************
                          357
                          358      ;CHANNEL B TRANSMIT DATA SERVICE ROUTINE
                          359
0608 52                   360      XMTINB: PUSH    DX              ;SAVE REGISTERS
0609 57                   361              PUSH    DI
060A 50                   362              PUSH    AX
060B E80201               363              CALL    EOI             ;SEND EOI COMMAND TO 8274
060E FF062002             364              INC     TX_POINTER_CHB  ;POINT TO NEXT CHARACTER
0612 FF0E2202             365              DEC     TX_LENGTH_CHB   ;DEC LENGTH COUNTER
0616 740E                 366              JE      XIB             ;TEST IF DONE
0618 BA0400               367              MOV     DX, DATA_PORT_CHB       ;NOT DONE - GET NEXT CHARACTER
061B 8B3E2002             368              MOV     DI, TX_POINTER_CHB
061F 8A05                 369              MOV     AL, [DI]        ;PUT CHARACTER IN AL
0621 EE                   370              OUT     DX, AL          ;OUTPUT IT TO 8274
```

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ                 LINE    SOURCE

0622 58                  371             POP     AX              ;RESTORE REGISTERS
0623 5F                  372             POP     DI
0624 5A                  373             POP     DX
0625 CF                  374             IRET                    ;RETURN TO FOREGROUND
0626 BA0600              375     XIB:    MOV     DX, COMMAND_PORT_CHB    ;ALL CHARACTERS HAVE BEEN SEND
0629 B028                376             MOV     AL, 28H         ;RESET TRANSMITTER INTERRUPT PENDING
062B EE                  377             OUT     DX, AL
062C C606280201          378             MOV     TX_EMPTY_CHB, 1 ;DONE - SO SET TX EMPTY FLAG CHB
0631 58                  379             POP     AX              ;RESTORE REGISTERS
0632 5F                  380             POP     DI
0633 5A                  381             POP     DX
0634 CF                  382             IRET                    ;RETURN TO FOREGROUND
                         383
                         384     ;CHANNEL B STATUS CHANGE SERVICE ROUTINE
                         385
0635 52                  386     STAINB: PUSH    DX              ;SAVE REGISTERS
0636 57                  387             PUSH    DI
0637 50                  388             PUSH    AX
0638 E8D500              389             CALL    EOI             ;SEND EOI COMMAND TO 8274
063B BA0600              390             MOV     DX, COMMAND_PORT_CHB
063E EC                  391             IN      AL, DX          ;READ RR0
063F A22A02              392             MOV     STATUS_MSG_CHB, AL      ;PUT RR0 IN STATUS MESSAGE
0642 B010                393             MOV     AL, 10H         ;SEND RESET STATUS INT COMMAND TO 8274
0644 EE                  394             OUT     DX, AL
0645 58                  395             POP     AX              ;RESTORE REGISTERS
0646 5F                  396             POP     DI
0647 5A                  397             POP     DX
0648 CF                  398             IRET
                         399
                         400     ;CHANNEL B RECEIVED DATA SERVICE ROUTINE
                         401
0649 52                  402     RCVINB. PUSH    DX              ;SAVE REGISTERS
064A 57                  403             PUSH    DI
064B 50                  404             PUSH    AX
064C E8C100              405             CALL    EOI             ;SEND EOI COMMAND TO 8274
064F 8B3E2402            406             MOV     DI, RX_POINTER_CHB      ;GET RX CHB BUFFER POINTER
0653 BA0400              407             MOV     DX, DATA_PORT_CHB
0656 EC                  408             IN      AL, DX          ;READ CHARACTER
0657 8805                409             MOV     [DI], AL        ;STORE IN BUFFER
0659 FF062402            410             INC     RX_POINTER_CHB  ;BUMP THE BUFFER POINTER
065D FF062602            411             INC     RX_COUNT_CHB    ;BUMP THE COUNTER
0661 3C0D                412             CMP     AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
0663 750E                413             JNE     RIB
0665 C606290201          414             MOV     RX_READY_CHB, 1 ;YES, SET READY FLAG
066A BA0600              415             MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
066D B003                416             MOV     AL, 3           ;POINT AT WR3
066F EE                  417             OUT     DX, AL
0670 B0C0                418             MOV     AL, 0C0H        ;DISABLE RX
0672 EE                  419             OUT     DX, AL
0673 58                  420     RIB.    POP     AX              ;EITHER WAY, RESTORE REGISTERS
0674 5F                  421             POP     DI
0675 5A                  422             POP     DX
0676 CF                  423             IRET                    ;RETURN TO FOREGROUND
                         424
                         425     ;CHANNEL B ERROR SERVICE ROUTINE
                         426
0677 52                  427     ERRINB  PUSH    DX              ;SAVE REGISTERS
0678 50                  428             PUSH    AX
0679 E89400              429             CALL    EOI             ;SEND EOI COMMAND TO 8274
067C BA0600              430             MOV     DX, COMMAND_PORT_CHB
```

```
MCS-86 MACRO ASSEMBLER   ASYNCB

LOC  OBJ              LINE    SOURCE

067F B001               431           MOV     AL, 1           ;POINT AT RR1
0681 EE                 432           OUT     DX, AL
0682 EC                 433           IN      AL, DX          ;READ RR1
0683 A22B02             434           MOV     ERROR_MSG_CHB, AL       ;SAVE IT IN ERROR FLAG
0686 B030               435           MOV     AL, 30H         ;SEND RESET ERROR COMMAND TO 8274
0688 EE                 436           OUT     DX, AL
0689 58                 437           POP     AX              ;RESTORE REGISTERS
068A 5A                 438           POP     DX
068B CF                 439           IRET                    ;RETURN TO FOREGROUND
                        440
                        441   ;CHANNEL A TRANSMIT DATA SERVICE ROUTINE
                        442
068C 52                 443   XMTINA: PUSH    DX              ;SAVE REGISTERS
068D 57                 444           PUSH    DI
068E 50                 445           PUSH    AX
068F E87E00             446           CALL    EOI             ;SEND EOI COMMAND TO 8274
0692 FF062C02           447           INC     TX_POINTER_CHA  ;POINT TO NEXT CHARACTER
0696 FF0E2E02           448           DEC     TX_LENGTH_CHA   ;DEC LENGTH COUNTER
069A 740E               449           JE      XIA             ;TEST IF DONE
069C BA0000             450           MOV     DX, DATA_PORT_CHA       ;NOT DONE - GET NEXT CHARACTER
069F 8B3E2C02           451           MOV     DI, TX_POINTER_CHA
06A3 8A05               452           MOV     AL, [DI]        ;PUT CHARACTER IN AL
06A5 EE                 453           OUT     DX, AL          ;OUTPUT IT TO 8274
06A6 58                 454           POP     AX              ;RESTORE REGISTERS
06A7 5F                 455           POP     DI
06A8 5A                 456           POP     DX
06A9 CF                 457           IRET                    ;RETURN TO FOREGROUND
06AA BA0200             458   XIA     MOV     DX, COMMAND_PORT_CHA    ;ALL CHARACTERS HAVE BEEN SEND
06AD B028               459           MOV     AL, 28H         ;RESET TRANSMITTER INTERRUPT PENDING
06AF EE                 460           OUT     DX, AL
06B0 C606340201         461           MOV     TX_EMPTY_CHA, 1 ;DONE - SO SET TX EMPTY FLAG CHB
06B5 58                 462           POP     AX              ;RESTORE REGISTERS
06B6 5F                 463           POP     DI
06B7 5A                 464           POP     DX
06B8 CF                 465           IRET                    ;RETURN TO FOREGROUND
                        466
                        467   ;CHANNEL A STATUS CHANGE SERVICE ROUTINE
                        468
06B9 52                 469   STAINA: PUSH    DX              ;SAVE REGISTERS
06BA 57                 470           PUSH    DI
06BB 50                 471           PUSH    AX
06BC E85100             472           CALL    EOI             ;SEND EOI COMMAND TO 8274
06BF BA0200             473           MOV     DX, COMMAND_PORT_CHA
06C2 EC                 474           IN      AL, DX          ;READ RR0
06C3 A23602             475           MOV     STATUS_MSG_CHA, AL      ;PUT RR0 IN STATUS MESSAGE
06C6 B010               476           MOV     AL, 10H         ;SEND RESET STATUS INT COMMAND TO 8274
06C8 EE                 477           OUT     DX, AL
06C9 58                 478           POP     AX              ;RESTORE REGISTERS
06CA 5F                 479           POP     DI
06CB 5A                 480           POP     DX
06CC CF                 481           IRET
                        482
                        483   ;CHANNEL A RECEIVED DATA SERVICE ROUTINE
                        484
06CD 52                 485   RCVINA: PUSH    DX              ;SAVE PEGISTERS
06CE 57                 486           PUSH    DI
06CF 50                 487           PUSH    AX
06D0 E82D00             488           CALL    EOI             ;SEND EOI COMMAND TO 8274
06D3 8B3E3002           489           MOV     DI, PX_POINTER_CHA      ;GET PX CHA BUFFER POINTER
06D7 BA0000             490           MOV     DX, DATA_PORT_CHA
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB


LOC  OBJ                  LINE     SOURCE

06DA EC                    491             IN      AL, DX          ;READ CHARACTER
06DB 8805                  492             MOV     [DI], AL        ;STORE IN BUFFER
06DD FF063002              493             INC     RX_POINTER_CHA  ;BUMP THE BUFFER POINTER
06E1 FF063202              494             INC     RX_COUNT_CHA    ;BUMP THE COUNTER
06E5 3C0D                  495             CMP     AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
06E7 750E                  496             JNE     RIA
06E9 C606350201            497             MOV     RX_READY_CHA, 1 ;YES, SET READY FLAG
06EE BA0200                498             MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
06F1 B003                  499             MOV     AL, 3           ;POINT AT WR3
06F3 EE                    500             OUT     DX, AL
06F4 B0C0                  501             MOV     AL, 0C0H        ;DISABLE RX
06F6 EE                    502             OUT     DX, AL
06F7 58                    503     RIA.    POP     AX              ;EITHER WAY, RESTORE REGISTERS
06F8 5F                    504             POP     DI
06F9 5A                    505             POP     DX
06FA CF                    506             IRET                    ;RETURN TO FOREGROUND
                           507
                           508     ;CHANNEL A ERROR SERVICE ROUTINE
                           509
06FB 52                    510     ERRINA: PUSH    DX              ;SAVE REGISTERS
06FC 50                    511             PUSH    AX
06FD E81000                512             CALL    EOI             ;SEND EOI COMMAND TO 8274
0700 BA0200                513             MOV     DX, COMMAND_PORT_CHA
0703 B001                  514             MOV     AL, 1           ;POINT AT RR1
0705 EE                    515             OUT     DX, AL
0706 EC                    516             IN      AL, DX          ;READ RR1
0707 A23702                517             MOV     ERROR_MSG_CHA, AL       ;SAVE IT IN ERROR FLAG
070A B030                  518             MOV     AL, 30H         ;SEND RESET ERROR COMMAND TO 8274
070C EE                    519             OUT     DX, AL
070D 58                    520             POP     AX              ;RESTORE REGISTERS
070E 5A                    521             POP     DX
070F CF                    522             IRET                    ;RETURN TO FOREGROUND
                           523
                           524     ;END-OF-INTERRUPT ROUTINE - SENDS EOI COMMAND TO 8274.
                           525     ;  THIS COMMAND MUST ALWAYS TO ISSUED ON CHANNEL A.
                           526
0710 50                    527     EOI:    PUSH    AX              ;SAVE REGISTERS
0711 52                    528             PUSH    DX
0712 BA0200                529             MOV     DX, COMMAND_PORT_CHA    ;ALWAYS FOR CHANNEL A !!!
0715 B038                  530             MOV     AL, 38H
0717 EE                    531             OUT     DX, AL
0718 5A                    532             POP     DX
0719 58                    533             POP     AX
071A C3                    534             RET
                           535
                           536     ;END OF CODE ROUTINE
                           537
----                       538             ABC     ENDS
                           539             END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

REFERENCES

1. 8274 Multiprotocol Serial Controller (MPSC) Data Sheet, Intel Corporation, California, 1980.

2. Basics of Data Communication, Electronics Book Series, McGraw-Hill, New York, 1976.

3. Telecommunications and the Computer, J. Martin, Prentice-Hall, New Jersey, 1976.

4. Technical Aspects of Data Communications, J. McNamara, DEC Press, Massachusetts, 1977.

5. Miscellaneous Data Communications Standards —EIA RS-232-C, EIA RS-422, EIA RS-423, EIA Standard Sales, Washington, D.C.

intel

APPLICATION NOTE

AP-145

June 1982

Synchronous Communication with the 8274 Multiple Protocol Serial Controller

Sikandar Naqvi
Application Engineer

ORDER NUMBER: 210403-001

INTRODUCTION:

The INTEL 8274 is a Multi-Protocol Serial Controller, capable of handling both asynchronous and synchronous communication protocols. Its programmable features allow it to be configured in various operating modes, providing optimization to given data communication application.

This application note describes the features of the MPSC in Synchronous Communication applications only. It is strongly recommended that the reader read the 8274 Data Sheet and Application Note AP134 "Asynchronous Communication with the 8274 Multi-Protocol Serial Controller" before reading this Application Note. This Application note assumes that the reader is familiar with the basic structure of the MPSC, in terms of pin description, Read/Write registers and asynchronous communication with the 8274. Appendix A contains the software listings of the Application Example and Appendix B shows the MPSC Read/Write Registers for quick reference.

The first section of this application note presents an overview of the various sysnchronous protocols. The second section discusses the block diagram description of the MPSC. This is followed by the description of MPSC interrupt structure and mode of operation in the third and fourth sections. The fifth section describes a hardware/software example, using the INTEL single board computer iSBC88/45 as the hardware vehicle. The sixth section consists of some specialized applications of the MPSC. Finally, in section seven, some useful programming hints are summarized.

OPENING FLAG BYTE	ADDRESS* FIELD(A)	CONTROL** FIELD(C)	DATA FIELD	FRAME CHECK SEQUENCE	CLOSING FLAG BYTE

Figure 1. HDLC/SDLC Frame Format

* Extendable to 2 or More Bytes
** Extendable to 2 Bytes

SYNCHRONOUS PROTOCOL OVERVIEW

This section presents an overview of various synchronous protocols. The contents of this section are fairly tutorial and may be skipped by the more knowledgeable reader.

Bit Oriented Protocols Overview

Bit oriented protocols have been defined to manage the flow of information on data communication links. One of the most widely known protocol is the one defined by the International Standards Organization: HDLC (High Level Data Link Control). The American Standard Associations' protocol, ADCCP is similar to HDLC. CCITT Recommendation X.25 layer 2 is also an acceptable version of HDLC. Finally, IBM's SDLC (Synchoronous Data Link Control) is also a subset of the HDLC.

In this section, we will concentrate most of our discussion on HDLC. Figure 1 shows a basic HDLC frame format.

A frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags - opening and closing flags. An address field is 8 bits wide, extendable to 2 or more bytes. The control field is also 8 bits wide, extendable to two bytes. The data field or information field may be any number of bits. The data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags.

ZERO BIT INSERTION

The flag has a unique binary bit pattern: 7E HEX. To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's.The 8274 performs zero bit insertion and deletion automatically in the SDLC/HDLC mode. The zero-bit stuffing ensures periodic transitions in the data stream. These transitions are necessary for a phase lock circuit, which may be used at the receiver end to generate a receive clock which is in phase to the received data. The inserted and deleted 0's are not included in the CRC checking. The *address* field is used to address a given secondary station. The *control* field contains the link-level control information which includes implied acknowledgement, supervisory commands and responses, etc. A more detailed discussion of higher level protocol functions is beyond the scope of this application note. Interested readers may refer to the references at the end of this application note.

The *data field* may be of any length and content in HDLC. Note that SDLC specifies that data field be a multiple of bytes only. In data communications, it is gen-

erally desirable to transmit data which may be of any content. This requires that data field should not contain characters which are defined to assist the transmission protocol (like opening flag 7EH in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion discussed earlier and the bit orientated nature of the protocol.

The last field is the FCS (Frame Check Sequence). The FCS uses the error detecting techniques called Cyclic Redundancy Check. In SDLC/HDLC, the CCITT-CRC must be used.

NON-RETURN TO ZERO INVERTED (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC protocol. It allows HDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while a 0 causes a change of state. NRZI coding ensures that an active data line will have transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for a phase lock circuit at the receiver end to derive a receive clock (from received data) which is synchronized to the received data and at the same time ensure data transparency.

Byte Synchronous Communication

As the name implies, Byte Synchronous Communication is a synchronous communication protocol which means that the transmitting station is synchronized to the receiving station through the recognition of a special sync character or characters. Two examples of Byte Synchronous protocol are the IBM Bisync and Monosync. Bisync has two starting sync characters per message while monosync has only one sync character. For the sake of abrevity, we will only discuss Bisync here. All the discussion is valid for Monosync also. Any exceptions will be noted. Figure 2 shows a typical Bisync message format.

The Bisync protocol is defined for half duplex communication between two or more stations over point to point or multipoint communication lines. Special characters control link access, transmission of data and termination of transmission operations for the system. A detailed discussion of these special control characters (SYN, ENQ, STX, ITB, ETB, ETX, DLE, SOH, ACK0, ACK1, WACK, NAK and EOT, etc) is beyond the scope of this Application Note. Readers interested in more detailed discussion are directed to the references listed at the end of this Application Note.

As shown in Figure 2, each message is preceded by two sync characters. Since the sync characters are defined at the beginning of the message only, the transmitter must insert fill characters (sync) in order to maintain synchronization with the receiver when no data is being transmitted.

TRANSPARENT TRANSMISSION

Bisync protocol requires special control characters to maintain the communication link over the line. If the data is EBCDIC encoded, then transparency is ensured by the fact that the data field will not contain any of the bisync control characters. However, if data does not conform to standard character encoding techniques, transparency in bisync is achieved by inserting a special character DLE (Data Link Escape) before and after a string of characters which are to be transmitted transparently. This ensures that any data charaters which match any of the special characters are not confused for special characters. An example of a transparent block is shown in Figure 3.

In a transparent mode, it is required that the CRC(BCC) is not performed on special characters. Later on, we will show how the 8274 can be used to achieve transparent transmission in Bisync mode.

SYNC	SYNC	SOH	HEADER	STX TEXT	ETX OR ETB	CRC 1	CRC 2

Figure 2. Bisync Message Format

DLE	STX	TRANSPARENT TRANSMISSION	DLE	ETX	BCC

Enter transparent mode — return to normal mode

Figure 3. Bisync Transparent Format

BLOCK DIAGRAM

This section discusses the block diagram view of the 8274. The CPU interface and serial interface is discussed separately. This will be followed by a hardware example in the fifth section, which will show how to interface the 8274 with the Intel CPU 8088. The 8274 block diagram is shown in Figure 4.

CPU Interface

The CPU interface to the system interface logic block utilizes the A0, A1, $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$ inputs to communicate with the internal registers of the 8274. Figure 5 shows the address of the internal registers. The DMA interface is achieved by utilizing DMA request lines for each channel: $TxDRQ_A$, $TxDRQ_B$, $RxDRQ_A$, $RxDRQ_B$. Note that

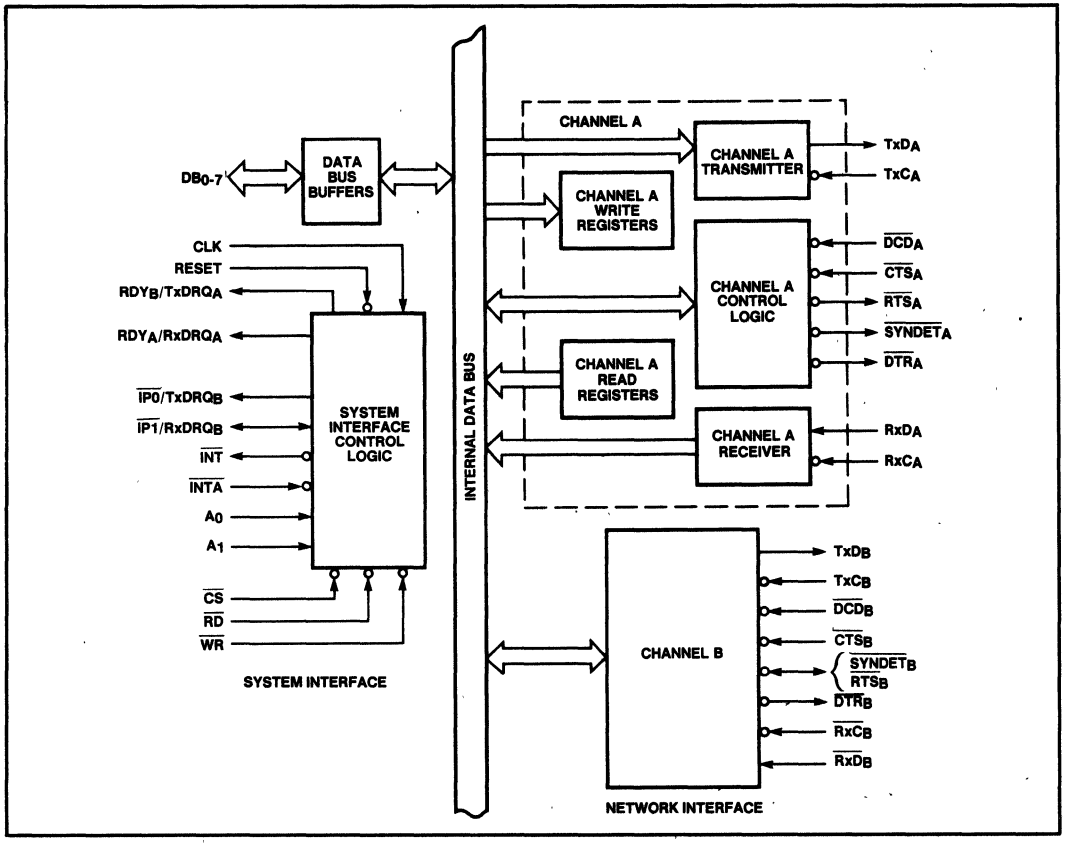


Figure 4. 8274 Block Diagram

$\overline{CS}$	A1	A0	Read Operation	Write Operation
0	0	0	CHA DATA READ	CHA DATA WRITE
0	1	0	CHA STATUS REGISTER (RR0,RR1)	CHA COMMAND/PARAMETER (WR0–WR7)
0	0	1	CHB DATA READ	CHB DATA WRITE
0	1	1	CHB STATUS REGISTER (RR0,RR1,RR2)	CHB COMMAND/PARAMETER (WR0–WR7)
1	X	X	HIGH Z	HIGH Z

Figure 5. Bus Interface

$TxDRQ_B$ and $RxDRQ_B$ becomes IPO and IPI respectively in non-DMA mode. IPI is the Interrupt Priority Input and IPO is the Interrrupt Priority Output. These two pins can be used for connecting multiple MPSCs in a daisy chain. If the Wait Mode is programmed, then $TxRDQ_A$ and $RxDRQ_A$ pins become RDY_B and RDY_A pins. These pins can be wire-or'ed and are usually hooked up to the CPU RDY line to synchronize the CPU for block transfers. The INT pin is activated whenever the MPSC requires CPU attention. The INTA may be used to utilize the powerful vectored mode feature of the 8274. Detailed discussion on these subjects will be done later in this Application Note. The Reset pin may be used for hardware reset while the clock is required to click the internal logic on the MPSC.

Serial Interface

On the serial side, there are two completely independent channels: Channel A and Channel B. Each channel consists of a transmitter block, receiver block and a set of read/write registers which are used to initialize the device. In addition, a control logic block provides the modem interface pins. Channel B serial interface logic is a mirror image of Channel A serial interface logic, except for one exception: there is only one pin for RTS_B and $SYNDET_B$. At a given time, this pin is either RTS_B or $SYNDET_B$. This mode is programmable through one of the internal registers on the MPSC.

Transmit And Receive Data Path

Figure 6 shows a block diagram for transmit and receive data path. Without describing each block on the diagram, a brief discussion of the block diagram will be presented here.

TRANSMIT DATA PATH

The transmit data is transferred to the twenty-bit serial shift register. The twenty-bits are needed to store two bytes of sync characters in bisync mode. The last three bits of the shift register are used to indicate to the internal control logic that the current data byte has been shifted out of the shift register. The transmit data in the transmit shift register is shifted out through a two bit delay onto the TxData line. This two bit delay is used to synchronize the internal shift clock with the external transmit clock. The data in the shift register is also presented to zero bit insertion logic which inserts a zero after sensing five contiguous ones in the data stream. In parallel to all this activity, the CRC-generator is computing CRC on the transmitted data and appends the frame with CRC bytes at the end of the data transmission.

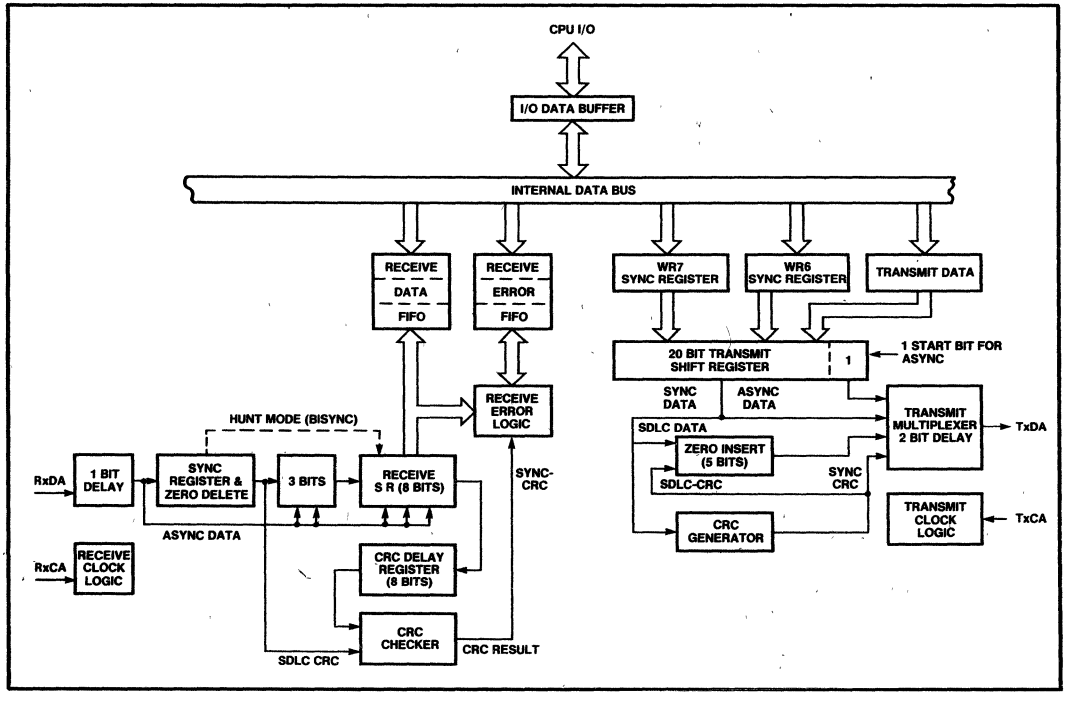


Figure 6. Transmit and Receive Data Path

RECEIVE DATA PATH

The received data is passed through a one-bit delay before it is presented for flag/sync comparison. In bisync mode, after the synchronization is achieved, the incoming data bypasses the sync register and enters directly into the three bit buffer on its way to receive shift register. In SDLC mode, the incoming data always passes through the sync register where data pattern is continously monitored for contiguous ones for zero deletion logic. The data then enters the three bit buffer and the receive shift register. From the receive shift register, the data is transferred to the three byte deep FIFO. The data is transferred to the top of the FIFO at the chip clock rate (not the receiver clock). It takes three chip clock/periods to transfer data from the serial shift register to the top of the FIFO. The three bit deep Receive Error FIFO shifts any error condition which may have occured during a frame reception. While all this is happening, the CRC checker is checking the CRC on the incoming data. The computed CRC is checked with the CRC bytes attached to the incoming frame and an error generated under a no-check condition. Note that the bisync data is presented to the CRC checker with an 8-bit delay. This is necessary to achieve transparency in bisync mode as will be shown later in this Application Note.

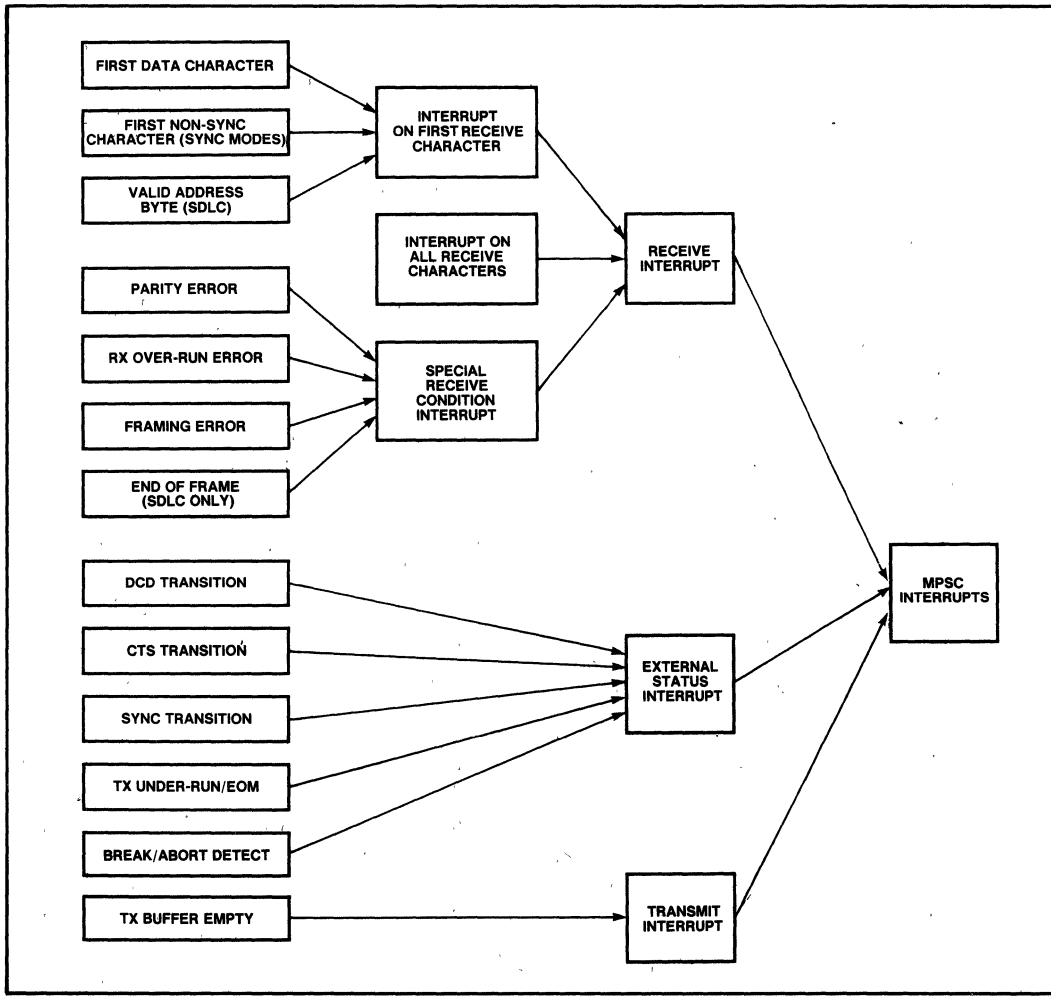


Figure 7. MPSC Interrupt Structure

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) INTERRUPT STRUCTURE

The MPSC offers a very powerful interrupt structure, which helps in responding to an interrupt condition very quickly. There are multiple sources of interrupts within the MPSC. However, the MPSC resolves the priority between various interrupting sources and interrupts the CPU for service through the interrupt line. This section presents a comprehensive discussion on all the 8274 interrupts and the priority resolution between these interrupts.

All the sources of interrupts on the 8274 can be grouped into three distinct catagories. (See Figure 7)

1. Receive Interrupts
2. Transmit Interrupts
3. External/Status Interrupts.

An internal interrupt priority structure sets the priority between the interrupts. There are two programmable options available on the MPSC. The priority is set by WR2A, D2. (Figure 8)

PRIORITY

WR2A:D2	Highest					Lowest
0	RxA	TxA	RxB	TxB	EXTA	EXTB
1	RxA	TxA	RxB	TxB	EXTA	EXTB

Figure 8. Interrupt Priority

Receive Interrupt

All receive interrupts may be catagorized into two distinct groups: Receive Interrupt on Receive Character and Special Receive Condition Interrupts.

RECEIVE INTERRUPT ON RECEIVE CHARACTER

A receive interrupt is generated when a character is received by the MPSC. However, as will be discussed later, this is a programmable feature on the MPSC . A Rx character available interrupt is generated by the MPSC after the receive character has been assembled by the MPSC. It may be noted that in DMA transfer mode too, a receive interrupt on the first receive character should be programmed. In SDLC mode, if address search mode has been programmed, this interrupt will be generated only after a valid address match has occured. In bisync mode, this interrupt is generated on receipt of a character after at least two valid sync characters. In monosync mode, a character followed by at least a single valid sync character will generate this interrupt. An interrupt on first receive character signifies the beginning of a valid frame. An end of the frame is characterized by an "End of Frame" Interrupt (RR1: D7).* This bit (RR1:D7) is set in SDLC/HDLC mode only and signifies that a valid ending flag (7EH) has been received. This bit gets reset either by an "Error Reset" command (WR0: D5D4D3 = 110) or upon reception of the first character of the next frame. In multiframe reception, on receiving the interrupt at the "End of Frame" the CPU may issue an Error Reset command which will reset the interrupt. In DMA mode, the interrupt on first receive character is accompanied by a RxDRQ (Receiver DMA request) on the appropriate channel. At the end of the frame, an End of Frame interrupt is generated. The CPU may use this interrupt to jump into a routine which may redefine the receive buffer for the next incoming frame.

*RR1:D7 is bit D7 in Read Register 1.

SPECIAL RECEIVE CONDITION INTERRUPTS

So far, we have assumed that the reception is error free. But this is not a 'typical' case in most real life applications. Any error condition during a frame reception generates yet another interrupt — special receive condition interrupt. There are four different error conditions which can generate this interrupt.

(i) Parity error
(ii) Receive Overrun error
(iii) Framing error
(iv) End of Frame

(i) Parity error: Parity error is encountered in asynchronous (start-stop bits) and in bisync/monosync protocols. Both odd or even parity can be programmed. A parity error in a received byte will generate a special receive condition interrupt and sets bit 4 in RR1.

(ii) Receive Overrun error: If the CPU or the DMA controller (in DMA mode) fails to read a received character within three byte times after the received character interrupt (or DMA request) was generated, the receiver buffer will overflow and this will generate a special receive condition interrupt and sets bit 5 in RR1.

(iii) Framing error: In asynchronous mode, a framing error will generate a special receive interrupt and set bit D6 in RR1. This bit is not latched and is updated on the next received character.

(iv) End of frame: This interrupt is encountered in SDLC/HDLC mode only. When the MPSC receives the closing flag, it generates the special receive condition interrupt and sets bit D7 in RR1.

All the special receive condition interrupts may be reset by issuing an Error Reset Command.

CRC Error: In SDLC/HDLC and synchronous modes, a CRC error is indicated by bit D6 in RR1. When used to check CRC error, this bit is normally set until a correct CRC match is obtained which resets this bit. After receiving a frame, the CPU must read this bit (RR1:D6) to determine if a valid CRC check had occured. It may be noted that a CRC error does not generate an interrupt.

It may be also be pointed out that in SDLC/HDLC mode, receive DMA requests are disabled by a special receive condition and can only be re-enabled by issuing an Error Reset Command.

Transmit Interrupt

A transmit buffer empty generates a transmit interrupt. This has been discussed earlier under "Transmit in Interrupt Mode" and it would be sufficient to note here that a transmit buffer empty interrupt is generated only when the transmit buffer gets empty — assuming it had a data character loaded into it earlier. This is why on starting a frame transmission, the first data character is loaded by the CPU without a transmit empty interrupt (or DMA request in DMA mode). After this character is loaded into the serial shift register, the buffer becomes empty, and an interrupt (or DMA request) is generated. This interrupt is reset by a "Reset Tx Interrupt/DMA Pending" command (WR0: D5 D4 D3 = 101).

External/Status Interrupt

Continuing our discussion on transmit interrupt, if the transmit buffer is empty and the transmit serial shift register also becomes empty (due to the data character shifted out of the MPSC), a transmit under-run interrupt will be generated. This interrupt may be reset by "Reset/External Status Interrupt" command (WR0: D5 D4 D3 = 101).

The External Status Interrupt can be caused by five different conditions:
(i) DCD Transition
(ii) CTS Transition
(iii) Sync/Hunt Transition
(iv) Tx under-run/EOM condition
(v) Break/Abort Detection.

DCD,CTS TRANSITION

Any transition on these inputs on the serial interface will generate an External/Status interrupt and set the corresponding bits in status register RR0. This interrupt will also be generated in DMA as well as in Wait Mode. In order to find out the state of the CTS or DCD pins *before* the transition had occurred, RR0 must be read before issuing a Reset External/Status Command through WR0. A read of RR0 after the Reset External/Status Command will give the condition of CTS or DCD pins *after* the transition had occurred. Note that bit D5 in RR0 gives the complement of the state of CTS pin while D3 in RR0 reflects the actual state of the DCD pin.

SYNC HUNT TRANSITION

Any transition on the SYNDET input generates an interrupt. However, sync input has different functions in different modes and we shall discuss them individually.

SDLC Mode

In SDLC mode, the SYNDET pin is an output. Status register RR1, D4 contains the state of the SYNDET pin. The Enter Hunt Mode initially sets this bit in R0. An opening flag in a received SDLC frame resets this bit and generates an external status interrupt. Every time the receiver is enabled or the Enter Hunt Code Command is issued, an external status interrupt will be generated on receiving a valid flag followed by a valid address/data character. This interrupt may be reset by the "Reset External Status Interrupt" command.

External SYNC Mode

The MPSC can be programmed into External Sync Mode by setting WR4, D5 D4 = 11. The SYNDET pin is an input in this case and must be held high until an external character synchronization is established. However, the External Sync mode is enabled by the Enter Hunt Mode control bit (WR3: D4). A high at the SYNDET pin holds the sync/Hunt bit (RR0,D4) in the reset state. When external synchronization is established, SYNDET must be driven low on second rising edge of RxC after the rising edge of RxC on which the last bit of sync character was received. This high to low transition sets the Sync/Hunt bit and generates an external status interrupt, which must be reset by the Reset External/Status command. If the SYNDET input goes high again, another External Status Interrupt is generated, which may be cleared by Reset External Status command.

Mono-Sync/Bisync Mode

SYNDET pin acts as an output in this case. The Enter Hunt Mode sets the Sync/Hunt bit in R0. Sync/Hunt bit is reset when the MPSC achieves character sysnchronization. This high to low transition will generate an external status interrupt. The SYNDET pin goes active every time a sync pattern is detected in the data stream. Once again, the external status interrupt may be reset by the Reset External Status command.

Tx UNDER-RUN/END OF MESSAGE (EOM)

The transmitter logic includes a transmit buffer and a transmit serial shift register. The CPU loads the character into the transmit buffer which is transferred into the transmit shift register to be shifted out of the MPSC. If the transmit buffer gets empty, a transmit buffer empty interrupt is generated (as discussed earlier). However, if the transmit buffer gets empty *and* the serial shift register gets empty, a transmit under-run condition will be created. This generates an External Status Interrupt and the interrupt can be cleared by the Reset External Status command. The status register RR0, D6 bit is set when the transmitter under-runs. This bit plays an important role in controlling a transmit operation, as will be discussed later in this application note.

BREAK/ABORT DETECTION

In asynchronous mode, bit D7 in RR0 is set when a break condition is detected on the receive data line. This also generates an External/Status interrupt which may be reset by issuing a Reset External/Status Interrupt command to the MPSC. Bit D7 in RR0 is reset when the break condition is terminated on the receive data line and this causes another External/Status interrupt to be generated. Again, a Reset External/Status Interrupt command will reset this interrupt and will enable the break detection logic to look for the next break sequence.

In SDLC Receive Mode, an Abort sequence (seven or more 1's) detection on the receive data line will generate an External/Status interrupt and set RR0,D7. A Reset External/Status command will clear this interrupt. However, a termination of the Abort sequence will generate another interrupt and set RR0, D7 again. Once again, it may be cleared by issuing Reset External/Status Command.

This concludes our discussion on External Status Interrupts.

Interrupt Priority Resolution

The internal interrupt priority between various interrupt sources is resolved by an internal prority logic circuit, according to the priority set in WR2A. We will now discuss the interrupt timings during the priority resolution. Figures 9 and 10 show the timing diagrams for vectored and non-vectored modes.

VECTORED MODE

We shall assume that the MPSC accepted an internal request for an interrupt by activating the internal INT signal. This leads to generating an external interrupt signal on the INT pin. The CPU responds with an interrupt acknowledge (INTA) sequence. The leading edge of the first INTA pulse sets an internal interrupt acknowledge signal (we will call it Internal INTA). Internal INTA is reset by the high going edge of the third INTA pulse. The MPSC will not accept any internal requests for an interrupt during the period when Internal INTA is active (high). The MPSC resolves the priority during various existing internal interrupt requests during the Interrupt Request Priority Resolve Time, which is defined as the time between the leading edge of the first INTA and the leading edge of the second INTA from the CPU. Once the internal priorities have been resolved, an internal Interrupt-in-service Latch is set. The external INT is also deactivated when the Interrupt-in-Service Latch is set.

The lower priority interrupt requests are not accepted internally until an EOI (WR0: D5 D4 D3 = 111) command is issued by the CPU. The EOI command enables the lower priority interrupts. However, a higher priority interrupt

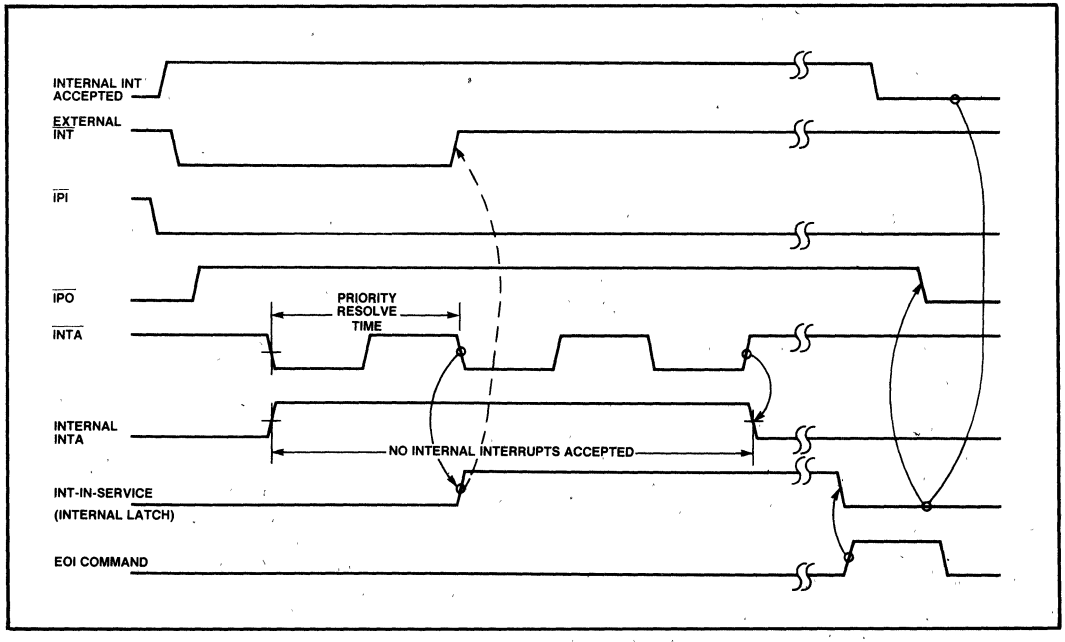


Figure 9. 8274 in 8085 Vectored Mode Priority Resolution Time

AFN-02213A

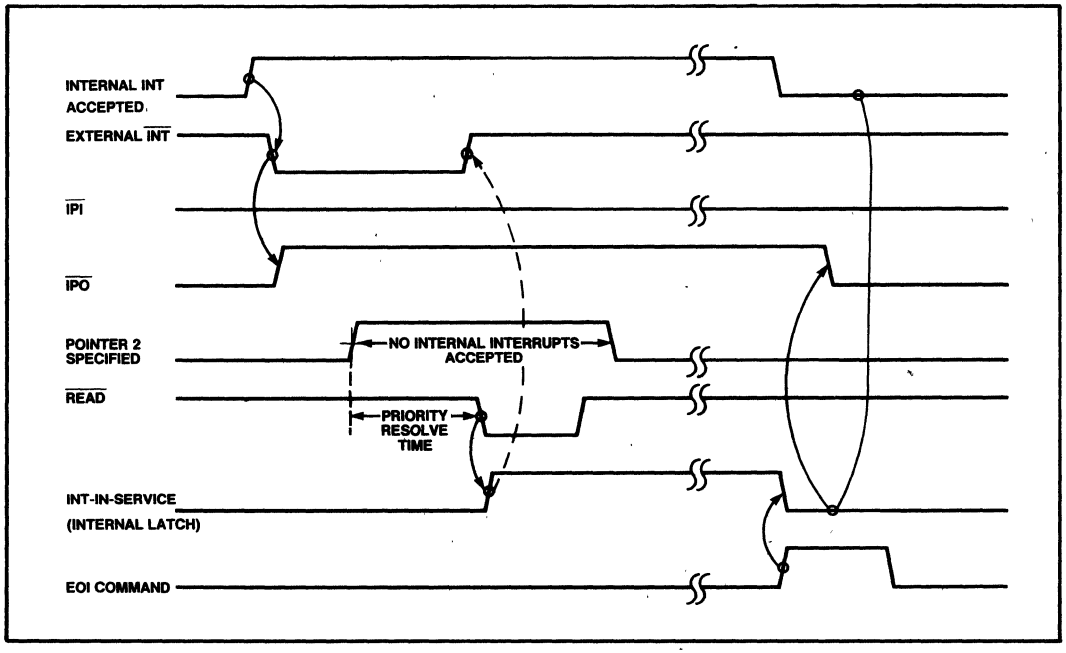


Figure 10. 8274 Non Vectored Mode Priority Resolve Time

request will still be accepted (except during the period when internal INTA is active) even though the Internal-in-Service Latch is set. This higher priority request will generate another external INT and will have to be handled by the CPU according to how the CPU is set up. If the CPU is set up to respond to this interrupt, a new INTA cycle will be repeated as discussed earlier. It may also be noted that a transmitter buffer empty and receive character available interrupts are cleared by loading a character into the MPSC and by reading the character received by the MPSC respectively.

NON-VECTORED MODE

Figure 10 shows the timing of interrupt sequence in non-vectored mode. The explanation for non-vectored is similar to the vector mode, except for the following exceptions.

— No internal priority requests are accepted during the time when pointer 2 for Channel B is specified.

— The interrupt request priority resolution time is the time between the leading edge of pointer 2 and leading edge of RD active. It may be pointed out that in non-vectored mode, it is assumed that the status affects vector mode is used to expedite interrupt response.

On getting an interrupt in non-vectored mode, the CPU must read status register RR2 to find out the cause of the interrupt. In order to do so, first a pointer to status register RR2 is specified and then the status read from RR2. It may be noted here that after specifying the pointer, the CPU must read status register RR2 otherwise, no new interrupt requests will be accepted internally.

Just like the vectored mode, no lower internal priority requests are accepted until an EOI command is issued by the CPU. A higher priority request can still interrupt the CPU (except during the priority request inhibit time). It is important to note here that if the CPU does not perform a read operation after specifying the pointer 2 for Channel B, the interrupt request accepted before the pointer 2 was activated will remain valid and no other request (high or low priority) will be accepted internally. In order to complete a correct priority resolution, it is advised that a read operation be done after specifying the pointer 2B.

IPI and IPO

So far, we have ignored the IPI and IPO signals shown in Figures 9 and 10. We may recall that IPI is the Interrupt-Priority-Input to the MPSC. In conjunction with the IPO (Interrupt Priority Output), it is used to daisy chain multiple MPSC's. MPSC daisy chaining will be discussed in detail later in this application note.

EOI Command

The EOI command as explained earlier, enables the lower priority interrupts by resetting the internal In-Service-Latch, which consequently resets the IPO output to a low state. See Figures 9 and 10 for details. Note that before issuing any EOI command, the internal interrupting source must be satisfied otherwise, same source will interrupt again. The Internal Interrupt is the signal which gets reset when the internal interrupting source is satisfied (see Figure 9).

This concludes our discussion on the MPSC Interrupt Structure.

MULTI-PROTOCOL SERIAL CONTROLLER (MPSC) MODES OF OPERATION

The MPSC provides two fully independent channels that may be configured in various modes of operations. Each channel can be configured into full duplex mode and may operate in a mode or protocol different from the other channel. This feature will be very efficient in an application which requires two data link channels operating in different protocols and possibly at different data rates. This section presents a detailed discussion on all the 8274 modes and shows how to configure it into these modes.

Interrupt Driven Mode

In the interrupt mode, all the transmitter and receiver operations are reported to the processor through interrupts. Interrupts are generated by the MPSC whenever it requires service. In the following discusson, we will discuss how to transmit and receive in interrupt driven mode.

TRANSMIT IN INTERRUPT MODE

The MPSC can be configured into interrupt mode by appropriately setting the bits in WR2 A (Write Register 2, Channel A). Figure 11 shows the modes of operation.

WR2A		
D1	D0	MODE
0	0	CH A and CH B in Interrupt Mode
0	1	CH A in DMA and CH B in Interrupt Mode
1	0	CH A and CH B in DMA Mode
1	1	Illegal

Figure 11. MPSC Mode Selection for Channel A and Channel B.

We will limit our discussion to SDLC transmit and receive only. However, exceptions for other synchronous protocols will be pointed out. To initiate a frame transmission, the first data character must be loaded from the CPU, in all cases. (DMA Mode too, as you will notice later in this application note). Note that in SDLC mode, this first data character may be the address of the station addressed by the MPSC. The transmit buffer consists of a transmit buffer and a serial shift register. When the character is transferred from the buffer into the serial shift register, an interrupt due to transmit buffer empty is generated. The CPU has one byte time to service this interrupt and load another character into the transmitter buffer. The MPSC will generate an interrupt due to transmit buffer under-run condition if the CPU does not service the Transmit Buffer Empty Interrupt within one byte time.

This process will continue until the CPU is out of any more data characters to be sent. At this point, the CPU does not respond to the interrupt with a character but simply issues a Reset Tx INT/DMA pending command (WR0: D5 D4 D3 = 1 0 1). The MPSC will ultimately under-run, which simply means that both the transmit buffer and transmit shift registers are empty. At this point, flag character (7EH) or CRC byte is loaded into the transmit shift register. This sets the transmit under-run bit in RR0 and generates "Transmit Under-run/EOM" interrupt (RR0:D6=1).

You will recall that an SDLC frame has two CRC bytes after the data field. 8274 generates the CRC on all the data that is loaded from the CPU. During initialization, there is a choice of selecting a CRC-16 or CCITT-CRC (WR5: D2). In SDLC/HDLC operation, CCITT-CRC must be selected. We will now see how the CRC gets inserted at the end of the data field. Here we have a choice of having the CRC attached to the data field or sending the frame without the CRC bytes. During transmission, a "Reset Tx Under-run/EOM Latch" command (WR0: D7 D6 = 11) will ensure that at the end of the frame when the transmitter underruns, CRC bytes will be automatically inserted at the end of the data field. If the "Reset Tx Under-run/EOM Latch" command was not issued during the transmission of data characters, no CRC would be inserted and the MPSC will transmit flags (7EH) instead.

However, in case of CRC transmission, the CRC transmission sets the Tx Under-run/EOM bit and generates a Transmitter Under-run/EOM Interrupt as discussed earlier. This will have to be reset in the next frame to ensure CRC insertion in the next frame. It is recommended that Tx Under-run/EOM latch be reset very early in the transmission mode, preferably after loading the first character. It may be noted here that Tx Under-run/EOM latch cannot be reset if there is no data in the transmit buffer. This means that at least one character has to be loaded into the MPSC before a "Reset Transmit Under-run/EOM Latch" command will be accepted by the MPSC.

When the transmitter is under-run, an interrupt is generated. This interrupt is generated at the beginning of the CRC transmission, thus giving the user enough time (minimum 22 transmit clock cycles) to issue an Abort

command (WR0: D5 D4 D3 = 0 0 1) in case if the transmitted data had an error. The Abort Command will ensure that the MPSC transmits at least eight 1's but less than fourteen 1's before the line reverts to continuous flags. The receiver will scratch this frame because of bad CRC.

However, assuming the transmission was good (no Abort Command issued), after the CRC bytes have been transmitted, closing flag (7EH) is loaded into the transmit buffer. When the flag (7EH) byte is transferred to the serial shift register, a transmit buffer empty interrupt is generated. If another frame has to be transmitted, a new data character has to be loaded into the transmit buffer and the complete transmit sequence repeated. If no more frames are to be transmitted, a "Reset Transmit INT/DMA Pending" command (WR0: D5 D4 D3 = 1 0 1) will reset the transmit buffer empty interrrupt.

For character oriented protocols (Bisync, Monosync), the same discussion is valid, except that during transmit under-run condition and transmit under-run/EOM bit in set state, instead of flags, filler sync characters are transmitted.

CRC Generation:

The transmit CRC enable bit (WR5: D0) must be set before loading any data into the MPSC. The CRC generator must be reset to all 1's at the beginning of each frame before CRC computation has begun. The CRC computation starts on the first data character loaded from the CPU and continues until the last data character. The CRC generated is inverted before it is sent on the Tx Data line.

Transmit Termination:

A successful transmission can be terminated by issuing a "Reset Transmit Interrupt/DMA Pending" command, as discussed earlier. However, the transmitter may be disabled any time during the transmission and the results will be as shown in Figure 12.

Transmitter Disabled during	Result
1. Data Transmission	Tx Data will send idle characters* which will be zero inserted.
2. CRC Transmission	16 bit transmission, corresponding to 16 bits of CRC will be completed. However, flag bits will be substituted in the CRC field.
3. Immediately after issuing ABORT command.	Abort will still be transmitted — output will be in the mark state.

Figure 12. Transmitter Disabled During Transmission

*Idle characters are defined as a string of 15 or more contiguous ones.

RECEIVE IN INTERRUPT MODE

The receiver has to be initialized into the appropriate receive mode (see sample program later in this application note). The receiver must be programmed into Hunt Mode (WR3: D4) before it is enabled (WR3: D0). The receiver will remain in the Hunt Mode until a flag (or sync character) is received. While in the SDLC/Bisync/Monosync mode, the receiver does not enter the Hunt Mode unless the Hunt bit (WR3, D4) is set again or the receiver is enabled again.

SDLC Address byte is stored in WR6. A global address (FFH) has been hardwired on the MPSC. In address search mode (WR3: D2 = 1), any frame with address matching with the address in WR6 will be received by the MPSC. Frames with global address (FFH) will also be received, irrespective of the condition of address search mode bit (WR3: D2). In general receive mode (WR3: D2=0), all frames will be received.

Since the MPSC only recognizes single byte address field, extended address recognition will have to be done by the CPU on the data passed on by the MPSC. If the first address byte is checked by the MPSC, and the CPU determines that the second address byte does not have the correct address field, it must set the Hunt Mode (WR3: D2 = 1) and the MPSC will start searching for a new address byte preceded by a flag.

Programmable Interrupts: The receiver may be programmed into any one of the four modes. See Figure 13 for details.

WR1, CHA		
D4	D3	Rx Interrupt Mode
0	0	Rx INT/DMA disable
0	1	Rx INT on first character
1	0	INT on all Rx characters (Parity affects vector)
1	1	INT on all Rx characters (Parity does not affect vector)

Figure 13. Receiver Interrupt Modes

All receiver interrupts can be disabled by WR1: D4 D3 = 0 0. Receiver interrupt on first character is normally used to start a DMA transfer or a block transfer sequence using WAIT to synchronize the data transfer to received or transmitted data.

External Status Interrupts:

Any change in DCD input or Abort detection in the received data, will generate an interrupt if External Status Interrupt was enabled (WR1: D0).

Special Receive Conditions:

The receiver buffer is quadruply buffered. If the CPU fails to respond to "receive character" available interrupt within a period of three byte times (received bytes), the receiver buffer will overflow and generate an interrupt. Finally, at the end of the received frame, an interrupt will be generated when a valid ending flag has been detected.

Receive Character Length:

The receive character length (6,7 or 8 bits/character) may be changed during reception. However, to ensure that the change is effective on the next received character, this must be done fast enough such that the bits specified for the next character have not been assembled.

CRC Checking:

The opening flag in the frame resets the receive CRC generator and any field between the opening and closing flag is checked for the CRC. In case of a CRC error, the CRC/Framing Error bit in status register 1 is set (RR1: D6=1). Receiver CRC may be disabled/enabled by WR3,D3. The CRC bytes on the received frame are passed on to the CPU just like data, and may be discarded by the CPU.

Receive Terminator:

An end of frame is indicated by End of Frame interrupt. The CPU may issue an "Error Reset" command to reset this interrupt.

DMA (Direct Memory Access) Mode

The 8274 can be interfaced directly to the Intel DMA Controllers 8237A, 8257A and Intel I/0 Processor 8089. The 8274 can be programmed into DMA mode by setting appropriate bits in WR2A. See Figure 11 for details.

TRANSMIT IN DMA MODE:

After initializing the 8274 into the DMA mode, the first character must be loaded from the CPU to start the DMA cycle. When the first data character (may be the address byte in SDLC) is tranferred from the transmit buffer to the transmit serial shift register, the transmit buffer gets empty and a transmit DMA request (TxDRQ) is generated for the channel. Just like the interrupt mode, to ensure that the CRC bytes are included in the frame, the transmit under-run/EOM latch must be reset. This should preferably be done after loading the first character from the CPU. The DMA will progress without any CPU intervention. When the DMA controller reaches the terminal count, it will not respond to the DMA request, thus letting the MPSC under-run. This will ensure CRC transmission. However, the under-run condition will generate an interrupt due to the Tx under-run/EOM bit getting set (RR0: D6). The CPU should issue a "Reset TxInt/DRQ pending" command to reset TxDRQ and issue a "Reset External Status" command to reset Tx Under-run/EOM interrupt. Following the CRC transmission, flag (7EH) will be loaded into the transmit buffer. This will also generate the TxDRQ since the transmit buffer is empty following the transmission of the CRC bytes. The CPU may issue a "Reset TxINT/DRQ pending" command to reset the TxDRQ. "Reset TxINT/DRQ pending" command must be issued before setting up the transmit DMA channel on the DMA Controller, otherwise the MPSC will start the DMA transfer immediately after the DMA channel is set up.

RECEIVE IN DMA MODE

The receiver must be programmed in RxINT on first receive character mode (WR1: D4 D3 = 0 1). Upon receiving the first character, which may be the address byte in SDLC, the MPSC generates an interrupt and also generates a Rx DMA Request (Rx DRQ) for the appropriate channel. The CPU has three byte times to service this interrupt (enable the DMA controller, etc.) before the receiver buffer will overflow. It is advisable to initialize the DMA controller before receiving the first character. In case of high bit rates, the CPU will have to service the interrupt very fast in order to avoid receiver over-run.

Once the DMA is enabled, the received data is transfered to the memory under DMA control. Any received error conditions or external status change condition will generate an interrupt as in the interrupt driven mode. The End of Frame is indicated by the End of Frame interrupt which is generated on reception of the closing flag of the SDLC frame. This End of Frame condition also disables the Receive DMA request. The End of Frame interrupt may be reset by issuing an "Error Reset" command to the MPSC. The "Error Reset" command also re-enables the Receive DMA request. It may be noted that the End of Frame condition sets bits D7 in RR1. This bit gets reset by "Error Reset" command. However, End of Frame bit (RR1:D7) can also be reset by the flag of the next incoming frame. For proper operation, Error Reset Command should be issued "after" the End of Frame Bit (RR1:D7) is set. In a more general case, "Error Reset" command should be issued after End of Frame, Receive over-run or Receive parity bit are set in RR1.

Wait Mode

The wait mode is normally used for block transfer by synchronizing the data transfer through the Ready output from the MPSC, which may be connected to the Ready input of the CPU. The mode can be programmed by WR 1, D7 D5 and may be programmed separately and independently on CH A and CH B. The Wait Mode will be operative if the following conditions are satisfied.

(i) Interrupts are enabled.
(ii) Wait Mode is enabled (WR1: D7)
(iii) CS = 0, Al = 0

The RDY output becomes active when the transmitter buffer is full or receiver buffer is empty. This way the RDY output from the MPSC can be used to extend the CPU read and write cycle by inserting WAIT states. RDY_A or RDY_B are in high impedance state when the corresponding channel is not selected. This makes it possible to connect RDY_A and RDY_B outputs in wired OR configuration. Caution must be exercised here in using the RDY outputs of the MPSC or else the CPU may hang up for indefinite period. For example, let us assume that transmitter buffer is full and RDY_A is active, forcing the CPU into a wait state. If the CTS goes inactive during this period, the RDY_A will remain active for indefinite period and CPU will continue to insert wait states.

Vectored/Non-Vectored Mode

The MPSC is capable of providing an interrupt vector in response to the interrupt acknowledge sequence from the CPU. WR2, CH B contains this vector and the vector can be read in status register RR2. WR2, CH A (bit D5) can program the MPSC in vectored or non-vectored mode. See Figure 14 for details.

In both cases, WR2 may still have the vector stored in it. However, in vectored mode, the MPSC will put the vector on the data bus in response the INTA (Interrupt Acknowledge) sequence as shown in Figure 15. In non-vectored mode, the MPSC will not respond to the INTA sequence. However, the CPU can read the vector by polling Status Register RR2. WR2A, D4 and D3 can be programmed to respond to 8085 or 8086 INTA sequence. It may be noted here that IPI (Interrupt Priority In) pin on the MPSC must be active for the vector to appear on the data bus.

WR2A,D5	Interrupt Mode
0	Non-vectored Interrupt
1	Vectored Interrupt

Figure 14. Vectored Interrupts

Status Affect Vector

The vector stored in WR2B can be modified by the source of the interrupt. This can be done by setting the Status Affect Vector bit (WR1: D2). This powerful feature of the MPSC provides fast interrupt response time, by eliminating the need of writing a routine to read the status of the MPSC. Three bits of the vector are modified in eight different ways as shown on Figure 16. Bits V4,V3,V2 are modified in 8085 based system and bits V2, V1, V0 are modified in 8086/88 based system.

In non-vectored mode, the status affect vector mode can still be used and the vector read by the CPU. Status Register RR2B (Read Register 2 in Channel B) will contain this modified vector.

WR2A D5	D4	D3	IPI	MODE	1ST INTA	2ND INTA	3RD INTA
0	X	X	X	NON-VECTORED	HIGH-Z	HI-Z	HI-Z
1	0	0	0	8085-1	1100 1101	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	0	1	8085-1	1100 1101	HI-Z	HI-Z
1	0	1	0	8085-2	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	0000 0000
1	0	1	1	8085-2	HI-Z	HI-Z	HI-Z
1	1	0	0	8086	HI-Z	V7 V6 V5 V4 V3 V2 V1 V0	—
1	1	0	1	8086	HI-Z	HI-Z	—

Figure 15. MPSC Vectored Interrupts

(8085) V4 V3 V2 (8086) V2 V1 V0	Channel	Interrupt Source
0 0 0	B	Tx BUFFER EMPTY
0 0 1		EXT/STAT CHANGE
0 1 0		RX CHAR AVAILABLE
0 1 1		SPECIAL Rx CONDITION
1 0 0	A	Tx BUFFER EMPTY
1 0 1		EXT/STAT CHANGE
1 1 0		RX CHAR AVAILABLE
1 1 1		SPECIAL Rx CONDITION

Rx Special Condition: Parity Error, Framing Error, Rx Over-run Error, EOF (SDLC)

EXT/STAT Change: Change in Modem Control Pin Status: CTS, DCD, SYNC, EOM, Break/Abort Detection

Figure 16. Status Affect Vector Mode

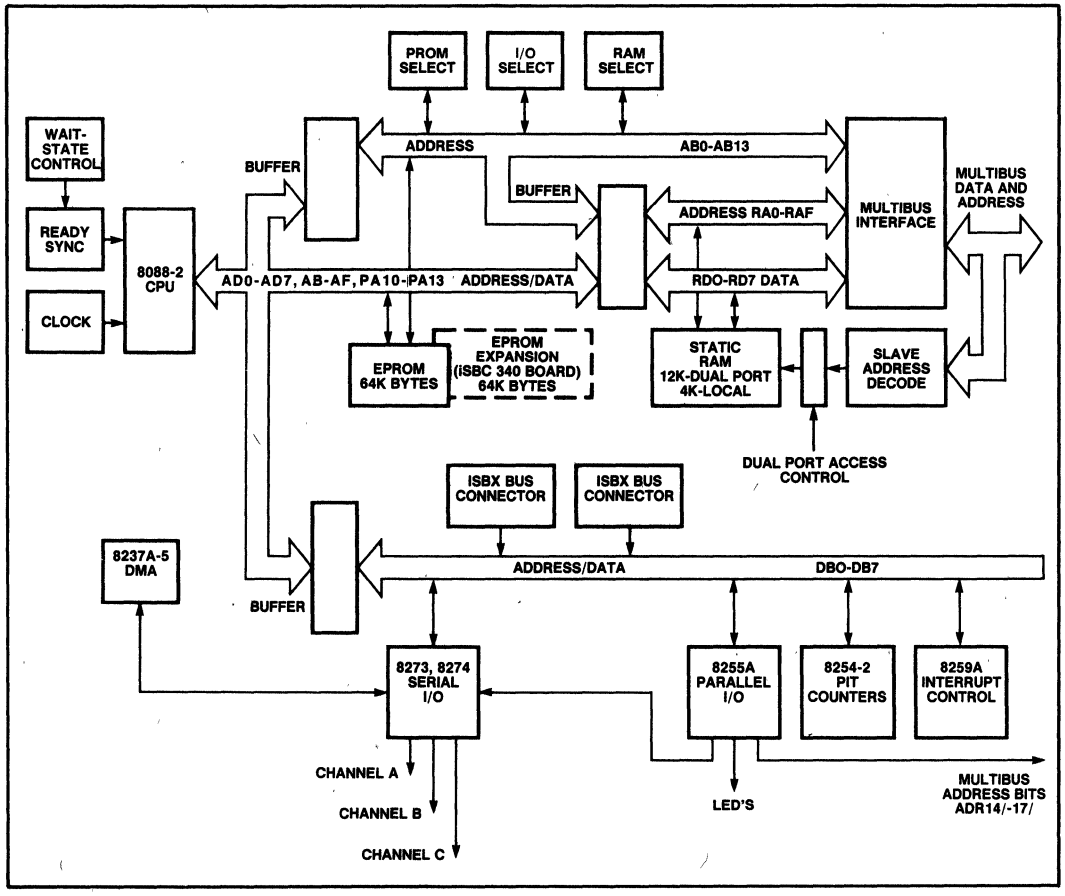


Figure 17. Functional Block Diagram — iSBC® 88/45

APPLICATION EXAMPLE

This section describes the hardware and software of an 8274/8088 system. The hardware vehicle used is the INTEL Single Board Computer iSBC 88/45 - Advanced Communication Controller. The software which exercises the 8274 is written in PLM 86.This example will demonstrate how 8274 can be configured into the SDLC mode and transfer data through DMA control. The hardware example will help the reader configure his hardware and the software examples will help in developing an application software. Most software examples closely approximates a real data link controller software in the SDLC communication and may be used with very little modification.

iSBC® 88/45

A brief description of the iSBC 88/45 board will be presented here. For more detailed information on the board and the schematics, refer to Hardware Manual for the iSBC 88/45, Advanced Communication Controller. iSBC 88/45 is an intelligent slave/multimaster communication board based on the 8088 processor, the 8274 and the 8273 SDLC/HDLC controller. Figure 17 shows the functional block diagram of the board. The iSBC 88/45 has the following features.

- 8 MHz processor
- 16K bytes of static RAM (12K dual port)
- Multimaster/Intelligent Slave Multibus Interface
- Nine Interrupt Levels 8259A
- Two serial channels through 8274
- One Serial channel through 8273
- S/W programmable baud rate generator
- Interfaces: RS 232, RS422/449, CCITT V.24
- 8237A DMA controller
- Baud Rate to 800k Baud

```
INITIALIZE_8274: PROCEDURE PUBLIC;

/*************************************************************************/
/*                                                                       */
/*       INITIALIZE THE 8274 FOR SDLC MODE                               */
/*                                                                       */
/*       1.  RESET CHANNEL                                               */
/*       2.  EXTERNAL INTERRUPTS ENABLED                                 */
/*       3.  NO WAIT                                                     */
/*       4.  PIN 10 = RTS                                                */
/*       5.  NON-VECTORED INTERRUPT-8086 MODE                            */
/*       6.  CHANNEL A DMA, CH B INT                                     */
/*       7.  TX AND RX = 8 BITS/CHAR                                     */
/*       9.  ADDRESS SEARCH MODE                                         */
/*       10. CD AND CTS AUTO ENABLE                                      */
/*       11. X1 CLOCK                                                    */
/*       12. NO PARITY                                                   */
/*       13. SDLC/HDLC MODE                                              */
/*       14. RTS AND DTR                                                 */
/*       15. CCITT - CRC                                                 */
/*       16. TRANSMITTER AND RECEIVER ENABLED                            */
/*       17. 7EH = FLAG                                                  */
/*                                                                       */
/*************************************************************************/

DECLARE C BYTE;

/* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B    */
/* FORMAT IS: WRITE REGISTER, REGISTER DATA        */
/* INITIALIZE CHANNEL A ONLY                       */

DECLARE TABLE_74_A(*) BYTE DATA
       (00H, 18H,          /* CHANNEL RESET */
        00H, 80H,          /* RESET TX CRC */
        02H, 11H,          /* PIN 10=RTSB, A DMA, B INT */
        04H, 20H,          /* SDLC/HDLC MODE, NO PARITY */
        07H, 07EH,         /* SDLC FLAG */
        01H, 0BH,          /* RX DMA ENABLE */
        05H, 0EBH,         /* DTR, RTS, 8 TX BITS, TX ENABLE, */
                           /* SDLC CRC, TX CRC ENABLE */
        06H, 55H,          /* DEFAULT ADDRESS */
        03H, 0D9H,         /* 8 RX BITS, AUTO ENABLES, HUNT MODE, */
                           /* RX CRC ENABLE */
        0FFH);             /* END OF INITIALIZATION TABLE */

DECLARE TABLE_74_B(*) BYTE DATA
        (02H, 00H,         /* INTERRUPT VECTOR */
        01H, 1CH,          /* STATUS AFFECTS VECTOR */
        0FFH);             /* END */

/* INITIALIZE THE 8274 */

C=0;
DO WHILE TABLE_74_B(C) <> 0FFH;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
        OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
        C=C+1;
END;

C=0;
DO WHILE TABLE_74_A(C) <> 0FFH;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
        OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
        C=C+1;
END;
RETURN;
END INITIALIZE_8274;
```

Figure 18. Typical MPSC SDLC Initialization Sequence

For this application, the CPU is run at 8 MHz. The board is configured to operate the 8274 in SDLC operation with the data transfer in DMA mode using the 8237A. 8274 is configured first in non-vectored mode in which case the INTEL Priority Interrupt Controller 8259A is used to resolve priority betweeen various interrupting sources on the board and subsequently interrupt the CPU. However, the vectored mode of the 8274 is also verified by disabling the 8259A and reading the vectors from the 8274. Software examples for each case will be shown later.

The application example is interrupt driven and uses DMA for all data transfers under 8237A control. The 8254 provides the transmit and receive clocks for the 8274. The 8274 was run at 400K baud with a local loopback (jumper wire) on Channel A data.The board was also run at 800K baud by modifying the software as will be discussed later in the Special Applications section. One detail to note is that the Rx Channel DMA request line from the 8274 has higher priority than the Tx Channel DMA request line. The 8274 master clock was 4.0 MHz. The on-board RAM is used to define transmit and receive data buffers. In this application, the data is read from memory location 800H through 810H and transferred to memory location 900H to 910H through the 8274 Serial Link. The operation is full duplex. 8274 modem control pins, $\overline{CTS}$ and $\overline{CD}$ have been tied low (active).

Software

The software consists of a monitor program and a program to exercise the 8274 in the SDLC mode. Appendix A contains the entire program listing. For the sake of clarity, each source module has been rewritten in a simple language and will be discussed here individually. Note that some labels in the actual listings in the Appendix will not match with the labels here. Also the listing in the Appendix sets up some flags to communicate with the monitor. Some of these flags are not explained in detail for the reason that they are not pertinent to this discussion. The monitor takes the command from a keyboard and executes this program, logging any error condition which might occur.

8274 Initialization

The MPSC is initialized in the SDLC mode for Channel A. Channel B is disabled. See Figure 18 for the initialization routine. Note that WR4 is initialized before setting up the transmitter and receive parameters. However, it may also be pointed out that other than WR4, all the other registers may be programmed in any order. Also SDLC-CRC has been programmed for correct operation. An incorrect CRC selection will result in incorrect operation. Also note that receive interrupt on first receive character has been programmed although Channel A is in the DMA mode.

Interrupt Routines

The 8274 interrupt routines will be discussed here. On an 8274 interrupt, program branches off to the "Main Interrupt Routine". In main interrupt routine, status register RR2 is read. RR2 contains the modified vector. The cause of the interrupt is determined by reading the modified bits of the vector. Note that the 8274 has been programmed in the non-vectored mode and status affects vector bit has been set. Depending on the value of the modified bits, the appropriate interrupt routine is called. See Figure 19 for the flow diagram and Figure 20 for the source code. Note that an End of Interrupt Command is issued after servicing the interrupt. This is necessary to enable the lower priority interrupts.

Figure 21 shows all the interrupt routines called by the Main Interrupt Routine. "Ignore - Interrupt" as the name implies, ignores any interrupts and sets the FAIL flag. This is done because this program is for Channel A only and we are ignoring any Channel B interrupts. The important thing to note is the Channel A Receiver Character available routine. This routine is called after receiving the first character in the SDLC frame. Since the transfer mode is DMA, we have a maximum of three character times to service this interrupt by enabling the DMA controller.

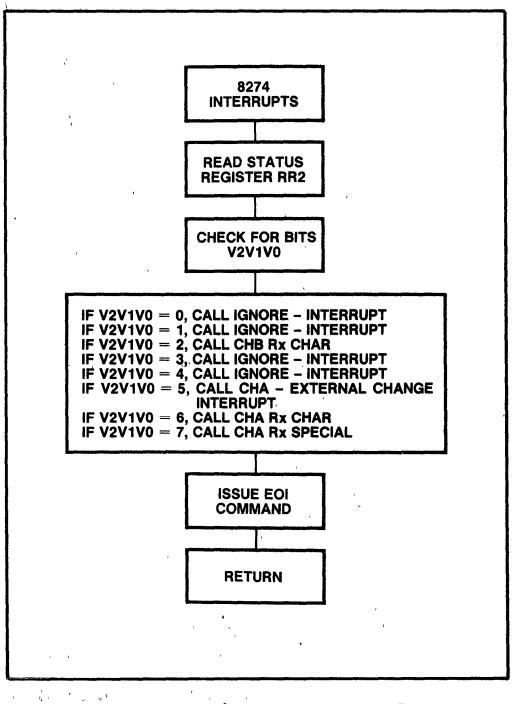


Figure 19. Interrupt Response Flow Diagram

```
/*************************/
/* MAIN INTERRUPT ROUTINE */
/*************************/
OUTPUT(COMMAND_B_74) = 2;                  /* SET POINTER TO 2*/
TEMP = INPUT(STATUS_B_74) AND 07H;         /* READ INTERRUPT VECTOR */
                                           /* CHECK FOR CHA INT ONLY*/
/* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
DO CASE TEMP;
        CALL    IGNORE_INT;                /* V2V1V0 = 000*/
        CALL    IGNORE_INT;                /* V2V1V0 = 001*/
        CALL    CHB_RX_CHAR;               /* V2V1V0 = 010*/
        CALL    IGNORE_INT;                /* V2V1V0 = 011*/
        CALL    IGNORE_INT;                /* V2V1V0 = 100*/
        CALL    CHA_EXTERNAL_CHANGE;       /* V2V1V0 = 101*/
        CALL    CHA_RX_CHAR;               /* V2V1V0 = 110*/
        CALL    CHA_RX_SPECIAL;            /* V2V1V0 = 111*/
END;
OUTPUT(COMMAND_A_74) =38H;       /* END OF INTERRUPT FOR 8274 */
RETURN;
END INTERRUPT_8274;
```

Figure 20. Typical Main Interrupt Routine

```
/*******************************************************/
/* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
/*******************************************************/
CHA_EXTERNAL_CHANGE: PROCEDURE;

TEMP = INPUT(STATUS_A_74);       /* STATUS REG 1*/
IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
        TXDONE_S=DONE;
ELSE DO;
        TXDONE_S=DONE;
        RESULTS_S=FAIL;
     END;
OUTPUT(COMMAND_A_74) = 10H;      /* RESET EXT/STATUS INTERRUPTS */
RETURN;
END CHA_EXTERNAL_CHANGE;
/*************************************************************/
/* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
/*************************************************************/
CHA_RX_SPECIAL: PROCEDURE;

        OUTPUT(COMMAND_A_74) = 1;
        TEMP = INPUT(STATUS_A_74);
        IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
              DO;
                IF(TEMP AND 040H) = 040H THEN
                  RESULTS_S = FAIL;          /* CRC ERROR */
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
              END;
              ELSE DO;
                IF (TEMP AND 20H) = 20H THEN DO;
                RESULTS_S = FAIL;            /* RX OVERRUN ERROR*/
                RXDONE_S = DONE;
                OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
                END;
              END;
RETURN;
END CHA_RX_SPECIAL;

/*************************************/
/* CHANNEL A RECEIVE CHARACTER AVAILABLE */
/*************************************/
CHA_RX_CHAR: PROCEDURE;
OUTPUT(SINGLE_MASK) = CH0_SEL;           /*ENABLE RX DMA CHANNEL*/
RETURN;
END CHA_RX_CHAR;
```

Figure 21. 8274 Typical Interrupt Handling Routines

It may be recalled that the receiver buffer is three bytes deep in addition to the receiver shift register. At very high data rates, it may not be possible to have enough time to read RR2, enable the DMA controller without overrunning the receiver. In a case like this, the DMA controller may be left enabled before receiving the Receive Character Interrupt. Remember, the Rx DMA request and interrupt for the receive character appears at the same time. If the DMA controller is enabled, it would service the DMA request by reading the received character. This will make the 8274 interrupt line go inactive. However, the 8259A has latched the interrupt and a regular interrupt acknowledge sequence still occurs after the DMA controller has completed the transfer and given up the bus. The 8259A will return Level 7 interrupt since the 8274 interrupt has gone away. The user software must take this into account, otherwise the CPU will hang up.

The procedure shown for the Special Receive Condition Interrupt checks if the interrupt is due to the End of Frame. If this is not TRUE, the FAIL flag is set and the program aborted. For a real life system, this must be followed up by error recovery procedures which obviously are beyond the scope of this Application Note.

The transmission is terminated when the End of Message (RR0, D6) interrupt is generated. This interrupt is serviced in the Channel A External/Status Change interrupt procedure. For any other change in external status conditions, the program is aborted and a FAIL flag set.

Main Program

Finally, we will briefly discuss the main program. Figure 22 shows the source program. It may be noted that the Transmit Under-run latch is reset after loading the first character into the 8274. This is done to ensure CRC transmission at the end of the frame. Also, the first character is loaded from the CPU to start DMA transfer of subsequent data. This concludes our discussion on hardware and software example. Appendix A also includes the software written to exercise the 8274 in the vectored mode by disabling the 8259A.

```
CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;

        CALL    ENABLE_INTERRUPTS_S;
        CALL    INIT_8274_SDLC_S;
        ENABLE;
        OUTPUT(COMMAND_A_74) = 28H;   /* RESET TX INT/DMA          */
        OUTPUT(COMMAND_B_74) = 28H;   /* BEFORE INITIALIZING 8237*/
        CALL    INIT_8237_S;
        OUTPUT(DATA_A_74) = 55H;      /*LOAD FIRST CHARACTER FROM */
                                      /*CPU */
        /* TO ENSURE CRC TRANSMISSION, RESET TX UNDERRUN LATCH   */
        OUTPUT(COMMAND_A_74) = 0C0H;
        RXDONE_S, TXDONE_S=NOT_DONE;  /* CLEAR ALL FLAGS           */
        RESULTS_S=PASS;               /* FLAG SET FOR MONITOR      */
        DO WHILE TXDONE_S=NOT_DONE;   /* DO UNTIL TERMINAL COUNT   */
        END;

        DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
        /* WAIT FOR CRC TO GET TRANSMITTED */
        /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
        END;
        DO WHILE RXDONE_S=NOT_DONE;   /* DO UNTIL TERMINAL COUNT   */
        END;
        CALL    STOP_8237_S;
END CHA_SDLC_TEST;
```

Figure 22. Typical 8274 Transmit/Receive Set-Up in SDLC Mode

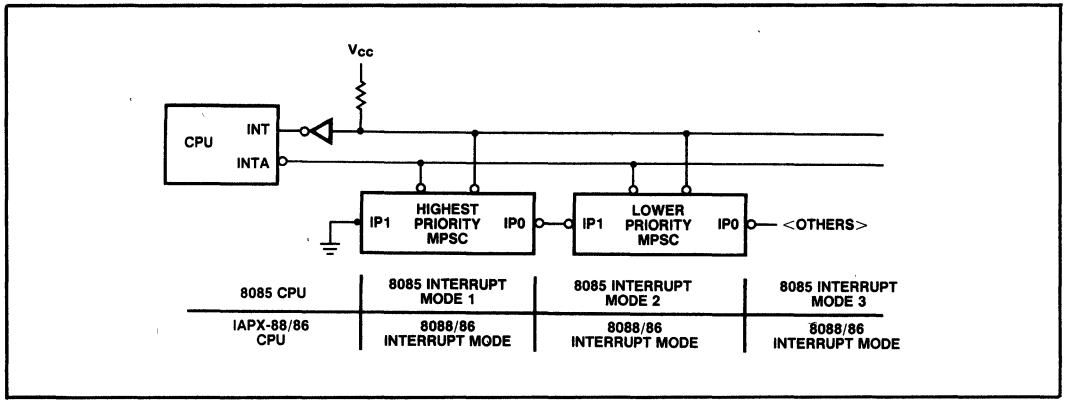


Figure 23. 8274 Daisy Chain Vectored Mode

SPECIAL APPLICATIONS

In this section, some special application issues will be discussed. This will be useful to a user who may be using a mode which is possible with the 8274 but not explicitly explained in the data sheet.

MPSC Daisy Chain Operation

Multiple MPSC can be connected in a daisy-chain configuration (see Figure 23). This feature may be useful in an application where multiple communication channels may be required and because of high data rates, conventional interrupt controller is not used to avoid long interrupt response times. To configure the MPSCs for the daisy chain operation, the interrupt priority input pins (IPI) and interrupt priority output pins (IPO) of the MPSC should be connected as shown. The highest priority device has its IPI pin connected to ground. Each MPSC is programmed in a vectored mode with status affects vector bit set. In the 8085 basic systems, only one MPSC should be programmed in the 8085 Mode 1. This is the MPSC which will put the call vector (CD Hex) on the data bus in response to the first INTA pulse (See Figure 15). It may be pointed out that the MPSC in 8085 Mode 1 will provide the call vector irrespective of the state of IPI pin. Once a higher priority MPSC generates an interrupt, its IPO pin goes inactive thus preventing lower priority MPSCs from interrupting the CPU. Preferably the highest priority MPSC should be programmed in 8085 Mode 1. It may be recalled that the Priority Resolve Time on a given MPSC extends from th falling edge of the first INTA pulse to the falling edge of the second INTA pulse. During this period, no new internal interrupt requests are accepted. The maximum number of the MPSCs that can be connected in a daisy chain is limited by the Priority Resolution Time. Figure 24 shows a maximum number of MPSCs that can be connected in various CPU systems. It may be pointed out that IPO to IPI delay time specification is 100ns.

Bisync Transparent Communication

Bisync applications generally require that data transparency be established during communication. This requires that the special control characters may not be included in the CRC accumulation. Refer to the Synchronous Protocol Overview section for a more detailed discussion on data transparency. The 8274 can be used for transparent communication in Bisync communications. This is made

System Configuration	Priority Resolution Time Min (ns)	Number of 8274s Daisy Chained (Max)
8086-1	400	4
8086-2	500	5
8086	800	8
8088	800	8
8085-2	1200	12
8085A	1920	19

Note: Zero wait states have been assumed.

Figure 24. 8274 Daisy Chain Operation

possible by the capability of the MPSC to selectively turn-on/turnoff the CRC accumulation while transmitting or receiving. In bisync transparent transmit mode, the special characters (DLE, DLE SYN etc) are excluded from CRC calculation. This can be easily accomplished by turning off the transmit CRC calculation (WR5: D5=0) before loading the special character into the transmit buffer. If the next character is to be included in the CRC accumulation, then the CRC can be enabled (WR5: D5=1). See Figure 25 for a typical flow diagram.

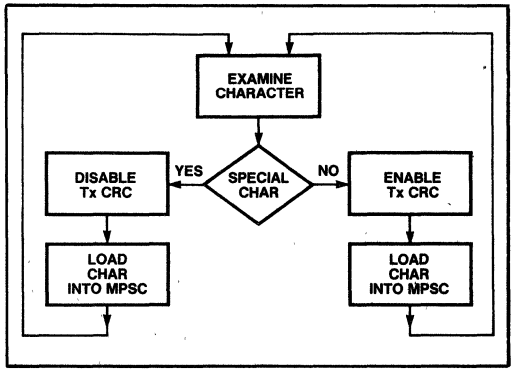


Figure 25. Transmit in Bisync Transparent Mode

During reception, it is possible to exclude received character from CRC calculation by turning off the Receive CRC after reading the special character. This is made possible by the fact that the received data is presented to receive CRC checker 8 bit times after the character has been received. During this 8 bit times, the CPU must read the character and decide if it wants it to be included in the CRC calculation. Figure 26 shows the typical flow diagram to achieve this.

It should be noted that the CRC generator must be enabled during CRC reception. Also, after reading the CRC bytes, two more characters (SYNC) must be read before checking for CRC check result in RR1.

Auto Enable Mode

In some data communication applications, it may be required to enable the transmitter or the receiver when the CTS or the DCD lines respectively, are activated by the modems. This may be done very easily by programming the 8274 into the Auto Enable Mode.The auto enable mode is set by writing a '1' to WR3,D5. The function of this mode is to enable the transmitter automatically when $\overline{CTS}$ goes active. The receiver is enabled when DCD goes active. An in-active state of $\overline{CTS}$ or $\overline{DCD}$ pin will disable the transmitter or the receiver respectively. However, the Transmit Enable bit (WR5:D3) and Receive Enable bit

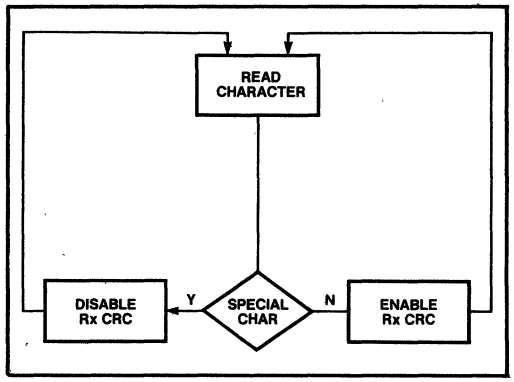


Figure 26. Receive in Bisync Transparent Mode

(WR3:D1) must be set in order to use the auto enable mode. In non-auto mode, the transmitter or receiver is enabled if the corresponding bits are set in WR5 and WR3, irrespective of the state $\overline{CTS}$ or $\overline{DCD}$ pins. It may be recalled that any transition on $\overline{CTS}$ or $\overline{DCD}$ pin will generate External/Status Interrupt with the corresponding bits set in RR1. This interrupt can be cleared by issuing a Reset External/Status interrupt command as discussed earlier.

Note that in auto enable mode, the character to be transmitted must be loaded into the transmit buffer after the $\overline{CTS}$ becomes active, not before. Any character loaded into the transmit buffer before the $\overline{CTS}$ became active will not be transmitted.

High Speed DMA Operation

In the section titled Application Example, the MPSC has been programmed to operate in DMA mode and receiver is programmed to generate an interrupt on the first receive character. You may recall that the receive FIFO is three bytes deep. On receiving the interrupt on the first receive character, the CPU must enable the DMA controller within three received byte times to avoid receiver over-run condition. In the application example, at 400K baud, the CPU had approximately 60 μs to enable the DMA controller to avoid receiver buffer overflow. However, at higher baud rates, the CPU may not have enough time to enable the DMA controller in time. For example, at 1M baud, the CPU should enable the DMA controller within approximately 24 μs to avoid receiver buffer overrun. In most applications, this is not sufficient time. To solve this problem, the DMA controller should be left enabled before getting the interrupt on the first receive character (which is accompanied by the Rx DMA request for the appropriate channel). This will allow the DMA controller to start DMA transfer as soon as the Rx DMA request becomes active without giving the CPU enough time to re-

spond to the interrupt on the first receive character. The CPU will respond to the interrupt *after* the DMA transfer has been completed and will find the 8259A (See Application Example) responding with interrupt level 7, the lowest priority level. Note that the 8274 interrupt request was satisfied by the DMA controller, hence the interrupt on the first receive character was cleared and the 8259A had no pending interrupt. Because of no pending interrupt, the 8259A returned interrupt level 7 in response to the INTA sequence from the CPU. The user software should take care of this interrupt.

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1:D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit underrun/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3:D1 = 1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI Command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

APPENDIX A
APPLICATION EXAMPLE: SOFTWARE LISTINGS

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8274_S
OBJECT MODULE PLACED IN :F1:SINI74.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI74.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

                /*************************************************/
                /*                                               */
                /*       INITIALIZE THE 8274 FOR SDLC MODE       */
                /*                                               */
                /*       1. RESET CHANNEL                        */
                /*       2. EXTERNAL INTERRUPTS ENABLED          */
                /*       3. NO WAIT                              */
                /*       4. PIN 10 = RTS                         */
                /*       5. NON-VECTORED INTERRUPT-8086 MODE     */
                /*       6. CHANNEL A DMA, CH B INT              */
                /*       7. TX AND RX = 8 BITS/CHAR              */
                /*       9. ADDRESS SEARCH MODE                  */
                /*       10. CD AND CTS AUTO ENABLE              */
                /*       11. X1 CLOCK                            */
                /*       12. NO PARITY                           */
                /*       13. SDLC/HDLC MODE                      */
                /*       14. RTS AND DTR                         */
                /*       15. CCITT - CRC                         */
                /*       16. TRANSMITTER AND RECEIVER ENABLED    */
                /*       17. 7EH = FLAG                          */
                /*                                               */
                /*************************************************/

  1             INIT_8274_S: DO;

                $INCLUDE (:F1:PORTS.PLM)

          =     /*********************************************/
          =     /*                                           */
          =     /*       iSBC 88/45 PORT ASSIGNMENTS         */
          =     /*                                           */
          =     /*********************************************/

  2   1   =     DECLARE LIT LITERALLY 'LITERALLY';

          =     /* 8237A-5  PORTS */

  3   1   =     DECLARE CH0_ADDR          LIT     '080H',
          =             CH0_COUNT         LIT     '081H',
          =             CH1_ADDR          LIT     '082H',
          =             CH1_COUNT         LIT     '083H',
          =             CH2_ADDR          LIT     '084H',
          =             CH2_COUNT         LIT     '085H',
          =             CH3_ADDR          LIT     '086H',
          =             CH3_COUNT         LIT     '087H',
          =             STATUS_37         LIT     '088H',
          =             COMMAND_37        LIT     '088H',
          =             REQUEST_REG_37    LIT     '089H',
          =             SINGLE_MASK       LIT     '08AH',
          =             MODE_REG_37       LIT     '08BH',

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


          =             CLR_BYTE_PTR_37   LIT     '08CH',
          =             TEMP_REG_37       LIT     '08DH',
          =             MASTER_CLEAR_37   LIT     '08DH',
          =             ALL_MASK_37       LIT     '08FH';

          =     /* 8254-2 PORTS */

  4   1   =     DECLARE CTR_00            LIT     '090H',
          =             CTR_01            LIT     '091H',
          =             CTR_02            LIT     '092H',
```

```
                =           CONTROLO_54     LIT     '093H',
                =           STATUSO_54      LIT     '093H',
                =           CTR_10          LIT     '098H',
                =           CTR_11          LIT     '099H',
                =           CTR12           LIT     '09AH',
                =           CONTROL1_54     LIT     '09BH',
                =           STATUS1_54      LIT     '09BH';

                =   /* 8255 PORTS */

  5     1       =   DECLARE PORTA_55        LIT     '0A0H',
                =           PORTB_55        LIT     '0A1H',
                =           PORTC_55        LIT     '0A2H',
                =           CONTROL_55      LIT     '0A3H';

                =   /* 8274 PORTS */

  6     1       =   DECLARE DATA_A_74       LIT     '0D0H',
                =           DATA_B_74       LIT     '0D1H',
                =           STATUS_A_74     LIT     '0D2H',
                =           COMMAND_A_74    LIT     '0D2H',
                =           STATUS_B_74     LIT     '0D3H',
                =           COMMAND_B_74    LIT     '0D3H';

                =   /* 8259A PORTS */

  7     1       =   DECLARE STATUS_POLL_59  LIT     '0E0H',
                =           ICW1_59         LIT     '0E0H',
                =           OCW2_59         LIT     '0E0H',
                =           OCW3_59         LIT     '0E0H',
                =           OCW1_59         LIT     '0E1H',
                =           ICW2_59         LIT     '0E1H',
                =           ICW3_59         LIT     '0E1H',
                =           ICW4_59         LIT     '0E1H';

                =   /* 8274 REGISTER BIT ASSIGNMENTS */
                =   /* READ REGISTER 0 */

  8     1       =   DECLARE RX_AVAIL        LIT     '01H',
                =           INT_PENDING     LIT     '02H',
                =           TX_EMPTY        LIT     '04H',
                =           CARRIER_DETECT  LIT     '08H',
                =           SYNC_HUNT       LIT     '10H',
                =           CLEAR_TO_SEND   LIT     '20H',

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

                =           END_OF_TX_MESSAGE LIT   '40H',
                =           BREAK_ABORT     LIT     '80H';

                =   /* READ REGISTER 1 */

  9     1       =   DECLARE ALL_SENT        LIT     '01H',
                =           PARITY_ERROR    LIT     '10H',
                =           RX_OVERRUN      LIT     '20H',
                =           CRC_ERROR       LIT     '40H',
                =           END_OF_FRAME    LIT     '80H';

                =   /* READ REGISTER 2 */

 10     1       =   DECLARE TX_B_EMPTY      LIT     '00H',
                =           EXT_B_CHANGE    LIT     '01H',
                =           RX_B_AVAIL      LIT     '02H',
                =           RX_B_SPECIAL    LIT     '03H',
                =           TX_A_EMPTY      LIT     '04H',
                =           EXT_A_CHANGE    LIT     '05H',
                =           RX_A_AVAIL      LIT     '06H',
                =           RX_A_SPECIAL    LIT     '07H';
```

```
          =   /* 8237 BIT ASSIGNMENTS */

   11   1  =   DECLARE CH0_SEL          LIT     '00H',
          =           CH1_SEL          LIT     '01H',
          =           CH2_SEL          LIT     '02H',
          =           CH3_SEL          LIT     '03H',
          =           WRITE_XFER       LIT     '04H',
          =           READ_XFER        LIT     '08H',
          =           DEMAND_MODE      LIT     '00H',
          =           SINGLE_MODE      LIT     '40H',
          =           BLOCK_MODE       LIT     '80H',
          =           SET_MASK         LIT     '04H';

   12   1      DELAY_S: PROCEDURE PUBLIC;
   13   2      DECLARE D WORD;
   14   2      D=0;
   15   2      DO WHILE D<800H;
   16   3      D=D+1;
   17   3      END;
   18   2      END DELAY_S;


   19   1      INIT_8274_SDLC_S:    PROCEDURE PUBLIC;

   20   2      DECLARE C    BYTE;

               $EJECT



PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


               /* TABLE TO INITIALIZE THE 8274 CHANNEL A AND B */

               /*  FORMAT IS: WRITE REGISTER, REGISTER DATA  */
               /*       INITIALIZE CHANNEL ONLY                    */

   21   2      DECLARE TABLE_74_A(*) BYTE DATA
                       (00H, 18H,          /* CHANNEL RESET */
                       00H, 80H,           /* RESET TX CRC */
                       02H, 11H,           /* PIN 10=RTSB, A DMA, B INT */
                       04H, 20H,           /* SDLC/HDLC MODE, NO PARITY */
                       07H, 07EH,          /* SDLC FLAG */
                       01H, 0BH,           /* RX DMA ENABLE */
                       05H, 0EBH,          /* DTR, RTS, 8 TX BITS, TX ENABLE, TX CRC ENABLE */
                       06H, 55H,           /* DEFAULT ADDRESS */
                       03H, 0D9H,          /* 8 RX BITS, AUTO ENABLES, HUNT MODE, */
                                           /* RX CRC ENABLE */
                       0FFH);              /* END OF INITIALIZATION TABLE */

   22   2      DECLARE TABLE_74_B(*) BYTE DATA
                       (02H, 00H,          /* INTERRUPT VECTOR */
                       01H, 1CH,           /* STATUS AFFECTS VECTOR */
                       0FFH);              /* END */


               /* INITIALIZE THE 8254 */

   23   2      OUTPUT(CONTROL0_54)=36H;
   24   2      OUTPUT(CTR_00) = LOW(20);       /* BAUD RATE = 400K_BAUD*/
   25   2      OUTPUT(CTR_00) = HIGH(20);      /* BAUD RATE = 400K_BAUD*/


               /* INITIALIZE THE 8274 */

   26   2      C=0;
   27   2      DO WHILE TABLE_74_B(C) <> 0FFH;
   28   3          OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
   29   3          C=C+1;
   30   3          OUTPUT(COMMAND_B_74) = TABLE_74_B(C);
   31   3          C=C+1;
   32   3      END;
```

```
  33   2       C=0;
  34   2       DO WHILE TABLE_74_A(C) <> OFFH;
  35   3           OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
  36   3           C=C+1;
  37   3           OUTPUT(COMMAND_A_74) = TABLE_74_A(C);
  38   3           C=C+1;
  39   3       END;
  40   2           CALL    DELAY_S;

  41   2       RETURN;
  42   2       END INIT_8274_SDLC_S;
  43   1       END INIT_8274_S;
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

MODULE INFORMATION:

     CODE AREA SIZE      = 00A8H    168D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 0003H      3D
     MAXIMUM STACK SIZE  = 0006H      6D
     213 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST

SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INIT_8237_CHA
OBJECT MODULE PLACED IN :F1:SINI37.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:SINI37.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC
TEST) COMPACT NOINTVECTOR ROM

                /*************************************************************************/
                /*                                                                       */
                /*      8237    INITIALIZATION ROUTINE  FOR DMA TRANSFER                 */
                /*                                                                       */
                /*************************************************************************/

   1            INIT_8237_CHA:  DO;

                $NOLIST

  12    1       INIT_8237_S:  PROCEDURE PUBLIC;


  13    2       OUTPUT(MASTER_CLEAR_37)=0;
  14    2       OUTPUT(COMMAND_37) = 20H;        /* EXTENDED WRITE */
  15    2       OUTPUT(ALL_MASK_37) = 0FH;       /* MASK ALL REQUESTS */
  16    2       OUTPUT(MODE_REG_37) = (SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
  17    2       OUTPUT(MODE_REG_37) = (SINGLE_MODE OR READ_XFER OR CH1_SEL);
  18    2       OUTPUT(CLR_BYTE_PTR_37) = 0;
  19    2       OUTPUT(CH0_ADDR) = 00;       /* RECEIVE BUFF AT 900H */
  20    2       OUTPUT(CH0_ADDR) = 09H;
  21    2       OUTPUT(CH0_COUNT) = 0H;
  22    2       OUTPUT(CH0_COUNT) = 01;
  23    2       OUTPUT(CH1_ADDR) = 00;       /* TRANSMIT BUFF AT 800H */
  24    2       OUTPUT(CH1_ADDR) = 08H;
  25    2       OUTPUT(CH1_COUNT) = 010H;
  26    2       OUTPUT(CH1_COUNT) = 00H;
```

```
                /* ENABLE TRANSFER */
 27    2        OUTPUT(SINGLE_MASK) = CH1_SEL;     /* ENABLE TX DMA */
 28    2        RETURN;

 29    2        END INIT_8237_S;

                /* TURN OFF THE 8237 CHANNELS 0 AND 1 */

 30    1        STOP_8237_S: PROCEDURE PUBLIC;
 31    2        OUTPUT(SINGLE_MASK) = CH1_SEL OR SET_MASK;
 32    2        OUTPUT(SINGLE_MASK) = CH0_SEL OR SET_MASK;
 33    2        RETURN;
 34    2        END STOP_8237_S;
 35    1        END INIT_8237_CHA;


MODULE INFORMATION:

     CODE AREA SIZE      = 004CH     76D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 0000H      0D



PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


     MAXIMUM STACK SIZE = 0002H      2D
     163 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION


PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE INTR_8274_S
OBJECT MODULE PLACED IN .F1:SINTR.OBJ
COMPILER INVOKED BY:  PLM86 86.F1 SINTR.PLM TITLE(iSBC 88/45 8274 CHANNEL
A SDLC TEST) COMPACT NOINTVECTOR ROM

                /*****************************************/
                /*                                       */
                /*      8274 INTERRUPT ROUTINE           */
                /*                                       */
                /*****************************************/

   1            INTR_8274_S: DO;
                $NOLIST
  12    1       DECLARE TEMP BYTE;
  13    1       DECLARE (RESULTS_S,TXDONE_S,RXDONE_S) BYTE EXTERNAL;
  14    1       DECLARE INT_VEC POINTER AT (140);
  15    1       DECLARE INT_VEC_STORE POINTER;
  16    1       DECLARE MASK_59 BYTE;
  17    1       DECLARE DONE            LIT     'OFFH',
                        NOT_DONE        LIT     '00H',
                        PASS            LIT     'OFFH',
                        FAIL            LIT     '00H';

                /*****************************/
                /* IGNORE INTERRUPT HANDLER */
                /*****************************/

  18    1       IGNORE_INT: PROCEDURE;

  19    2       RESULTS_S = FAIL;
  20    2       RETURN;
  21    2       END IGNORE_INT;
```

```
                /*******************************************************/
                /* CHANNEL A EXTERNAL/STATUS CHANGE INTERRUPT HANDLER */
                /*******************************************************/

  22    1       CHA_EXTERNAL_CHANGE: PROCEDURE;

  23    2       TEMP = INPUT(STATUS_A_74);          /* STATUS REG 1*/
  24    2       IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
  25    2           TXDONE_S=DONE;
  26    2       ELSE DO;
  27    3           TXDONE_S=DONE;
  28    3           RESULTS_S=FAIL;
  29    3            END;
  30    2       OUTPUT(COMMAND_A_74) = 10H; /* RESET EXT/STATUS INTERRUPTS */
  31    2       RETURN;
  32    2       END CHA_EXTERNAL_CHANGE;

                $EJECT

PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

                /*******************************************************/
                /* CHANNEL A SPECIAL RECEIVE CONDITIONS INTERRUPT HANDLER */
                /*******************************************************/

  33    1       CHA_RX_SPECIAL: PROCEDURE;

  34    2           OUTPUT(COMMAND_A_74) = 1;
  35    2           TEMP = INPUT(STATUS_A_74);
  36    2           IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
  37    2                       DO;
  38    3                         IF(TEMP AND 040H) = 040H THEN
  39    3                           RESULTS_S = FAIL;          /* CRC ERROR */
  40    3                         RXDONE_S = DONE;
  41    3                         OUTPUT(COMMAND_A_74) = 30H; /*ERROR RESET*/
  42    3                       END;
  43    2                       ELSE DO;
  44    3                         IF (TEMP AND 20H) = 20H THEN DO;
  46    4                         RESULTS_S = FAIL;          /* RX OVERRUN ERROR*/
  47    4                         RXDONE_S = DONE;
  48    4                         OUTPUT(COMMAND_A_74) = 30H, /*ERROR RESET*/
  49    4                         END;
  50    3                       END;
  51    2       RETURN;
  52    2       END CHA_RX_SPECIAL;


                /*****************************************/
                /* CHANNEL A RECEIVE CHARACTER AVAILABLE */
                /*****************************************/

  53    1       CHA_RX_CHAR: PROCEDURE;
  54    2       OUTPUT(SINGLE_MASK) = CH0_SEL,          /*ENABLE RX DMA CHANNEL*/
  55    2       RETURN;
  56    2       END CHA_RX_CHAR;

                $EJECT


PL/M-86 COMPILER    iSBC 88/45 8274 CHANNEL A SDLC TEST

                /* ENABLE 8274 INTERRUPTS - SET UP THE 8259A */

  57    1       ENABLE_INTERRUPTS_S: PROCEDURE PUBLIC;

  58    2       DECLARE CHA_INT_ON LIT '0F7H';

  59    2       DISABLE;

  60    2       CALL    SET$INTERRUPT(39, INT_39);
```

```
 61   2      INT_VEC_STORE = INT_VEC;
 62   2      INT_VEC = INTERRUPT$PTR(INT_8274_S),
 63   2      MASK_59 = INPUT(OCW1_59),

 64   2      OUTPUT(OCW1_59) = MASK_59 AND CHA_INT_ON,

 65   2      RETURN,
 66   2      END ENABLE_INTERRUPTS_S,


              /* DISABLE 8274 INTERRUPTS - SET UP THE 8259A */

 67   1      DISABLE_INTERRUPTS_S  PROCEDURE PUBLIC,

 68   2      DISABLE,

 69   2      INT_VEC = INT_VEC_STORE,

 70   2      OUTPUT(OCW1_59) = MASK_59,
 71   2      ENABLE,

 72   2      RETURN,
 73   2      END DISABLE_INTERRUPTS_S,


              /* CHANNEL B RECEIVE CHARACTER AVAILABLE */

 74   1      CHB_RX_CHAR  PROCEDURE,

 75   2      TEMP=INPUT(DATA_B_74),

 76   2      OUTPUT(COMMAND_B_74) = 38H;
 77   2      RETURN,
 78   2      END CHB_RX_CHAR,


              $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


              /**************************/
              /* MAIN INTERRUPT ROUTINE */
              /**************************/

 79   1      INT_8274_S: PROCEDURE INTERRUPT 35 PUBLIC;

 80   2      OUTPUT(COMMAND_B_74) = 2,                /* SET POINTER TO 2*/
 81   2      TEMP = INPUT(STATUS_B_74) AND 07H,       /* READ INTERRUPT VECTOR */
                                                      /* CHECK FOR CHA INT ONLY*/
              /* FOR THIS APPLICATION CH B INTERRUPTS ARE IGNORED*/
 82   2      DO CASE TEMP,
 83   3          CALL     IGNORE_INT;              /* V2V1V0 = 000*/
 84   3          CALL     IGNORE_INT;              /* V2V1V0 = 001*/
 85   3          CALL     CHB_RX_CHAR,             /* V2V1V0 = 010*/
 86   3          CALL     IGNORE_INT;              /* V2V1V0 = 011*/
 87   3          CALL     IGNORE_INT;              /* V2V1V0 = 100*/
 88   3          CALL     CHA_EXTERNAL_CHANGE;     /* V2V1V0 = 101*/
 89   3          CALL     CHA_RX_CHAR;             /* V2V1V0 = 110*/
 90   3          CALL     CHA_RX_SPECIAL;          /* V2V1V0 = 111*/
 91   3      END,
 92   2      OUTPUT(COMMAND_A_74) =38H,   /* END OF INTERRUPT FOR 8274 */
 93   2      OUTPUT(OCW2_59) = 63H;       /* 8259 EOI */
 94   2      OUTPUT(OCW1_59) = INPUT(OCW1_59) AND 0F7H;
 95   2      RETURN;
 96   2      END INT_8274_S,



              /* DEFAULT INTERRUPT ROUTINE - 8259A INTERRUPT 7 */
              /* REQUIRED ONLY WHEN DMA CONTROLLER IS ENABLED  */
              /* BEFORE RECEIVING FIRST CHARACTER WHICH IS     */
              /* AT HIGH BAUD RATES  LIKE 800K BAUD  READ APP  */
              /* NOTE SECTION 6 FOR DETAILS                    */
```

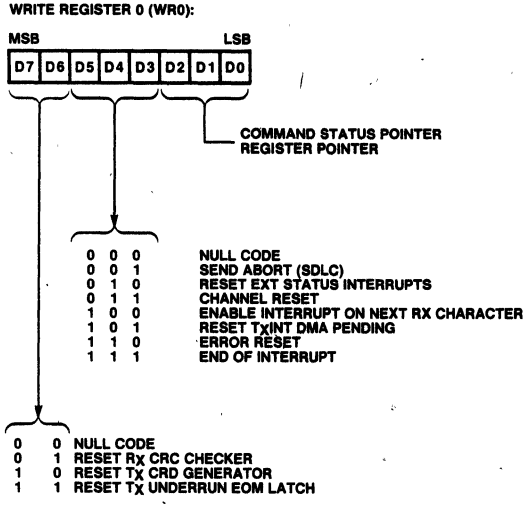
```
 97   1       INT_39: PROCEDURE INTERRUPT 39;
 98   2               OUTPUT(OCW2_59) = 20H;     /* NON-SPECIFIC EOI */
 99   2               OUTPUT(OCW1_59) = INPUT(OCW1_59) AND OF7H;
100   2               RESULTS_S = FAIL;
101   2       END INT_39;

102   1       END INTR_8274_S;


MODULE INFORMATION:

     CODE AREA SIZE     = 01BFH    447D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0006H      6D
     MAXIMUM STACK SIZE = 0022H     34D
     295 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE STEST
OBJECT MODULE PLACED IN :F1:STEST.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:STEST.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)
COMPACT NOINTVECTOR ROM

              /*************************************************************************/
              /*                                                                       */
              /*       iSBC 545 PORT A (8274) SDLC TEST                                */
              /*                                                                       */
              /*************************************************************************/

  1           STEST: DO;


  2   1       DELAY_S: PROCEDURE EXTERNAL;
  3   2       END DELAY_S;

  4   1       ENABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  5   2       END ENABLE_INTERRUPTS_S;

  6   1       DISABLE_INTERRUPTS_S: PROCEDURE EXTERNAL;
  7   2       END DISABLE_INTERRUPTS_S;

  8   1       INIT_8274_SDLC_S: PROCEDURE EXTERNAL;
  9   2       END INIT_8274_SDLC_S;

 10   1       INIT_8237_S: PROCEDURE EXTERNAL;
 11   2       END INIT_8237_S;

 12   1       STOP_8237_S: PROCEDURE EXTERNAL;
 13   2       END STOP_8237_S;

 14   1       VERIFY_TRANSFER_S: PROCEDURE EXTERNAL;
 15   2       END VERIFY_TRANSFER_S;

 16   1       INT_8274_S: PROCEDURE INTERRUPT 35 EXTERNAL;
 17   2       END INT_8274_S;
              $NOLIST
              $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


 28   1       DECLARE (RESULTS_S,TXDONE_S,RXDONE_S) BYTE PUBLIC;
 29   1       DECLARE DONE            LIT     'OFFH',
                      NOT_DONE        LIT     '00H',
                      PASS            LIT     'OFFH',
                      FAIL            LIT     '00H';
```

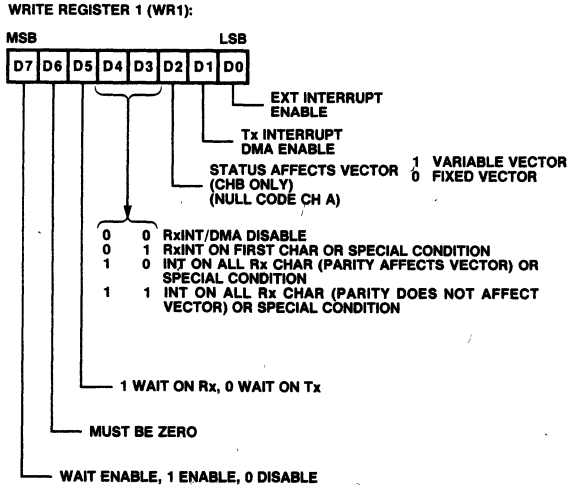
```
          $EJECT

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


  30   1       CHA_SDLC_TEST: PROCEDURE BYTE PUBLIC;


  31   2               CALL    ENABLE_INTERRUPTS_S;
  32   2               CALL    INIT_8274_SDLC_S;
  33   2               ENABLE;
  34   2               OUTPUT(COMMAND_A_74) = 28H;   /* RESET TX INT/DMA         */
  35   2               OUTPUT(COMMAND_B_74) = 28H;   /* BEFORE INITIALIZING 8237*/
  36   2               CALL    INIT_8237_S;
  37   2               OUTPUT(DATA_A_74) = 55H;   /* LOAD FIRST CHARACTER FROM CPU*/

                        /* TO ENSURE CRC TRANSMISSION RESET TX UNDERRUN LATCH*/
  38   2               OUTPUT(COMMAND_A_74) = 0C0H;
  39   2               RXDONE_S, TXDONE_S=NOT_DONE;        /* CLEAR ALL FLAGS */
  40   2               RESULTS_S=PASS;                     /* FLAG SET FOR MONITOR*/

  41   2               DO WHILE TXDONE_S=NOT_DONE;         /* DO UNTIL TERMINAL COUNT*/
  42   3               END;

  43   2               DO WHILE(INPUT(STATUS_A_74) AND 04H) <> 04H;
                        /* WAIT FOR CRC TO GET TRANSMITTED */
                        /* TEST FOR TX BUFFFER EMPTY TO VERIFY THIS*/
  44   3               END;
  45   2               DO WHILE RXDONE_S=NOT_DONE;         /* DO UNTIL TERMINAL COUNT*/
  46   3               END;

  47   2               CALL    STOP_8237_S;

  48   2               CALL    DISABLE_INTERRUPTS_S;

  49   2               CALL    VERIFY_TRANSFER_S;

  50   2               RETURN RESULTS_S;

  51   2       END CHA_SDLC_TEST;
  52   1       END STEST;


MODULE INFORMATION:

     CODE AREA SIZE     = 0063H     99D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0004H      4D
     198 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION


PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE VECTOR_MODE
OBJECT MODULE PLACED IN :F1:VECTOR.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:VECTOR.PLM TITLE(iSBC 88/45 8274 CHANNEL A SDLC TEST)


              /*****************************************************************/
              /*                                                               */
              /*            8274 INTERRUPT HANDLING ROUTINE FOR                */
              /*                   8274 VECTOR MODE                            */
              /*                STATUS AFFECTS VECTOR                          */
              /*                                                               */
              /*****************************************************************/
```

```
                /* THIS IS AN EXAMPLE OF HOW 8274 CAN BE USED IN VECTORED MODE.      */
                /* THE iSBC88/45 BOARD WAS REWIRED TO DISABLE THE PIT 8259A AND      */
                /* ENABLE THE 8274 TO PLACE ITS VECTOR ON THE DATABUS IN RESPONSE    */
                /* TO THE INTA SEQUENCE FROM THE 8088. OTHER MODIFICATIONS INCLUDED */
                /* CHANGES TO 8274 INITIALIZATION PROGRAM (SINI74) TO PROGRAM 8274   */
                /* INTO VECTORED MODE (WRITE REGISTER 2A D5=1).                      */

  1             VECTOR_MODE: DO;
                $NOLIST

 12    1        DECLARE TEMP BYTE;
 13    1        DECLARE (RESULTS_S,TXDONE,RXDONE) BYTE EXTERNAL;
 14    1        DECLARE DONE  LITERALLY 'OFFH',
                        NOT_DONE LITERALLY '00H',
                        PASS     LITERALLY 'OFFH',
                        FAIL     LITERALLY '00H';

                /*************************************************************************/
                /*      TRANSMIT INTERRUPT CHANNEL A INTERRUPT WILL NOT BE SEEN IN THE */
                /*      DMA OPERATION.                                                   */
                /*************************************************************************/

 15    1            TX_INTERRUPT_CHA: PROCEDURE INTERRUPT 84;
 16    2            OUTPUT(COMMAND_A_74) = 00101000B;        /*RESET TXINT PENDING*/
 17    2            OUTPUT(COMMAND_A_74) = 00111000B;        /*EOI*/
 18    2            END TX_INTERRUPT_CHA;


                /*************************************************************************/
                /*      EXTERNAL/STATUS INTERRUPT PROCEDURE: CHECKS FOR END OF MESSAGE */
                /*      ONLY. IF THIS IS NOT TRUE THEN THE FAIL FLAG IS SET. HOWEVER,  */
                /*      A USER PROGRAM SHOULD CHECK FOR OTHER EXT/STATUS CONDITIONS    */
                /*      ALSO IN RR1 AND THEN TAKE APPROPRIATE ACTION BASED ON THE      */
                /*      APPLICATION.                                                   */
                /*************************************************************************/

 19    1        EXT_STAT_CHANGE_CHA: PROCEDURE INTERRUPT 85;
 20    2                TEMP  = INPUT(STATUS_A_74);
 21    2                IF (TEMP AND END_OF_TX_MESSAGE) = END_OF_TX_MESSAGE THEN
 22    2                   TXDONE = DONE;
 23    2                ELSE DO;
 24    3                   TXDONE = DONE;

PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


 25    3                   RESULTS_S = FAIL;
 26    3                END;

 27    2            OUTPUT(COMMAND_A_74) = 00010000B;        /*RESET EXT STAT INT*/
 28    2                OUTPUT(COMMAND_A_74) = 00111000B;        /*EOI*/
 29    2                RETURN;
 30    2        END EXT_STAT_CHANGE_CHA;,

                /*************************************************************************/
                /*      RECEIVER CHARACTER AVAILABLE INTERRUPT WILL APPEAR ONLY ON FIRST*/
                /*      RECEIVE CHARACTER. SINCE DMA CONTROLLER HAS BEEN ENABLED BEFORE */
                /*      THE FIRST CHARACTER IS RECEIVED, THE RECEIVER REQUEST IS       */
                /*      SERVICED BY THE DMA CONTROLLER.                                */
                /*************************************************************************/

 31    1        RX_CHAR_AVAILABLE_CHA: PROCEDURE INTERRUPT 86;
 32    2            OUTPUT(COMMAND_A_74) = 00111000B;        /*EOI*/
 33    2                RETURN;
 34    2        END RX_CHAR_AVAILABLE_CHA;
                $EJECT
```

```
PL/M-86 COMPILER     iSBC 88/45 8274 CHANNEL A SDLC TEST


                /************************************************************************/
                /*       SPECIAL RECEIVE CONDITION INTERRUPT SERVICE ROUTINE CHECKS FOR */
                /*       END OF FRAME BIT ONLY. SEE SPECIAL SERVICE ROUTINE FOR NON-    */
                /*       VECTORED MODE FOR CRC CHECK AND OVERRUN ERROR CHECK.           */
                /************************************************************************/

  35    1       SPECIAL_RX_CONDITION_CHA: PROCEDURE INTERRUPT 87;

  36    2               OUTPUT(COMMAND_A_74) = 1;                  /*POINTER 1*/
  37    2               TEMP = INPUT(STATUS_A_74);
  38    2               IF (TEMP AND END_OF_FRAME) = END_OF_FRAME THEN
  39    2                   RXDONE = DONE;
  40    2               ELSE DO;
  41    3                     RXDONE = DONE;
  42    3                     RESULTS_S = FAIL;
  43    3                    END;
  44    2           OUTPUT(COMMAND_A_74) = 00110000B;          /*ERROR RESET*/
  45    2           OUTPUT(COMMAND_A_74) = 00111000B;          /*EOI*/
  46    2               RETURN;
  47    2       END SPECIAL_RX_CONDITION_CHA;


  48    1       ENABLE_INTERRUPTS: PROCEDURE PUBLIC;
  49    2       DISABLE;
  50    2       CALL SET$INTERRUPT(84, TX_INTERRUPT_CHA);
  51    2       CALL SET$INTERRUPT(85, EXT_STAT_CHANGE_CHA);
  52    2       CALL SET$INTERRUPT(86, RX_CHAR_AVAILABLE_CHA);
  53    2       CALL SET$INTERRUPT(87, SPECIAL_RX_CONDITION_CHA);
  54    2       RETURN;
  55    2       END ENABLE_INTERRUPTS;

  56    1       END VECTOR_MODE;
                /************************************************************************/
                /************************************************************************/


MODULE INFORMATION:

     CODE AREA SIZE     = 012EH    302D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 001EH     30D
     226 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

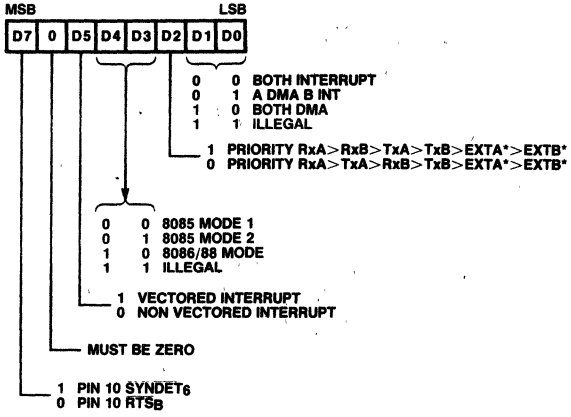
APPENDIX B
MPSC READ/WRITE REGISTER DESCRIPTIONS


WRITE REGISTER 0 (WR0):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
COMMAND STATUS POINTER REGISTER POINTER
0 0 0 NULL CODE
0 0 1 SEND ABORT (SDLC)
0 1 0 RESET EXT STATUS INTERRUPTS
0 1 1 CHANNEL RESET
1 0 0 ENABLE INTERRUPT ON NEXT RX CHARACTER
1 0 1 RESET TxINT DMA PENDING
1 1 0 ERROR RESET
1 1 1 END OF INTERRUPT
0 0 NULL CODE
0 1 RESET Rx CRC CHECKER
1 0 RESET Tx CRD GENERATOR
1 1 RESET TX UNDERRUN EOM LATCH


WRITE REGISTER 1 (WR1):
MSB
LSB
D7 D6 D5 D4 D3 D2 D1 D0
EXT INTERRUPT ENABLE
Tx INTERRUPT DMA ENABLE
STATUS AFFECTS VECTOR (CHB ONLY) (NULL CODE CH A)
1 VARIABLE VECTOR
0 FIXED VECTOR
0 0 RxINT/DMA DISABLE
0 1 RxINT ON FIRST CHAR OR SPECIAL CONDITION
1 0 INT ON ALL Rx CHAR (PARITY AFFECTS VECTOR) OR SPECIAL CONDITION
1 1 INT ON ALL Rx CHAR (PARITY DOES NOT AFFECT VECTOR) OR SPECIAL CONDITION
1 WAIT ON Rx, 0 WAIT ON Tx
MUST BE ZERO
WAIT ENABLE, 1 ENABLE, 0 DISABLE

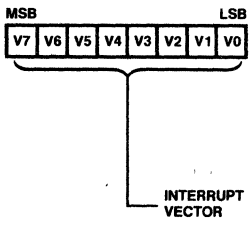
AFN-02213A

WRITE REGISTER 2 (WR2): CHANNEL A

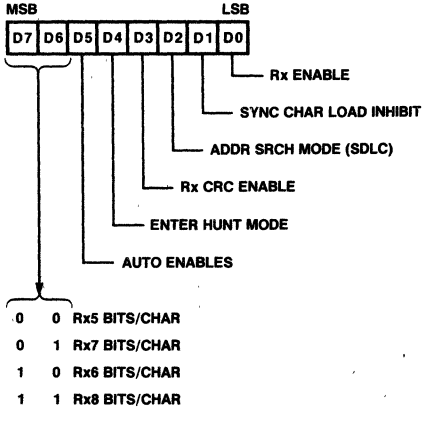


* EXTERNAL STATUS INTERRUPT ONLY IF EXT INTERRUPT ENABLE (WR1:D0) IS SET

WRITE REGISTER 2 (WR2): CHANNEL B

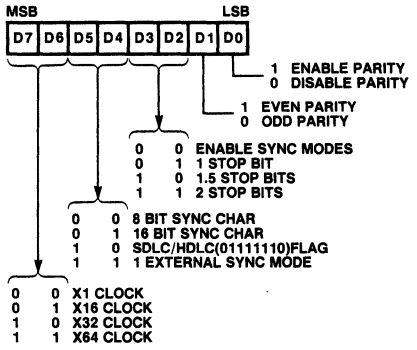


WRITE REGISTER 3 (WR3):

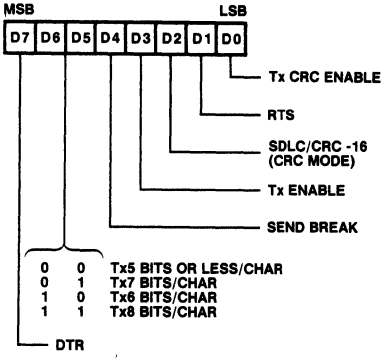


AFN-02213A

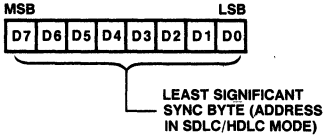
WRITE REGISTER 4 (WR4):



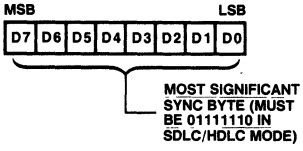
WRITE REGISTER 5 (WR5):



WRITE REGISTER 6 (WR6):

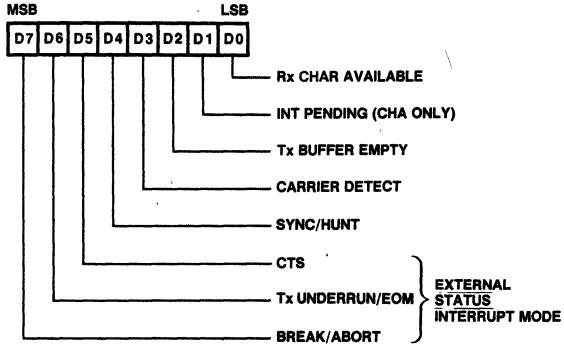


WRITE REGISTER 7 (WR7):

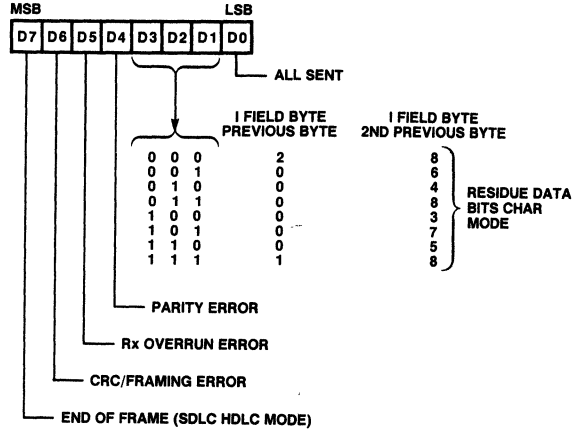


AFN-02213A

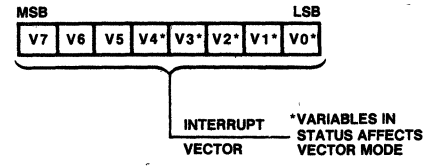
READ REGISTER 0 (RR0):



READ REGISTER 1 (RR1):
(SPECIAL RECEIVE CONDITION MODE)



READ REGISTER 2 (RR2):



AFN-02213A

8251A
PROGRAMMABLE COMMUNICATION INTERFACE

- **Synchronous and Asynchronous Operation**
- **Synchronous 5-8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion**
- **Asynchronous 5-8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling**
- **Synchronous Baud Rate—DC to 64K Baud**
- **Asynchronous Baud Rate—DC to 19.2K Baud**
- **Full-Duplex, Double-Buffered Transmitter and Receiver**
- **Error Detection—Parity, Overrun and Framing**
- **Compatible with an Extended Range of Intel Microprocessors**
- **28-Pin DIP Package**
- **All Inputs and Outputs are TTL Compatible**
- **Available in EXPRESS**
  —Standard Temperature Range
  —Extended Temperature Range

The Intel® 8251A is the enhanced version of the industry standard, Intel 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel's microprocessor families such as MCS-48, 80, 85, and iAPX-86, 88. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM "bi-sync"). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPTY. The chip is fabricated using N-channel silicon gate technology.

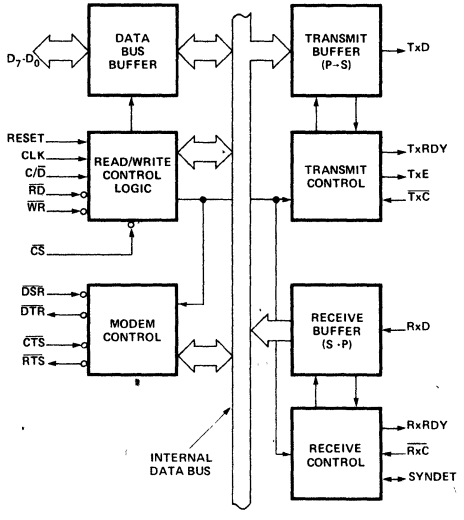


Figure 1. Block Diagram

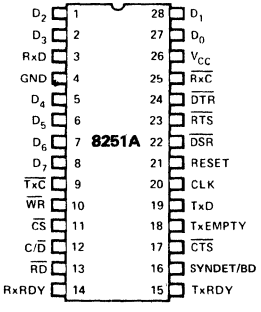


Figure 2. Pin Configuration

FEATURES AND ENHANCEMENTS

The 8251A is an advanced design of the industry standard USART, the Intel® 8251. The 8251A operates with an extended range of Intel microprocessors and maintains compatibility with the 8251. Familiarization time is minimal because of compatibility and involves only knowing the additional features and enhancements, and reviewing the AC and DC specifications of the 8251A.

The 8251A incorporates all the key features of the 8251 and has the following additional features and enhancements:

- 8251A has double-buffered data paths with separate I/O registers for control, status, Data In, and Data Out, which considerably simplifies control programming and minimizes CPU overhead.
- In asynchronous operations, the Receiver detects and handles "break" automatically, relieving the CPU of this task.
- A refined Rx initialization prevents the Receiver from starting when in "break" state, preventing unwanted interrupts from a disconnected USART.
- At the conclusion of a transmission, TxD line will always return to the marking state unless SBRK is programmed.
- Tx Enable logic enhancement prevents a Tx Disable command from halting transmission until all data previously written has been transmitted. The logic also prevents the transmitter from turning off in the middle of a word.
- When External Sync Detect is programmed, Internal Sync Detect is disabled, and an External Sync Detect status is provided via a flip-flop which clears itself upon a status read.
- Possibility of false sync detect is minimized by ensuring that if double character sync is programmed, the characters be contiguously detected and also by clearing the Rx register to all ones whenever Enter Hunt command is issued in Sync mode.
- As long as the 8251A is not selected, the $\overline{RD}$ and $\overline{WR}$ do not affect the internal operation of the device.
- The 8251A Status can be read at any time but the status update will be inhibited during status read.
- The 8251A is free from extraneous glitches and has enhanced AC and DC characteristics, providing higher speed and better operating margins.
- Synchronous Baud rate from DC to 64K.

FUNCTIONAL DESCRIPTION

General

The 8251A is a Universal Synchronous/Asynchronous Receiver/Transmitter designed for a wide range of Intel microcomputers such as 8048, 8080, 8085, 8086 and 8088. Like other I/O devices in a microcomputer system, its functional configuration is programmed by the system's software for maximum flexibility. The 8251A can support most serial data techniques in use, including IBM "bi-sync."

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8251A to the system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer. The Command Status, Data-In and Data-Out registers are separate, 8-bit registers communicating with the system bus through the Data Bus Buffer.

This functional block accepts inputs from the system Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for the device functional definition.

RESET (Reset)

A "high" on this input forces the 8251A into an "Idle" mode. The device will remain at "Idle" until a new set of control words is written into the 8251A to program its functional definition. Minimum RESET pulse width is 6 t_{CY} (clock must be running).

A command reset operation also puts the device into the "Idle" state.

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

$\overline{WR}$ (Write)

A "low" on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

$\overline{RD}$ (Read)

A "low" on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

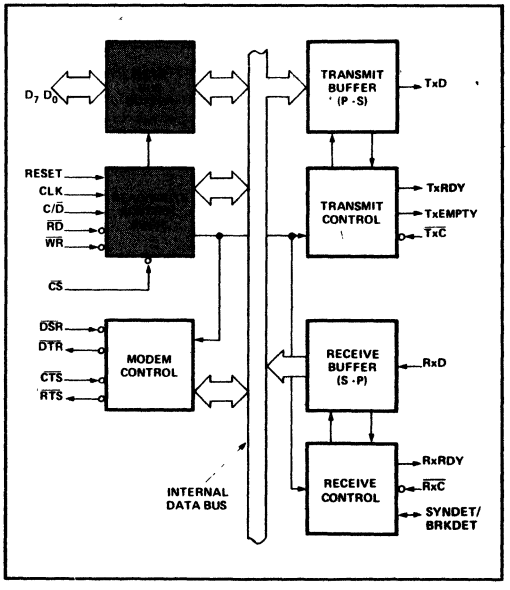


Figure 3. 8251A Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

C/D	RD	WR	CS	
0	0	1	0	8251A DATA ⇒ DATA BUS
0	1	0	0	DATA BUS ⇒ 8251A DATA
1	0	1	0	STATUS ⇒ DATA BUS
1	1	0	0	DATA BUS ⇒ CONTROL
X	1	1	0	DATA BUS ⇒ 3-STATE
X	X	X	1	DATA BUS ⇒ 3-STATE

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs, informs the 8251A that the word on the Data Bus is either a data character, control word or status information.

1 = CONTROL/STATUS; 0 = DATA.

$\overline{CS}$ (Chip Select)

A "low" on this input selects the 8251A. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus is in the float state and $\overline{RD}$ and $\overline{WR}$ have no effect on the chip.

Modem Control

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. The modem control signals are general purpose in nature and can be used for functions other than modem control, if necessary.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input signal is a general-purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test modem conditions such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready)

The $\overline{DTR}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{DTR}$ output signal is normally used for modem control such as Data Terminal Ready.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{RTS}$ output signal is normally used for modem control such as Request to Send.

$\overline{CTS}$ (Clear to Send)

A "low" on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a "one." If either a Tx Enable off or $\overline{CTS}$ off condition occurs while the Tx is in operation, the Tx will transmit all the data in the USART, written prior to Tx Disable command before shutting down.

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of $\overline{TxC}$. The transmitter will begin transmission upon being enabled if $\overline{CTS}$ = 0. The TxD line will be held in the marking state immediately upon a master Reset or when Tx Enable or $\overline{CTS}$ is off or the transmitter is empty.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. The TxRDY output pin can be used as an interrupt to the system, since it is masked by TxEnable; or, for Polled operation, the CPU can check TxRDY using a Status Read operation. TxRDY is automatically reset by the leading edge of $\overline{WR}$ when a data character is loaded from the CPU.

Note that when using the Polled operation, the TxRDY status bit is *not* masked by TxEnable, but will only indicate the Empty/Full Status of the Tx Data Input Register.

TxE (Transmitter Empty)

When the 8251A has no characters to send, the TxEMPTY output will go "high." It resets upon receiving a character from CPU if the transmitter is enabled. TxEMPTY remains high when the transmitter is disabled. TxEMPTY can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplex operational mode.

In the Synchronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be or are being transmitted automatically as "fillers." TxEMPTY does not go low when the SYNC characters are being shifted out.

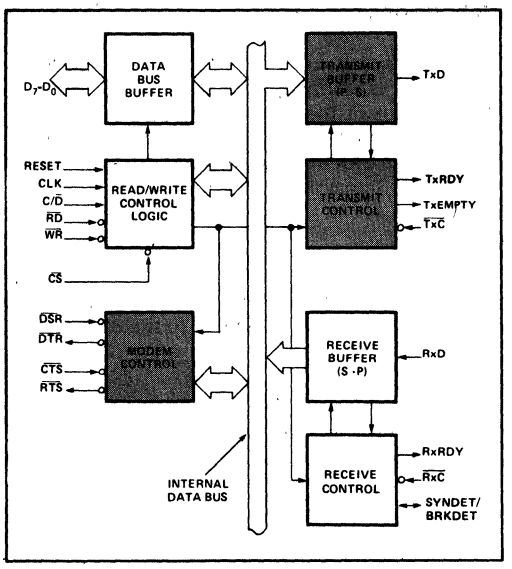


Figure 4. 8251A Block Diagram Showing Modem and Transmitter Buffer and Control Functions

$\overline{TxC}$ (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the $\overline{TxC}$ frequency. In Asynchronous transmission mode, the baud rate is a fraction of the actual $\overline{TxC}$ frequency. A portion of the mode instruction selects this factor; it can be 1, 1/16 or 1/64 the $\overline{TxC}$.

For Example:

If Baud Rate equals 110 Baud,
$\overline{TxC}$ equals 110 Hz in the 1x mode.
$\overline{TxC}$ equals 1.72 kHz in the 16x mode.
$\overline{TxC}$ equals 7.04 kHz in the 64x mode.

The falling edge of $\overline{TxC}$ shifts the serial data out of the 8251A.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to RxD pin, and is clocked in on the rising edge of $\overline{RxC}$.

Receiver Control

This functional block manages all receiver-related activities which consists of the following features.

The RxD initialization circuit prevents the 8251A from mistaking an unused input line for an active low data line in the "break condition." Before starting to receive serial characters on the RxD line, a valid "1" must first be detected after a chip master Reset. Once this has been determined, a search for a valid low (Start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master Reset.

The False Start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the nominal center of the Start bit (RxD = low).

Parity error detection sets the corresponding status bit.

The Framing Error status bit is set if the Stop bit is absent at the end of the data byte (asynchronous mode).

RxRDY (Receiver Ready)

This output indicates that the 8251A contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or, for polled operation, the CPU can check the condition of RxRDY using a Status Read operation.

RxEnable, when off, holds RxRDY in the Reset Condition. For Asynchronous mode, to set RxRDY, the Receiver must be enabled to sense a Start Bit and a complete character must be assembled and transferred to the Data Output Register. For Synchronous mode, to set RxRDY, the Receiver must be enabled and a character must finish assembly and be transferred to the Data Output Register.

Failure to read the received character from the Rx Data Output Register prior to the assembly of the next Rx Data character will set overrun condition error and the previous character will be written over and lost. If the Rx Data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the Baud Rate (1x) is equal to the actual frequency of $\overline{RxC}$. In Asynchronous Mode, the Baud Rate is a fraction of the actual $\overline{RxC}$ frequency. A portion of the mode instruction selects this factor: 1, 1/16 or 1/64 the $\overline{RxC}$.

For example:

Baud Rate equals 300 Baud, if
$\overline{RxC}$ equals 300 Hz in the 1x mode;
$\overline{RxC}$ equals 4800 Hz in the 16x mode;
$\overline{RxC}$ equals 19.2 kHz in the 64x mode.

Baud Rate equals 2400 Baud, if
$\overline{RxC}$ equals 2400 Hz in the 1x mode;
$\overline{RxC}$ equals 38.4 kHz in the 16x mode;
$\overline{RxC}$ equals 153.6 kHz in the 64x mode.

Data is sampled into the 8251A on the rising edge of $\overline{RxC}$.

NOTE: In most communications systems, the 8251A will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

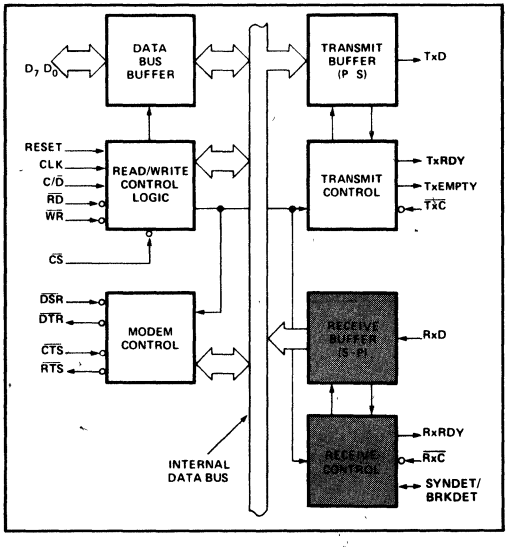


Figure 5. 8251A Block Diagram Showing Receiver Buffer and Control Functions

SYNDET (SYNC Detect/ BRKDET Break Detect)

This pin is used in Synchronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bisync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, Internal SYNC Detect is disabled.

BREAK (Async Mode Only)

This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits). Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset or Rx Data returning to a "one" state.

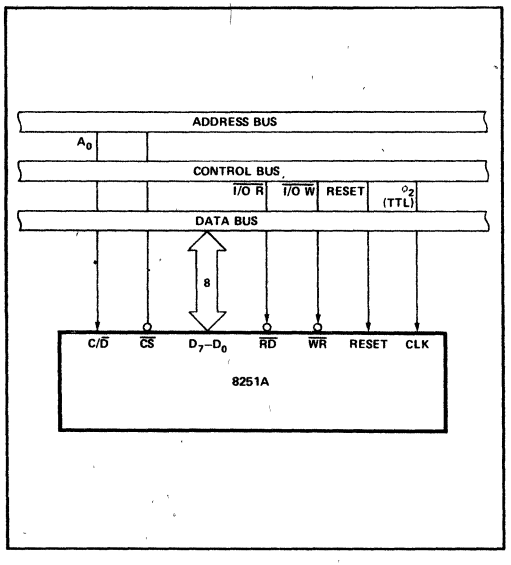


Figure 6. 8251A Interface to 8080 Standard System Bus

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251A is programmed by the system's software. A set of control words must be sent out by the CPU to initialize the 8251A to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD/OFF PARITY, etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251A is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251A is ready to receive a data character from the CPU. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251A. On the other hand, the 8251A receives serial data from the MODEM or I/O device. Upon receiving an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251A has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU data read operation.

The 8251A cannot begin transmission until the Tx Enable (Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send ($\overline{CTS}$) input. The TxD output will be held in the marking state upon Reset.

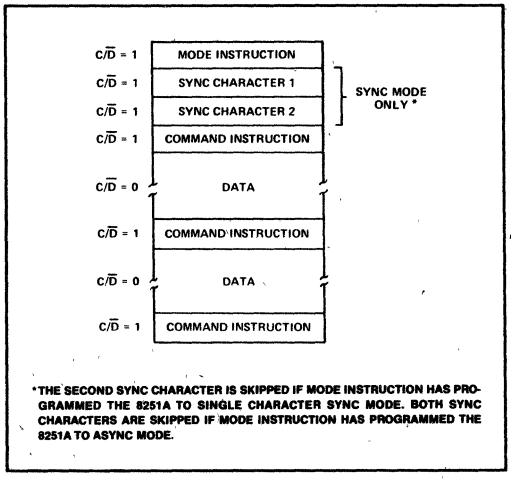


Figure 7. Typical Data Block

Programming the 8251A

Prior to starting data transmission or reception, the 8251A must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251A and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

This instruction defines the general operational characteristics of the 8251A. It must follow a Reset operation (internal or external). Once the Mode Instruction has been written into the 8251A by the CPU, SYNC characters or Command Instructions may be written.

Command Instruction

This instruction defines a word that is used to control the actual operation of the 8251A.

Both the Mode and Command Instructions must conform to a specified sequence for proper device operation (see Figure 7). The Mode Instruction must be written immediately following a Reset operation, prior to using the 8251A for data communication.

All control words written into the 8251A after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251A at any time in the data block during the operation of the 8251A. To return to the Mode Instruction format, the master Reset bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251A back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.

Mode Instruction Definition

The 8251A can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251A, the designer can best view the device as two separate components, one Asynchronous and the other Synchronous, sharing the same package. The format definition can be changed only after a master chip Reset. For explanation purposes the two formats will be isolated.

NOTE: When parity is enabled it is not considered as one of the data bits for the purpose of programming the word length. The actual parity bit received on the Rx Data line cannot be read on the Data Bus. In the case of a programmed character length of less than 8 bits, the least significant Data Bus bits will hold the data; unused bits are "don't care" when writing data to the 8251A, and will be "zeros" when reading the data from the 8251A.

Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251A automatically adds a Start bit (low level) followed by the data bits (least significant bit first), and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, 1/16, or 1/64 that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

When no data characters have been loaded into the 8251A the TxD output reamins "high" (marking) unless a Break (continuously low) has been programmed.

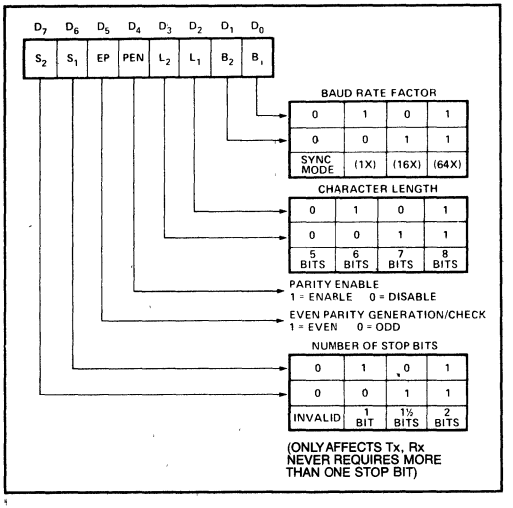


Figure 8. Mode Instruction Format, Asynchronous Mode

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center (16X or 64X mode only). If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter thus locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of $\overline{RxC}$. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. Note that the *receiver* requires only *one* stop bit, regardless of the number of stop bits programmed. This character is then loaded into the parallel I/O buffer of the 8251A. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN Error flag is raised (thus the previous character is lost). All of the error flags can be reset by an Error Reset Instruction. The occurrence of any of these errors will not affect the operation of the 8251A.

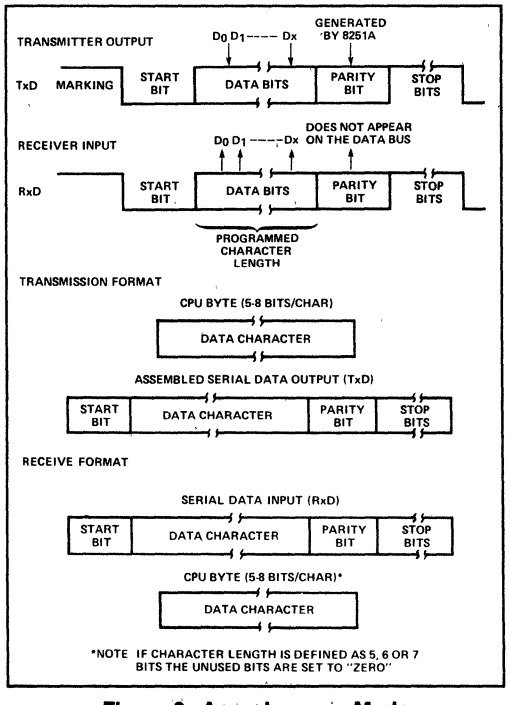
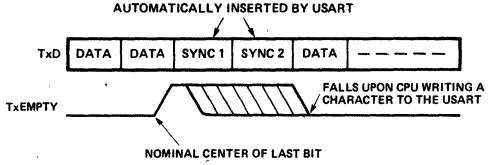


Figure 9. Asynchronous Mode

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251A which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at the TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251A with a data character before the 8251A Transmitter Buffers become empty, the SYNC characters (or character if in single SYNC character mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251A is empty and SYNC characters are being sent out. TxEMPTY does not go low when the SYNC is being shifted out (see figure below). The TxEMPTY pin is internally reset by a data character being written into the 8251A.



Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the SYNC mode has been programmed, ENTER HUNT command should be included in the first command instruction word written. Data on the RxD pin is then sampled on the rising edge of $\overline{RxC}$. The content of the Rx buffer is compared at every bit boundary with the first SYNC character until a match occurs. If the 8251A has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ. If parity is programmed, SYNDET will not be set until the middle of the parity bit instead of the middle of the last data bit.

In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin, thus forcing the 8251A out of the HUNT mode. The high level can be removed after one $\overline{RxC}$ cycle. An ENTER HUNT command has no effect in the asynchronous mode of operation.

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode. Parity is checked when not in Hunt, regardless of whether the Receiver is enabled or not.

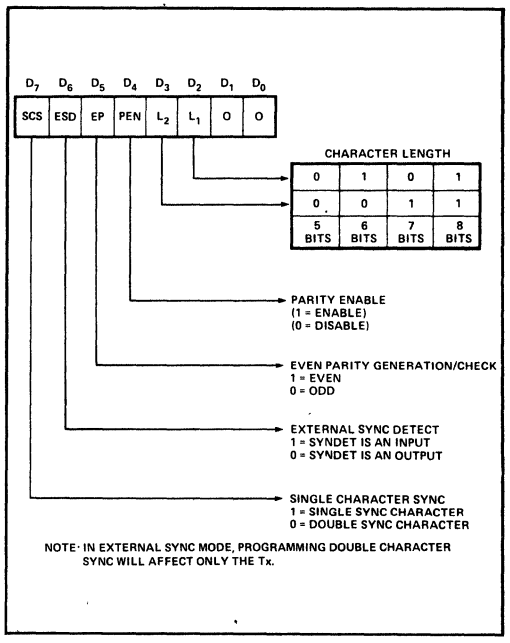


Figure 10. Mode Instruction Format, Synchronous Mode

The CPU can command the receiver to enter the HUNT mode if synchronization is lost. This will also set all the used character bits in the buffer to a "one," thus preventing a possible false SYNDET caused by data that happens to be in the Rx Buffer at ENTER HUNT time. Note that the SYNDET F/F is reset at each Status Read, regardless of whether internal or external SYNC has been programmed. This does not cause the 8251A to return to the HUNT mode. When in SYNC mode, but not in HUNT, Sync Detection is still functional, but only occurs at the "known" word boundaries. Thus, if one Status Read indicates SYNDET and a second Status Read also indicates SYNDET, then the programmed SYNDET characters have been received since the previous Status Read. (If double character sync has been programmed, then both sync characters have been contiguously received to gate a SYNDET indication.) When external SYNDET mode is selected, internal Sync Detect is disabled, and the SYNDET F/F may be set at any bit boundary.

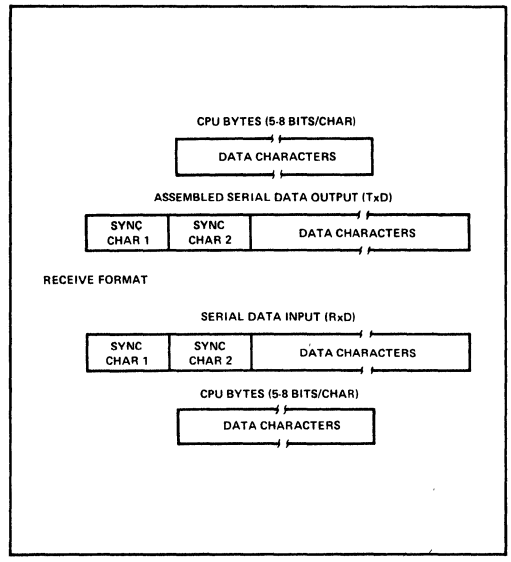


Figure 11. Data Format, Synchronous Mode

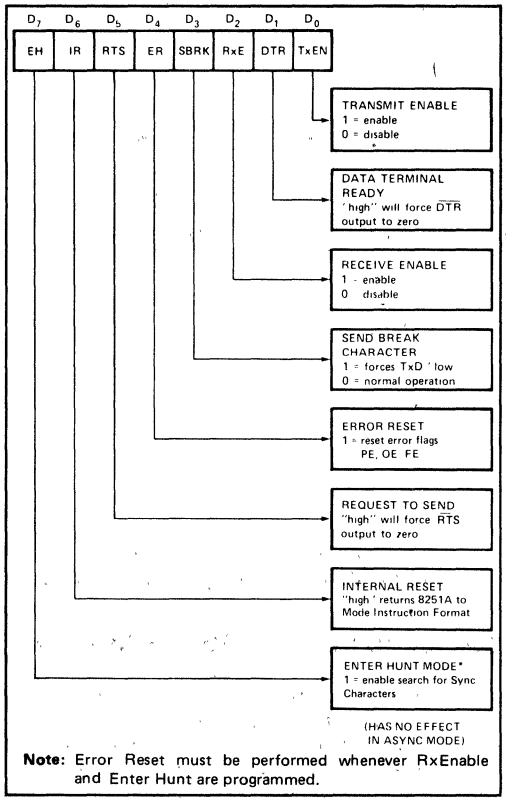
COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251A has been programmed by the Mode Instruction and the sync characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

Once the Mode Instruction has been written into the 8251A and Sync characters inserted, if necessary, then all further "control writes" (C/D = 1) will load a Command Instruction. A Reset Operation (internal or external) will return the 8251A to the Mode Instruction format.

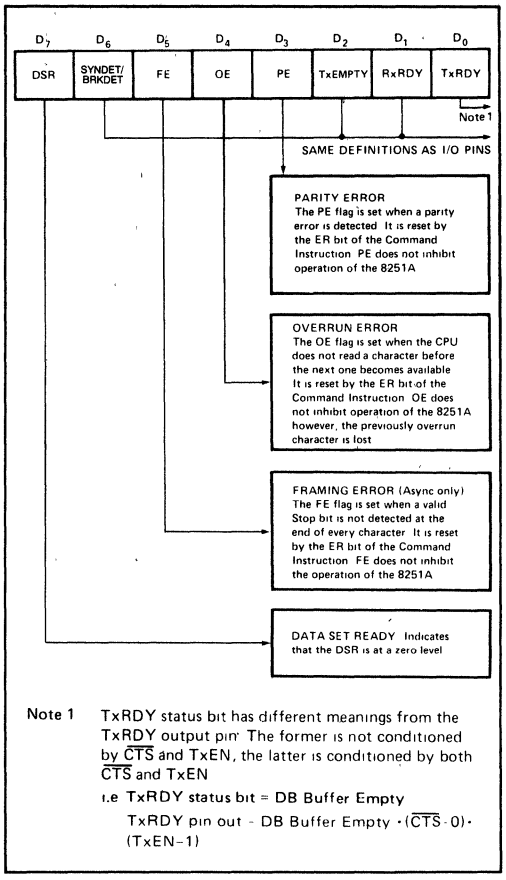
Note: Internal Reset on Power-up

When power is first applied, the 8251A may come up in the Mode, Sync character or Command format. To guarantee that the device is in the Command Instruction format before the Reset command is issued, it is safest to execute the worst-case initialization sequence (sync mode with two sync characters). Loading three 00Hs consecutively into the device with C/$\overline{D}$ = 1 configures sync operation and writes two dummy 00H sync characters. An Internal Reset command (40H) may then be issued to return the device to the "Idle" state.



Note: Error Reset must be performed whenever RxEnable and Enter Hunt are programmed.

Figure 12. Command Instruction Format



Note 1 TxRDY status bit has different meanings from the TxRDY output pin The former is not conditioned by $\overline{CTS}$ and TxEN, the latter is conditioned by both $\overline{CTS}$ and TxEN

i.e TxRDY status bit = DB Buffer Empty

TxRDY pin out - DB Buffer Empty $\cdot(\overline{CTS}\cdot 0)\cdot$ (TxEN-1)

Figure 13. Status Read Format

STATUS READ DEFINITION

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251A has facilities that allow the programmer to "read" the status of the device at any time during the functional operation. (Status update is inhibited during status read.)

A normal "read" command is issued by the CPU with $C/\overline{D}$ = 1 to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251A can be used in a completely polled or interrupt-driven environment. TxRDY is an exception.

Note that status update can have a maximum delay of 28 clock periods from the actual event affecting the status.

APPLICATIONS OF THE 8251A

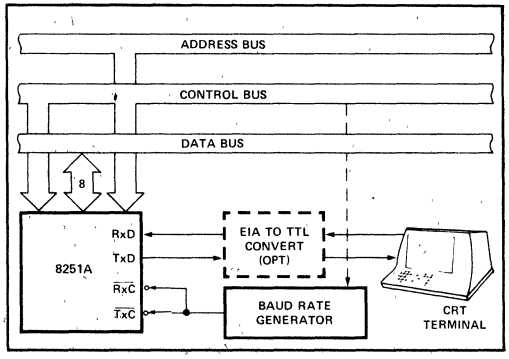


Figure 14. Asynchronous Serial Interface to CRT Terminal, DC—9600 Baud

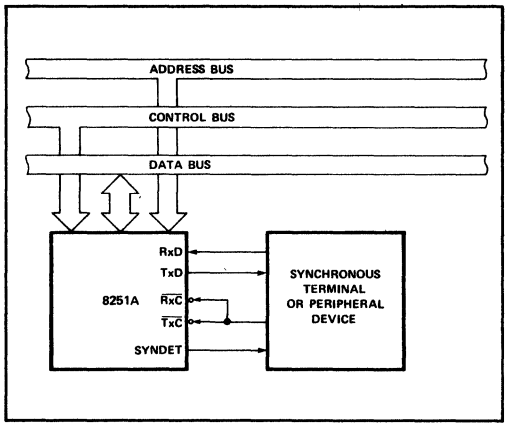


Figure 15. Synchronous Interface to Terminal or Peripheral Device

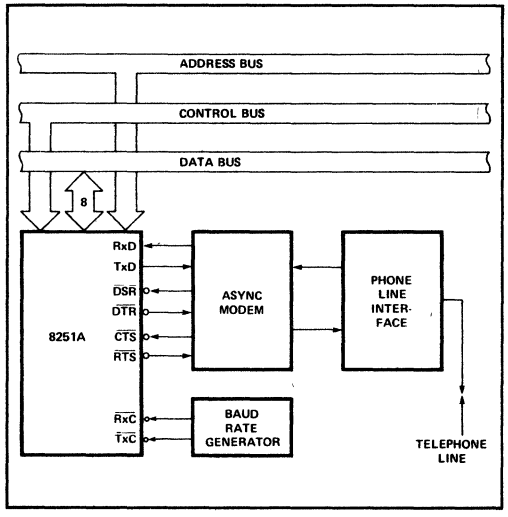


Figure 16. Asynchronous Interface to Telephone Lines

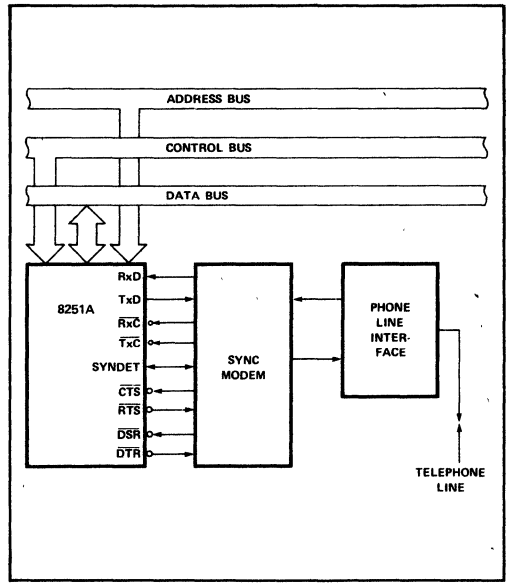


Figure 17. Synchronous Interface to Telephone Lines

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect To Ground ..... −0.5V to +7V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 5%, GND = 0V)*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OL} = −400 μA
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} TO 0.45V
I_{IL}	Input Leakage		±10	μA	V_{IN} = V_{CC} TO 0.45V
I_{CC}	Power Supply Current		100	mA	All Outputs = High

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance		10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V) *

Bus Parameters (Note 1)

READ CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	Address Stable Before $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	0		ns	Note 2
t_{RA}	Address Hold Time for $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	0		ns	Note 2
t_{RR}	$\overline{READ}$ Pulse Width	250		ns	
t_{RD}	Data Delay from $\overline{READ}$		200	ns	3, C_L = 150 pF
t_{DF}	$\overline{READ}$ to Data Floating	10	100	ns	

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Condtions
t_{AW}	Address Stable Before $\overline{WRITE}$	0		ns	
t_{WA}	Address Hold Time for $\overline{WRITE}$	0		ns	
t_{WW}	$\overline{WRITE}$ Pulse Width	250		ns	
t_{DW}	Data Set-Up Time for $\overline{WRITE}$	150		ns	
t_{WD}	Data Hold Time for $\overline{WRITE}$	20		ns	
t_{RV}	Recovery Time Between WRITES	6		t_{CY}	Note 4

A.C. CHARACTERISTICS (Continued)

OTHER TIMINGS

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CY}	Clock Period	320	1350	ns	Notes 5, 6
t_{ϕ}	Clock High Pulse Width	120	$t_{CY}-90$	ns	
$t_{\overline{\phi}}$	Clock Low Pulse Width	90		ns	
t_R, t_F	Clock Rise and Fall Time		20	ns	
t_{DTx}	TxD Delay from Falling Edge of $\overline{TxC}$		1	μs	
f_{Tx}	Transmitter Input Clock Frequency				
	1x Baud Rate	DC	64	kHz	
	16x Baud Rate	DC	310	kHz	
	64x Baud Rate	DC	615	kHz	
t_{TPW}	Transmitter Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{TPD}	Transmitter Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
f_{Rx}	Receiver Input Clock Frequency				
	1x Baud Rate	DC	64	kHz	
	16x Baud Rate	DC	310	kHz	
	64x Baud Rate	DC	615	kHz	
t_{RPW}	Receiver Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{RPD}	Receiver Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
t_{TxRDY}	TxRDY Pin Delay from Center of Last Bit		8	t_{CY}	Note 7
$t_{TxRDY\ CLEAR}$	TxRDY ↓ from Leading Edge of $\overline{WR}$		400	ns	Note 7
t_{RxRDY}	RxRDY Pin Delay from Center of Last Bit		26	t_{CY}	Note 7
$t_{RxRDY\ CLEAR}$	RxRDY ↓ from Leading Edge of $\overline{RD}$		400	ns	Note 7
t_{IS}	Internal SYNDET Delay from Rising Edge of $\overline{RxC}$		26	t_{CY}	Note 7
t_{ES}	External SYNDET Set-Up Time After Rising Edge of $\overline{RxC}$	18		t_{CY}	Note 7
$t_{TxEMPTY}$	TxEMPTY Delay from Center of Last Bit	20		t_{CY}	Note 7
t_{WC}	Control Delay from Rising Edge of WRITE (TxEn, $\overline{DTR}$, $\overline{RTS}$)	8		t_{CY}	Note 7
t_{CR}	Control to READ Set-Up Time ($\overline{DSR}$, $\overline{CTS}$)	20		t_{CY}	Note 7

***NOTE:**

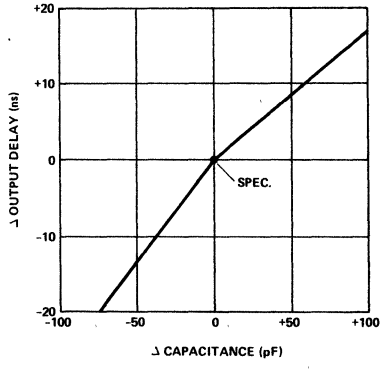
1. For Extended Temperature EXPRESS, use M8251A electrical parameters.

A.C. CHARACTERISTICS (Continued)

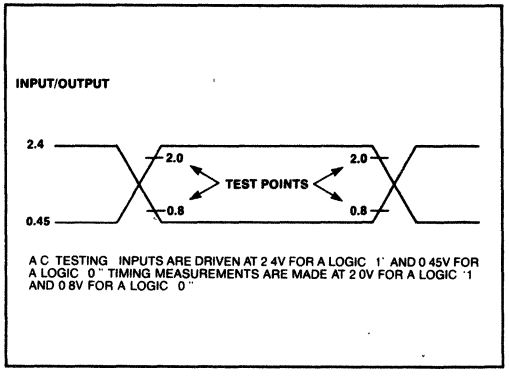
NOTES:

1. AC timings measured V_{OH} = 2.0 V_{OL} = 2.0, V_{OL} = 0.8, and with load circuit of Figure 1.
2. Chip Select (CS) and Command/Data (C/D) are considered as Addresses.
3. Assumes that Address is valid before $R_D\downarrow$.
4. This recovery time is for Mode Initialization only. Write Data is allowed only when TxRDY = 1. Recovery Time between Writes for Asynchronous Mode is 8 t_{CY} and for Synchronous Mode is 16 t_{CY}.
5. The TxC and RxC frequencies have the following limitations with respect to CLK: For 1x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(30\ t_{CY})$:
   For 16x and 64x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(4.5\ t_{CY})$.
6. Reset Pulse Width = 6 t_{CY} minimum; System Clock must be running during Reset.
7. Status update can have a maximum delay of 28 clock periods from the event affecting the status.

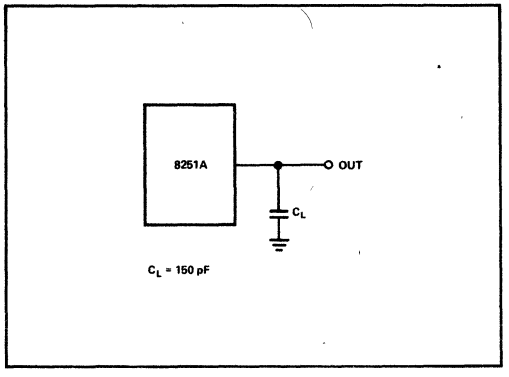
TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (pF)



A.C. TESTING INPUT, OUTPUT WAVEFORM

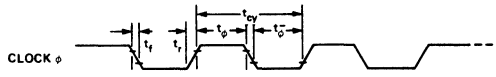


A.C. TESTING LOAD CIRCUIT

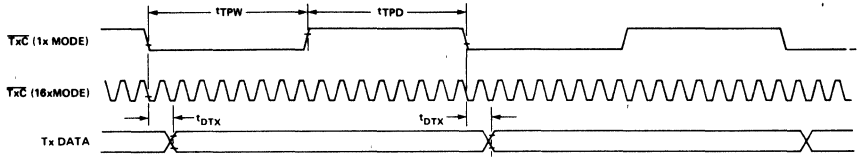


WAVEFORMS

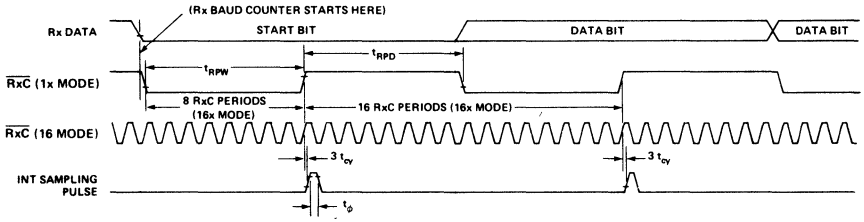
SYSTEM CLOCK INPUT



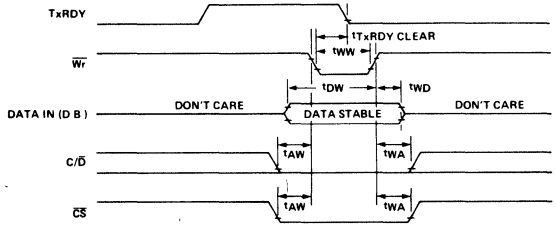
TRANSMITTER CLOCK AND DATA



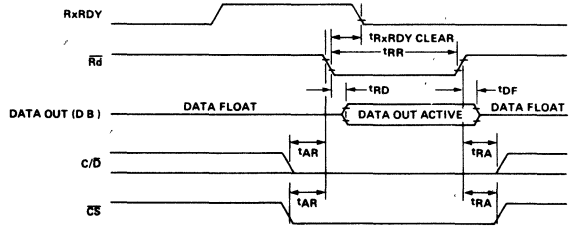
RECEIVER CLOCK AND DATA



WRITE DATA CYCLE (CPU → USART)

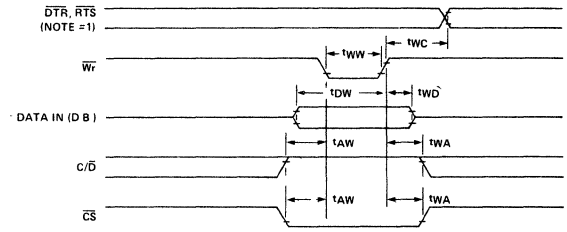


READ DATA CYCLE (CPU ← USART)

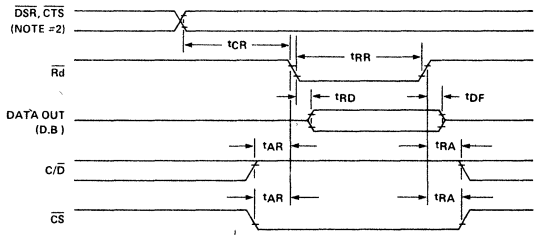


WAVEFORMS (Continued)

WRITE CONTROL OR OUTPUT PORT CYCLE (CPU → USART)

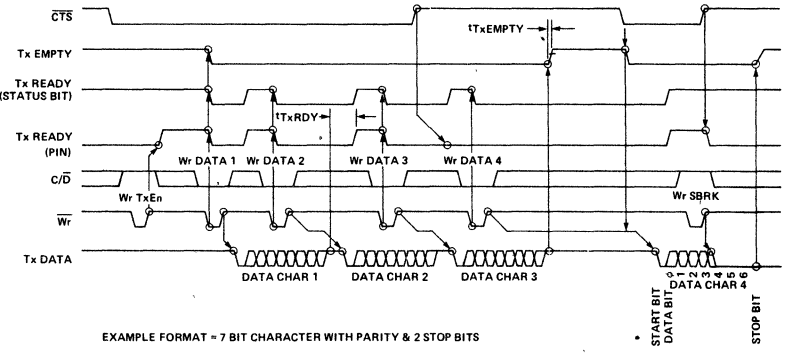


READ CONTROL OR INPUT PORT (CPU ← USART)



NOTE #1 T_{WC} INCLUDES THE RESPONSE TIMING OF A CONTROL BYTE.
NOTE #2 T_{CR} INCLUDES THE EFFECT OF CTS ON THE TxENBL CIRCUITRY.

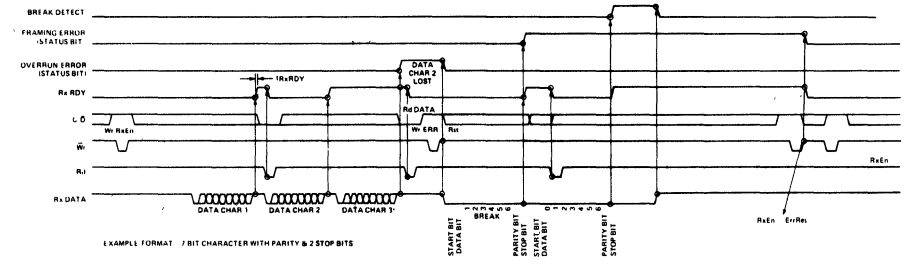
TRANSMITTER CONTROL AND FLAG TIMING (ASYNC MODE)



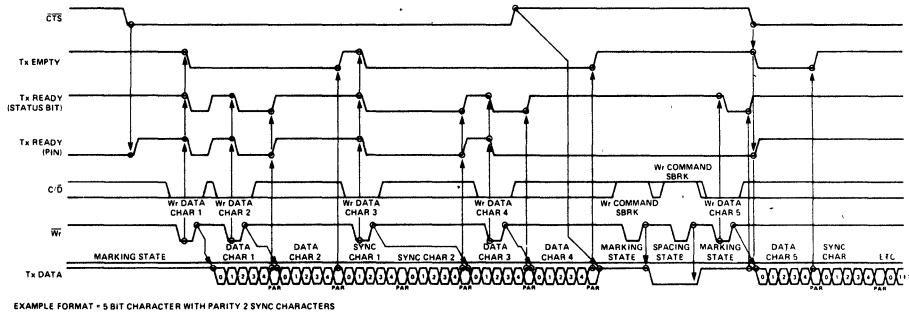
EXAMPLE FORMAT = 7 BIT CHARACTER WITH PARITY & 2 STOP BITS

WAVEFORMS (Continued)

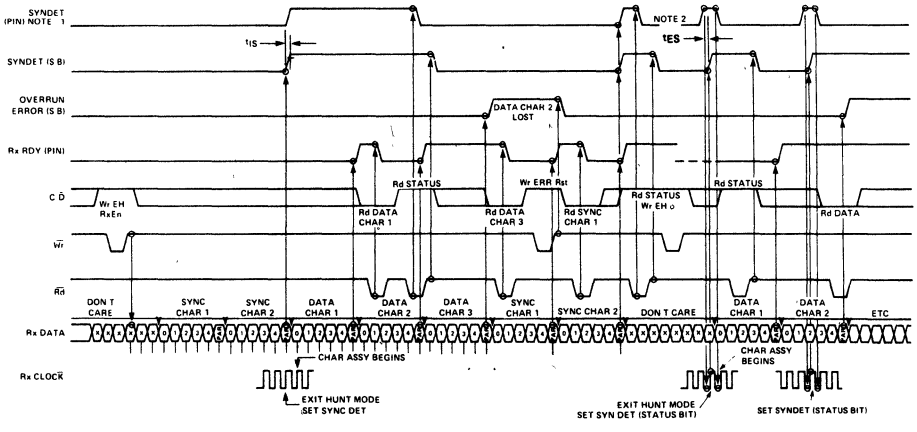
RECEIVER CONTROL AND FLAG TIMING (ASYNC MODE)



TRANSMITTER CONTROL AND FLAG TIMING (SYNC MODE)



RECEIVER CONTROL AND FLAG TIMING (SYNC MODE)



NOTE 1 INTERNAL SYNC 2 SYNC CHARACTERS 5 BITS WITH PARITY
NOTE 2 EXTERNAL SYNC 5 BITS WITH PARITY

AFN-01573D

8273, 8273-4
PROGRAMMABLE HDLC/SDLC PROTOCOL CONTROLLER

- **CCITT X.25 Compatible**
- **HDLC/SDLC Compatible**
- **Full Duplex, Half Duplex, or Loop SDLC Operation**
- **Up to 64K Baud Synchronous Transfers (56K Baud with 8273-4)**
- **Automatic FCS (CRC) Generation and Checking**
- **Up to 9.6K Baud with On-Board Phase Locked Loop**
- **Programmable NRZI Encode/Decode**
- **Two User Programmable Modem Control Ports**
- **Digital Phase Locked Loop Clock Recovery**
- **Minimum CPU Overhead**
- **Fully Compatible with 8048/8080/8085/8088/8086/80188/80186 CPUs**
- **Single +5V Supply**

The Intel® 8273 Programmable HDLC/SDLC Protocol Controller is a dedicated device designed to support the ISO/CCITT's HDLC and IBM's SDLC communication line protocols. It is fully compatible with Intel's new high performance microcomputer systems such as the MCS1 88/186™. A frame level command set is achieved by a unique microprogrammed dual processor chip architecture. The processing capability supported by the 8273 relieves the system CPU of the low level real-time tasks normally associated with controllers.

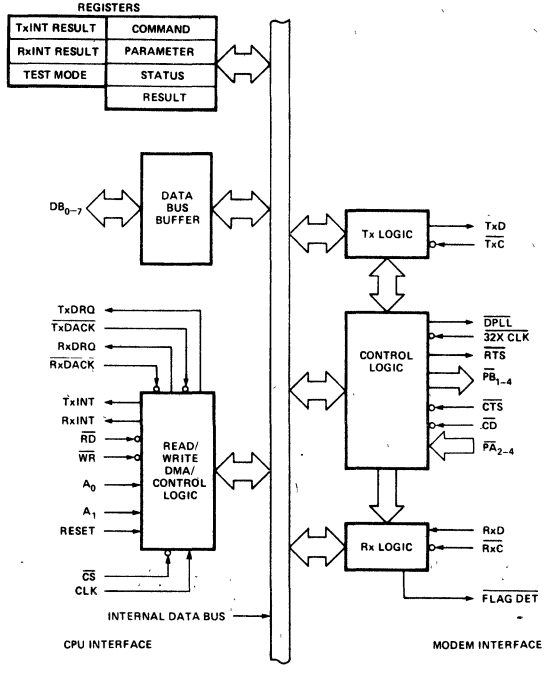


Figure 1. Block Diagram

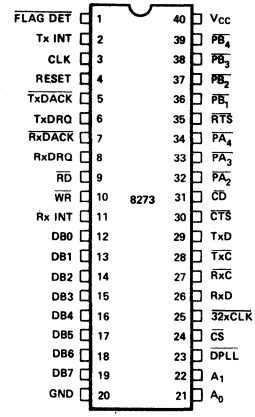


Figure 2. Pin Configuration

NOVEMBER 1983
ORDER NUMBER 210479-002

A BRIEF DESCRIPTION OF HDLC/SDLC PROTOCOLS

General

The High Level Data Link Control (HDLC) is a standard communication link protocol established by International Standards Organization (ISO). HDLC is the discipline used to implement ISO X.25 packet switching systems.

The Synchronous Data Link Control (SDLC) is an IBM communication link protocol used to implement the System Network Architecture (SNA). Both the protocols are bit oriented, code independent, and ideal for full duplex communication. Some common applications include terminal to terminal, terminal to CPU, CPU to CPU, satellite communication, packet switching and other high speed data links. In systems which require expensive cabling and interconnect hardware, any of the two protocols could be used to simplify interfacing (by going serial), thereby reducing interconnect hardware costs. Since both the protocols are speed independent, reducing interconnect hardware could become an important application.

Network

In both the HDLC and SDLC line protocols, according to a pre-assigned hierarchy, a PRIMARY (Control) STATION controls the overall network (data link) and issues commands to the SECONDARY (Slave) STATIONS. The latter comply with instructions and respond by sending appropriate RESPONSES. Whenever a transmitting station must end transmission prematurely it sends an ABORT character. Upon detecting an abort character, a receiving station ignores the transmission block called a FRAME. Time fill between frames can be accomplished by transmitting either continuous frame preambles called FLAGS or an abort character. A time fill within a frame is not permitted. Whenever a station receives a string of more that fifteen consecutive ones, the station goes into an IDLE state.

Frames

A single communication element is called a FRAME which can be used for both Link Control and data transfer purposes. The elements of a frame are the beginning eight bit FLAG (F) consisting of one zero, six ones, and a zero, an eight bit ADDRESS FIELD (A), an eight bit CONTROL FIELD (C), a variable (N-bit) INFORMATION FIELD (I), a sixteen bit FRAME CHECK SEQUENCE (FCS), and an eight bit end FLAG (F), having the same bit pattern as the beginning flag. In HDLC the Address (A) and Control (C) bytes are extendable. The HDLC and the SDLC use three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

Frame Characteristics

An important characteristic of a frame is that its contents are made code transparent by use of a zero bit insertion and deletion technique. Thus, the user can adopt any format or code suitable for his system — it may even be a computer word length or a "memory dump". The frame is bit oriented that is, bits, not characters in each field, have specific meanings. The Frame Check Sequence (FCS) is an error detection scheme similar to the Cyclic Redundancy Checkword (CRC) widely used in magnetic disk storage devices. The Command and Response information frames contain sequence numbers in the control fields identifying the sent and received frames. The sequence numbers are used in Error Recovery Procedures (ERP) and as implicit acknowledgement of frame communication, enhancing the true full-duplex nature of the HDLC/SDLC protocols.

In contrast, BISYNC is basically half-duplex (two way alternate) because of necessity to transmit immediate acknowledgement frames. HDLC/SDLC therefore saves propagation delay times and have a potential of twice the throughput rate of BISYNC.

It is possible to use HDLC or SDLC over half duplex lines but there is a corresponding loss in throughput because both are primarily designed for full-duplex communication. As in any synchronous system, the bit rate is determined by the clock bits supplied by the modem, protocols themselves are speed independent.

A byproduct of the use of zero-bit insertion-deletion technique is the non-return-to-zero invert (NRZI) data transmission/reception compatibility. The latter allows HDLC/SDLC protocols to be used with asynchronous data communication hardware in which the clocks are derived from the NRZI encoded data.

References

IBM Synchronous Data Link Control General Information, IBM, GA27-3093-1.
Standard Network Access Protocol Specification, DATAPAC, Trans-Canada Telephone System CCG111
Recommendation X.25, ISO/CCITT March 2, 1976.
IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA 27-3098-0
Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715
IBM Introduction to Teleprocessing, IBM, GC 20-8095-02
System Network Architecture, Technical Overview, IBM, GA 27-3102
System Network Architecture Format and Protocol, IBM GA 27-3112

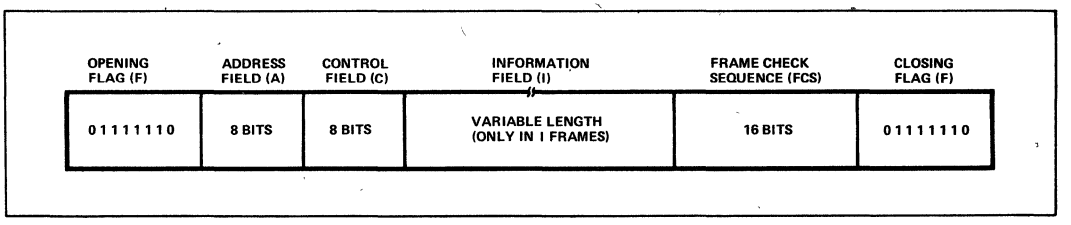


Figure 3. Frame Format

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	40		**Power Supply:** +5V Supply.
GND	20		**Ground:** Ground.
RESET	4	I	**Reset:** A high signal on this pin will force the 8273 to an idle state. The 8273 will remain idle until a command is issued by the CPU. The modem interface output signals are forced high. Reset must be true for a minimum of 10 TCY.
$\overline{CS}$	24	I	**Chip Select:** The RD and WR inputs are enabled by the chip select input.
DB_7-DB_0	19-12	I/O	**Data Bus:** The Data Bus lines are bi-directional three-state lines which interface with the system Data Bus.
$\overline{WR}$	10	I	**Write Input:** The Write signal is used to control the transfer of either a command or data from CPU to the 8273.
$\overline{RD}$	9	I	**Read Input:** The Read signal is used to control the transfer of either a data byte or a status word from the 8273 to the CPU.
TxINT	2	O	**Transmitter Interrupt:** The Transmitter interrupt signal indicates that the transmitter logic requires service.
RxINT	11	O	**Receiver Interrupt:** The Receiver interrupt signal indicates that the Receiver logic requires service.
TxDRQ	6	O	**Transmitter Data Request:** Requests a transfer of data between memory and the 8273 for a transmit operation.
RxRDQ	8	O	**Receiver DMA Request:** Requests a transfer of data between the 8273 and memory for a receive operation.
$\overline{TxDACK}$	5	I	**Transmitter DMA Acknowledge:** The Transmitter DMA acknowledge signal notifies the 8273 that the TxDMA cycle has been granted.
$\overline{RxDACK}$	7	I	**Receiver DMA Acknowledge:** The Receiver DMA acknowledge signal notifies the 8273 that the RxDMA cycle has been granted.
A_1-A_0	22-21	I	**Address:** These two lines are CPU Interface Register Select lines.
TxD	29	O	**Transmitter Data:** This line transmits the serial data to the communication channel.
$\overline{TxC}$	28	I	**Transmitter Clock:** The transmitter clock is used to synchronize the transmit data.
RxD	26	I	**Receiver Data:** This line receives serial data from the communication channel.
$\overline{RxC}$	27	I	**Receiver Clock:** The Receiver Clock is used to synchronize the receive data.
$\overline{32X\ CLK}$	25	I	**32X Clock:** The 32X clock is used to provide clock recovery when an asynchronous modem is used. In loop configuration the loop station can run without an accurate 1X clock by using the 32X CLK in conjunction with the DPLL output. (This pin must be grounded when not used.)
$\overline{DPLL}$	23	O	**Digital Phase Locked Loop:** Digital Phase Locked Loop output can be tied to RxC and/or TxC when 1X clock is not available. DPLL is used with 32X CLK.
$\overline{FLAG\ DET}$	1	O	**Flag Detect:** Flag Detect signals that a flag (01111110) has been received by an active receiver.
$\overline{RTS}$	35	O	**Request to Send:** Request to Send signals that the 8273 is ready to transmit data.
$\overline{CTS}$	30	I	**Clear to Send:** Clear to Send signals that the modem is ready to accept data from the 8273.
$\overline{CD}$	31	I	**Carrier Detect:** Carrier Detect signals that the line transmission has started and the 8273 may begin to sample data on RxD line.
$\overline{PA_{2\text{-}4}}$	32-34	I	**General purpose Input ports:** The logic levels on these lines can be Read by the CPU through the Data Bus Buffer.
$\overline{PB_{1\text{-}4}}$	36-39	O	**General purpose output ports:** The CPU can write these output lines through Data Bus Buffer.
CLK	3	I	**Clock:** A square wave TTL clock.

FUNCTIONAL DESCRIPTION

General

The Intel® 8273 HDLC/SDLC controller is a microcomputer peripheral device which supports the International Standards Organization (ISO) High Level Data Link Control (HDLC), and IBM Synchronous Data Link Control (SDLC) communications protocols. This controller minimizes CPU software by supporting a comprehensive frame-level instruction set and by hardware implementation of the low level tasks associated with frame assembly/disassembly and data integrity. The 8273 can be used in either synchronous or asynchronous applications.

In asynchronous applications the data can be programmed to be encoded/decoded in NRZI code. The clock is derived from the NRZI data using a digital phase locked loop. The data transparency is achieved by using a zero-bit insertion/deletion technique. The frames are automatically checked for errors during reception by verifying the Frame Check Sequence (FCS); the FCS is automatically generated and appended before the final flag in transmit.

The 8273 recognizes and can generate flags (01111110' Abort, Idle, and GA (EOP) characters.

The 8273 can assume either a primary (control) or a secondary (slave) role. It can therefore be readily implemented in an SDLC loop configuration as typified by the IBM 3650 Retail Store System by programming the 8273 into a one-bit delay mode. In such a configuration, a two wire pair can be effectively used for data transfer between controllers and loop stations. The digital phase locked loop output pin can be used by the loop station without the presence of an accurate Tx clock.

CPU Interface

The CPU interface is optimized for the MCS-80/85™ bus with an 8257 DMA controller. However, the interface is flexible, and allows either DMA or non-DMA data transfers, interrupt or non-interrupt driven. It further allows maximum line utilization by providing early interrupt mechanism for buffered (only the information field can be transferred to memory) Tx command overlapping. It also provides separate Rx and Tx interrupt output channels for efficient operation. The 8273 keeps the interrupt request active until all the associated interrupt results have been read.

The CPU utilizes the CPU interface to specify commands and transfer data. It consists of seven registers addressed via $\overline{CS}$, A_1, A_0, $\overline{RD}$ and $\overline{WR}$ signals and two independent data registers for receive data and transmit data. A_1, A_0 are generally derived from two low order bits of the address bus. If an 8080 based CPU is utilized, the $\overline{RD}$ and $\overline{WR}$ signals may be driven by the 8228 $\overline{I/OR}$ and $\overline{I/OW}$. The table shows the seven register select decoding:

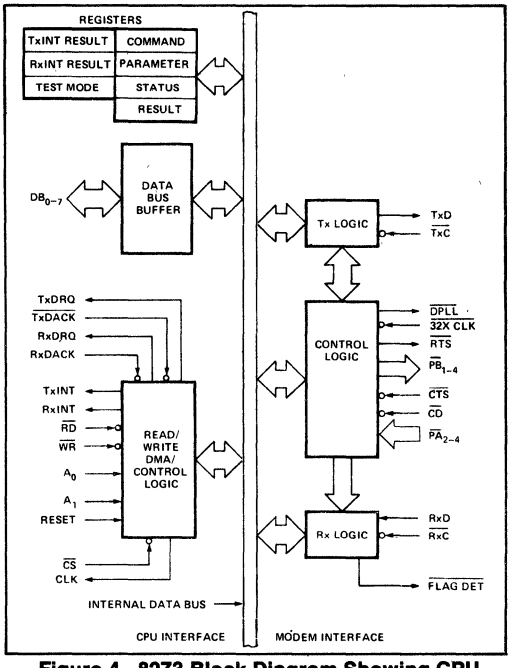


Figure 4. 8273 Block Diagram Showing CPU Interface Functions

A_1	A_0	$\overline{TxDACK}$	$\overline{RxDACK}$	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	Register
0	0	1	1	0	1	0	Command
0	0	1	1	0	0	1	Status
0	1	1	1	0	1	0	Parameter
0	1	1	1	0	0	1	Result
1	0	1	1	0	1	0	Reset
1	0	1	1	0	0	1	TxINT Result
1	1	1	1	0	1	0	—
1	1	1	1	0	0	1	RxINT Result
X	X	0	1	1	1	0	Transmit Data
X	X	1	0	1	0	1	Receive Data

Register Description

Command

Operations are initiated by writing an appropriate command in the Command Register.

Parameter

Parameters of commands that require additional information are written to this register.

Result

Contains an immediate result describing an outcome of an executed command.

Transmit Interrupt Result

Contains the outcome of 8273 transmit operation (good/bad completion).

Receive Interrupt Result

Contains the outcome of 8273 receive operation (good/bad completion), followed by additional results which detail the reason for interrupt.

Status

The status register reflects the state of the 8273 CPU Interface.

DMA Data Transfers

The 8273 CPU interface supports two independent data interfaces: receive data and transmit data. At high data transmission speeds the data transfer rate of the 8273 is great enough to justify the use of direct memory access (DMA) for the data transfers. When the 8273 is configured in DMA mode, the elements of the DMA interfaces are:

TxDRQ: Transmit DMA Request

Requests a transfer of data between memory and the 8273 for a transmit operation

$\overline{TxDACK}$: Transmit DMA Acknowledge

The $\overline{TxDACK}$ signal notifies the 8273 that a transmit DMA cycle has been granted. It is also used with $\overline{WR}$ to transfer data to the 8273 in non-DMA mode. Note: $\overline{RD}$ must not be asserted while TxDACK is active.

RxDRQ: Receive DMA Request

Requests a transfer of data between the 8273 and memory for a receive operation.

$\overline{RxDACK}$: Receive DMA Acknowledge

The $\overline{RxDACK}$ signal notifies the 8273 that a receive DMA cycle has been granted. It is also used with $\overline{RD}$ to read data from the 8273 in non-DMA mode. Note: $\overline{WR}$ must not be asserted while $\overline{RxDACK}$ is active.

$\overline{RD}$, $\overline{WR}$: Read, Write

The $\overline{RD}$ and $\overline{WR}$ signals are used to specify the direction of the data transfer.

DMA transfers require the use of a DMA controller such as the Intel 8257. The function of the DMA controller is to provide sequential addresses and timing for the transfer, at a starting address determined by the CPU. Counting of data block lengths is performed by the 8273.

To request a DMA transfer the 8273 raises the appropriate DMA REQUEST. DMA ACKNOWLEDGE and READ enables DMA data onto the bus (independently of CHIP SELECT). DMA ACKNOWLEDGE and WRITE transfers DMA data to the 8273 (independent of CHIP SELECT).

It is also possible to configure the 8273 in the non-DMA data transfer mode. In this mode the CPU module must pass data to the 8273 in response to non-DMA data requests indicated by the status word.

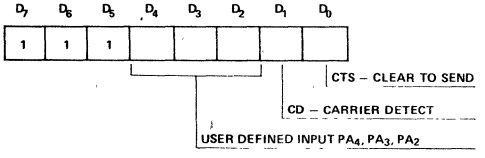
Modem Interface

The 8273 Modem interface provides both dedicated and user defined modem control functions. All the control signals are active low so that EIA RS-232C inverting drivers (MC 1488) and inverting receivers (MC 1489) may be used to interface to standard modems. For asynchronous operation, this interface supports programmable NRZI data encode/decode, a digital phase locked loop for efficient clock extraction from NRZI data, and modem control ports with automatic $\overline{CTS}$, $\overline{CD}$ monitoring and $\overline{RTS}$ generation. This interface also allows the 8273 to operate in PRE-FRAME SYNC mode in which the 8273 prefixes 16 transitions to a frame to synchronize idle lines before transmission of the first flag.

It should be noted that all the 8273 port operations deal with logical values, for instance, bit D0 of Port A will be a one when $\overline{CTS}$ (Pin 30) is a physical zero (logical one).

Port A — Input Port

During operation, the 8273 interrogates input pins $\overline{CTS}$ (Clear to Send) and $\overline{CD}$ (Carrier Detect). $\overline{CTS}$ is used to condition the start of a transmission. If during transmission $\overline{CTS}$ is lost the 8273 generates an interrupt. During reception, if $\overline{CD}$ is lost, the 8273 generates an interrupt.



The user defined input bits correspond to the 8273 PA_4, PA_3 and PA_2 pins. The 8273 does not interrogate or manipulate these bits.

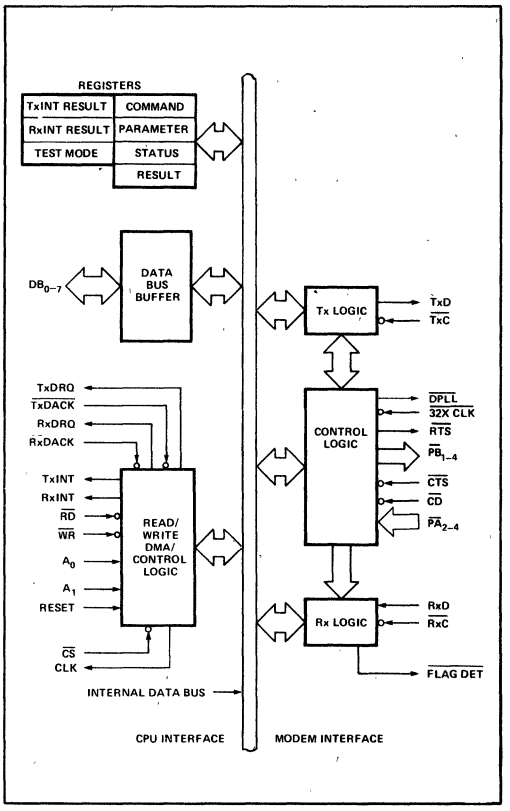
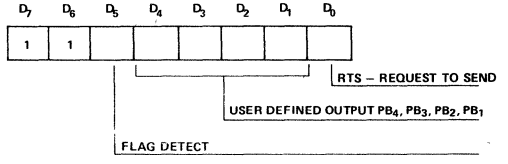


Figure 5. 8273 Block Diagram Showing Control Logic Functions

Port B - Output Port

During normal operation, if the CPU sets $\overline{RTS}$ active, the 8273 will not change this pin; however, if the CPU sets $\overline{RTS}$ inactive, the 8273 will activate it before each transmission and deactivate it one byte time after transmission. While the receiver is active the flag detect pin is pulsed each time a flag sequence is detected in the receive data stream. Following an 8273 reset, all pins of Port B are set to a high, inactive level.



The user defined output bits correspond to the state of PB_4-PB_1 pins. The 8273 does not interrogate or manipulate these bits.

Serial Data Logic

The Serial data is synchronized by the user transmit ($\overline{TxC}$) and receive ($\overline{RxC}$) clocks. The leading edge of $\overline{TxC}$ generates new transmit data and the trailing edge of $\overline{RxC}$ is used to capture receive data. The NRZI encoding/decoding of the receive and transmit data is programmable.

The diagnostic features included in the Serial Data logic are programmable loop back of data and selectable clock for the receiver. In the loop-back mode, the data presented to the TxD pin is internally routed to the receive data input circuitry in place of the RxD pin, thus allowing a CPU to send a message to itself to verify operation of the 8273.

In the selectable clock diagnostic feature, when the data is looped back, the receiver may be presented incorrect sample timing by the external circuitry. The user may select to substitute the $\overline{TxC}$ pin for the $\overline{RxC}$ input on-chip so that the clock used to generate the loop back data is used to sample it. Since TxD is generated off the leading edge of $\overline{TxC}$ and RxD is sampled on the trailing edge, the selected clock allows bit synchronism.

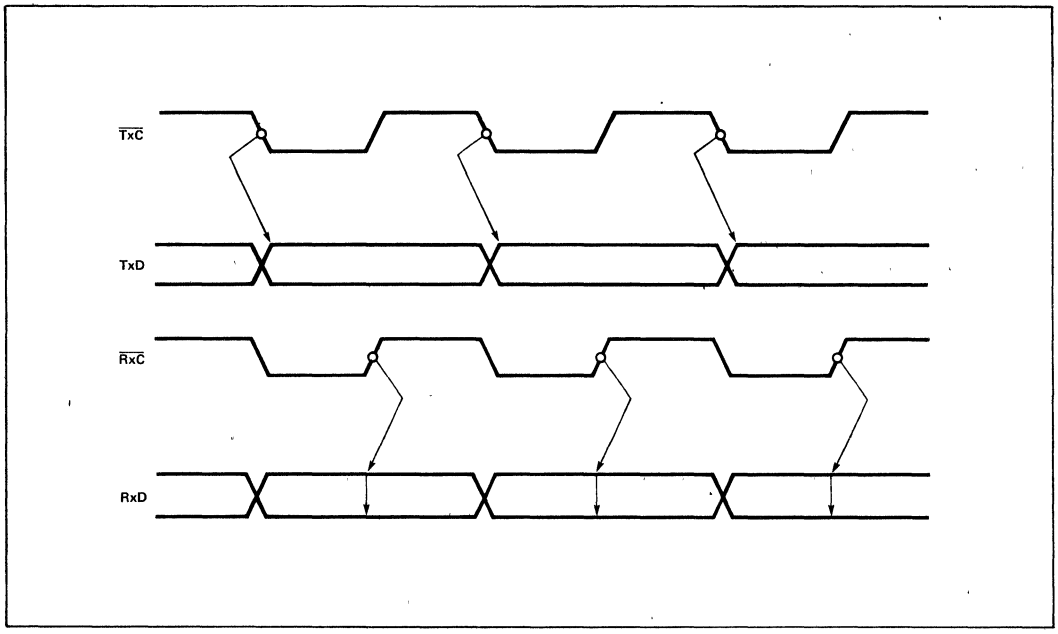


Figure 6. Transmit/Receive Timing

Asynchronous Mode Interface

Although the 8273 is fully compatible with the HDLC/SDLC communication line protocols, which are primarily designed for synchronous communication, the 8273 can also be used in asynchronous applications by using this interface. The interface employs a digital phase locked loop (DPLL) for clock recovery from a receive data stream and programmable NRZI encoding and decoding of data. The use of NRZI coding with SDLC transmission guarantees that within a frame, data transitions will occur at least every five bit times — the longest sequence of ones which may be transmitted without zero-bit insertion. The DPLL should be used only when NRZI coding is used since the NRZI coding will transmit zero sequence as line transitions. The digital phase locked loop also facilitates full-duplex and half-duplex asynchronous implementation with, or without modems.

Digital Phase Locked Loop

In asynchronous applications, the clock is derived from the receiver data stream by the use of the digital phase locked loop (DPLL). The DPLL requires a clock input at 32 times the required baud rate. The receive data (RxD) is sampled with this $\overline{32X\ CLK}$ and the 8273 DPLL supplies a sample pulse nominally centered on the RxD bit cells. The DPLL has a built-in "stiffness" which reduces sensitivity to line noise and bit distortion. This is accomplished by making phase error adjustments in discrete increments. Since the nominal pulse is made to occur at 32 counts of the $\overline{32X\ CLK}$, these counts are subtracted or added to the nominal, depending upon which quadrant of the four error quadrants the data edge occurs in. For example if an RxD edge is detected in quadrant A1, it is apparent that the $\overline{DPLL}$ sample "A" was placed too close to the trailing edge of the data cell; sample "B" will then be placed at T = ($T_{nominal}$ - 2 counts) = 30 counts of the $\overline{32X\ CLK}$ to move the sample pulse "B" toward the nominal center of the next bit cell. A data edge occuring in quadrant B1 would cause a smaller adjustment of phase with T = 31 counts of the $\overline{32X\ CLK}$. Using this technique the $\overline{DPLL}$ pulse will converge to nominal bit center within 12 data bit times, worst case, with constant incoming RxD edges.

A method of attaining bit synchronism following a line idle is to use PRE-FRAME SYNC mode of transmission.

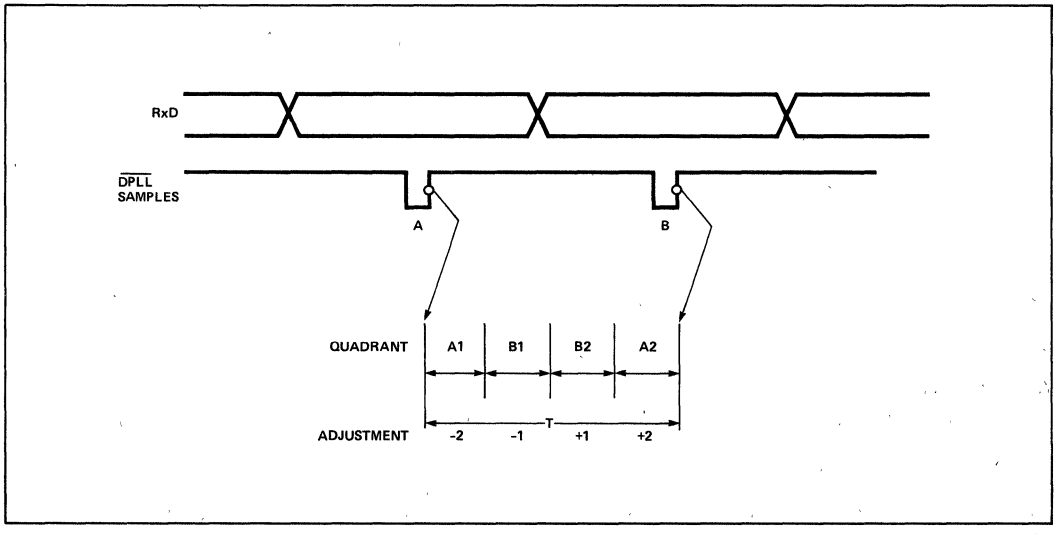
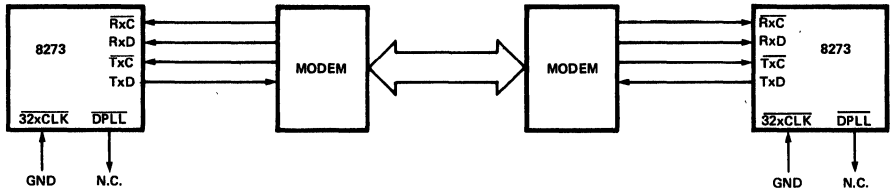
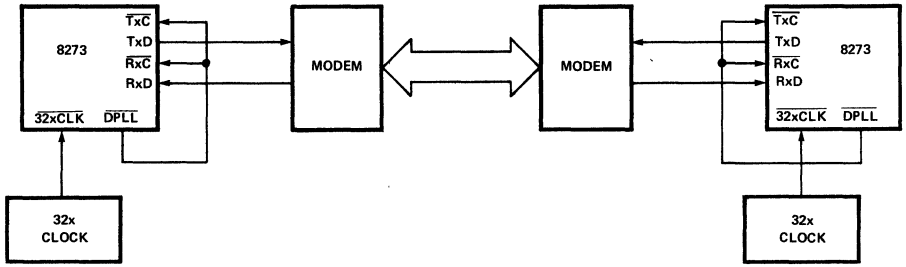


Figure 7. DPLL Sample Timing

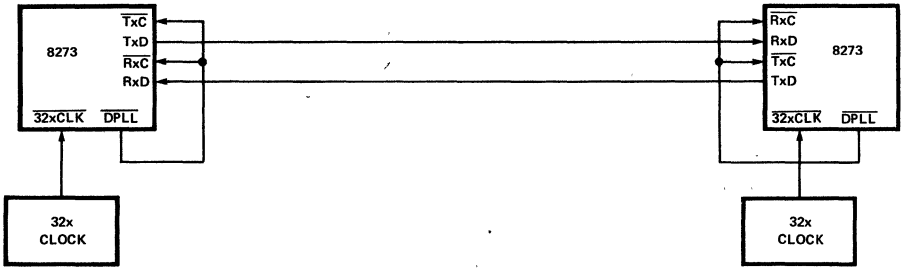
Synchronous Modem — Duplex or Half Duplex Operation



Asynchronous Modems — Duplex or Half Duplex Operation



Asynchronous — No Modems — Duplex or Half Duplex



AFN-00743C

SDLC Loop

The DPLL simplifies the SDLC loop station implementation. In this application, each secondary station on a loop data link is a repeater set in one-bit delay mode. The signals sent out on the loop by the loop controller (primary station) are relayed from station to station then, back to the controller. Any secondary station finding its address in the A field captures the frame for action at that station. All received frames are relayed to the next station on the loop.

Loop stations are required to derive bit timing from the incoming NRZI data stream. The DPLL generates sample Rx clock timing for reception and uses the same clock to implement Tx clock timing.

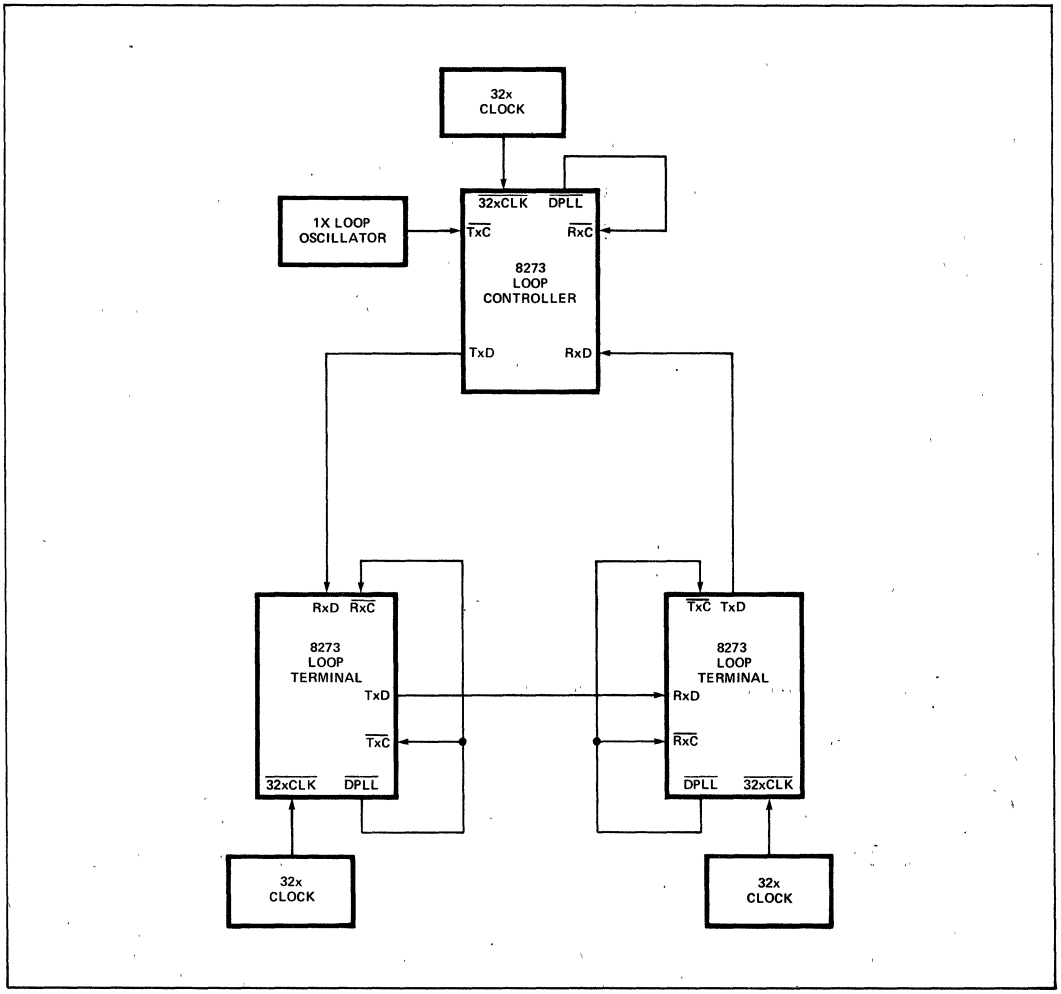


Figure 8. SDLC Loop Application

PRINCIPLES OF OPERATION

The 8273 is an intelligent peripheral controller which relieves the CPU of many of the rote tasks associated with constructing and receiving frames. It is fully compatible with the MCS-80/85™ system bus. As a peripheral device, it accepts commands from a CPU, executes these commands and provides an Interrupt and Result back to the CPU at the end of the execution. The communication with the CPU is done by activation of $\overline{CS}$, $\overline{RD}$, $\overline{WR}$ pins, while the A_1, A_0 select the appropriate registers on the chip as described in the Hardware Description Section.

The 8273 operation is composed of the following sequence of events:

COMMAND PHASE	CPU WRITES COMMAND AND PARAMETERS INTO THE 8273 COMMAND AND PARAMETER REGISTERS.
EXECUTION PHASE	THE 8273 IS ON ITS OWN TO CARRY OUT THE COMMAND.
RESULT PHASE	THE 8273 SIGNALS THE CPU THAT THE EXECUTION HAS FINISHED. THE CPU MUST PERFORM A READ OPERATION OF ONE OR MORE OF THE REGISTERS.

The Command Phase

During the command phase, the software writes a command to the command register. The command bytes provide a general description of the type of operation requested. Many commands require more detailed information about the command. In such a case up to four parameters are written into the parameter register. The flowchart of the command phase indicates that a command may not be issued if the Status Register indicates that the device is busy. Similarly if a parameter is issued when the Parameter Buffer shows full, incorrect operation will occur.

The 8273 is a duplex device and both transmitter and receiver may each be executing a command or passing results at any given time. For this reason separate interrupt pins are provided. However, the command register must be used for one command sequence at a time.

Status Register

The status register contains the status of the 8273 activity. The description is as follows.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CBSY	CBF	CPBF	CRBF	RxINT	TxINT	RxIRA	TxIRA

Bit 7 CBSY (Command Busy)

Indicates in-progress command, set for CPU poll when Command Register is full, reset upon command phase completion. It is improper to write a command when CBSY is set; it results in incorrect operation.

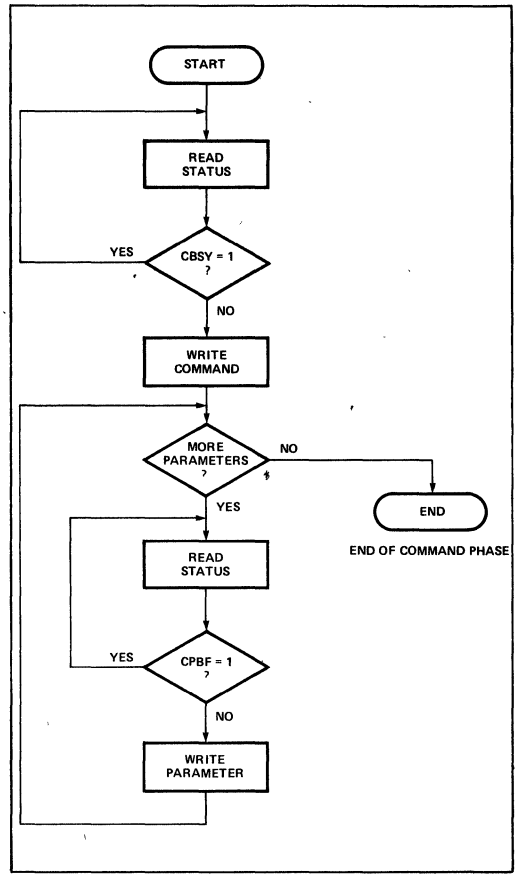


Figure 9. Command Phase Flowchart

Bit 6 CBF (Command Buffer Full)

Indicates that the command register is full, it is reset when the 8273 accepts the command byte but does not imply that execution has begun.

Bit 5 CPBF (Command Parameter Buffer Full)

CPBF is set when the parameter buffer is full, and is reset by the 8273 when it accepts the parameter. The CPU may poll CPBF to determine when additional parameters may be written.

Bit 4 CRBF (Command Result Buffer Full)

Indicates that an executed command immediate result is present in the Result Register. It is set by 8273 and reset when CPU reads the result.

Bit 3 RxINT (Receiver Interrupt)

RxINT indicates that the receiver requires CPU attention. It is identical to RxINT (pin 11) and is set by the 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has received a data byte from the 8273 in a Non-DMA data transfer.

Bit 2 TxINT (Transmitter Interrupt)

The TxINT indicates that the transmitter requires CPU attention. It is identical to TxINT (pin 2). It is set by 8273 either upon good/bad completion of a specified command or by Non-DMA data transfer. It is reset only after the CPU has read the result byte or has transferred transmit data byte to the 8273 in a Non-DMA transfer.

Bit 1 RxIRA (Receiver Interrupt Result Available)

The RxIRA is set by the 8273 when an interrupt result byte is placed in the RxINT register. It is reset after the CPU has read the RxINT register.

Bit 0 TxIRA (Transmitter Interrupt Result Available)

The TxIRA is set by the 8273 when an interrupt result byte is placed in the TxINT register. It is reset when the CPU has read the TxINT register.

The Execution Phase

Upon accepting the last parameter, the 8273 enters into the Execution Phase. The execution phase may consist of a DMA or other activity, and may or may not require CPU intervention. The CPU intervention is eliminated in this phase if the system utilizes DMA for the data transfers, otherwise, for non-DMA data transfers, the CPU is interrupted by the 8273 via TxINT and RxINT pins, for each data byte request.

The Result Phase

During the result phase, the 8273 notifies the CPU of the execution outcome of a command. This phase is initiated by:

1. The successful completion of an operation
2. An error detected during an operation.

To facilitate quick network software decisions, two types of execution results are provided:

1. An Immediate Result
2. A Non-Immediate Result

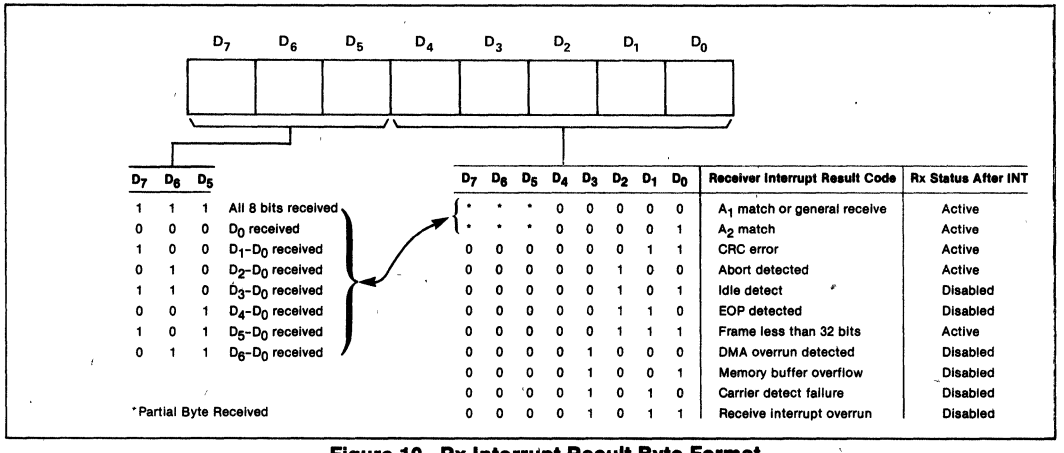


Figure 10. Rx Interrupt Result Byte Format

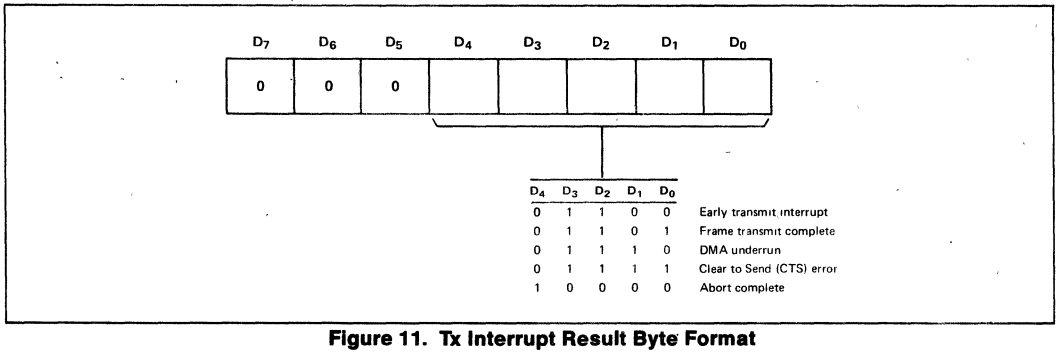


Figure 11. Tx Interrupt Result Byte Format

Immediate result is provided by the 8273 for commands such as Read Port A and Read Port B which have information ($\overline{CTS}$, $\overline{CD}$, $\overline{RTS}$, etc.) that the network software needs to make quick operational decisions.

A command which cannot provide an immediate result will generate an interrupt to signal the beginning of the Result phase. The immediate results are provided in the Result Register; all non-immediate results are available upon device interrupt, through Tx Interrupt Result Register TxI/R or Rx Interrupt Result Register RxI/R. The result may consist of a one-byte interrupt code indicating the condition for the interrupt and, if required, one or more bytes which detail the condition.

Tx and Rx Interrupt Result Registers

The Result Registers have a result code, the three high order bits D_7-D_5 of which are set to zero for all but the receive command. This command result contains a count that indicates the number of bits received in the last byte. If a partial byte is received, the high order bits of the last data byte are indeterminate.

All results indicated in the command summary must be read during the result phase.

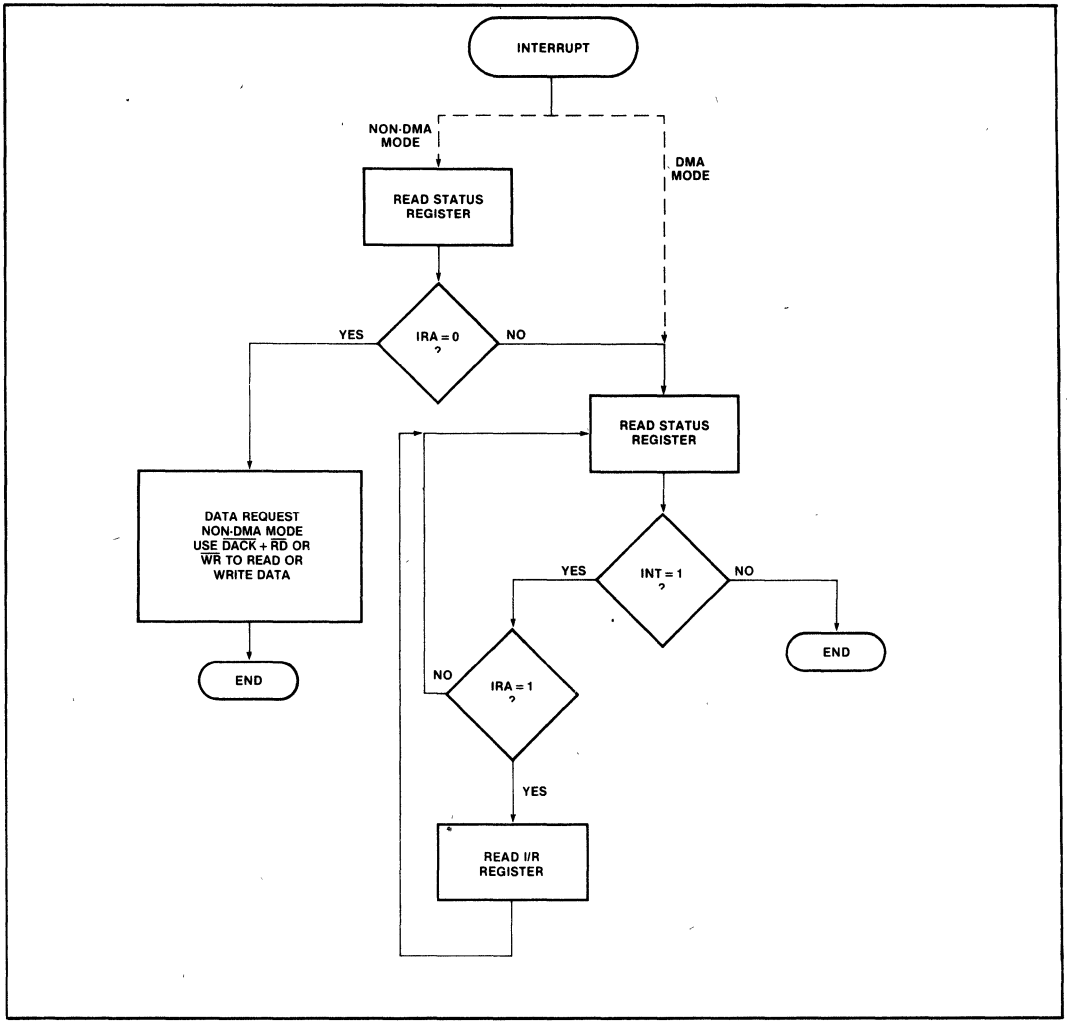


Figure 12. Result Phase Flowchart—Interrupt Results

IMMEDIATE RESULTS

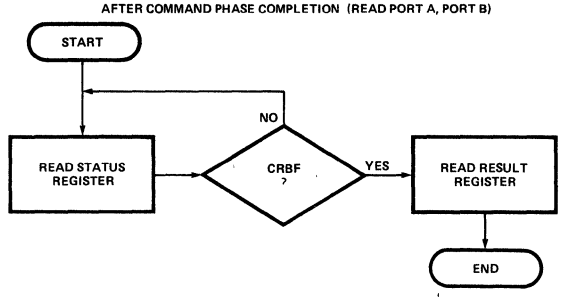


Figure 13. (Rx Interrupt Service)

DETAILED COMMAND DESCRIPTION

General

The 8273 HDLC/SDLC controller supports a comprehensive set of high level commands which allows the 8273 to be readily used in full-duplex, half-duplex, synchronous, asynchronous and SDLC loop configuration, with or without modems. These frame-level commands minimize CPU and software overhead. The 8273 has address and control byte buffers which allow the receive and transmit commands to be used in buffered or non-buffered modes.

In buffered transmit mode, the 8273 transmits a flag automatically, reads the Address and Control buffer registers and transmits the fields, then via DMA, it fetches the information field. The 8273, having transmitted the information field, automatically appends the Frame Check Sequence (FCS) and the end flag. Correspondingly, in buffered read mode, the Address and Control fields are stored in their respective buffer registers and only Information Field is transferred to memory.

In non-buffered transmit mode, the 8273 transmits the beginning flag automatically, then fetches and transmits the Address, Control and Information fields from the memory, appends the FCS character and an end flag. In the non-buffered receive mode the entire contents of a frame are sent to memory with the exception of the flags and FCS.

HDLC Implemenation

HDLC Address and Control field are extendable. The extension is selected by setting the low order bit of the field to be extended to a one, a zero in the low order bit indicates the last byte of the respective field.

Since Address/Control field extension is normally done with software to maximize extension flexibility, the 8273 does not create or operate upon contents of the extended HDLC Address/Control fields. Extended fields are transparently passed by the 8273 to user as either interrupt results or data transfer requests. Software must assemble the fields for transmission and interrogate them upon reception.

However, the user can take advantage of the powerful 8273 commands to minimize CPU/Software overhead and simplify buffer management in handling extended fields. For instance buffered mode can be used to separate the first two bytes, then interrogate the others from buffer. Buffered mode is perfect for a two byte address field.

The 8273 when programmed, recognizes protocol characters unique to HDLC such as Abort, which is a string of seven or more ones (01111111). Since Abort character is the same as the GA (EOP) character used in SDLC Loop applications, Loop Transmit and Receive commands are not recommended to be used in HDLC. HDLC does not support Loop mode.

Initialization Set/Reset Commands

These commands are used to manipulate data within the 8273 registers. The Set commands have a single parameter which is a mask that corresponds to the bits to be set. (They perform a logical-OR of the specified register with the mask provided as a parameter). The Register commands have a single parameter which is a mask that has a zero in the bit positions that are to be reset. (They perform a logical-AND of the specified register with the mask).

Set One-Bit Delay (CMD Code A4)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	1	0	1	0	0	1	0	0
PAR	0	1	1	0	0	0	0	0	0	0

When one bit delay is set, 8273 retransmits the received data stream one bit delayed. This mode is entered at a receiver character boundary, and should only be used by Loop Stations.

Reset One-Bit Delay (CMD Code 64)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	0	1	1	0	0	1	0	0
PAR	0	1	0	1	1	1	1	1	1	1

The 8273 stops the one bit delayed retransmission mode.

Set Data Transfer Mode (CMD Code 97)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	1	0	0	1	0	1	1	1
PAR	0	1	0	0	0	0	0	0	0	1

When the data transfer mode is set, the 8273 will interrupt when data bytes are required for transmission or are available from a receive. If a transmit interrupt occurs and the status indicates that there is no Transmit Result (TxIRA = 0), the interrupt is a transmit data request. If a receive interrupt occurs and the status indicates that there is no receive result (RxIRA = 0), the interrupt is a receive data request.

Reset Data Transfer Mode (CMD Code 57)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	0	1	0	1	0	1	1	1
PAR	0	1	1	1	1	1	1	1	1	0

If the Data Transfer Mode is reset, the 8273 data transfers are performed through the DMA requests without interrupting the CPU.

Set Operating Mode (CMD Code 91)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	0	1	0	0	0	1
PAR:	0	1	0	0						

1 = FLAG STREAM MODE
1 = PREFRAME SYNC MODE
1 = BUFFERED MODE
1 = EARLY INTERRUPT MODE
1 = EOP INTERRUPT MODE
1 = HDLC MODE

Reset Operating Mode (CMD Code 51)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	1	0	1	0	0	0	1
PAR	0	1	1	1						

Any mode switches set in CMD code 91 can be reset using this command by placing zeros in the appropriate positions.

(D5) HDLC Mode

In HDLC mode, a bit sequence of seven ones (01111111) is interpreted as an abort character. Otherwise, eight ones (011111111) signal an abort.

(D4) EOP Interrupt Mode

In EOP interrupt mode, an interrupt is generated whenever an EOP character (01111111) is detected by an active receiver. This mode is useful for the implementation of an SDLC loop controller in detecting the end of a message stream after a loop poll.

(D3) Transmitter Early Interrupt Mode (Tx)

The early interrupt mode is specified to indicate when the 8273 should generate an end of frame interrupt. When set, an early interrupt is generated when the last data character has been passed to the 8273. If the user software responds with another transmit command before the final flag is sent, the final flag interrupt will not be generated and a new frame will immediately begin when the current frame is complete. This permits frames to be separated by a single flag. If no additional Tx commands are provided, a final interrupt will follow.

Note: In buffered mode, if a supervisory frame (no Information) Transmit command is sent in response to an early Transmit Interrupt, the 8273 will repeatedly transmit the same supervisory frame with one flag in between, until a non-supervisory transmit is issued.

Early transmitter interrupt can be used in buffered mode by waiting for a transmit complete interrupt instead of early Transmit Interrupt before issuing a transmit frame command for a supervisory frame. See Figure 14.

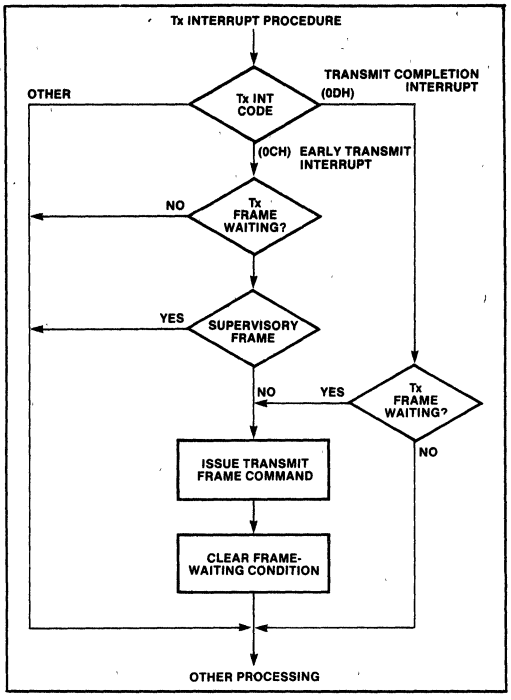


Figure 14.

If this bit is zero, the interrupt will be generated only after the final flag has been transmitted.

(D2) Buffered Mode

If the buffered mode bit is set to a one, the first two bytes (normally the address (A) and control (C) fields) of a frame are buffered by the 8273. If this bit is a zero the address and control fields are passed to and from memory.

(D1) Preframe Sync Mode

If this bit is set to a one the 8273 will transmit two characters before the first flag of a frame.
To guarantee sixteen line transitions, the 8273 sends two bytes of data $(00)_H$ if NRZI is set or data $(55)_H$ if NRZI is not set.

(D0) Flag Stream Mode

If this bit is set to a one, the following table outlines the operation of the transmitter.

TRANSMITTER STATE	ACTION
Idle	Send Flags immediately.
Transmit or Transmit-Transparent Active	Send Flags after the transmission complete
Loop Transmit Active	Ignore command.
1 Bit Delay Active	Ignore command.

If this bit is reset to zero the following table outlines the operation of the transmitter.

TRANSMITTER STATE	ACTION
IDLE	Send Idles on next character boundary.
Transmit or Transmit-Transparent Active	Send Idles after the transmission is complete.
Loop Transmit Active	Ignore command.
1 Bit Delay Active	Ignore command.

Set Serial I/O Mode (CMD Code A0)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	0	1	0	0	0	0	0
PAR.	0	1	0	0	0	0	0			

D_0 = 1 = NRZ1 MODE
D_1 = 1 = TxC → RxC
D_2 = 1 = LOOP BACK TxD → RxD

Reset Serial I/O Mode (CMD Code 60)

This command allows bits set in CMD code A0 to be reset by placing zeros in the appropriate positions.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	0	1	1	0	0	0	0	0
PAR.	0	1	1	1	1	1	1			

(D2) Loop Back

If this bit is set to a one, the transmit data is internally routed to the receive data circuitry.

(D1) TxC → RxC

If this bit is set to a one, the transmit clock is internally routed to the receive clock circuitry. It is normally used with the loop back bit (D2).

(D0) NRZI Mode

If this bit is set to a one, NRZI encoding and decoding of transmit and receive data is provided. If this bit is a zero, the transmit and receive data is treated as a normal positive logic bit stream.

NRZI encoding specifies that a zero causes a change in the polarity of the transmitted signal and a one causes no polarity change. NRZI is used in all asynchronous operations. Refer to IBM document GA27-3093 for details.

Reset Device Command

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TMR:	1	0	0	0	0	0	0	0	0	1
TMR:	1	0	0	0	0	0	0	0	0	0

An 8273 reset command is executed by outputing a $(01)_H$ followed by $(00)_H$ to the reset register (TMR). See 8273 AC timing characteristics for Reset pulse specifications.

The reset command emulates the action of the reset pin.

1. The modem control signals are forced high (inactive level).
2. The 8273 status register flags are cleared.
3. Any commands in progress are terminated immediately.
4. The 8273 enters an idle state until the next command is issued.
5. The Serial I/O and Operating Mode registers are set to zero and DMA data register transfer mode is selected.
6. The device assumes a non-loop SDLC terminal role.

Receive Commands

The 8273 supports three receive commands: General Receive, Selective Receive, and Selective Loop Receive.

General Receive (CMD Code C0)

General receive is a receive mode in which frames are received regardless of the contents of the address field.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	0	0	0	0
PAR·	0	1	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)							
PAR	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)							

NOTES:

1. If buffered mode is specified, the R0, R1 receive frame length (result) is the number of data bytes received.
2. If non-buffered mode is specified, the R0, R1 receive frame length (result) is the number of data bytes received plus two (the count includes the address and control bytes).
3. The frame check sequence (FCS) is not transferred to memory.
4. Frames with less than 32 bits between flags are ignored (no interrupt generated) if the buffered mode is specified
5. In the non-buffered mode an interrupt is generated when a less than 32 bit frame is received, since data transfer requests have occurred.
6. The 8273 receiver is always disabled when an Idle is received after a valid frame. The CPU module must issue a receive command to re-enable the receiver.
7. The intervening ABORT character between a final flag and an IDLE does not generate an interrupt.
8. If an ABORT Character is not preceded by a flag and is followed by an IDLE, an interrupt will be generated for the ABORT followed by an IDLE interrupt one character time later. The reception of an ABORT will disable the receiver.

Selective Receive (CMD Code C1)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	1	1	0	0	0	0	0·	1
PAR	0	1	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)							
PAR.	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)							
PAR:	0	1	RECEIVE FRAME ADDRESS MATCH FIELD ONE (A1)							
PAR.	0	1	RECEIVE FRAME ADDRESS MATCH FIELD TWO (A2)							

Selective receive is a receive mode in which frames are ignored unless the address field matches any one of two address fields given to the 8273 as parameters.

When selective receive is used in HDLC the 8273 looks at the first character, if extended, software must then decide if the message is for this unit.

Selective Loop Receive (CMD Code C2)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
CMD	0	0	1	1	0	0	0	0	1	0	
PAR	0	0	LEAST SIGNIFICANT BYTE OF THE RECEIVE BUFFER LENGTH (B0)								
PAR	0	1	MOST SIGNIFICANT BYTE OF RECEIVE BUFFER LENGTH (B1)								
PAR	0	1	RECEIVE FRAME ADDRESS MATCH FIELD ONE (A1)								
PAR	0	1	RECEIVE FRAME ADDRESS MATCH FIELD TWO (A2)								

Selective loop receive operates like selective receive except that the transmitter is placed in flag stream mode automatically after detecting an EOP (01111111) following a valid received frame. The one bit delay mode is also reset at the end of a selective loop receive.

Receive Disable (CMD Code C5)

Terminates an active receive command immediately.

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	1	1	0	0	0	1	0	1
PAR	NONE									

Transmit Commands

The 8273 supports three transmit commands: Transmit Frame, Loop Transmit, Transmit Transparent.

Transmit Frame (CMD Code C8)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	1	1	0	0	1	0	0	0
PAR	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							
PAR	0	1	ADDRESS FIELD OF TRANSMIT FRAME (A)							
PAR	0	1	CONTROL FIELD OF TRANSMIT FRAME (C)							

Transmits one frame including: initial flag, frame check sequence, and the final flag.

If the buffered mode is specified, the L0, L1, frame length provided as a parameter is the length of the information field and the address and control fields must be input.

In unbuffered mode the frame length provided must be the length of the information field plus two and the address and control fields must be the first two bytes of data. Thus only the frame length bytes are required as parameters.

Loop Transmit (CMD Code CA)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	1	1	0	0	1	0	1	0
PAR	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							
PAR	0	1	ADDRESS FIELD OF TRANSMIT FRAME (A)							
PAR	0	1	CONTROL FIELD OF TRANSMIT FRAME (C)							

Transmits one frame in the same manner as the transmit frame command except:

1. If the flag stream mode is not active transmission will begin after a received EOP has been converted to a flag.
2. If the flag stream mode is active transmission will begin at the next flag boundary for buffered mode or at the third flag boundary for non-buffered mode.
3. At the end of a loop transmit the one-bit delay mode is entered and the flag stream mode is reset.

Transmit Transparent (CMD Coded C9)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	1	1	0	0	1	0	0	1
PAR:	0	1	LEAST SIGNIFICANT BYTE OF FRAME LENGTH (L0)							
PAR	0	1	MOST SIGNIFICANT BYTE OF FRAME LENGTH (L1)							

The 8273 will transmit a block of raw data without protocol, i.e., no zero bit insertion, flags, or frame check sequences.

Abort Transmit Commands

An abort command is supported for each type of transmit command. The abort commands are ignored if a transmit command is not in progress.

Abort Transmit Frame (CMD Code CC)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD.	0	0	1	1	0	0	1	1	0	0
PAR.	NONE									

After an abort character (eight contiguous ones) is transmitted, the transmitter reverts to sending flags or idles as a function of the flag stream mode specified.

Abort Loop Transmit (CMD Code CE)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	1	1	0	0	1	1	1	0
PAR	NONE									

After a flag is transmitted the transmitter reverts to one bit delay mode.

Abort Transmit Transparent (CMD Code CD)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	1	1	0	0	1	1	0	1
PAR:	NONE									

The transmitter reverts to sending flags or idles as a function of the flag stream mode specified.

Modem Control Commands

The modem control commands are used to manipulate the modem control ports.

When read Port A or Port B commands are executed the result of the command is returned in the result register. The Bit Set Port B command requires a parameter that is a mask that corresponds to the bits to be set. The Bit Reset Port B command requires a mask that has a zero in the bit positions that are to be reset.

Read Port A (CMD Code 22)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD:	0	0	0	0	1	0	0	0	1	0

PAR: NONE

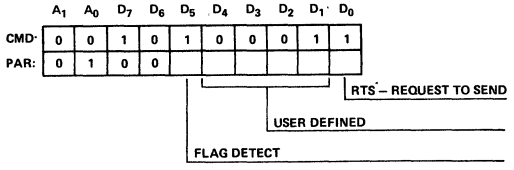
Read Port B (CMD Code 23)

	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CMD	0	0	0	0	1	0	0	0	1	1

PAR NONE

Set Port B Bits (CMD Code A3)

This command allows user defined Port B pins to be set.



(D_5) Flag Detect

This bit can be used to set the flag detect pin. However, it will be reset when the next flag is detected.

(D_4-D_1) User Defined Outputs

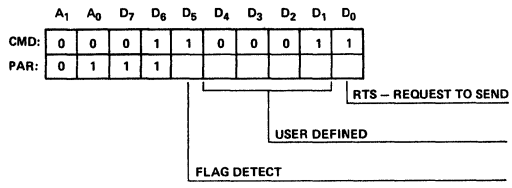
These bits correspond to the state of the PB_4-PB_1 output pins.

(D_0) Request to Send

This is a dedicated 8273 modem control signal, and reflects the same logical state of RTS pin.

Reset Port B Bits (CMD Code 63)

This command allows Port B user defined bits to be reset.



This command allows Port B (D_4-D_1) user defined bits to be reset. These bits correspond to Output Port pins (PB_4-PB_1).

8273 Command Summary

Command Description	Command (HEX)	Parameter	Results	Result Port	Completion Interrupt
Set One Bit Delay	A4	Set Mask	None	—	No
Reset One Bit Delay	64	Reset Mask	None	—	No
Set Data Transfer Mode	97	Set Mask	None	—	No
Reset Data Transfer Mode	57	Reset Mask	None	—	No
Set Operating Mode	91	Set Mask	None	—	No
Reset Operating Mode	51	Reset Mask	None	—	No
Set Serial I/O Mode	A0	Set Mask	None	—	No
Reset Serial I/O Mode	60	Reset Mask	None	—	No
General Receive	C0	B0,B1	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Selective Receive	C1	B0,B1,A1,A2	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Selective Loop Receive	C2	B0,B1,A1,A2	RIC,R0,R1,(A,C)(2)	RXI/R	Yes
Receive Disable	C5	None	None	—	No
Transmit Frame	C8	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Loop Transmit	CA	L0,L1,(A,C)(1)	TIC	TXI/R	Yes
Transmit Transparent	C9	L0,L1	TIC	TXI/R	Yes
Abort Transmit Frame	CC	None	TIC	TXI/R	Yes
Abort Loop Transmit	CE	None	TIC	TXI/R	Yes
Abort Transmit Transparent	CD	None	TIC	TXI/R	Yes
Read Port A	22	None	Port Value	Result	No
Read Port B	23	None	Port Value	Result	No
Set Port B Bit	A3	Set Mask	None	—	No
Reset Port B Bit	63	Reset Mask	None	—	No

NOTES:
1. Issued only when in buffered mode.
2. Read as results only in buffered mode.

8273 Command Summary Key

B0 — Least significant byte of the receive buffer length.
B1 — Most significant byte of the receive buffer length.
L0 — Least significant byte of the Tx frame length.
L1 — Most significant byte of the Tx frame length.
A1 — Receive frame address match field one.
A2 — Receive frame address match field two.
A — Address field of received frame. If non-buffered mode is specified, this result is not provided.
C — Control field of received frame. If non-buffered mode is specified this result is not provided.
RXI/R — Receive interrupt result register.
TXI/R — Transmit interrupt result register.
R0 — Least significant byte of the length of the frame received.
R1 — Most significant byte of the length of the frame received.
RIC — Receiver interrupt result code.
TIC — Transmitter interrupt result code.

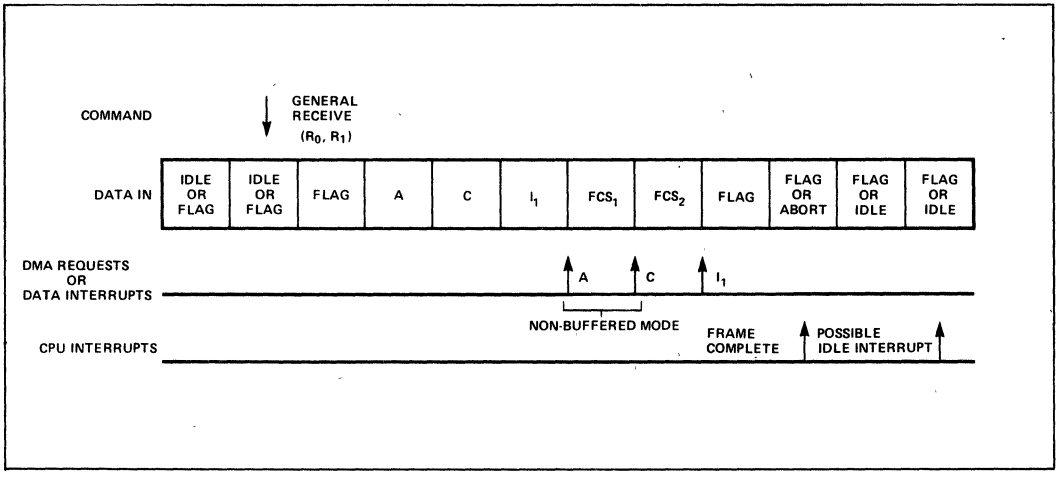


Figure 15. Typical Frame Reception

NOTE:
In order to ensure proper operation to the maximum baud rate, Receive commands or Read/Write Port commands should be written only when either the transmitter or the receiver is inactive. In full duplex systems, it is recommended that these commands be issued after servicing a transmitter interrupt but before a new transmit command is issued.

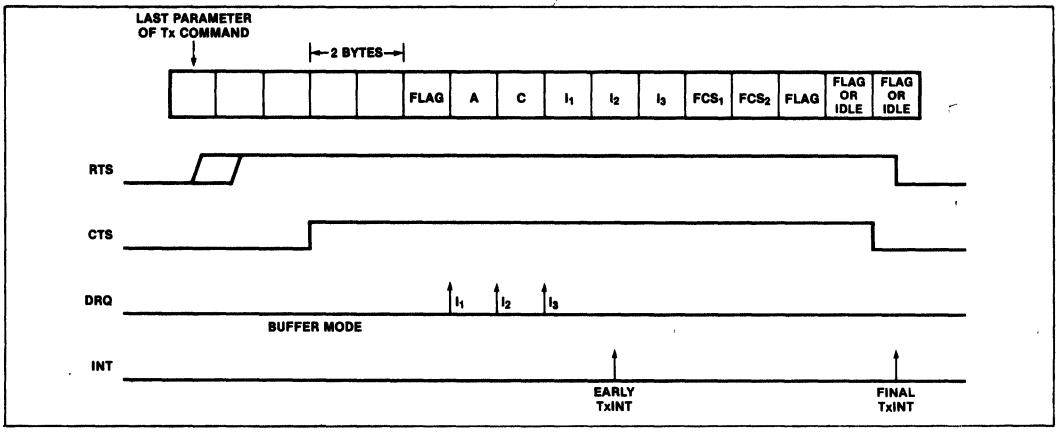


Figure 16a. Typical Frame Transmission, Buffered Mode

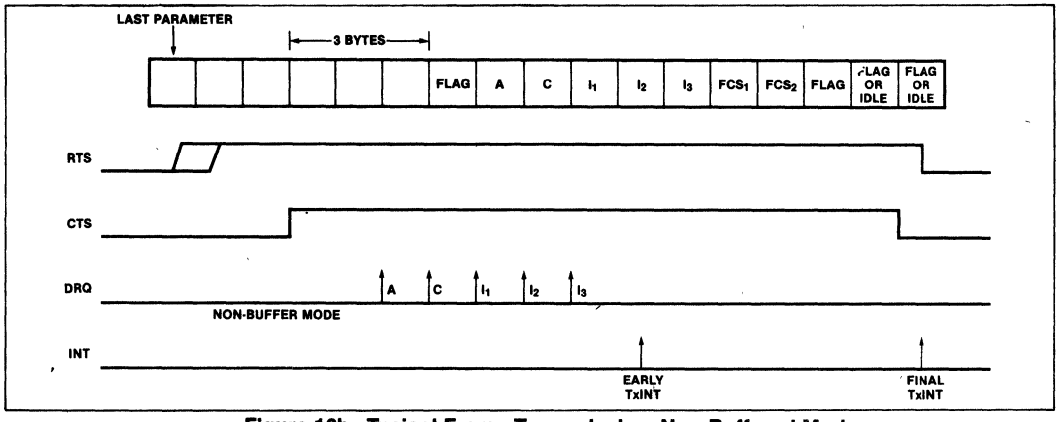


Figure 16b. Typical Frame Transmission, Non-Buffered Mode

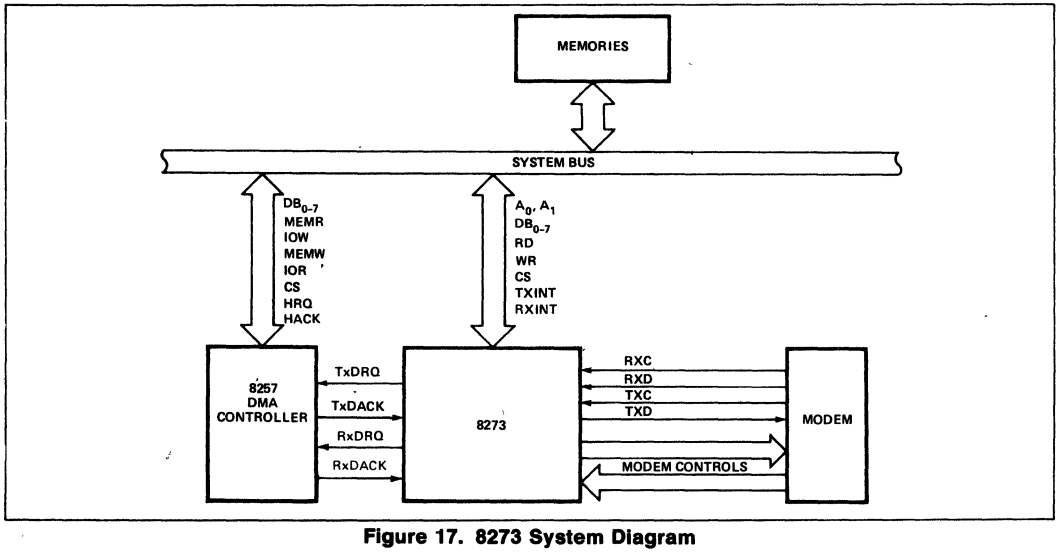


Figure 17. 8273 System Diagram

WAVEFORMS

COMMAND PHASE

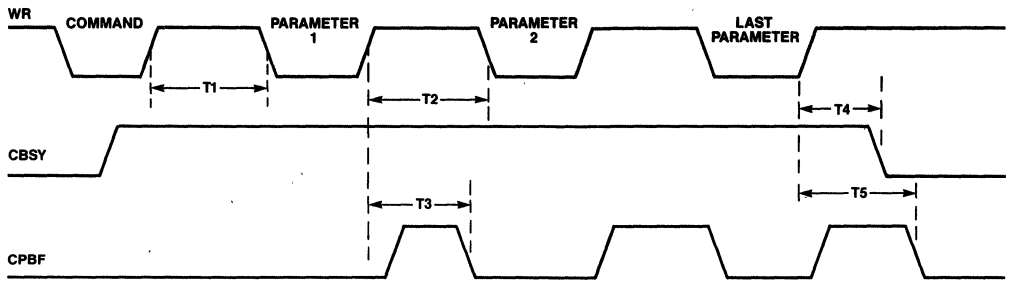


Table 2. Command Phase Timing (Full Duplex)

Symbol	Timing Parameter	Buffered		Non-Buffered		Unit
		Min.	Max.	Min.	Max.	
T1	Between command & first parameter	13	756	13	857	tcy
T2	Between consecutive parameters	10	604	10	705	tcy
T3	Command Parameter Buffer full bit Reset after Parameter loaded	10	604	10	705	tcy
T4	Command busy bit reset after last parameter	128	702	128	803	tcy
T5	CPBF bit reset after last parameter	10	604	10	705	tcy

WAVEFORMS (Continued)

RECEIVER INTERRUPT

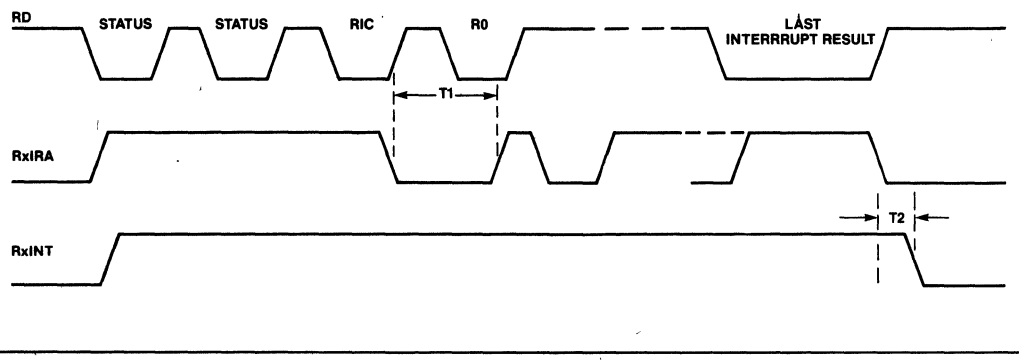


Table 3. Receiver Interrupt Result Timing

Symbol	Timing Parameter (clock cycles)	Buffered		Non-Buffered		Unit
		Min.	Max.	Min.	Max.	
T1	RxIRA bit set after RIC read	18	29	18	29	tcy
T2	RxINT goes away after last Int. Result read	16	27	16	27	tcy

WAVEFORMS (Continued)

TRANSMIT INTERRUPT

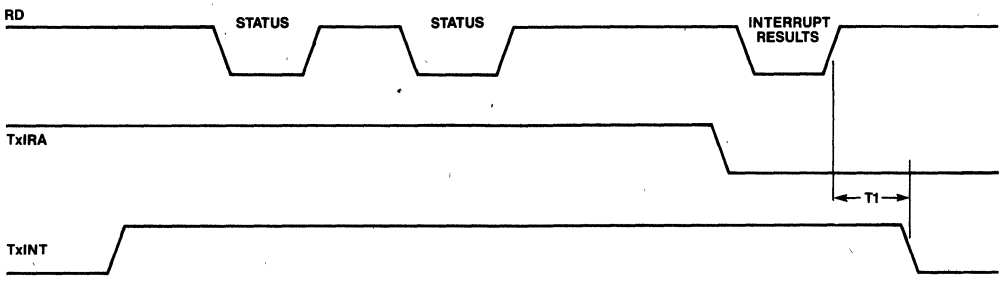


Table 4. Transmit Interrupt Result

Symbol	Timing (Clock Cycle)	Buffered		Non-Buffered		Unit
		Min.	Max.	Min.	Max.	
T1	TxINT inactive after Int. Results read	13	353	13	454	tcy

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin With Respect to Ground ..... -0.5V to +7V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (8273, 8273-4) (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 5%)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA for Data Bus Pins I_{OL} = 1.0 mA for Output Port Pins I_{OL} = 1.6 mA for All Other Pins
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −200 μA for Data Bus Pins I_{OH} = −100 μA for All Other Pins
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Leakage Current		±10	μA	V_{OUT} = V_{CC} to .45V
I_{CC}	V_{CC} Supply Current		180	mA	

CAPACITANCE (8273, 8273-4) (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	t_c = 1 MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured Pins Returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 5%)

CLOCK TIMING (8273)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
t_{CY}	Clock	250		1000	ns	64K Baud Max Operating Rate
t_{CL}	Clock Low	120			ns	
t_{CH}	Clock High	120			ns	

CLOCK TIMING (8273-4)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
t_{CY}	Clock	286		1000	ns	56K Baud Max Operating Rate
t_{CL}	Clock Low	135			ns	
t_{CH}	Clock High	135			ns	

A.C. CHARACTERISTICS (8273, 8273-4) (T_A = 0°C to 70°C, V_{CC} = +5.0V ± 5%)

READ CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{RD}$	0		ns	Note 2
t_{CA}	Select Hold from $\overline{RD}$	0		ns	Note 2
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	Data Delay from Address		300	ns	Note 2
t_{RD}	Data Delay from $\overline{RD}$		200	ns	C_L = 150pF, Note 2
t_{DF}	Output Float Delay	20	100	ns	C_L = 20pF for Minimum; 150pF for Maximum
t_{DC}	DACK Setup to $\overline{RD}$	25		ns	
t_{CD}	DACK Hold from $\overline{RD}$	25		ns	
t_{KD}	Data Delay from DACK		300	ns	

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AC}	Select Setup to $\overline{WR}$	0		ns	
t_{CA}	Select Hold from $\overline{WR}$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}$	150		ns	
t_{WD}	Data Hold from $\overline{WR}$	0		ns	
t_{DC}	DACK Setup to $\overline{WR}$	25		ns	
t_{CD}	DACK Hold from $\overline{WR}$	25		ns	

DMA

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CQ}	Request Hold from $\overline{WR}$ or $\overline{RD}$ (for Non-Burst Mode)		200	ns	

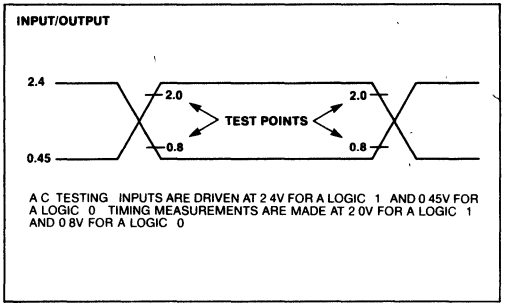
OTHER TIMING

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{RSTW}	Reset Pulse Width	10		t_{CY}	
t_r	Input Signal Rise Time		20	ns	
t_f	Input Signal Fall Time		20	ns	
t_{RSTS}	Reset to First $\overline{IOWR}$	2		t_{CY}	
t_{CY32}	32X Clock Cycle Time	13.02 · t_{CY}		ns	
t_{CL32}	32X Clock Low Time	4 · t_{CY}		ns	
t_{CH32}	32X Clock High Time	4 · t_{CY}		ns	
t_{DPLL}	$\overline{DPLL}$ Output Low	1 · t_{CY} − 50		ns	
t_{DCL}	Data Clock Low	1 · t_{CY} − 50		ns	
t_{DCH}	Data Clock High	2 · t_{CY}		ns	
t_{DCY}	Data Clock	62.5 · t_{CY}		ns	Note 3
t_{TD}	Transmit Data Delay		200	ns	
t_{DS}	Data Setup Time	200		ns	
t_{DH}	Data Hold Time	100		ns	
t_{FLD}	$\overline{FLAG\ DET}$ Output Low	8 · t_{CY} ± 50		ns	

NOTES:

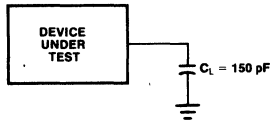
1. All timing measurements are made at the reference voltages unless otherwise specified: Input "1" at 2.0V, "0" at 0.8V; Output "1" at 2.0V, "0" at 0.8V.
2. t_{AD}, t_{RD}, t_{AC}, and t_{CA} are not concurrent specs.
3. If receive commands or Read/Write Port commands are issued while both the transmitter and receiver are active, this specification will be 81.5 T_{CY} min.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

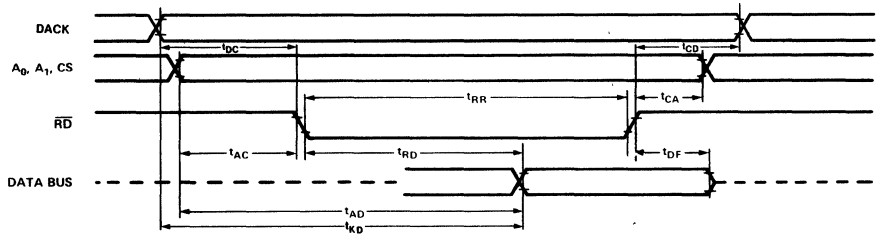
A.C. TESTING LOAD CIRCUIT



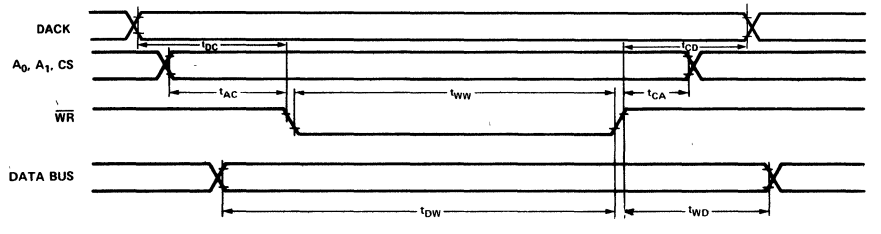
C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

READ

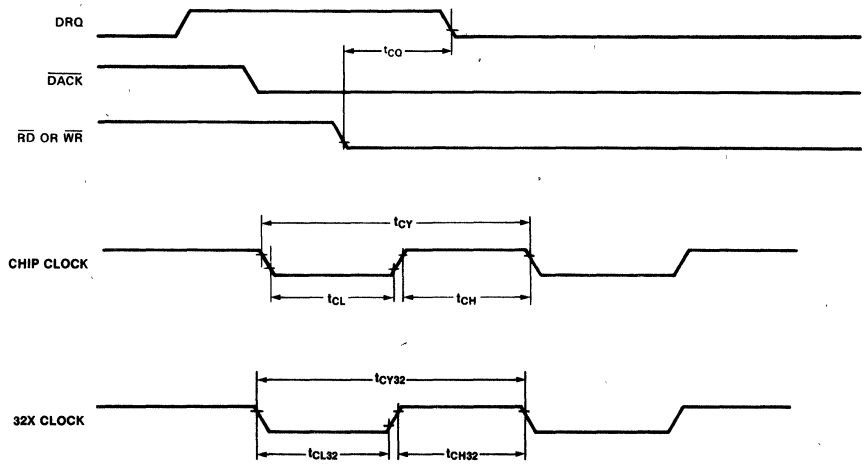


WRITE

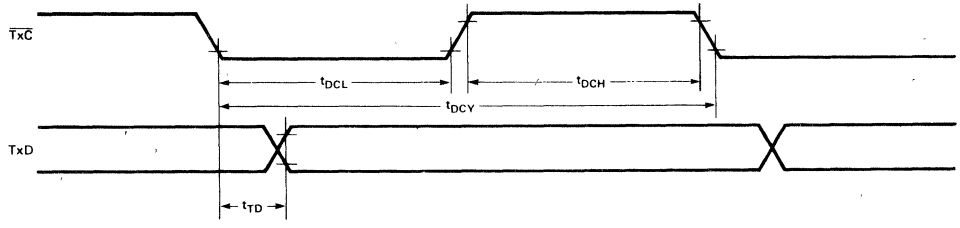


WAVEFORMS (Continued)

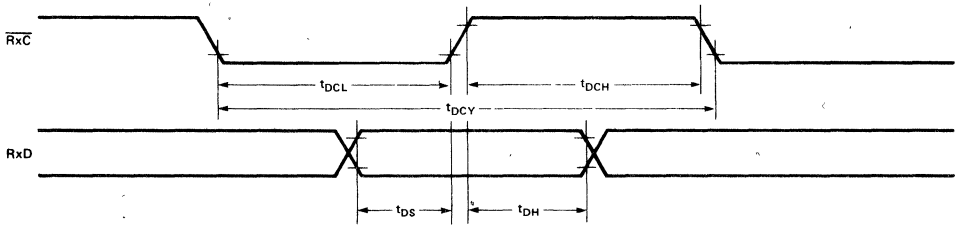
DMA



TRANSMIT

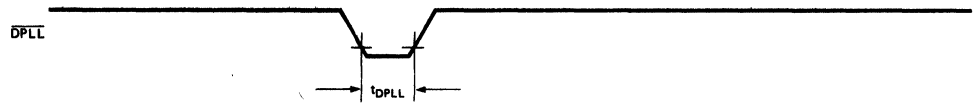


RECEIVE

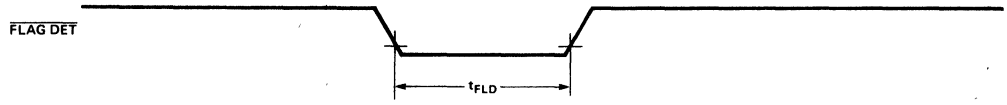


WAVEFORMS (Continued)

DPLL OUTPUT



FLAG DETECT OUTPUT



8274
MULTI-PROTOCOL SERIAL CONTROLLER (MPSC)

- **Asynchronous, Byte Synchronous and Bit Synchronous Operation**
- **Two Independent Full Duplex Transmitters and Receivers**
- **Fully Compatible with 8048, 8051, 8085, 8088, 8086, 80188 and 80186 CPU's; 8257 and 8237 DMA Controllers; and 8089 I/O Proc.**
- **4 Independent DMA Channels**
- **Baud Rate: DC to 880K Baud**
- **Asynchronous:**
  - **—5-8 Bit Character; Odd, Even, or No Parity; 1, 1.5 or 2 Stop Bits**
  - **—Error Detection: Framing, Overrun, and Parity**
- **Byte Synchronous:**
  - **– Character Synchronization, Int. or Ext.**
  - **– One or Two Sync Characters**
  - **– Automatic CRC Generation and Checking (CRC-16)**
  - **– IBM Bisync Compatible**
- **Bit Synchronous:**
  - **– SDLC/HDLC Flag Generation and Recognition**
  - **– 8 Bit Address Recognition**
  - **– Automatic Zero Bit Insertion and Deletion**
  - **– Automatic CRC Generation and Checking (CCITT-16)**
  - **– CCITT X.25 Compatible**
- **Available in EXPRESS**
  - **—Standard Temperature Range**

The Intel® 8274 Multi-Protocol Series Controller (MPSC) is designed to interface High Speed Communications Lines using Asynchronous, IBM Bisync, and SDLC/HDLC protocol to Intel microcomputer systems. It can be interfaced with Intel's MCS-48, -85, -51; iAPX-86, -88, -186 and -188 families, the 8237 DMA Controller, or the 8089 I/O Processor in polled, interrupt driven, or DMA driven modes of operation.

The MPSC is a 40 pin device fabricated using Intel's High Performance HMOS Technology.

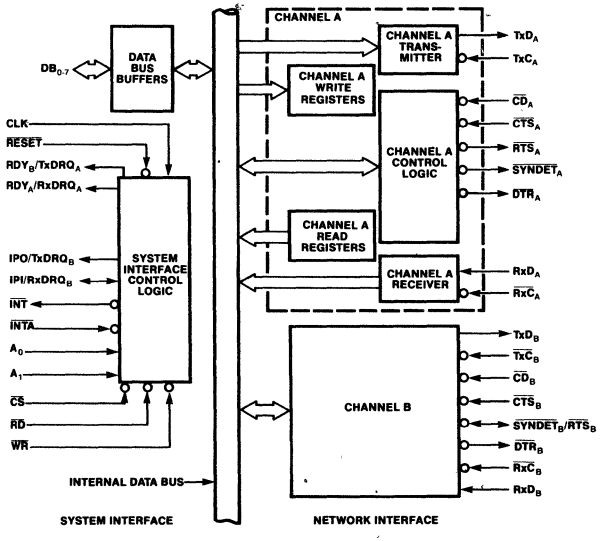


Figure 1. Block Diagram

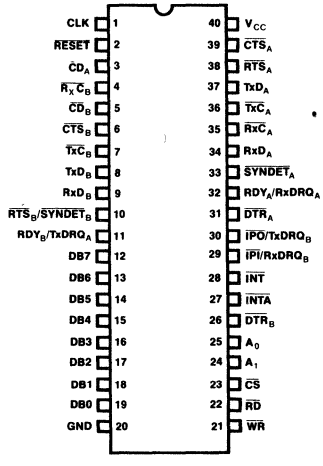


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
CLK	1	I	**Clock:** System clock, TTL compatible.
$\overline{RESET}$	2	I	**Reset:** A low signal on this pin will force the MPSC to an idle state TxD_A and TxD_B are forced high. The modem interface output signals are forced high. The MPSC will remain idle until the control registers are initialized. Reset must be true for one complete CLK cycle.
$\overline{CD_A}$	3	I	**Carrier Detect (Channel A):** This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD_A line If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD_A}$ has been activated.
$\overline{RxC_B}$	4	I	**Receive Clock (Channel B):** The serial data are shifted into the Receive Data input (RxD_B) on the rising edge of the Receive Clock
$\overline{CD_B}$	5	I	**Carrier Detect (Channel B):** This interface signal is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD_B line If the auto enable control is set the 8274 will not enable the serial receiver until $\overline{CD_B}$ has been activated.
$\overline{CTS_B}$	6	I	**Clear to Send (Channel B):** This interface signal is supplied by the modem in response to an active $\overline{RTS}$ signal. $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated
$\overline{TxC_B}$	7	I	**Transmit Clock (Channel B):** The serial data are shifted out from the Transmit Data output (TxD_B) on the falling edge of the Transmit Clock
TxD_B	8	O	**Transmit Data (Channel B):** This pin transmits serial data to the communications channel (Channel B)
RxD_B	9	I	**Receive Data (Channel B):** This pin receives serial data from the communications channel (Channel B)
$\overline{SYNDET_B}$ /$\overline{RTS_B}$	10	I/O	**Synchronous Detection (Channel B):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input) In SDLC mode, this pin is an output indicating Flag detection In asynchronous mode it is a general purpose input (Channel B) **Request to Send (Channel B):** General purpose output, generally used to signal that Channel B is ready to send data $\overline{SYNDET_B}$ or $\overline{RTS_B}$ selection is done by WR2, D7 (Channel A)
RDY_B/ $TxDRQ_A$	11	O	**Ready (Channel B)/Transmitter DMA Request (Channel A):** In mode 0 this pin is RDY_B and is used to synchronize data transfers between the processor and the MPSC (Channel B). In modes 1 and 2 this pin is $TxDRQ_A$ and is used by the Channel A transmitter to request a DMA transfer
DB7	12	I/O	**Data Bus:** The Data Bus lines are bidirectional three state lines which interface with the system's Data Bus.
DB6	13		
DB5	14		
DB4	15		
DB3	16		
DB2	17		
DB1	18		
DB0	19		
GND	20		**Ground.**
V_{CC}	40		**Power:** +5V Supply
$\overline{CTS_A}$	39	I	**Clear to Send (Channel A):** This interface signal is supplied by the Modem in response to an active $\overline{RTS}$ signal $\overline{CTS}$ indicates that the data terminal/computer equipment is permitted to transmit data In addition, if the auto enable control is set, the 8274 will not transmit data bytes until $\overline{CTS}$ has been activated
$\overline{RTS_A}$	38	O	**Request To Send (Channel A):** general purpose output commonly used to signal that Channel A is ready to send data
TxD_A	37	O	**Transmit Data (Channel A):** This pin transmits serial data to the communications channel (Channel A)
$\overline{TxC_A}$	36	I	**Transmit Clock (Channel A):** The serial data are shifted out from the Transmit Data output (TxD_A) on the falling edge of the Transmit Clock
$\overline{RxC_A}$	35	I	**Receive Clock (Channel A):** The serial data are shifted into the Receive Data input (RxD_A) on the rising edge of the Receive Clock.
RxD_A	34	I	**Receive Data (Channel A):** This pin receives serial data from the communications channel (Channel A)
$\overline{SYNDET_A}$	33	I/O	**Synchronous Detection (Channel A):** This pin is used in byte synchronous mode as either an internal sync detect (output) or as a means to force external synchronization (input) In SDLC mode, this pin is an output indicating flag detection In asynchronous mode it is a general purpose input (Channel A)
RDY_A/ $RxDRQ_A$	32	O	**Ready:** In mode 0 this pin is RDY_A and is used to synchronize data transfers between the processor and the MPSC (Channel A). In modes 1 and 2 this pin is $RxDRQ_A$ and is used by the channel A receiver to request a DMA transfer
$\overline{DTR_A}$	31	O	**Data Terminal Ready (Channel A):** General purpose output

AFN-01701B

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{IPO}$/ $TxDRQ_B$	30	O	**Interrupt Priority Out/Transmitter DMA Request (Channel B):** In modes 0 and 1, this pin is Interrupt Priority Out. It is used to establish a hardware interrupt priority scheme with $\overline{IPI}$. It is low only if $\overline{IPI}$ is low and the controlling processor is not servicing an interrupt from this MPSC. In mode 2 it is $TxDRQ_B$ and is used to request a DMA cycle for a transmit operation (Channel B)
$\overline{IPI}$/ $RxDRQ_B$	29	I/O	**Interrupt Priority In/Receiver DMA Request (Channel B):** In modes 0 and 1, $\overline{IPI}$ is Interrupt Priority In. A low on $\overline{IPI}$ means that no higher priority device is being serviced by the controlling processor's interrupt service routine. In mode 2 this pin is $RxDRQ_B$ and is used to request a DMA cycle for a receive operation (Channel B). In Interrupt mode, this pin must be tied low.
$\overline{INT}$	28	O	**Interrupt:** The interrupt signal indicates that the highest priority internal interrupt requires service (open collector) Priority can be resolved via an external interrupt controller or a daisy-chain scheme.
$\overline{INTA}$	27	I	**Interrupt Acknowledge:** This Interrupt Ackowledge signal allows the highest priority interrupting device to generate an interrupt vector. This pin must be pulled high (inactive) in non-vector mode.
$\overline{DTR_B}$	26	O	**Data Terminal Ready (Channel B):** This is a general purpose output
A_0	25	I	**Address:** This line selects Channel A or B during data or command transfers. A low selects Channel A.
A_1	24	I	**Address:** This line selects between data or command information transfer A low means data.
$\overline{CS}$	23	I	**Chip Select:** This signal selects the MSPC and enables reading from or writing into its registers
$\overline{RD}$	22	I	**Read:** Read controls a data byte or status byte transfer from the MPSC to the CPU.
$\overline{WR}$	21	I	**Write:** Write controls transfer of data or commands to the MPSC.

RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals ($\overline{RTS}$, $\overline{DTR}$) are forced high. In addition, the pointers registers are set to zero.

GENERAL DESCRIPTION

The Intel 8274 Multi-Protocol Serial Controller is a microcomputer peripheral device which supports Asynchronous, Byte Synchronous (Monosync, IBM Bisync), and Bit Synchronous (ISO's HDLC, IBM's SDLC) protocols. This controller's flexible architecture allows easy implementation of many variations of these three protocols with low software and hardware overhead.

The Multi-Protocol Serial controller (MPSC) implements two independent serial receiver/transmitter channels.

The MPSC supports several microprocessor interface options: Polled, Wait, Interrupt driven and DMA driven. The MPSC is designed to support INTEL'S MCS-85 and iAPX 86, 88, 186, 188 families.

FUNCTIONAL DESCRIPTION

Additional information on Asynchronous and Synchronous Communications with the 8274 is available respectively in the Applications Notes AP 134 and AP 145.

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B).

In the following discussion, the writable registers will be referred to as WRO through WR7 and the readable registers will be referred to as RRO through RR2.

This section of the data sheet describes how the Asynchronous and Synchronous protocols are implemented in the MPSC. It describes general considerations, transmit operation, and receive operation for Asynchronous, Byte Synchronous, and Bit Synchronous protocols.

ASYNCHRONOUS OPERATIONS

TRANSMITTER/RECEIVER INITIALIZATION

(See Detailed Command Description Section for complete information)

In order to operate in asynchronous mode, each MPSC channel must be initialized with the following information:

1. Transmit/Receive Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 1, 16, 32, or 64 times the data-link bit rate. If the X1 clock mode is selected, the bit synchronization must be accomplished externally.

2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1 1/2, or 2.

3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4.

4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3.

5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3.

6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 0).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

								Number of Bits Transmitted
D7	D6	D5	D4	D3	D2	D1	D0	(Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5

8. Interrupt Mode.

For data transmission via a modem or RS-232-C interface, the following information must also be specified:

1. The Request To Send (RTS) (WR5; D1) and Data Terminal Ready (DTR) (WR5; D7) bits must be set along with the Transmit Enable bit (WR5; D3).

2. Auto Enable may be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. However, the Transmit Enable bit (WR3; D3) and Receive Enable bit (WR3; D1) must be set in order to use the Auto Enable mode. Auto Enable is controlled by bit 5 of WR3.

When loading Initialization parameters into the MPSC, WR4 information must be written before the WR1, WR3, WR5 parameters commands.

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Intel Application Note AP 134.

TRANSMIT

The transmit function begins when the Transmit Enable bit (WR5; D3) is set. The MPSC automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits (1, 1.5 or 2 bits) to the data character being transmitted. 1.5 stop bits option must be used with X16, X32 or X64 clock options only.

The serial data are shifted out from the Transmit Data (TxD) output on the falling edge of the Transmit Clock (TxC) input at a rate programmable to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the TxC input.

The TxD output is held high when the transmitter has no data to send, unless, under program control, the Send Break (WR5; D4) command is issued to hold the TxD low.

If the External/Status Interrupt bit (WR1; D0) is set, the status of $\overline{CD}$, $\overline{CTS}$ and $\overline{SYNDET}$ are monitored and, if any changes occur for a period of time greater than the minimum specified pulse width, an interrupt is generated. $\overline{CTS}$ is usually monitored using this interrupt feature (e.g. Auto Enable option).

The Transmit Buffer Empty bit (RRO; D2) is set by the MPSC when the data byte from the buffer is loaded in the transmit shift register. Data should be written to the MPSC only when the Tx buffer becomes empty to prevent overwriting.

Receive

The receive function begins when the Receive Enable (WR3; D0) bit is set. If the Auto Enable (WR3: D5) option is selected, then Carrier Detect ($\overline{CD}$) must also be low. A valid start bit is detected if a low persists for at least 1/2 bit time on the Receive Data (RxD) input.

The data is sampled at mid-bit time, on the rising edge of RxC, until the entire character is assembled. The receiver inserts 1's when a character is less than 8 bits. If parity (WR4; D0) is enabled and the character is less than 8 bits the parity bit is not stripped from the character.

Error Reporting

The receiver also stores error status for each of the 3 data characters in the data buffer. Three error conditions may be encountered during data reception in the asynchronous mode:

1. **Parity.** If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4). When a parity error is detected, the parity error flag (RR1; D4) is set and remains set until it is reset by the Error Reset command (WR0; D5, D4, D3).

2. **Framing.** A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled). When a Framing Error is detected, the Framing Error bit (RR1; D6) is set. The detection of a Framing Error adds an additional 1/2 bit time to the character time so the Framing Error is not interpreted as a new start bit.

3. **Overrun.** If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1; D5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. Only the overwritten character is flagged with the Receive Overrun bit. The Receive Overrun bit (RR1, D5) is reset by the Error Reset command (WR0; D5, D4, D3).

External/Status Latches

The MPSC continuously monitors the state of five external/status conditions:

1. CTS — clear-to-send input pin.

2. CD — carrier-detect input pin.

3. SYNDET — sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.

4. BREAK — a break condition (series of space bits on the receiver input pin).

5. Tx UNDERRUN/EOM — Transmitter Underrun/End of Message.

A change of state in any of these monitored conditions will cause the associated status bit in RR0 to be latched (and optionally cause an interrupt).

If the External/Status Interrupt bit (WR1; D0) is enabled, Break Detect (RR0; D7) and Carrier Detect (RR0; D3) will cause an interrupt. Reset External/Status interrupts (WR0; D5, D4, D3) will clear Break Detect and Carrier Detect bits if they are set.

Asynchronous Mode Register Setup

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	0	0	0	0	Rx ENABLE
WR4	00 X1 Clock 01 X16 Clock 10 X32 Clock 11 X64 Clock		0	0	01 1 STOP BIT 10 1½ STOP BITS 11 2 STOP BITS		EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx≤5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	0	RTS	0

SYNCHRONOUS OPERATION—MONOSYNC, BISYNC

General

The MPSC must be initialized with the following parameters: odd or even parity (WR4; D1,D0), X1 clock mode (WR4; D7, D6), 8- or 16-bit sync character (WR4; D5, D4), CRC polynomial (WR5; D2), Transmitter Enable (WR5; D3), interrupt modes (WR1, WR2), transmit character length (WR5; D6, D5) and receive character length (WR3; D7, D6). WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The data is transmitted on the falling edge of the Transmit Clock, (TxC) and is received on the rising edge of Receive Clock (RxC). The X1 clock is used for both transmit and receive operations for all three sync modes: Mono, Bi and External.

Transmit Set-Up—Monosync, Bisync

Transmit data is held high after channel reset, or if the transmitter is not enabled. A break may be programmed to generate a spacing line that begins as soon as the Send Break (WR5; D4) bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

Using interrupts for data transfer requires that the Transmit Interrupt/DMA Enable bit (WR1; D1) be set. An interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied

Synchronous Mode Register Setup—Monosync, Bisync

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5 b/char 01 Rx 7 b/char 10 Rx 6 b/char 11 Rx 8 b/char		AUTO ENABLE	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	Rx ENABLE
WR4	0	0	00 8 bit Sync 01 16 bit Sync 11 Ext Sync		0	0	EVEN/ $\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 Tx≤5 b/char 01 Tx 7 b/char 10 Tx 6 b/char 11 Tx 8 b/char		SEND BREAK	Tx ENABLE	1 (SELECTS CRC-16)	RTS	Tx CRC ENABLE

either by writing another character into the transmitter or by resetting the Transmitter Interrupt/DMA Pending latch with a Reset Transmitter Interrupt/DMA Pending Command (WR0; D5, D4, D3). If nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupt, but this situation does cause a Transmit Underrun condition (RR0; D6).

Data Transfers using the RDY signal are for software controlled data transfers such as block moves. RDY tells the CPU that the MPSC is not ready to accept/provide data and that the CPU must extend the output/input cycle. DMA data transfers use the TxDRQ A/B signals which indicate that the transmit buffer is empty, and that the MPSC is ready to accept the next data character. If the data character is not loaded into the MPSC by the time the transmit shift register is empty, the MPSC enters the Transmit Underrun condition.

The MPSC has two programmable options for solving the transmit underrun condition: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0; D6) is in a set condition allowing the insertion of sync characters when there is no data to send. The CRC is not calculated on these automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data to send.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded in the Transmit Shift Register. The status register indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded into the Tx Shift Register). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5; D3).

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5; D0) indicates CRC accumulation when the program sends the first data character to the MPSC. Although the MPSC automatically transmits up to two sync characters (16 bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The Transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded into the transmit shift register. To ensure this bit in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the MPSC.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16 bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the MPSC.

In the transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16) (WR4; D5, D4). When in the Monosync mode, the transmitter sends from WR6 and the receiver compares against WR7. One of two CRC polynomials, CRC 16 or SDLC, may be used with synchronous modes. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command (WR0; D7, D6).

The External/Status interrupt (WR1; D0) mode can be used to monitor the status of the $\overline{CTS}$ input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enable (WR3; D5) feature can be used to enable the transmitter when $\overline{CTS}$ is active. The first data transfer to the MPSC can begin when the External/Status interrupt occurs (CTS (RR0; D5) status bit set) following the Transmit Enable command (WR5; D3).

Receive

After a channel reset, the receiver is in the Hunt phase, during which the MPSC looks for character synchronization. The Hunt begins only when the receiver is enabled and data transfer begins only when character synchronization has been achieved. If character synchronization is lost, the hunt phase can be re-entered by writing the Enter Hunt Phase (WR3; D4) bit. The assembly of received data continues until the MPSC is reset or until the receiver is

disabled (by command or by $\overline{CD}$ while in the Auto Enables mode) or until the CPU sets the Enter Hunt Phase bit. Under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit (WR3; D1) bit. After character synchronization is achieved the assembled characters are transferred to the receive data FIFO. After receiving the first data character, the Sync Character Load Inhibit bit should be reset to zero so that all characters are received, including the sync characters. This is important because the received CRC may look like a sync character and not get received.

Data may be transferred with or without interrupts. Transferring data without interrupts is used for a purely polled operation or for off-line conditions. There are three interrupt modes available for data transfer: Interrupt on First Character Only, Interrupt on Every Character, and Special Receive Conditions Interrupt.

Interrupt on First Character Only mode is normally used to start a polling loop, a block transfer sequence using RDY to synchronize the CPU to the incoming data rate, or a DMA transfer using the RxDRQ signal. The MPSC interrupts on the first character and thereafter only interrupts after a Special Receive Condition is detected. This mode can be reinitialized using the Enable Interrupt On Next Receive Character (WR0; D5, D4, D3) command which allows the next character received to generate an interrupt. Parity Errors do not cause interrupts, but End of Frame (SDLC operation) and Receive Overrun do cause interrupts in this mode. If the external status interrupts (WR1; D0) are enabled an interrupt may be generated any time the $\overline{CD}$ changes state.

Interrupt On Every Character mode generates an interrupt whenever a character enters the receive buffer. Errors and Special Receive Conditions generate a special vector if the Status Affects Vector (WR1B; D2) is selected. Also the Parity Error may be programmed (WR1; D4, D3) not to generate the special vector while in the Interrupt On Every Character mode.

The Special Receive Condition interrupt can only occur while in the Receive Interrupt On First Character Only or the Interrupt On Every Receive Character modes. The Special Receive Condition interrupt is caused by the Receive Overrun (RR1; D5) error condition. The error status reflects an error in the current word in the receive buffer, in addition to any Parity or Overrun errors since the last Error Reset (WR0; D5, D4, D3). The Receive Overrun and Parity error status bits are latched and can only be reset by the Error Reset (WR0; D5, D4, D3) command.

The CRC check result may be obtained by checking for CRC bit (RR1; D6). This bit gives the valid CRC result 16 bit times after the second CRC byte has been read from the MPSC. After reading the second CRC byte, the user software must read two more characters (may be sync characters) before checking for CRC result in RR1. Also for proper CRC computation by the receiver, the user software must reset the Receive CRC Checker (WR0; D7, D6) after receiving the first valid data character. The receive CRC Enable bit (WR3; D3) may also be enabled at this time.

SYNCHRONOUS OPERATION—SDLC

General

Like the other synchronous operations the SDLC mode must be initialized with the following parameters: SDLC mode (WR4; D5, D4), SDLC polynomial (WR5; D2), Request to Send, Data Terminal Ready,

Synchronous Mode Register Setup—SDLC/HDLC

	D7	D6	D5	D4	D3	D2	D1	D0
WR3	00 Rx 5b/char 01 Rx 7b/char 10 Rx 6b/char 11 Rx 8b/char		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1 0 (SELECTS SDLC/ HDLC MODE)		0	0	0	0
WR5	DTR	00 Tx ≤5b/char 01 Tx 7b/char 10 Tx 6b/char 11 Tx 8b/char		SEND BREAK	Tx ENABLE	0 (SELECTS SDLC/ HDLC CRC)	RTS	Tx CRC ENABLE

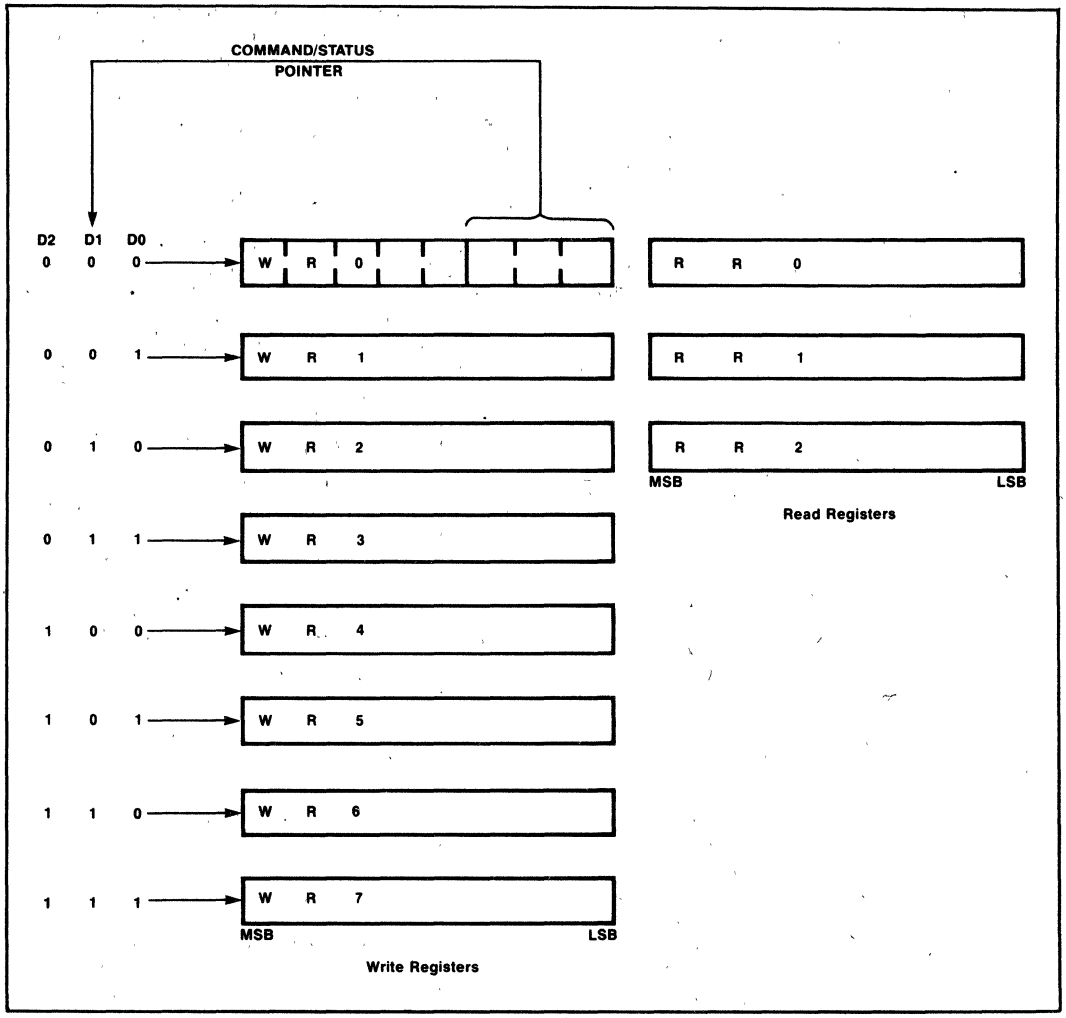


Figure 3. Command/Status Register Architecture (each serial channel)

Command, parameter, and status information is stored in 21 registers within the MPSC (8 writable registers for each channel, 2 readable registers for Channel A and 3 readable registers for Channel B). They are all accessed via the command ports.

An internal pointer register selects which of the command or status registers will be read or written during a command/status access of an MPSC channel.

After reset, the contents of the pointer registers are zero. The first write to a command register causes the data to be loaded into Write Register 0 (WR0). The three least significant bits of WR0 are loaded into the Command/Status Pointer. The next read or write operation accesses the read or write register selected by the pointer. The pointer is reset after the read or write operation is completed.

transmit character length (WR5; D6, D5), interrupt modes (WR1; WR2), Transmit Enable (WR5; D3), Receive Enable (WR3; D0), Auto Enable (WR3; D5) and External/Status Interrupt (WR1; D0).WR4 parameters must be written before WR1, WR3, WR5, WR6 and WR7.

The Interrupt modes for SDLC operation are similar to those discussed previously in the synchronous operations section.

Transmit

After a channel reset, the MPSC begins sending SDLC flags.

Following the flags in an SDLC operation the 8-bit address field, control field and information field may be sent to the MPSC by the microprocessor. The MPSC transmits the Frame Check Sequence using the Transmit Underrun feature. The MPSC automatically inserts a zero after every sequence of 5 consecutive 1's except when transmitting Flags or Aborts.

SDLC—like protocols do not have provision for fill characters within a message. The MPSC therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by sending the two bytes of CRC and then one or more flags. This allows very high-speed transmissions under DMA or CPU control without requiring the CPU to respond quickly to the end-of-message situation.

After a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Flag characters are sent. The MPSC begins to send the frame when data is written into the transmit buffer. Between the time the first data byte is written, and the end of the message, the Reset Transmit Underrun/EOM (WR0; D7, D6) command must be issued. The Transmit Underrun/EOM status bit (RR0; D6) is in the reset state at the end of the message which automatically sends the CRC characters.

The MPSC may be programmed to issue a send Abort command (WR0; D5, D4, D3). This command causes at least eight 1's but less than fourteen 1's to be sent before the line reverts to continuous flags.

Receive

After initialization, the MPSC enters the Hunt phase, and remains in the Hunt phase until the first Flag is received. The MPSC never again enters the Hunt phase unless the microprocessor writes the Enter Hunt command. The MPSC will also detect flags separated by a single zero. For example, the bit pattern 011111101111110 will be detected as two flags.

The MPSC can be programmed to receive all frames or it can be programmed to the Address Search Mode. In the Address Search Mode, only frames with addresses that match the value in WR6 or the global address (0FFH) are received by the MPSC. Extended address recognition must be done by the microprocessor software.

The control and information fields are received as data.

SDLC/HDLC CRC calculation does not have an 8-bit delay, since all characters are included in the calculation, unlike Byte Synchronous Protocols.

Reception of an abort sequence (7 or more 1's) will cause the Break/Abort bit (RR0; D7) to be set and will cause an External/Status interrupt, if enabled. After the Reset External/Status Interrupts Command has been issued, a second interrupt will occur at the end of the abort sequence.

MPSC

Detailed Command/Status Description

GENERAL

The MPSC supports an extremely flexible set of serial and system interface modes.

The system interface to the CPU consists of 8 ports or buffers:

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch A Data Read	Ch A Data Write
0	1	0	Ch A Status Read	Ch A Command/Parameter
0	0	1	Ch B Data Read	Ch B Data Write
0	1	1	Ch B Status Read	Ch B Command/Parameter
1	X	X	High Impedance	High Impedance

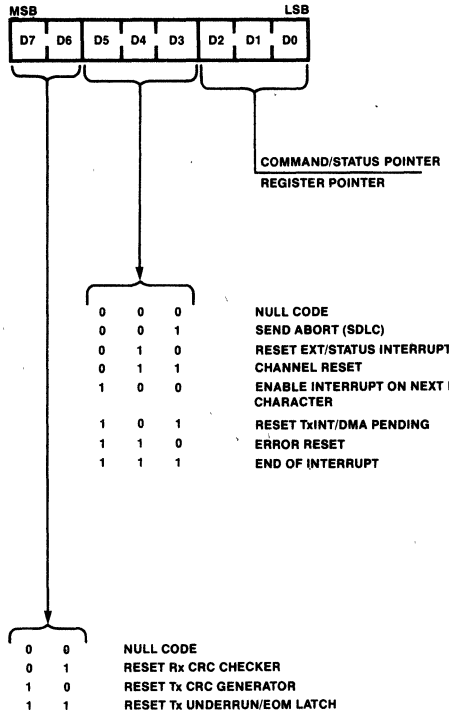
Data buffers are addressed by $A_1 = 0$, and Command ports are addressed by $A_1 = 1$.

COMMAND/STATUS DESCRIPTION

The following command and status bytes are used during initialization and execution phases of operation. All Command/Status operations on the two channels are identical, and independent, except where noted.

Detailed Register Description

Write Register 0 (WR0):



WR0

D2, D1, D0—Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5, D4, D3—Command bits determine which of the basic seven commands are to be performed.

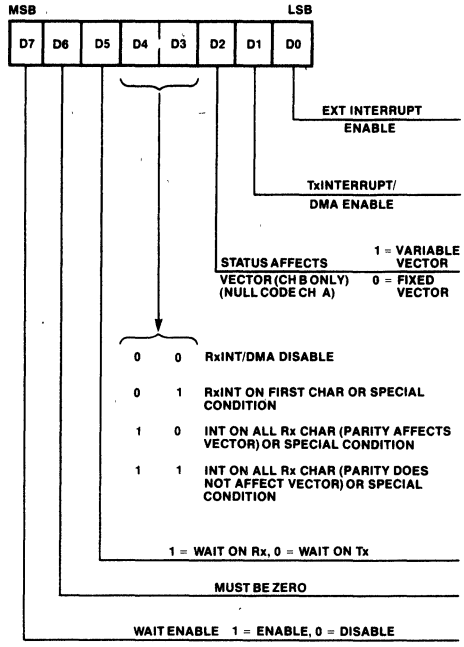
Command 0	Null—has no effect.
Command 1	Send Abort—causes the generation of eight to thirteen 1's when in the SDLC mode.
Command 2	Reset External/Status Interrupts—resets the latched status bits of RR0 and re-enables them, allowing interrupts to occur again.
Command 3	Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.
Command 4	Enable Interrupt on Next Receive Character—if the Interrupt on First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.
Command 5	Reset Transmitter Interrupt/DMA Pending—if The Transmit Interrupt/DMA Enable mode is selected, the MPSC automatically interrupts or requests DMA data transfer when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts or DMA requests until the next character has been completely sent.
Command 6	Error Reset—error latches, Parity and Overrun errors in RR1 are reset.
Command 7	End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

D7, D6	CRC Reset Code
00	Null—has no effect.
01	Reset Receive CRC Checker—resets the CRC checker to 0's. If in SDLC mode the CRC checker is initialized to all 1's.

10 Reset Transmit CRC Generator —resets the CRC generator to 0's. If in SDLC mode the CRC generator's initialized to all 1's.

11 Reset Tx Underrun/End of Message Latch.

Write Register 1 (WR1):



WR1

D0 External/Status Interrupt Enable —allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable —allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects vector—(WR1, D2 active in channel B only.) If this bit is not set, then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4, D3 Receive Interrupt Mode

0 0 Receive Interrupts/DMA Disabled

0 1 Receive Interrupt on First Character Only or Special Condition

1 0 Interrupt on All Receive Characters or Special Condition (Parity Error is a Special Receive Condition)

1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/$\overline{Transmit}$—when the following conditions are met the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}$ = 0, A0 = 0/1, and A1 = 0). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired OR connected since only one signal is active at any one time while the other is in the High Z state.

D6 Must be Zero

D7 Wait Enable—enables the wait function.

WR2 **Channel A**

D1, D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 Channel A and Channel B both use interrupts

0 1 Channel A uses DMA, Channel B uses interrupt

1 0 Channel A and Channel B both use DMA

1 1 Illegal Code

D2 Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

0 (Highest) RxA, TxA, RxB, TxB ExTA, ExTB (Lowest)

1 (Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest)

D5, D4, D3 Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table).

0 X X Non-vectored interrupts—intended for use with external DMA CONTROLLER. The Data Bus remains in a high impedence state during INTA sequences.

1 0 0 8085 Vector Mode 1—intended for use as the primary MPSC in a daisy chained priority structure. (See System Interface section)

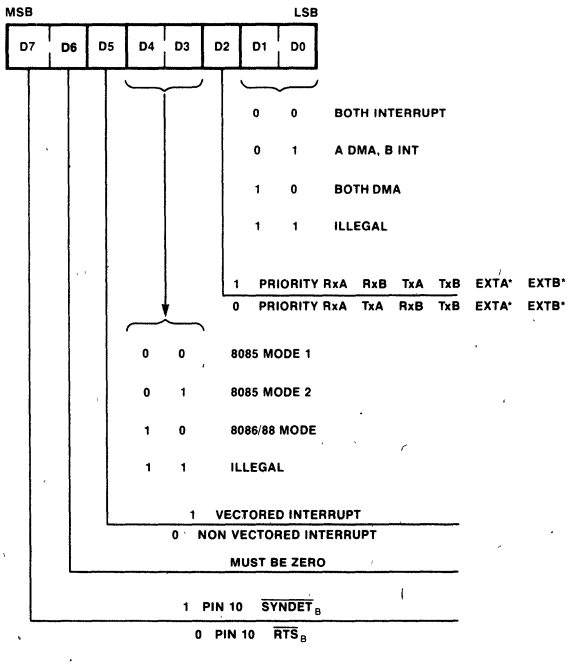
1 0 1 8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy chained priority structure. (See System Interface section)

1 1 0 8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy chained priority structure. (See System Interface section)

D6 Must be zero.

D7 zero Pin 10 = $\overline{RTS_B}$
one Pin 10 = $\overline{SYNDET_B}$

Write Register 2 (WR2): Channel A



The following table describes the MPSC's response to an interrupt acknowledge sequence:

D5	D4	D3	$\overline{IPI}$	MODE	INTA	Data Bus
0	X	X	X	Non-vectored	Any INTA	D7 D0 High Impedance
1	0	0	0	85 Mode 1	1st INTA 2nd INTA 3rd INTA	1 1 0 0 1 1 0 1 V7 V6 V5 V4* V3* V2* V1 V0 0 0 0 0 0 0 0 0
1	0	0	1	85 Mode 1	1st INTA 2nd INTA 3rd INTA	1 1 0 0 1 1 0 1 High Impedance High Impedance
1	1	0	0	86 Mode	1st INTA 2nd INTA	High Impedance V7 V6 V5 V4 V3 V2* V1*V0*
1	0	1	0	85 Mode 2	1st INTA 2nd INTA 3rd INTA	High Impedance V7 V6 V5 V4* V3* V2* V1 V0 0 0 0 0 0 0 0 0
1	0	1	1	85 Mode 2	1st INTA 2nd INTA 3rd INTA	High Impedance High Impedance High Impedance
1	1	0	1	86 Mode	1st INTA 2nd INTA	High Impedance High Impedance

These bits are variable if the "status affects vector" mode has been programmed, (WR1B, D2).

Interrupt/DMA Mode, Pin Functions, and Priority

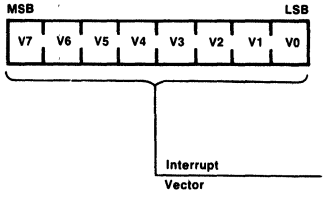
Ch. A WR2			Int/DMA Mode		Pin Functions				Priority
					RDY_A/ $RxDRQ_A$	RDY_B/ $TxDRQ_A$	$\overline{IPI}$/ $RxDRQ_B$	$\overline{IPO}$/ $TxDRQ_B$	
D_2	D_1	D_0	CH. A	CH. B	Pin 32	Pin 11	Pin 29	Pin 30	Highest Lowest
0	0	0	INT	INT	RDY_A	RDY_B	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA, RxB, TxB, EXT_A, EXT_B
1	0	0	INT	INT					RxA, RxB, TxA, TxB, EXT_A, EXT_B
0	0	1	DMA		$RxDRQ_A$	$TxDRQ_A$	$\overline{IPI}$	$\overline{IPO}$	RxA, TxA (DMA)
				INT					RxA[1], RxB, TxB, EXT_A, EXT_B (INT)
1	0	1	DMA						RxA, TxA (DMA)
				INT					RxA[1], RxB, TxB, EXT_A, EXT_B (INT)
0	1	0	DMA	DMA	$RxDRQ_A$	$TxDRQ_A$	$RxDRQ_B$	$TxDRQ_B$	RxA, TxA, RxB, TxB (DMA) RxA[1], RxB[1], EXT_A, EXT_B (INT)
1	1	0	DMA	DMA					RxA, RxB, TxA, TxB, (DMA) RxA[1], RxB[1], EXT_A, EXT_B (INT)

[1]Special Receive Condition

Interrupt Vector Mode Table

8085 Modes 8086/88 Mode	V_4 V_2	V_3 V_1	V_2 V_0	Channel	Condition
Note 1: Special Receive Condition = Parity Error, Rx Overrun Error, Framing Error, End of Frame (SDLC)	0 0 0 0	0 0 1 1	0 1 0 1	B	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)
	1 1 1 1	0 0 1 1	0 1 0 1	A	Tx Buffer Empty Ext/Status Change Rx Char. Available Special Rx Condition (Note 1)

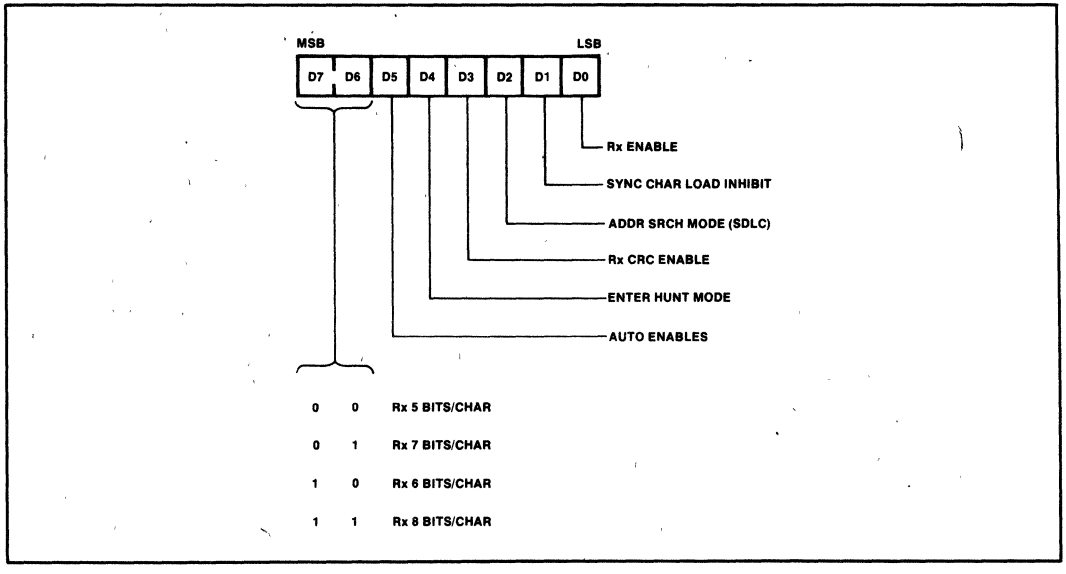
Write Register 2 (WR2): Channel B



WR2 CHANNEL B

D7–D0 Interrupt vector—This register contains the value of the interrupt vector placed on the data bus during interrupt acknowledge sequences.

Write Register 3 (WR3):



WR3

D0 Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.

D1 Sync Character Load Inhibit—A one prevents the receiver from loading sync characters into the receive buffers. In SDLC, this bit must be zero.

D2 Address Search Mode—If the SDLC mode has been selected, the MPSC will receive all frames unless this bit is a 1. If this bit is a 1, the MPSC will receive only frames with address bytes that match the global address (0FFH) or the value loaded into WR6. This bit must be zero in non-SDLC modes.

D3 Receive CRC Enable—A one in this bit enables (or re-enables) CRC calculation. CRC calculation starts with the last character placed in the Receiver FIFO. A zero in this bit disables, but does not reset, the Receiver CRC generator.

D4 Enter Hunt Phase--After initialization, the MPSC automatically enters the Hunt mode. If synchronization is lost, the Hunt phase can be re-entered by writing a one to this bit.

D5 Auto Enable—A one written to this bit causes $\overline{CD}$ to be automatic enable signal for the receiver and $\overline{CTS}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).

D7, D6 Receive Character length

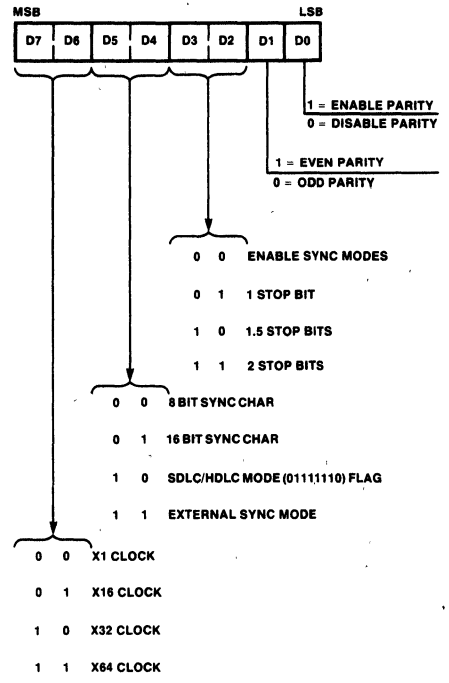
0 0 Receive 5 Data bits/character

0 1 Receive 7 Data bits/character

1 0 Receive 6 Data bits/character

1 1 Receive 8 Data bits/character

Write Register 4 (WR4):



WR4

D0 Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.

D1 Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and a zero causes it to send and expect odd parity.

D3, D2 Stop bits/sync mode

0 0 Selects synchronous modes.

0 1 Async mode, 1 stop bit/character

1 0 Async mode, 1-½ stop bits/character
1 1 Async mode, 2 stop bits/character

D5, D4 Sync mode select

0 0 8 bit sync character

0 1 16 bit sync character

1 0 SDLC mode (Flag sync)

1 1 External sync mode

D7, D6 Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. 1x mode must be selected for synchronous modes. If the 1x mode is selected, bit synchronization must be done externally.

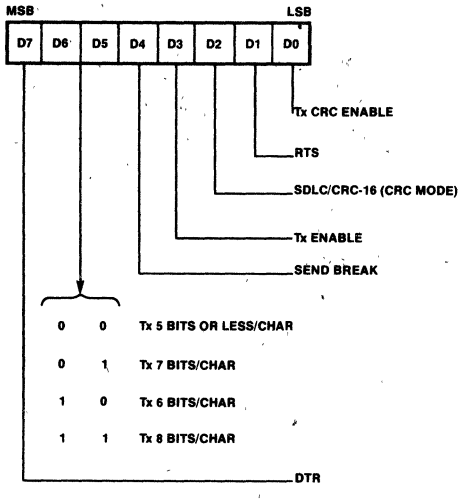
0 0 Clock rate = Data rate x 1

0 1 Clock rate = Data rate x 16

1 0 Clock rate = Data rate x 32

1 1 Clock rate = Data rate x 64

Write Register 5 (WR5):



WR5

D0 Transmit CRC Enable—a one in this bit enables the transmitter CRC generator. The CRC calculation is done when a character is moved from the transmit buffer into the shift register. A zero in this bit disables CRC calculations. If this bit is not set when a transmitter underrun occurs, the CRC will not be sent.

D1 Request to Send—a one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high).

D2 CRC Select—a one in this bit selects the CRC −16 polynomial ($X^{16} + X^{15} + X^2 + 1$) and a zero in this bit selects the CCITT-CRC polynomial ($X^{16} + X^{12} + X^5 + 1$). In SDLC mode, CCITT-CRC must be selected.

D3 Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.

D4 Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.

D6, D5 Transmit Character length

0 0 Transmit 1 - 5 bits/character

0 1 Transmit 7 bits/character

1 0 Transmit 6 bits/character

1 1 Transmit 8 bits/character

Bits to be sent must be right justified least significant bit first, eg:

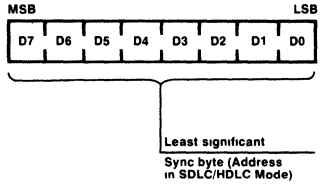
D7	D6	D5	D4	D3	D2	D1	D0
0	0	B5	B4	B3	B2	B1	B0

Five or less mode allows transmission of one to five bits per character. The microprocessor must format the data in the following way:

D7	D6	D5	D4	D3	D2	D1	D0	
1	1	1	1	0	0	0	B0	Sends one data bit
1	1	1	0	0	0	B1	B0	Sends two data bits
1	1	0	0	0	B2	B1	B0	Sends three data bits
1	0	0	0	B3	B2	B1	B0	Sends four data bits
0	0	0	B4	B3	B2	B1	B0	Sends five data bits

D7 Data Terminal Ready—when set, this bit forces the $\overline{DTR}$ pin active (low). When reset, this bit forces the $\overline{DTR}$ pin inactive (high).

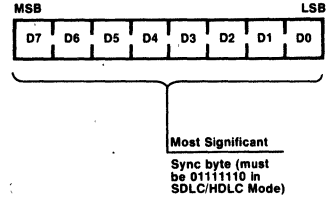
Write Register 6 (WR6):



WR6

D7–D0 Sync/Address—this register contains the transmit sync character in Monosync mode, the low order 8 sync bits in Bisync mode, or the Address byte in SDLC mode.

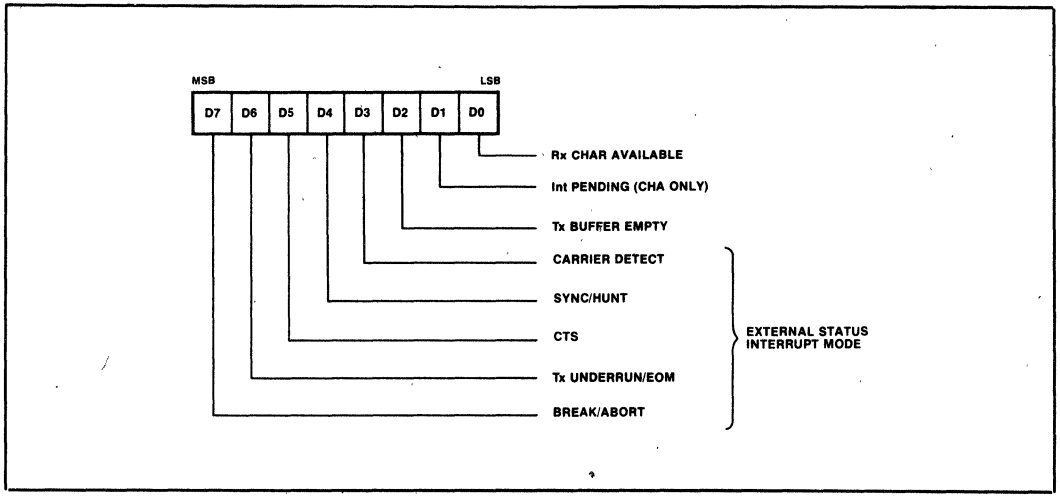
Write Register 7 (WR7):



WR7

D7–D0 Sync/Flag—this register contains the receive sync character in Monosync mode, the high order 8 sync bits in Bisync mode, or the Flag character (01111110) in SDLC mode. WR7 is not used in External Sync mode.

Read Register 0 (RR0):



RR0

D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending*—This Interrupt-Pending bit is reset when an EOI command is issued and there is no other interrupt request pending at that time.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the $\overline{CD}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CD}$ pin causes the CD bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{CD}$ pin immediately following a Reset External/Status Interrupt command.

*In vector mode this bit is set at the falling edge of the second $\overline{INTA}$ in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of $\overline{RD}$ input after pointer 2 is specified. This bit is always zero in Channel B.

D4 Sync/Hunt—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that Sync/Hunt shows the state of the $\overline{SYNDET}$ input. Any High-to-Low transition on the $\overline{SYNDET}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{SYNDET}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNDET}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{SYNDET}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{SYNDET}$ input holds the Sync/Hunt status in the reset condition.

When external synchronization is achieved, $\overline{SYNDET}$ must be driven Low on the second rising edge of $\overline{RxC}$ after the rising edge of $\overline{RxC}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{SYNDET}$ input. Once $\overline{SYNDET}$ is forced Low, it is good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. The High-to-Low transition of the $\overline{SYNDET}$ output sets the Sync/Hunt bit, which sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt Command.

When the $\overline{SYNDET}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit isset whenever character synchronization is lost or the end of message is detected. In this case, the MPSC again looks for a High-to-Low transition on the $\overline{SYNDET}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the MPSC is waiting for $\overline{SYNDET}$ to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the MPSC establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interupt command. This enables the MPSC to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status Interrupt, which must also be cleared by the Reset External/Status Interrupt Command. Note that the SYNDET pin acts as an output in this mode, and goes low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the MPSC. The External/Status interrupt is also generated, and should be handled as discussed previously .

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The MPSC automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

D5 Clear to Send—this bit contains the inverted state of the $\overline{CTS}$ pin at the time of the last change of any of the External/Status bits (CD, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{CTS}$ pin causes the CTS bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{CTS}$ pin immediately following a Reset External/Status Interrupt command.

D6 Transmitter Underrun/End of Message—this bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set, which causes the External/Status Interrupt which must be reset by issuing a Reset External/Status command (WR0; command 2).

D7 Break/Abort—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

SDLC Residue Code Table (I Field Bits in 2 Previous Bytes)

RR1 D3, D2, D1	8 bits/char		7 bits/char		6 bits/char		5 bits/char	
	Previous Byte	2nd Prev. Byte	Previous Byte	2nd Prev. Byte	Previous Byte	2nd Prev. Byte	Previous Byte	2nd Prev. Byte
1 0 0	0	3	0	2	0	1	0	5
0 1 0	0	4	0	3	0	2	0	1
1 1 0	0	5	0	4	0	3	0	2
0 0 1	0	6	0	5	0	4	0	3
1 0 1	0	7	0	6	0	5	—	
0 1 1	0	8	0	—	—	—	—	
1 1 1	1	8	—	—	—	—	—	
0 0 0	2	8	1	7	0	6	0	4

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

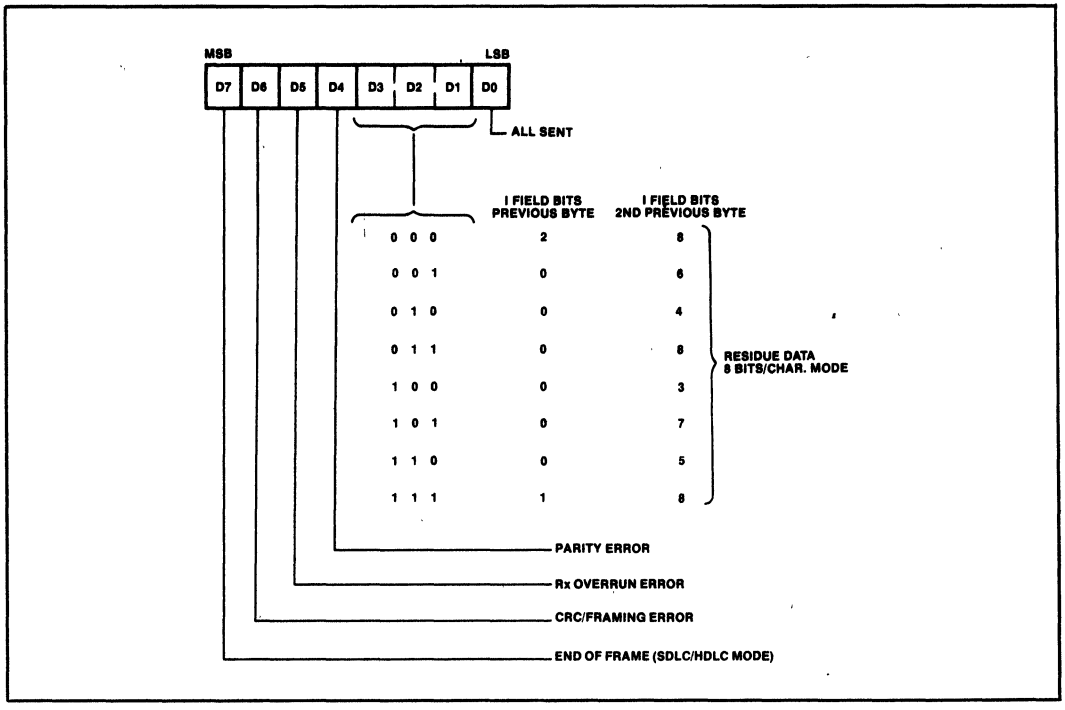
In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

D0 All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

D3, D2, D1 Residue Codes—bit synchronous protocols allow I-fields that are not an integral number of characters. Since transfers from the MPSC to the CPU are character oriented, the residue codes provide the capability of receiving leftover bits. Residue bits are right justified in the last two data bytes received.

D4 Parity Error—If parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

Read Register 1 (RR1): (Special Receive Condition Mode)

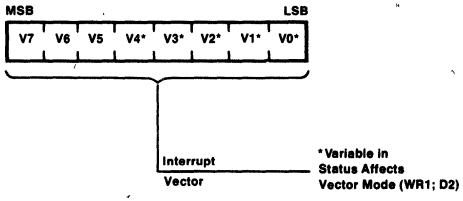


D5 Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the status affects vector mode, the overrun causes a special Receive Condition Vector.

D6 CRC/Framing Error—In async modes, a one in this bit indicates a receive framing error. In synchronous modes, a one in this bit indicates that the calculated CRC value does not match the last two bytes received. It can be reset by issuing an Error Reset command.

D7 End of Frame—this bit is valid only in SDLC mode. A one indicates that a valid ending flag has been received. This bit is reset either by an Error Reset command or upon reception of the first character of the next frame.

Read Register 2 (RR2):



RR2 Channel B

D7–D0 Interrupt vector—contains the interrupt vector programmed into WR2. If the status affects vector mode is selected (WR1; D2), it contains the modified vector (See WR2). RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

SYSTEM INTERFACE

General

The MPSC to Microprocessor System interface can be configured in many flexible ways. The basic interface types are polled, wait, interrupt driven, or direct memory access driven.

Polled operation is accomplished by repetitively reading the status of the MPSC, and making decisions based on that status. The MPSC can be polled at any time.

Wait operation allows slightly faster data throughput for the MPSC by manipulating the Ready input to the microprocessor. Block Read or Write Operations to the MPSC are started at will by the microprocessor and the MPSC deactivates its RDY signal if it is not yet ready to transmit the new byte, or if reception of new byte is not completed.

Interrupt driven operation is accomplished via an internal or external interrupt controller. When the MPSC requires service, it sends an interrupt request signal to the microprocessor, which responds with an interrupt acknowledge signal. When the internal or external interrupt controller receives the acknowledge, it vectors the microprocessor to a service routine, in which the transaction occurs.

DMA operation is accomplished via an external DMA controller. When the MPSC needs a data transfer, it request a DMA cycle from the DMA controller. The DMA controller then takes control of the bus and simultaneously does a read from the MPSC and a write to memory or vice-versa.

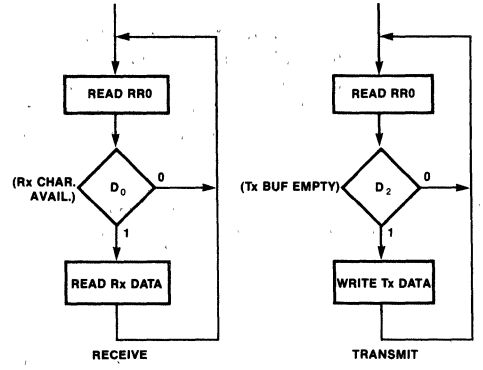
The following section describes the many configurations of these basic types of system interface techniques for both serial channels.

POLLED OPERATION:

In the polled mode, the CPU must monitor the desired conditions within the MPSC by reading the appropriate bits in the read registers. All data available, status, and error conditions are represented by the appropriate bits in read registers 0 and 1 for channels A and B.

There are two ways in which the software task of monitoring the status of the MPSC has been reduced. One is the "ORing" of all conditions into the Interrupt Pending bit. (RR0; D1 channel A only). This bit is set when the MPSC requires service, allowing the CPU to monitor one bit instead of four status registers. The other is available when the "status-affects-vector" mode is selected. By reading RR2 Channel B, the CPU can read a vector who's value will indicate that one or more of group of conditions has occurred, narrowing the field of possible conditions. See WR2 and RR2 in the Detailed Command Description section.

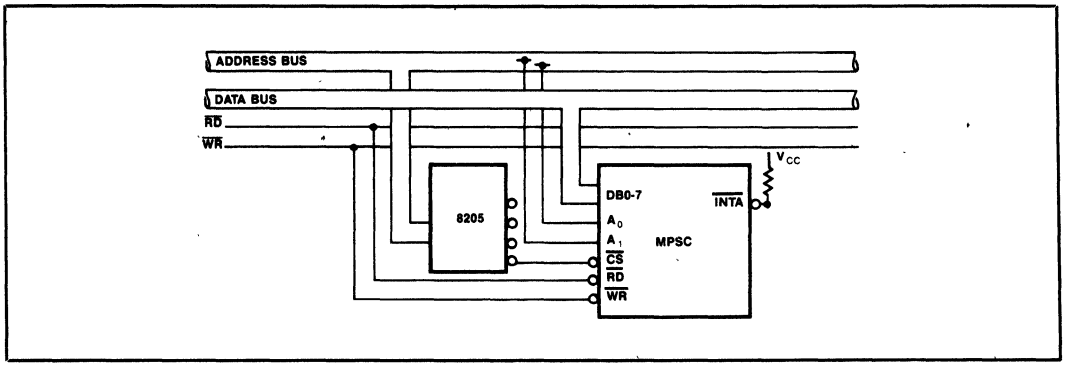
Software Flow, Polled Operation



RR0, D0 is reset automatically when the data is read

RR0, D2 is reset automatically when the data is written

Hardware Configuration, Polled Operation



WAIT OPERATION:
Wait Operation is intended to facilitate data transmission or reception using block move operations. If a block of data is to be transmitted, for example, the CPU can execute a String I/O instruction to the MPSC. After writing the first byte, the CPU will attempt to write a second byte immediately as is the case of block move. The MPSC forces the RDY signal low which inserts wait states in the CPU's write cycle until the transmit buffer is ready to accept a new byte. At that time, the RDY signal is high allowing the CPU to finish the write cycle. The CPU then attempts the third write and the process is repeated.

Similar operation can be programmed for the receiver. During initialization, wait on transmit (WR1; D5 = 0) or wait on receive (WR1; D5 = 1) can be selected.The wait operation can be enabled/disabled by setting/resetting the Wait Enable Bit (WR1; D7).

CAUTION: ANY CONDITION THAT CAN CAUSE THE TRANSMITTER TO STOP (EG, CTS GOES INACTIVE) OR THE RECEIVER TO STOP (EG, RX DATA STOPS) WILL CAUSE THE MPSC TO HANG THE CPU UP IN WAIT STATES UNTIL RESET. EXTREME CARE SHOULD BE TAKEN WHEN USING THIS FEATURE.

INTERRUPT DRIVEN OPERATION:
The MPSC can be programmed into several interrupt modes: Non-Vectored, 8085 vectored, and 8088/86 vectored. In both vectored modes, multiple MPSC's can be daisy-chained.

In the vectored mode, the MPSC responds to an interrupt acknowledge sequence by placing a call instruction (8085 mode) and interrupt vector (8085 and 8088/86 mode) on the data bus.

The MPSC can be programmed to cause an interrupt due to up to 14 conditions in each channel. The status of these interrupt conditions is contained in Read Registers 0 and 1. These 14 conditions are all directed to cause 3 different types of internal interrupt request for each channel: receive/interrupts, transmit interrupts and external/status interrupts (if enabled).

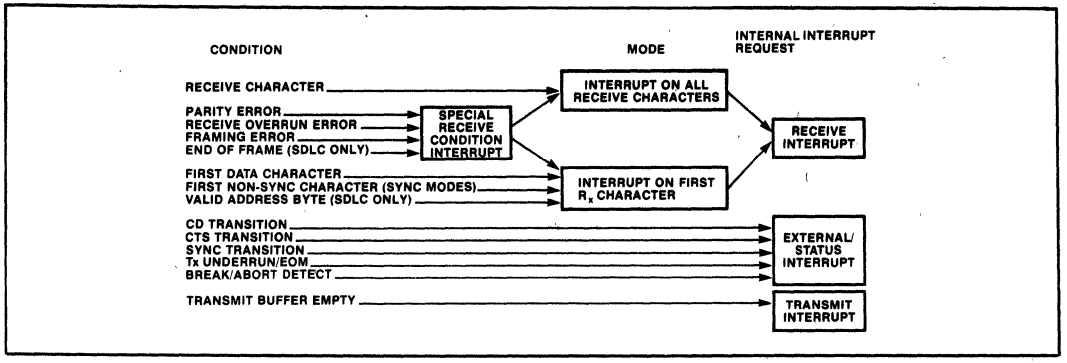
This results in up to 6 internal interrupt request signals. The priority of those signals can be programmed to one of two fixed modes:

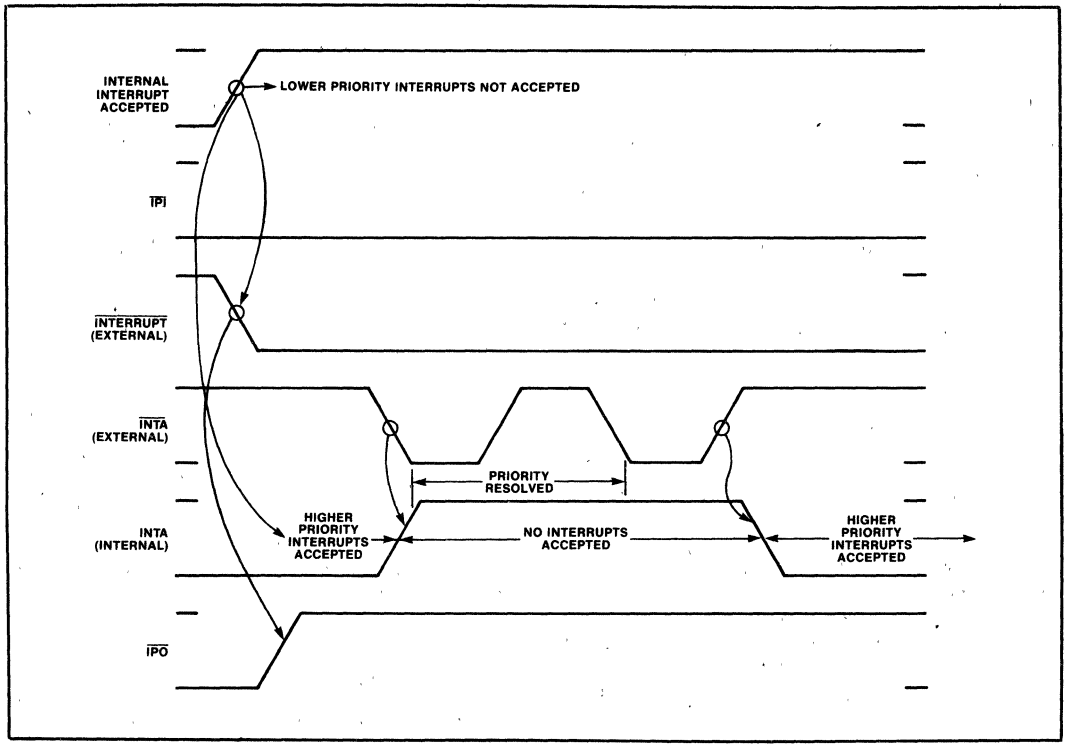
Highest Priority					Lowest Priority
RxA	RxB	TxA	TxB	ExTA	ExTB
RxA	TxA	RxB	TxB	ExTA	ExTB

The interrupt priority resolution works differently for vectored and non-vectored modes.

PRIORITY RESOLUTION: VECTORED MODE
Any interrupt condition can be accepted internally to the MPSC at any time, unless the MPSC's internal INTA signal is active, unless a higher priority interrupt is currently accepted, or if $\overline{IPI}$ is inactive (high). The MPSC's internal INTA is set on the leading (falling) edge of the first External $\overline{INTA}$ pulse and reset on the trailing (rising) edge of the second External $\overline{INTA}$ pulse. After an interrupt is accepted internally, an External $\overline{INT}$ request is generated and the $\overline{IPO}$ goes inactive. $\overline{IPO}$ and $\overline{IPI}$ are used for daisy-chaining MPSC's together.

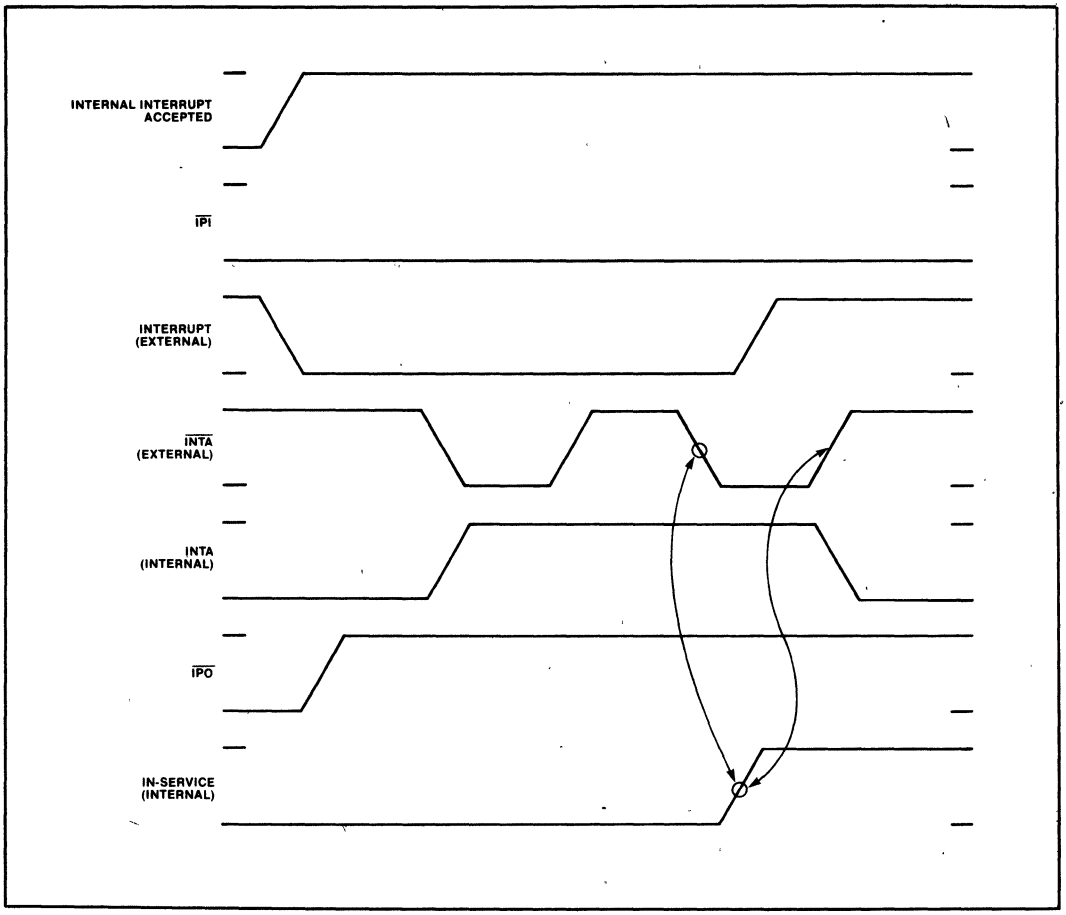
Interrupt Condition Grouping





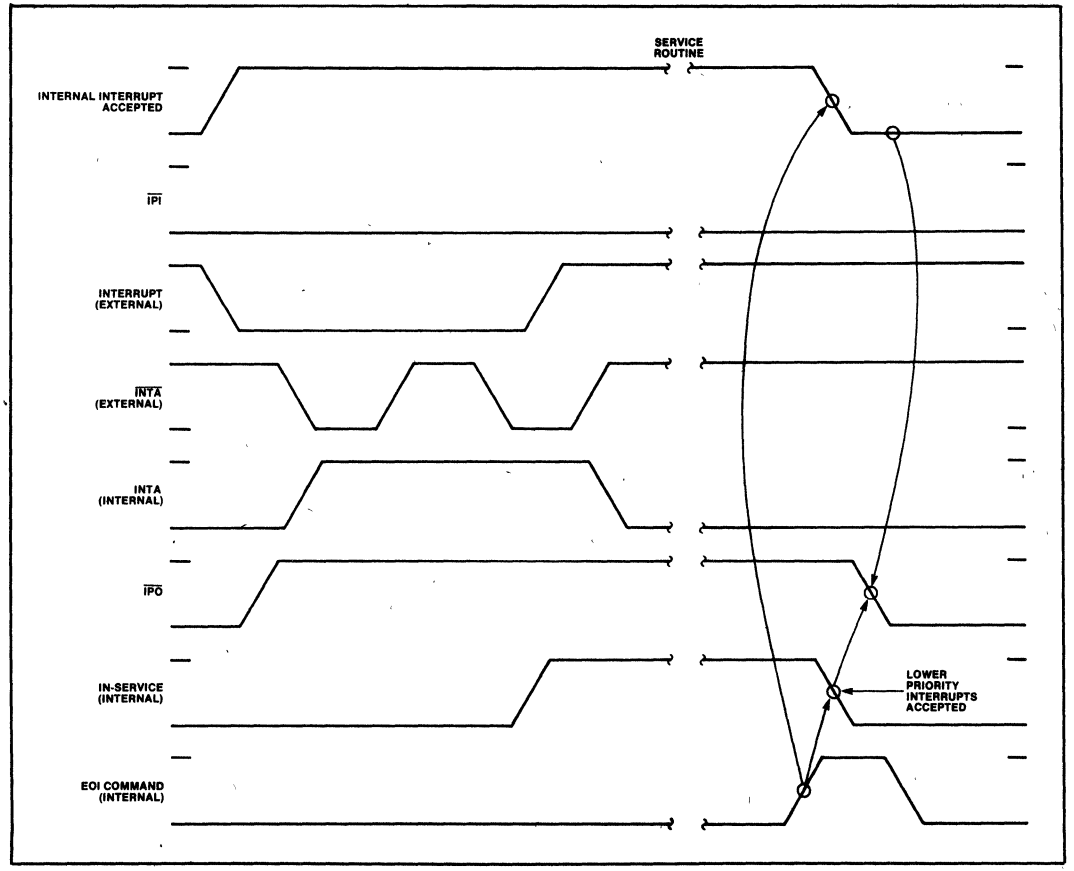
The MPSC's internal INTA is set on the leading (falling) edge of the first external $\overline{INTA}$ pulse, and reset on the trailing (rising) edge of the second external $\overline{INTA}$ pulse. After an interrupt is accepted internally, an external $\overline{INT}$ request is generated and $\overline{IPO}$ goes inactive (high). $\overline{IPO}$ and $\overline{IPI}$ are used for daisy-chaining MPSC's together.

In-Service Timing



Each of the six interrupt sources has an associated In-Service latch. After priority has been resolved, the highest priority In-Service latch is set. After the In-Service latch is set, the $\overline{INT}$ pin goes inactive (high).

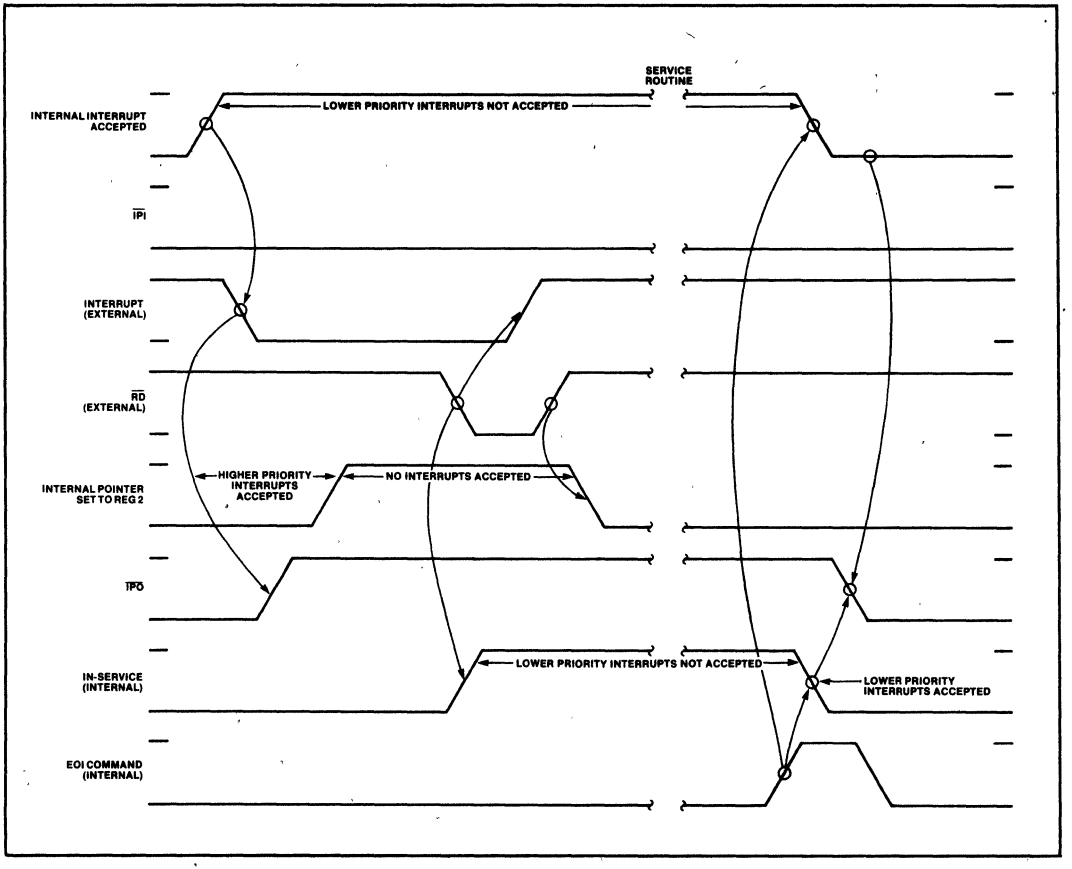
EOI Command Timing



Lower priority interrupts are not accepted internally while the In-Service latch is set. However, higher priority interrupts are accepted internally and a new external $\overline{INT}$ request is generated. If the CPU responds with a new INTA sequence, the MPSC will respond as before, suspending the lower priority interrupt.

After the interrupt is serviced, the End-of-Interrupt (EOI) command should be written to the MPSC. This command will cause an internal pulse that is used to reset the In-Service Latch which allows service for lower priority interrupts in the daisy-chain to resume, provided a new INTA sequence does not start for a higher priority interrupt (higher than the highest under service). If there is no interrupt pending internally, the $\overline{IPO}$ follows $\overline{IPI}$.

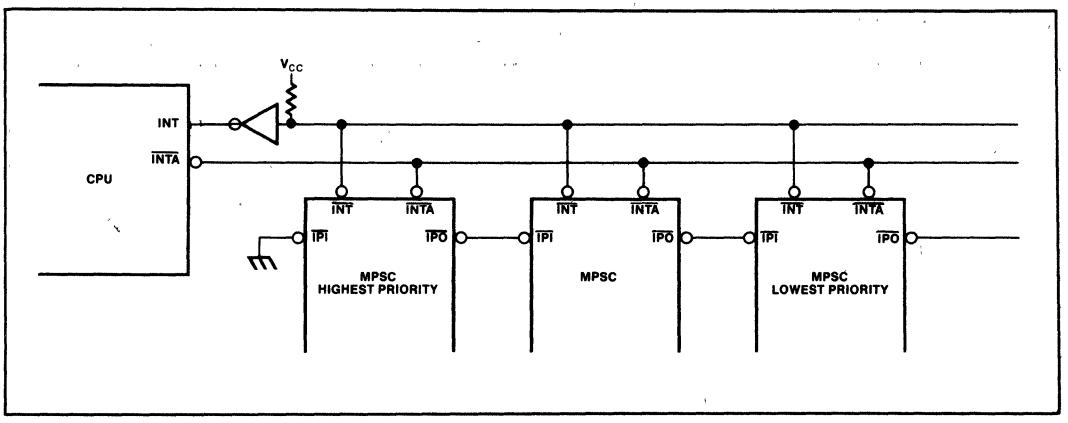
Non-Vectored Interrupt Timing



PRIORITY RESOLUTION: NON-VECTORED MODE

In non-vectored mode, the MPSC does not respond to interrupt acknowledge sequences. The $\overline{INTA}$ input (pin 27) must be pulled high for proper operation. The MPSC should be programmed to the Status-Affects-Vector mode, and the CPU should read RR2 (Ch. B) in its service routine to determine which interrupt requires service.

In this case, the internal pointer being set to RR2 provides the same function as the internal INTA signal in the vectored mode. It inhibits acceptance of any additional internal interrupts and its leading edge starts the interrupt priority resolution circuit. The interrupt priority resolution is ended by the leading edge of the read signal used by the CPU to retrieve the modified vector. The leading edge of read sets the In-Service latch and forces the external $\overline{INT}$ output inactive (high). The internal pointer is reset to zero after the trailing edge of the read pulse.



Note that if RR2 is specified but not read, no internal interrupts, regardless of priority, are accepted.

DAISY CHAINING MPSC:

In the vectored interrupt mode, multiple MPSC's can be daisy-chained on the same $\overline{INT}$, $\overline{INTA}$ signals. These signals, in conjunction with the $\overline{IPI}$ and $\overline{IPO}$ allow a daisy-chain-like interrupt resolution scheme. This scheme can be configured for either 8085 or 8086/88 based system.

In either mode, the same hardware configuration is called for. The $\overline{INT}$ request lines are wire-OR'ed together at the input of a TTL inverter which drives the INT pin of the CPU. The $\overline{INTA}$ signal from the CPU drives all of the daisy-chained MPSC's.

The MPSC drives $\overline{IPO}$ (Interrupt Priority Output) inactive (high) if $\overline{IPI}$ (Interrupt Priority Input) is inactive (high), or if the MPSC has an interrupt pending.

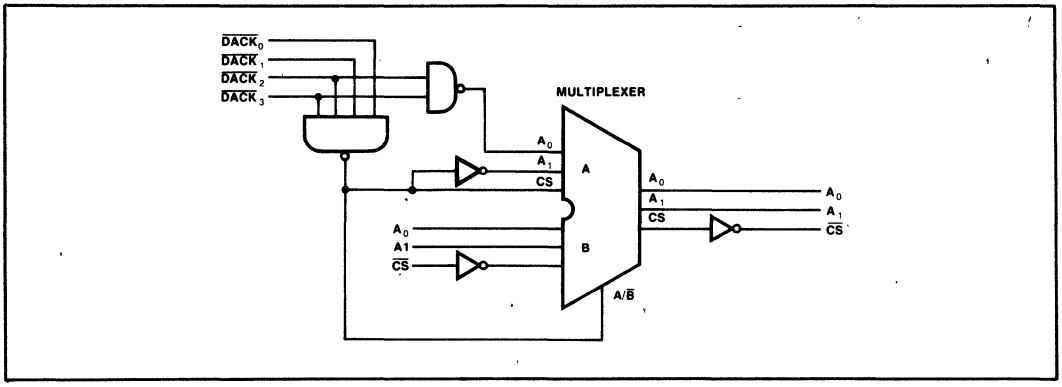
The $\overline{IPO}$ of the highest priority MPSC is connected to the $\overline{IPI}$ of the next highest priority MPSC, and so on.

If $\overline{IPI}$ is active (low), the MPSC knows that all higher priority MPSC's have no interrupts pending. The $\overline{IPI}$ pin of the highest priority MPSC is strapped active (low) to ensure that it always has priority over the rest.

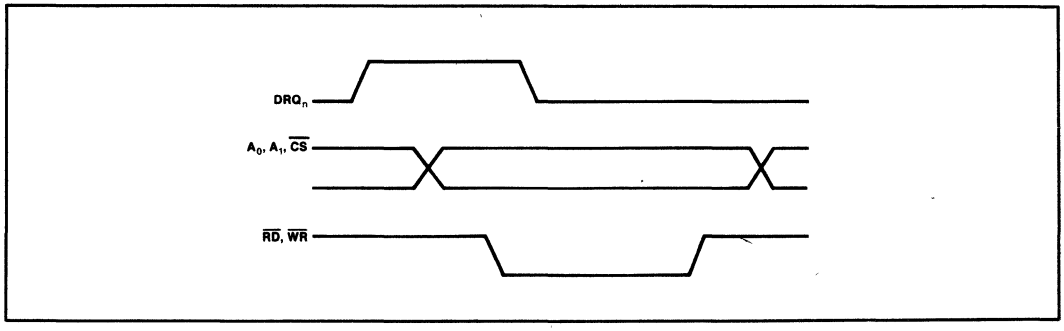
MPSC's Daisy-chained on an 8088/86 CPU should be programmed to the 8088/86 Interrupt mode (WR2; D4, D3 (Ch. A). MPSC's Daisy-chained on an 8085 CPU should be programmed to 8085 interrupt mode 1 if it is the highest priority MPSC. In this mode, the highest priority MPSC issues the CALL instruction during the first INTA cycle, and the interrupting MPSC provides the interrupt vector during the following INTA cycles. Lower priority MPSC's should be programmed to 8085 interrupt mode 2.

MPSC's used alone in 8085 systems should be programmed to 8085 mode 1 interrupt operation.

DMA Acknowledge Circuit



DMA Timing



DMA OPERATION

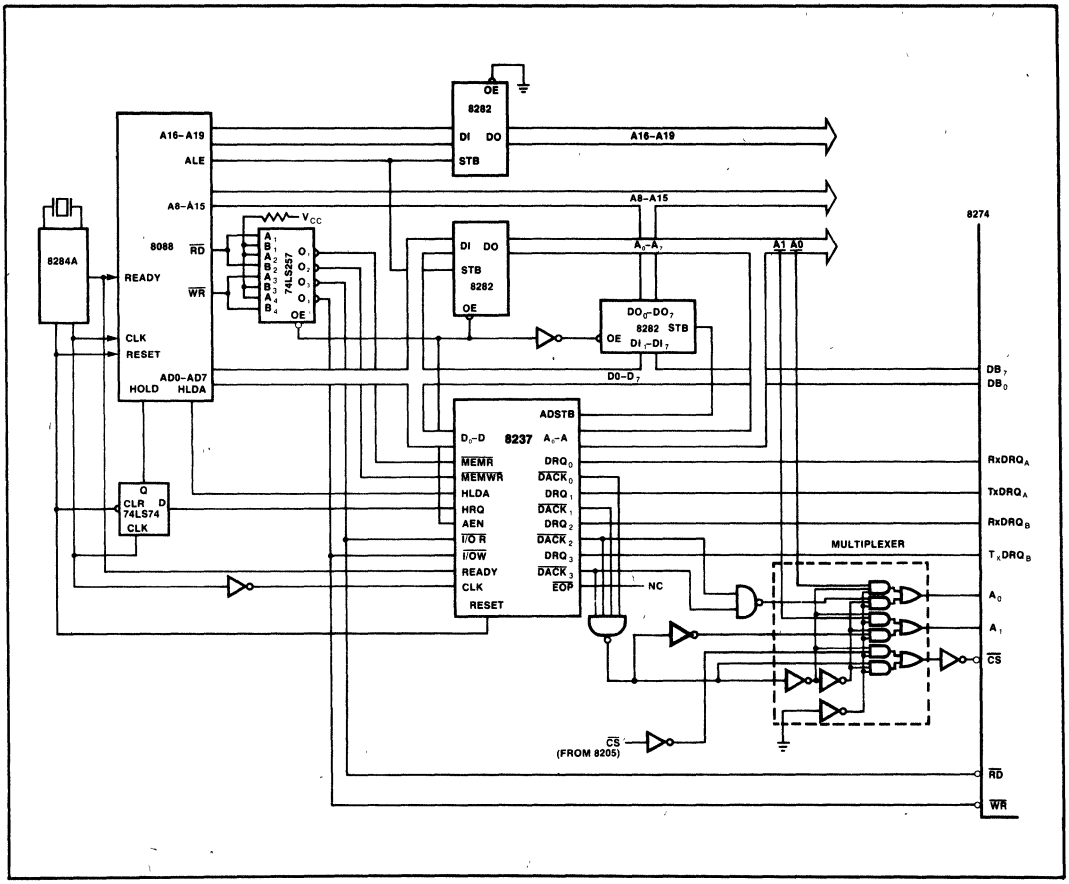
Each MPSC can be programmed to utilize up to four DMA channels: Transmit Channel A, Receive Channel A, Transmit Channel B, Receive Channel B. Each DMA Channel has an associated DMA Request line. Acknowledgement of a DMA cycle is done via normal data read or write cycles. This is accomplished by encoding the $\overline{DACK}$ signals to generate A_0, A_1, and $\overline{CS}$ signals, and multiplexing them with the normal A_0, A_1, and $\overline{CS}$ signals.

PERMUTATIONS

Channels A and B can be used with different system interface modes. In all cases it is impossible to poll the MPSC. The following table shows the possible permutations of interrupt, wait, and DMA modes for channels A and B. Bits D_1, D_0 of WR2 Ch. A determine these permutations.

Permutation WR2 Ch. A D_1 D_0	Channel A	Channel B
0 0	Wait Interrupt Polled	Wait Interrupt Polled
0 1	DMA Polled	Interrupt Polled
1 0	DMA Polled	DMA Polled

D1, D0 = 1, 1 is illegal.


8282
A16–A19
ALE
A8–A15
8088
RD
READY
WR
CLK
RESET
AD0–AD7
HOLD
HLDA
8284A
VCC
74LS257
8282
STB
OE
DO0–DO7
DI0–DI7
D0–D7
A0–A7
A1 A0
8237
ADSTB
D0–D
A0–A
MEMR
MEMWR
HLDA
HRQ
AEN
I/O R
I/O W
READY
CLK
RESET
DRQ0
DACK0
DRQ1
DACK1
DRQ2
DACK2
DRQ3
DACK3
EOP
NC
Q
CLR D
74LS74
CLK
CS
(FROM 8205)
MULTIPLEXER
8274
DB7
DB0
RxDRQA
TxDRQA
RxDRQB
TxDRQB
A0
A1
CS
RD
WR

PROGRAMMING HINTS

This section will describe some useful programming hints which may be useful in program development.

Asynchronous Operation

At the end of transmission, the CPU must issue "Reset Transmit Interrupt/DMA Pending" command in WR0 to reset the last transmit empty request which was not satisfied. Failing to do so will result in the MPSC locking up in a transmit empty state forever.

Non-Vectored Mode

In non-vectored mode, the Interrupt Acknowledge pin (INTA) on the MPSC must be tied high through a pull-up resistor. Failing to do so will result in unpredictable response from the 8274.

HDLC/SDLC Mode

When receiving data in SDLC mode, the CRC bytes must be read by the CPU (or DMA controller) just like any other data field. Failing to do so will result in receiver buffer overflow. Also, the End of Frame Interrupt indicates that the entire frame has been received. At this point, the CRC result (RR1:D6) and residue code (RR1;D3, D2, D1) may be checked.

Status Register RR2

RR2 contains the vector which gets modified to indicate the source of interrupt (See the section titled MPSC Modes of Operation). However, the state of the vector does not change if no new interrupts are generated. The contents of RR2 are only changed when a new interrupt is generated. In order to get the correct information, RR2 must be read only after an interrupt is generated, otherwise it will indicate the previous state.

Initialization Sequence

The MPSC initialization routine must issue a channel Reset Command at the beginning. WR4 should be defined before other registers. At the end of the initialization sequence, Reset External/Status and Error Reset commands should be issued to clear any spurious interrupts which may have been caused at power up.

Transmit Under-run/EOM Latch

In SDLC/HDLC, bisync and monosync mode, the transmit under-run/EOM must be reset to enable the CRC check bytes to be appended to the transmit frame or transmit message. The transmit under-run/EOM latch can be reset only after the first character is loaded into the transmit buffer. When the transmitter under-runs at the end of the frame, CRC check bytes are appended to the frame/message. The transmit under-run/EOM latch can be reset at any time during the transmission after the first character. However, it should be reset *before* the transmitter under-runs otherwise, both bytes of the CRC may not be appended to the frame/message. In the receive mode in bisync operation, the CPU must read the CRC bytes and two more SYNC characters before checking for valid CRC result in RR1.

Sync Character Load Inhibit

In bisync/monosync mode only, it is possible to prevent loading sync characters into the receive buffers by setting the sync character load inhibit bit (WR3;D1=1). Caution must be exercised in using this option. It may be possible to get a CRC character in the received message which may match the sync character and not get transferred to the receive buffer. However, sync character load inhibit should be enabled during all pre-frame sync characters so the software routine does not have to read them from the MPSC.

In SDLC/HDLC mode, sync character load inhibit bit must be reset to zero for proper operation.

EOI Command

EOI command can only be issued through channel A irrespective of which channel had generated the interrupt.

Priority in DMA Mode

There is no priority in DMA mode between the following four signals: TxDRQ(CHA), RxDRQ(CHA), TxDRQ(CHB), RxDRQ(CHB). The priority between these four signals must be resolved by the DMA controller. At any given time, all four DMA channels from the 8274 are capable of going active.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature
Under Bias ..... 0°C to +70°C
Storage Temperature
(Ceramic Package) ..... −65°C to +150°C
(Plastic Package) ..... −40°C to +125°C
Voltage On Any Pin With
Respect to Ground ..... −0.5V to +7.0V

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ±10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	+2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		+0.45	V	I_{OL} = 2.0mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = −200μA
I_{IL}	Input Leakage Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OL}	Output Leakage Current		±10	μA	V_{OUT} = V_{CC} to 0V
I_{CC}	V_{CC} Supply Current		180	mA	

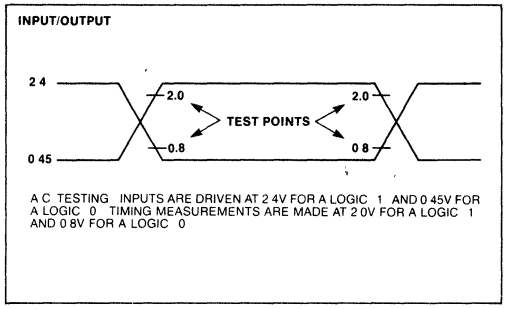
CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz;
C_{OUT}	Output Capacitance		15	pF	Unmeasured
$C_{I/O}$	Input/Output Capacitance		20	pF	pins returned to GND

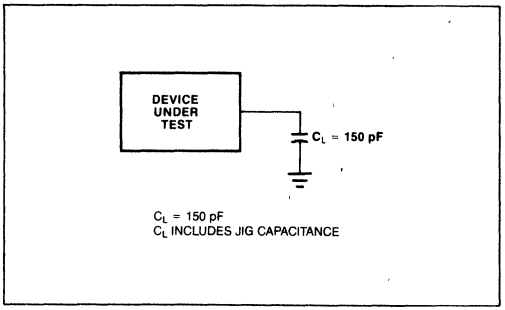
A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = +5V ±10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{CY}	CLK Period	250	4000	ns	
t_{CL}	CLK Low Time	105	2000	ns	
t_{CH}	CLK High Time	105	2000	ns	
t_r	CLK Rise Time	0	30	ns	
t_f	CLK Fall Time	0	30	ns	
t_{AR}	A0, A1 Setup to $\overline{RD}\downarrow$	0		ns	
t_{AD}	A0, A1 to Data Output Dlay		200	ns	C_L=150 pf
t_{RA}	A0, A1 Hold After $\overline{RD}\uparrow$	0		ns	
t_{RD}	$\overline{RD}\downarrow$ to Data Output Delay		200	ns	C_L=150 pf
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{DF}	Output Float Delay		120	ns	
t_{AW}	$\overline{CS}$, A0, A1 Setup to $\overline{WR}\downarrow$	0		ns	
t_{WA}	$\overline{CS}$, A0, A1 Hold after $\overline{WR}\uparrow$	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	250		ns	
t_{DW}	Data Setup to $\overline{WR}\uparrow$		150	ns	
t_{WD}	Data Hold After $\overline{WR}\uparrow$	0		ns	
t_{PI}	$\overline{IPI}$ Setup to $\overline{INTA}\downarrow$	0		ns	
t_{IP}	$\overline{IPI}$ Hold after $\overline{INTA}\uparrow$	10		ns	
t_{II}	$\overline{INTA}$ Pulse Width	250		ns	
t_{PIPO}	$\overline{IPI}\downarrow$ to $\overline{IPO}$ Delay		100	ns	
t_{ID}	$\overline{INTA}\downarrow$ to Data Output Deay		200	ns	
t_{CQ}	$\overline{RD}$ or $\overline{WR}$ to $\overline{DRQ}\downarrow$		150	ns	
t_{RV}	Recovery Time Between Controls	300		ns	
t_{CW}	$\overline{CS}$, A0, A1 to RDY_A or RDY_B Delay		140	ns	
t_{DCY}	Data Clock Cycle	4.5		tcy	
t_{DCL}	Data Clock Low Time	180		ns	
t_{DCH}	Data Clock High Time	180		ns	
t_{TD}	$\overline{TxC}$ to TxD Delay		300	ns	
t_{DS}	RxD Setup to $\overline{RxC}\uparrow$	0		ns	
t_{DH}	RxD Hold after $\overline{RxC}\uparrow$	140		ns	
t_{ITD}	$\overline{TxC}$ to $\overline{INT}$ Delay	4	6	tcy	
t_{IRD}	RxC to $\overline{INT}$ Delay	7	10	tcy	
t_{PL}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ Low Time	200		ns	
t_{PH}	$\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$ High Time	200		ns	
t_{IPD}	External $\overline{INT}$ from $\overline{CTS}$, $\overline{CD}$, $\overline{SYNDET}$		500	ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM

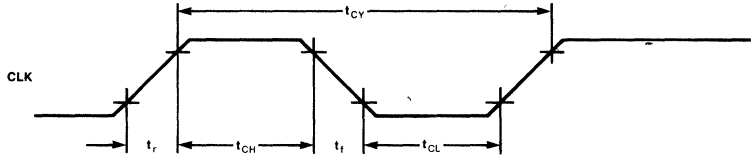


A.C. TESTING LOAD CIRCUIT

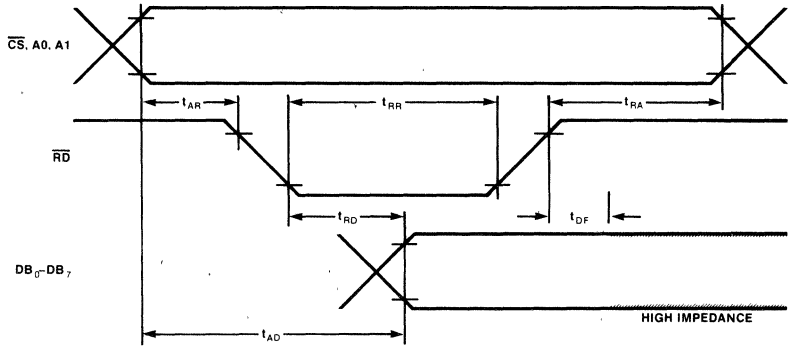


WAVEFORMS

CLOCK CYCLE

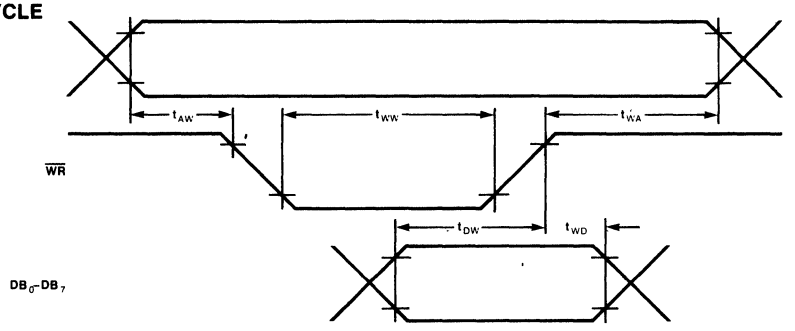


READ CYCLE

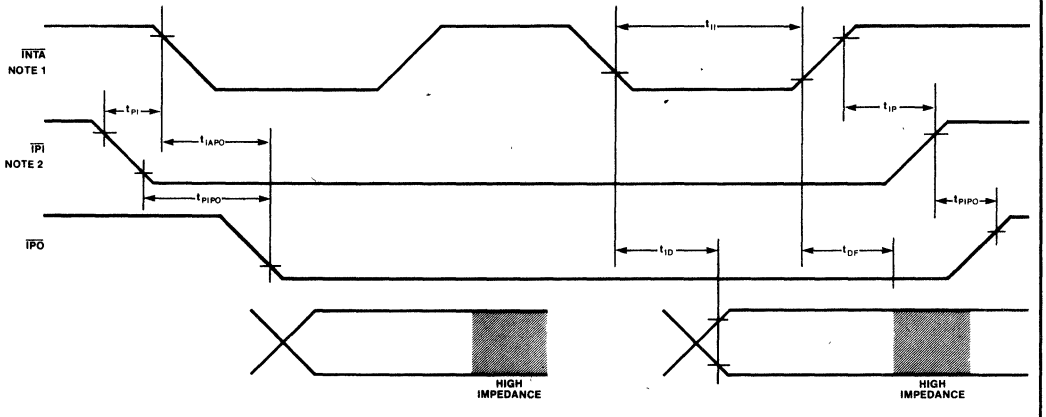


WAVEFORMS (Continued)

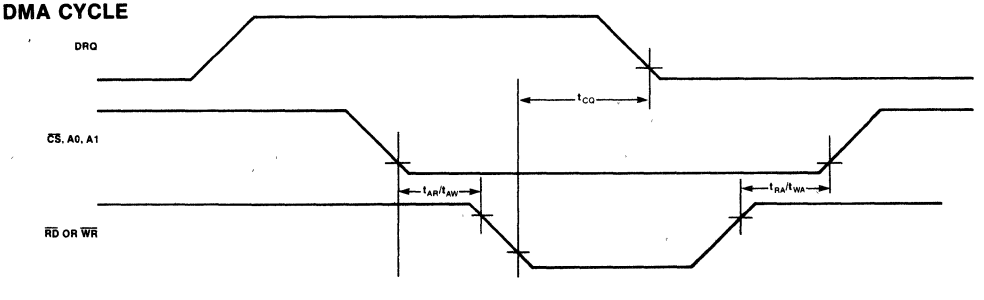
WRITE CYCLE



INTA CYCLE



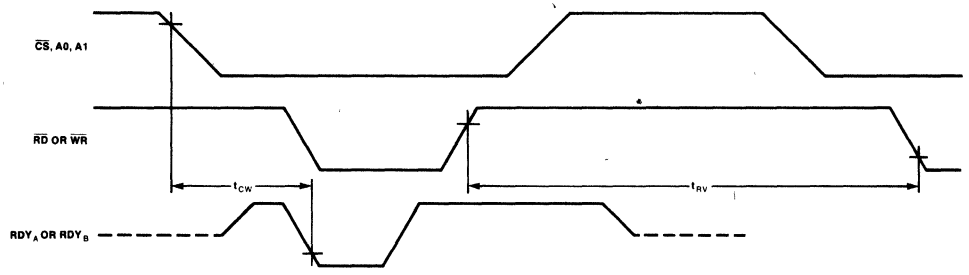
DMA CYCLE



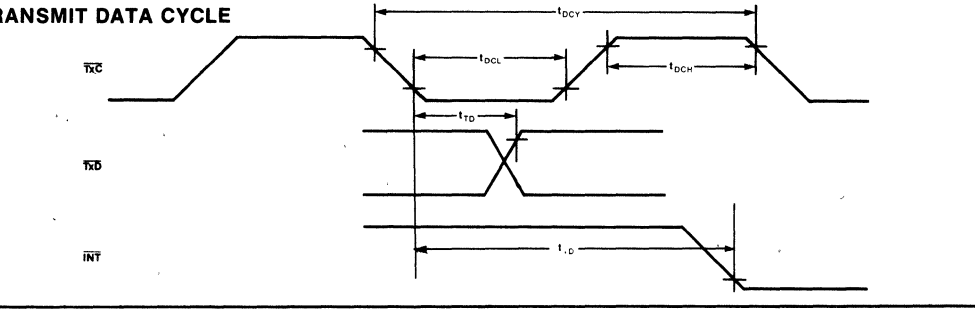
NOTES:
1. INTA signal acts as RD signal.
2. IPI signal acts as CS signal.

WAVEFORMS (Continued)

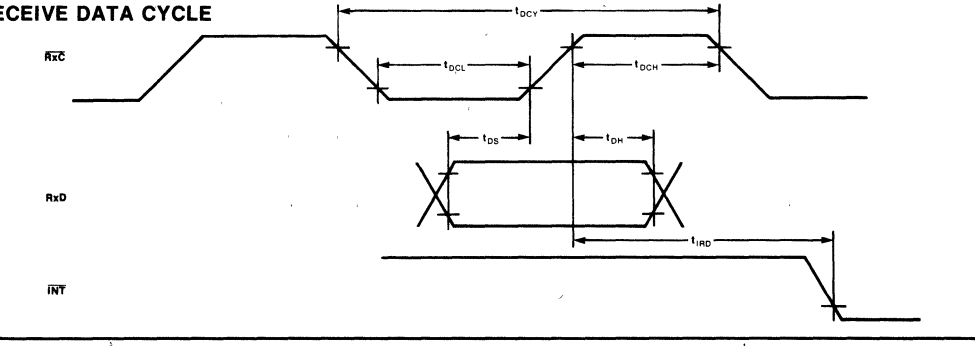
READ/WRITE CYCLE (SOFTWARE POLLED MODE)



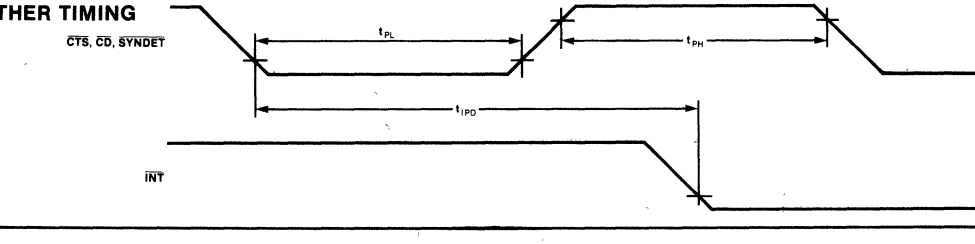
TRANSMIT DATA CYCLE



RECEIVE DATA CYCLE



OTHER TIMING



82530/82530-6
SERIAL COMMUNICATIONS CONTROLLER (SCC)

- Two independent full duplex serial channels
- On chip crystal oscillator, Baud-Rate Generator and Digital Phase Locked Loop for each channel
- Programmable for NRZ, NRZI or FM data encoding/decoding
- Diagnostic local loopback and automatic echo for fault detection and isolation
- System Clock Rates:
  —4 Mhz for 82530
  —6 Mhz for 82530-6
- Max Bit Rate (4 Mhz)
  — Externally clocked: 1Mbps
  — Self clocked:
    250 Kbps FM coding
    125 Kbps NRZI coding
- Interfaces easily with any INTEL CPU, DMA or I/O processor
- Asynchronous Modes
  — 5-8 bit character; odd, even or no parity; 1, 1.5 or 2 stop bits
  — Independent transmit and receive clocks. 1X, 16X, 32X or 64X programmable sampling rate
  — Error Detection: Framing, Overrun and Parity
  — Break detection and generation
- Bit synchronous Modes
  — SDLC/HDLC flag generation and recognition
  — Automatic zero bit insertion and deletion
  — Automatic CRC generation and detection (CRC 16 or CCITT)
  — Abort generation and detection
  — I-field residue handling
  — SDLC loop mode operation
  — CCITT X.25 compatible
- Byte synchronous Modes
  — Internal or external character synchronization (1 or 2 characters)
  — Automatic CRC generation and checking (CRC 16 or CCITT)
  — IBM Bisync compatible

NOVEMBER 1983
ORDER NUMBER: 230834-001

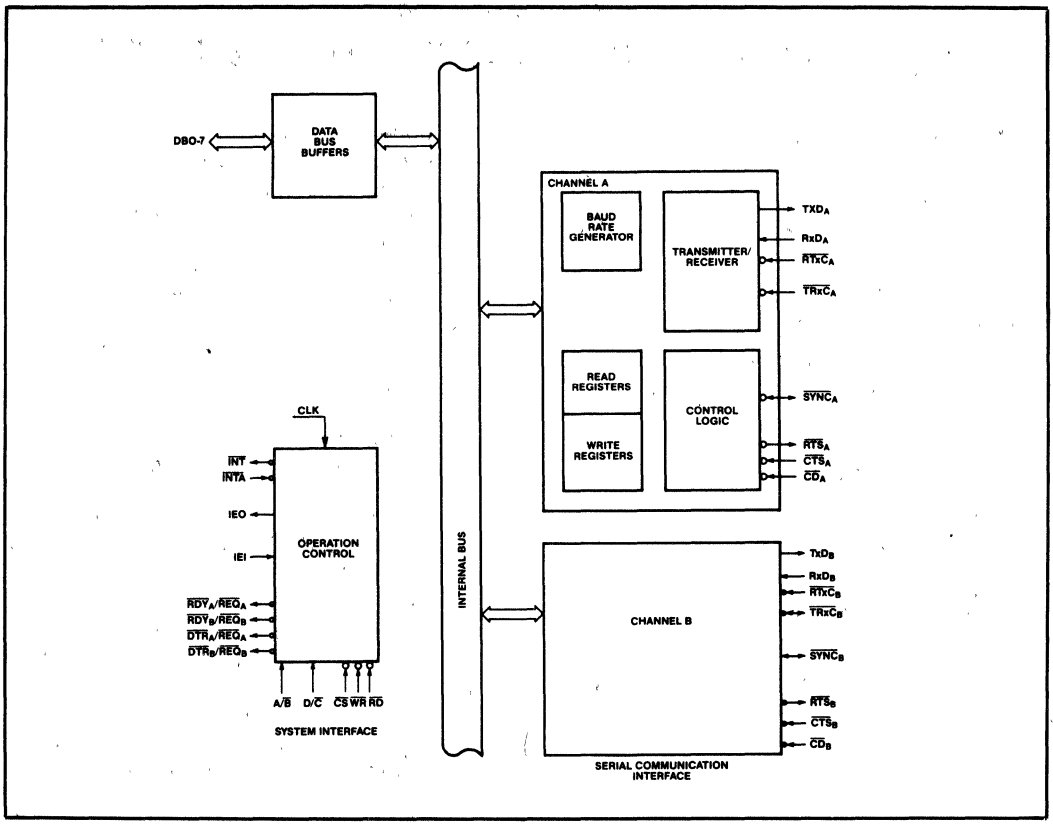


Figure 1. 82530 Internal Block Diagram

The INTEL 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. It is designed to interface high speed communications lines using Asynchronous. Byte synchronous and Bit synchronous protocols to INTEL's microprocessors based systems. It can be interfaced with Intel's MCS51, iAPX86/88/186 and 188 in polled, interrupt driven or DMA driven modes of operation.

The SCC is a 40 pin device manufactured using INTEL's high-performance HMOS II technology.

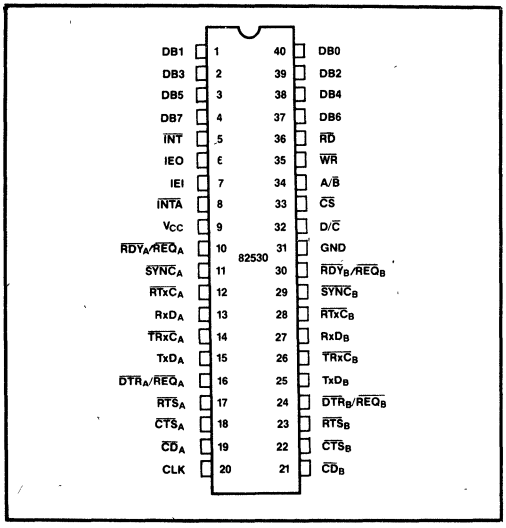


Figure 2. Pin configuration

The following section describes the pin functions of the SCC. Figure 2 details the pin assignments

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	40 1 39 2 38 3 37 4	I/O I/O I/O I/O I/O I/O I/O I/O	**Data Bus:** The Data Bus lines are bi-directional three-state lines which interface with the system's Data Bus. These lines carry data and commands to and from the SCC.
$\overline{INT}$	5	0	**Interrupt Request:** The interrupt signal is activated when the SCC requests an interrupt. It is an open drain output.
IEO	6	0	**Interrupt Enable Out:** IEO is High only if IEI is High and the CPU is not servicing an SCC interrupt or the SCC is not requesting an interrupt (Interrupt Acknowledge cycle only). IEO is connected to the next lower priority device's IEI input and thus inhibits interrupts from lower priority devices.
IEI	7	I	**Interrupt Enable In:** IEI is used with IEO to form an interrupt daisy chain when there is more than one interrupt-driven device. A High IEI indicates that no other higher priority device has an interrupt under service or is requesting an interrupt.
$\overline{INTA}$	8	I	**Interrupt Acknowledge:** This signal indicates an active Interrupt Acknowledge cycle. During this cycle, the SCC interrupt daisy chain settles. When $\overline{RD}$ becomes active, the SCC places an interrupt vector on the data bus (if IEI is High). $\overline{INTA}$ is latched by the rising edge of CLK.
V_{CC}	9		**Power:** +5V Power supply
$\overline{RDY_A}/\overline{REQ_A}$ $\overline{RDY_B}/\overline{REQ_B}$	10 30	0 0	**Ready/Request** (output, open-drain when programmed for a Ready function, driven High or Low when programmed for a Request function). These dual-purpose outputs may be programmed as Request lines for a DMA controller or as Ready lines to synchronize the CPU to the SCC data rate. The reset state is Ready.
$\overline{SYNC_A}$ $\overline{SYNC_B}$	11 29	I/O I/O	**Synchronization:** These pins can act either as inputs, outputs or part of the crystal oscillator circuit. In the Asynchronous Receive mode (crystal oscillator option not selected), these pins are inputs similar to $\overline{CTS}$ and $\overline{CD}$. In this mode, transitions on these lines affect the state of the Synchronous/Hunt status bits in Read Register 0 (Figure 9) but have no other function. In External Synchronization mode with the crystal oscillator not selected, these lines also act as inputs. In this mode, $\overline{SYNC}$ must be driven LOW two receive clock cycles after the last bit in the synchronous character is received. Character assembly begins on the rising edge of the receive clock immediately preceding the activation of $\overline{SYNC}$. In the Internal Synchronization mode (Monosync and Bisync) with the crystal oscillator not selected, these pins act as outputs and are active only during the part of the receive clock cycle in which synchronous characters are recognized. The synchronous condition is not latched, so these outputs are active each time a synchronization pattern is recognized (regardless of characters boundaries). In SDLC mode, these pins act as outputs and are valid on receipt of a flag.

Table 1. Pin Description (Cont.)

Symbol	Pin No.	Type	Name and Function
$\overline{RTxC}_A$ $\overline{RTxC}_B$	12 28	I I	**Receive/Transmit clocks:** These pins can be programmed in several different modes of operation. In each channel, $\overline{RTxC}$ may supply the receive clock, the transmit clock, the clock for the baud rate generator, or the clock for the Digital Phase Locked Loop. These pins can be programmed for use with the respective $\overline{SYNC}$ pins as a crystal oscillator. The receive clock may be 1, 16, 32, or 64 times the data rate in Asynchronous modes.
RxD_A RxD_B	13 27	I I	**Receive Data:** These lines receive serial data at standard TTL levels.
$\overline{TRxC}_A$ $\overline{TRxC}_B$	14 26	I/O I/O	**Transmit/Receive clocks:** These pins can be programmed in several different modes of operation. $\overline{TRxC}$ may supply the receive clock or the transmit clock in the input mode or supply the output of the Digital Phase Locked Loop, the crystal oscillator, the baud rate generator, or the transmit clock in the output mode.
TxDA TxDA	15 25	O O	**Transmit Data:** These output signals transmit serial data at standard TTL levels
$\overline{DTR}_A\overline{REQ}_A$ $\overline{DTR}_B\overline{REQ}_B$	16 24	O O	**Data Terminal Ready/Request:** These outputs follow the state programmed into the DTR bit. They can also be used as general purpose outputs or as Request lines for a DMA controller.
$\overline{RTS}_A$ $\overline{RTS}_B$	17 23	O O	**Request To Send:** When the Request to Send (RTS) bit in Write Register 5 is set (figure 10), the $\overline{RTS}$ signal goes Low. When the RTS bit is reset in the Asynchronous mode and Auto Enable is on, the signal goes High after the transmitter is empty. In Synchronous mode or in Asynchronous mode with Auto Enable off, the $\overline{RTS}$ pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.
$\overline{CTS}_A$ $\overline{CTS}_B$	18 22	I I	**Clear To Send:** If these pins are programmed as Auto Enables, a Low on the inputs enables the respective transmitters. If not programmed as Auto Enables, they may be used as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow rise-time inputs. The SCC detects pulses on these inputs and can interrupt the CPU on both logic level transitions.
$\overline{CD}_A$ $\overline{CD}_B$	19 21	I I	**Carrier Detect:** These pins function as receiver enables if they are programmed for Auto Enables; otherwise they may be used as general-purpose input pins. Both pins are Schmitt-trigger buffered to accommodate slow rise time signals. The SCC detects pulses on these pins and can interrupt the CPU on both logic level transitions.
CLK	20	I	**Clock:** This is the system SCC clock used to synchronize internal signals. CLK is a TTL level signal.
GND	31		**Ground**
$D/\overline{C}$	32	I	**Data/Command Select:** This signal defines the type of information transferred to or from the SCC A High means data is transferred; a Low indicates a command
$\overline{CS}$	33	1	**Chip Select:** This signal selects the SCC for a read or write operation.
$A/\overline{B}$	34	I	**Channel A/Channel B Select:** This signal selects the channel in which the read or write operation occurs
$\overline{WR}$	35	I	**Write:** When the SCC is selected this signal indicates a write operation. The coincidence of $\overline{RD}$ and $\overline{WR}$ is interpreted as a reset.
$\overline{RD}$	36	I	**Read:** This signal indicates a read operation and when the SCC is selected, enables the SCC's bus drivers. During the Interrupt Acknowledge cycle, this signal gates the interrupt vector onto the bus if the SCC is the highest priority device requesting an interrupt.

GENERAL DESCRIPTION

The INTEL 82530 Serial Communications Controller (SCC) is a dual-channel, multi-protocol data communications peripheral. The SCC functions as a serial-to-parallel, parallel-to-serial converter/controller. The SCC can be software-configured to satisfy a wide range of serial communications applications. The device contains new, sophisticated internal functions including on-chip baud rate generators, digital phase locked loops, various data encoding and decoding schemes, and crystal oscillators that dramatically reduce the need for external logic.

In addition, diagnostic capabilities - automatic echo and local loopback - allow the user to detect and isolate a failure in the network. They greatly improve the reliability and maintainability of the system.

The SCC handles Asynchronous formats, Synchronous byte-oriented protocols such as IBM Bisync, and Synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device supports virtually any serial data transfer application (Terminal, Personal Computer, Peripherals, Industrial Controller, Telecommunication system, etc.).

The 82530 can generate and check CRC codes in any Synchronous mode and can be programmed to check data integrity in various modes. The SCC also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

The INTEL 82530 is designed to support INTEL's MCS51, iAPX86/88 and iAPX186/188 families.

ARCHITECTURE

The 82530 internal structure includes two full-duplex channels, two baud rate generators, internal control and interrupt logic, and a bus interface to a non-multiplexed CPU bus. Associated with each channel are a number of read and write registers for mode control and status information, as well as logic necessary to interface to modems or other external devices.

The logic for both channels provides formats, synchronization, and validation for data transferred to and from the channel interface. The modem control inputs are monitored by the control logic under program control. All of the modem control signals are general-purpose in nature and can optionally be used for functions other than modem control.

The register set for each channel includes ten control (write) registers, two synchronous character (write) registers, and four status (read) registers. In addition, each baud rate generator has two (read/write) registers for holding the time constant that determines the baud rate. Finally, associated with the interrupt logic is a write register for the interrupt vector accessible through either channel, a write-only Master Interrupt Control register and three read registers: one containing the vector with status information (Channel B only), one containing the vector without status (A only), and one containing the Interrupt Pending bits (A only).

The registers for each channel are designated as follows:

WR0-WR15 - Write Registers 0 through 15.
RR0-RR3, RR10, RR12, RR13, RR15 - Read Registers 0 through 3, 10, 12, 13, 15

Table 2 lists the functions assigned to each read or write register. The SCC contains only one WR2 and WR9, but they can be accessed by either channel. All other registers are paired (one for each channel).

DATA PATH

The transmit and receive data path illustrated in Figure 3 is identical for both channels. The receiver has three 8-bit buffer registers in a FIFO arrangement, in addition to the 8-bit receive shift register. This scheme creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. Incoming data is routed through one of several paths (data or CRC) depending on the selected mode (the character length in asynchronous modes also determines the data path).

The transmitter has an 8-bit transmit data buffer register loaded from the internal data bus and a 20-bit transmit shift register that can be loaded either from the sync-character registers or from the transmit data register. Depending on the operational mode, outgoing data is routed through one of four main paths before it is transmitted from the Transmit Data output (TxD).

Table 2. Read and Write Register Functions

READ REGISTER FUNCTIONS		WRITE REGISTER FUNCTIONS	
RR0	Transmit/Receive buffer status and External status	**WR0**	CRC initialize, initialization commands for the various modes, shift right/shift left command
RR1	Special Receive Condition status	**WR1**	Transmit/Receive interrupt and data transfer mode definition
RR2	Modified interrupt vector (Channel B only) Unmodified interrupt (Channel A only)	**WR2**	Interrupt vector (accessed through either channel)
RR3	Interrupt Pending bits (Channel A only)	**WR3**	Receive parameters and control
RR8	Receive buffer	**WR4**	Transmit/Receive miscellaneous parameters and modes
RR10	Miscellaneous status	**WR5**	Transmit parameters and controls
RR12	Lower byte of baud rate generator time constant	**WR6**	Sync characters or SDLC address field
RR13	Upper byte of baud rate generator time constant	**WR7**	Sync character or SDLC flag
RR15	External/Status interrupt information	**WR8**	Transmit buffer
		WR9	Master interrupt control and reset (accessed through either channel)
		WR10	Miscellaneous transmitter/receiver control bits
		WR11	Clock mode control
		WR12	Lower Byte of baud rate generator time constant
		WR13	Upper byte of baud rate generator time constant
		WR14	Miscellaneous control bits
		WR15	External/Status interrupt control

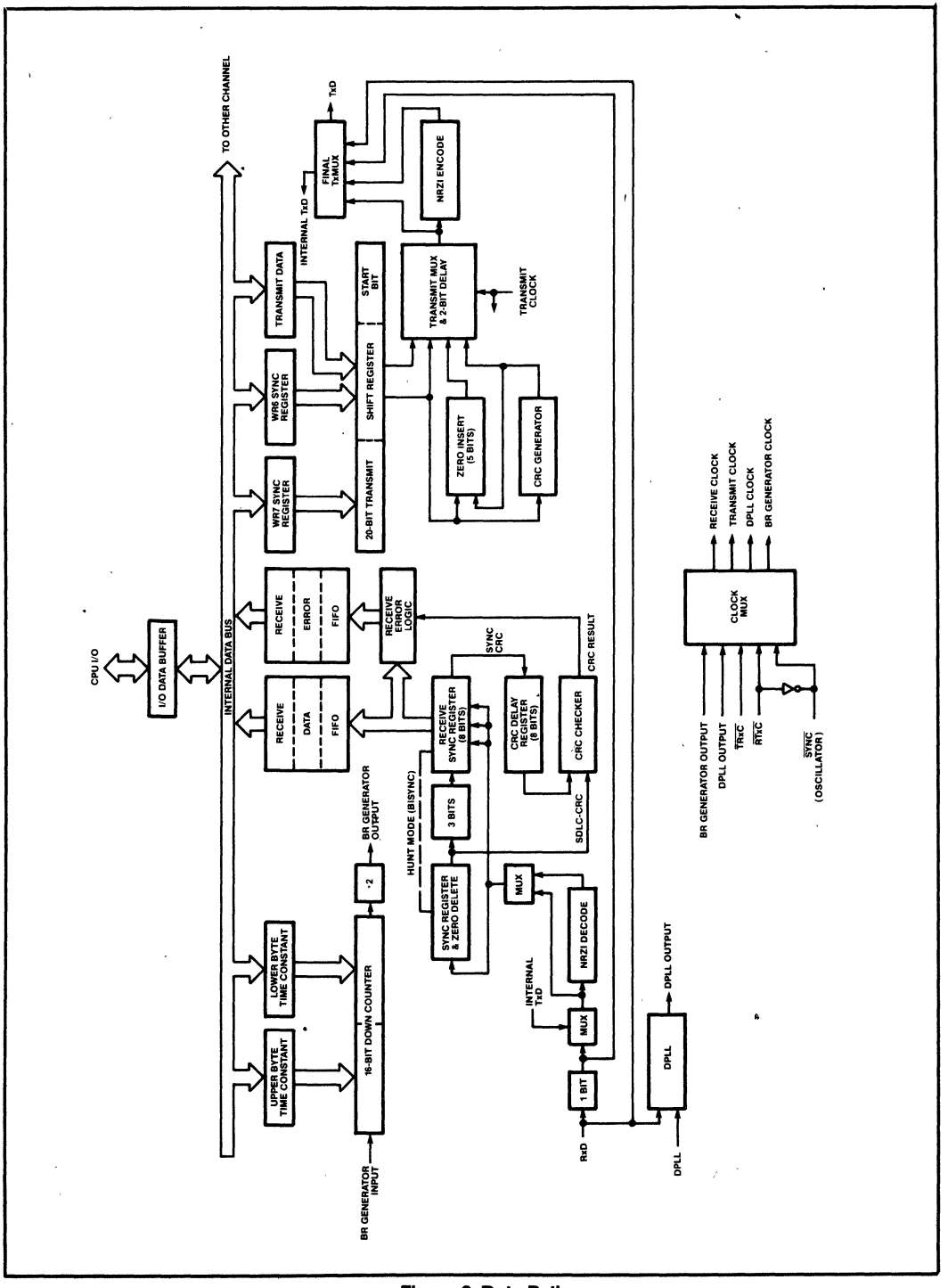


Figure 3 Data Path

230834-001

FUNCTIONAL DESCRIPTION

The functional capabilities of the SCC can be described from two different points of view: as a data communications device, it transmits and receives data in a wide variety of data communications protocols; as a microprocessor peripheral, it interacts with the CPU and provides vectored interrupts and handshaking signals.

DATA COMMUNICATIONS CAPABILITIES

The SCC provides two independent full-duplex channels programmable for use in any common asynchronous or synchronous data-communications protocol. Figure 4 and the following description briefly detail these protocols.

Asynchronous Modes

Transmission and reception can be accomplished independently on each channel with five to eight bits per character, plus optional even or odd parity. The transmitter can supply one, one-and-a-half or two stop bits per character and can provide a break output at any time. The receiver break-detection logic interrupts the CPU both at the start and at the end of a received break. Reception is protected from spikes by a transient spike-rejection mechanism that checks the signal one-half a bit time after a Low level is detected on the receive data input (RxD_A or RxD_B). If the Low does not persist (as in the case of a transient), the character assembly process does not start.

Framing errors and overrun errors are detected and buffered together with the partial character on which they occur. Vectored interrupts allow fast servicing or error conditions using dedicated routines. Furthermore, a built-in checking process avoids the interpretation of framing error as a new start bit: a framing error results in the addition of one-half a bit time to the point at which the search for the next start bit begins.

The SCC does not require symmetric transmit and receive clock signals — a feature allowing use of the wide variety of clock sources. The transmitter and receiver can handle data at a rate of 1, 1/16, 1/32, or 1/64 of the clock rate supplied to the receive and transmit clock inputs. In asynchronous modes, the $\overline{SYNC}$ pin may be programmed as an input used for functions such as monitoring a ring indicator.

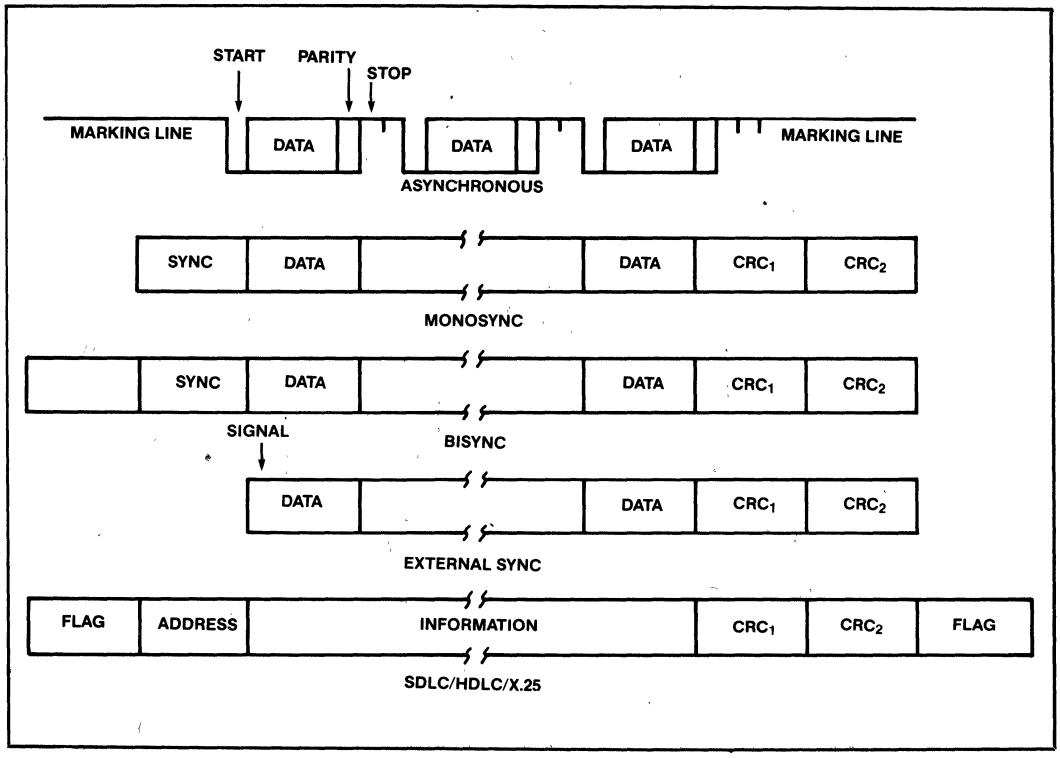


Figure 4. SCC Protocols

Synchronous Modes

The SCC supports both byte-oriented and bit-oriented synchronous communication. Synchronous-byte-oriented protocols can be handled in several modes allowing character synchronization with a 6-bit or 8-bit synchronous character (Monosync), any 12-bit synchronous pattern (Bisync), or with an external synchronous signal. Leading synchronous characters can be removed without interrupting the CPU.

Five- or 7-bit synchronous characters are detected with 8- or 16-bit patterns in the SCC by overlapping the larger pattern across multiple incoming synchronous characters as shown in Figure 5.

CRC checking for Synchronous byte-oriented mode is delayed by one character time so that the CPU may disable CRC checking on specific characters. This permits the implementation of protocols such as IBM Bisync.

Both CRC-16 ($X^{16} + X^{15} + X^2 + 1$) and CCITT ($X^{16} + X^{12} + X^5 + 1$) error checking polynomials are supported. Either polynomial may be selected in all synchronous modes. Users may preset the CRC generator and checker to all 1s or all 0s. The SCC also provides a feature that automatically transmits CRC data when no other data is available for transmission.
This allows for high-speed transmissions under DMA control, with no need for CPU intervention at the end of a message. When there is no data or CRC to send in synchronous modes, the transmitter inserts 6-, 8-, or 16-bit synchronous characters, regardless of the programmed character length.

The SCC supports synchronous bit-oriented protocols, such as SDLC and HDLC, by performing automatic flag sending, zero insertion, and CRC generation. A special command can be used to abort a frame in transmission. At the end of a message, the SCC automatically transmits the CRC and trailing flag when the transmitter underruns. The transmitter may also be programmed to send an idle line consisting of continuous flag characters or a steady marking condition.

If a transmit underrun occurs in the middle of a message, an external status interrupt warns the CPU of this status change so that an abort may be issued. The SCC may also be programmed to send an abort itself in case of an underrun, relieving the CPU of this task. One to eight bits per character can be sent allowing reception of a message with no prior information about the character structure in the information field of a frame.

The receiver automatically acquires synchronization on the leading flag of a frame in SDLC or HDLC and provides a synchronization signal on the SYNC pin (an interrupt can also be programmed). The receiver can be programmed to search for frames addressed by a single byte (or four bits within a byte) of a user-selected address or to a global broadcast address. In this mode, frames not matching either the user-selected or broadcast address are ignored. The number of address bytes can be extended under software control. For receiving data, an interrupt on the first received character, or an interrupt on every character, or on special condition only (end-of-frame) can be selected. The receiver automatically deletes all 0s inserted by the transmitter during character assembly. CRC is also calculated and is automatically checked to validate frame transmission. At the end of transmission, the status of a received frame is available in the status registers. In SDLC mode, the SCC must be programmed to use the SDLC CRC polynomial, but the generator and checker may be be preset to all 1s or all 0s. The CRC is inverted before transmission and the receiver checks against the bit pattern 0001110100001111.

NRZ, NRZI or FM coding may be used in any 1X mode. The parity options available in asynchronous modes are available in synchronous modes.

The SCC can be conveniently used under DMA control to provide high-speed reception or transmission. In reception, for example, the SCC can interrupt the CPU when the first character of a message is received. The CPU then enables the DMA to transfer the message to memory. The SCC then issues an end-of-frame interrupt and the CPU can

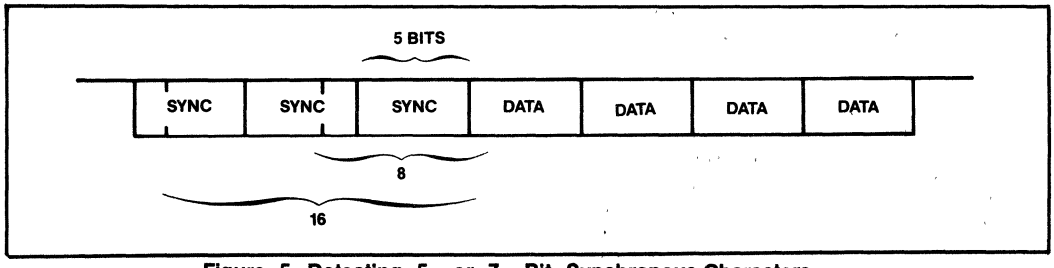


Figure 5. Detecting 5- or 7- Bit Synchronous Characters

check the status of the received message. Thus, the CPU is freed for other service while the message is being received. The CPU may also enable the DMA first and have the SCC interrupt only on end-of-frame. This procedure allows all data to be transferred via DMA.

SDLC LOOP MODE

The SCC supports SDLC Loop mode in addition to normal SDLC. In an SDLC Loop, there is a primary controller station that manages the message traffic flow and any number of secondary stations. In SDLC Loop mode, the SCC performs the functions of a secondary station while an SCC operating in regular SDLC mode can act as a controller (Figure 6).

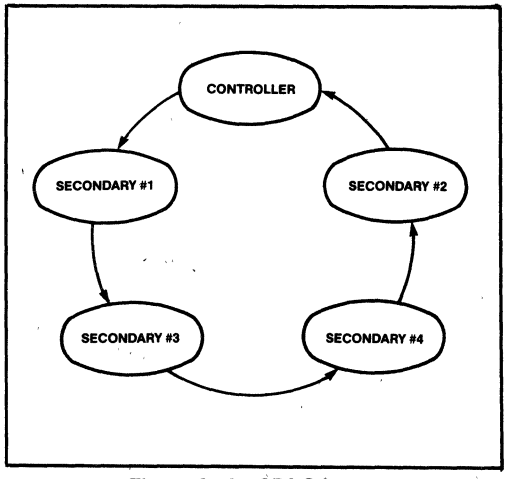


Figure 6. An SDLC Loop

A secondary station in an SDLC Loop is always listening to the messages being sent around the loop, and in fact must pass these messages to the rest of the loop by retransmitting them with a one-bit-time delay. The secondary station can place its own message on the loop only at specific times. The controller signals that secondary stations may transmit messages by sending a special character, called an EOP (End of Poll), around the loop. The EOP character is the bit pattern 11111110. Because of zero insertion during messages, this bit pattern is unique and easily recognized.

When a secondary station has a message to transmit and recognizes an EOP on the line, it changes the last binary one of the EOP to a zero before transmission. This has the effect of turning the EOP into a flag sequence. The secondary station now places its message on the loop and terminates the message with an EOP. Any secondary stations further down the loop with messages to transmit can then append their messages to the message of the first secondary station by the same process. Any secondary stations without messages to send merely echo the incoming messages and are prohibited from placing messages on the loop (except upon recognizing an EOP).

SDLC Loop mode is a programmable option in the SCC. NRZ, NRZI, and FM coding may all be used in SDLC Loop mode.

BAUD RATE GENERATOR

Each channel in the SCC contains a programmable Baud rate generator. Each generator consists of two 8-bit time constant registers that form a 16-bit time constant, a 16-bit down counter, and a flip-flop on the output producing a square wave. On startup, the flip-flop on the output is set in a High state, the value in the time constant register is loaded into the counter, and the counter starts counting down. The output of the baud rate generator toggles upon reaching zero, the value in the time constant register is loaded into the counter, and the process is repeated. The time constant may be changed at any time, but the new value does not take effect until the next load of the counter.

The output of the baud rate generator may be used as either the transmit clock, the receive clock, or both. It can also drive the digital phase-locked loop (see next section).

If the receive clock or transmit clock is not programmed to come from the $\overline{\text{TRxC}}$ pin, the output of the baud rate generator may be echoed out via the $\overline{\text{TRxC}}$ pin.

The following formula relates the time constant to the baud rate. (The baud rate is in bits/second and the BR clock period is in seconds.)

$$\text{baud rate} = \frac{1}{2\,(\text{time constant} + 2) \times (\text{BR clock period})}$$

Time Constant Values for Standard Baud Rates at BR Clock = 3.9936MHz

Rate (Baud)	Time Constant (decimal notation)	Error
19200	102	—
9600	206	—
7200	275	0.12%
4800	414	—
3600	553	0.06%
2400	830	—
2000	996	0.04%
1800	1107	0.03%
1200	1662	—
600	3326	—
300	6654	—
150	13310	—
134.5	14844	0.0007%
110	18151	0.0015%
75	26622	—
50	39934	—

DIGITAL PHASE LOCKED LOOP

The SCC contains a digital phase locked-loop (DPLL) to recover clock information from a datastream with NRZI or FM encoding. The DPLL is driven by a clock that is nominally 32 (NRZI) or 16 (FM) times the data rate. The DPLL uses this clock, along with the datastream, to construct a clock for the data. This clock may then be used as the SCC receive clock, the transmit clock, or both.

For NRZI coding, the DPLL counts the 32X clock to create nominal bit times. As the 32X clock is counted, the DPLL is searching the incoming datastream for edges (either 1/0 or 0/1). Whenever an edge is detected, the DPLL makes a count adjustment (during the next counting cycle), producing a terminal count closer to the center of the bit cell.

For FM encoding, the DPLL still counts from 1 to 31, but with a cycle corresponding to two bit times. When the DPLL is locked, the clock edges in the datastream should occur between counts 15 and 16 and between counts 31 and 0. The DPLL looks for edges only during a time centered on the 15/16 counting transition.

The 32X clock for the DPLL can be programmed to come from either the $\overline{RTxC}$ input or the output of the baud rate generator. The DPLL output may be programmed to be echoed out of the SCC via the $\overline{TRxC}$ pin (if this pin is not being used as an input).

DATA ENCODING

The SCC may be programmed to encode and decode the serial data in four different ways (Figure 7). In NRZ encoding, a 1 is represented by a High level and a 0 is represented by a Low level. In NRZI encoding, as 1 is represented by no change in level and a 0 is represented by a change in level. In FM_1 (more properly, bi-phase mark) a transition occurs at the beginning of every bit cell. A 1 is represented by an additional transition at the center of the bit cell and a 0 is represented by no additional transition at the center of the bit cell. In FM_0 (bi-phase space), a transition occurs at the beginning of every bit cell. A 0 is represented by an additional transition at the center of the bit cell, and a 1 is represented by no additional transition at the center of the bit cell. In addition to these four methods, the SCC can be used to decode Manchester (bi-phase level) data by using the DPLL in the FM mode and programming the receiver for NRZ data. Manchester encoding always produces a transition at the center of the bit cell. If the transition is 0/1 the bit is a 0. If the transition is 1/0 the bit is a 1.

AUTO ECHO AND LOCAL LOOPBACK

The SCC is capable of automatically echoing everything it receives. This feature is useful mainly in asynchronous modes, but works in synchronous and SDLC modes as well. In Auto Echo mode TxD is RxD. Auto Echo mode can be used with NRZI or FM encoding with no additional delay, because the datastream is not decoded before retransmission. In Auto Echo mode, the $\overline{CTS}$ input is ignored as a transmitter enable (although transitions on this input can still cause interrupts if programmed to do so). In this mode, the transmitter is actually bypassed and the programmer is responsible for disabling transmitter interrupts and $\overline{READY}/\overline{REQUEST}$ on transmit.

The SCC is also capable of local loopback. In this mode, TxD is RxD just as in Auto Echo mode. However, in Local Loopback mode, the internal transmit data is tied to the internal receive data and RxD is ignored (except to be echoed out via TxD). $\overline{CTS}$ and $\overline{CD}$ inputs are also ignored as transmit and receive enables. However, transitions on these inputs can still cause interrupts. Local Loopback works in asynchronous, synchronous and SDLC modes with NRZ, NRZI or FM coding of the data stream.

SERIAL BIT RATE

To run the 82530 (4Mhz) at 1Mbps the receive and transmit clocks must be externally generated and synchronized to the system clock. If the serial clocks ($\overline{RTxC}$ and $\overline{TRxC}$) and the system clock (CLK) are asynchronous, the maximum bit rate is 880 Kbps. For self-clocked operation, i.e using the on chip DPLL, the maximum bit rate is 125 Kbps if NRZI coding is used and 250 Kbps if FM coding is used.

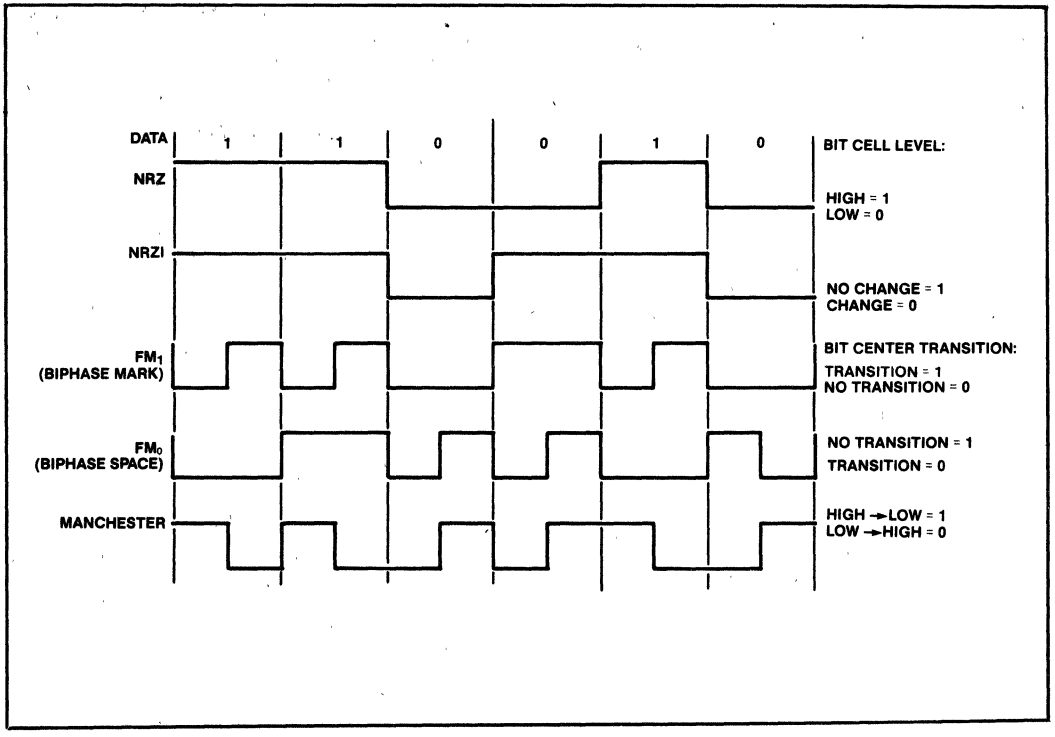


Figure 7. Data Encoding Methods

Mode	System clock	System clock/ Serial clock	Serial bit rate	Conditions
Serial clocks generated externally	4Mhz	4	1 Mbps	Serial clocks synchronized with system clock. Refer to parameter #3 and #10 in general timings.
	6 Mhz	4	1.5 Mbps	Serial clocks synchronized with system clock. Refer to parameter #3 and #10 in general timings.
	4 Mhz	4.5	880 Kbps	Serial clocks and system clock asynchronous.
	6 Mhz	4.5	1.3 Mbps	Serial clocks and system clock asynchronous
Self-clocked operation				
NRZI	4 Mhz	32	125 Kbps	
	6 Mhz	32	187 Kbps	
FM	4 Mhz	16	250 Kbps	
	6 Mhz	16	375 kbps	

I/O INTERFACE CAPABILITIES

The SCC offers the choice of Polling, Interrupt (vectored or nonvectored) and Block Transfer modes to transfer data, status, and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

POLLING

All interrupts are disabled. Three status registers in the SCC are automatically updated whenever any function is performed. For example, end-of-frame in SDLC mode sets a bit in one of these status registers. The idea behind polling is for the CPU to periodically read a status register until the register contents indicate the need for data to be transferred. Only one register needs to be read; depending on its contents, the CPU either writes data, reads data, or continues. Two bits in the register indicate the need for data transfer. An alternative is a poll of the Interrupt Pending register to determine the source of an interrupt. The status for both channels resides in one register.

INTERRUPTS

When a SCC responds to an Interrupt Acknowledge signal ($\overline{INTA}$) from the CPU, an interrupt vector may be placed on the data bus. This vector is written in WR2 and may be read in RR2A or RR2B (Figures 9 and 10).

To speed interrupt response time, the SCC can modify three bits in this vector to indicate status. If the vector is read in Channel A, status is never included; if it is read in Channel B, status is always included.

Each of the six sources of interrupts in the SCC (Transmit, Receive and External/Status interrupts in both channels) has three bits associated with the interrupt source: Interrupt Pending (IP), Interrupt Under Service (IUS), and Interrupt Enable (IE). Operation of the IE bit is straightforward. If the IE bit is set for a given interrupt source, then that source can request interrupts. The exception is when the MIE (Master Interrupt Enable) bit in WR9 is reset and no interrupts may be requested. The IE bits are write-only.

The other two bits are related to the interrupt priority chain (Figure 8). As a peripheral, the SCC may request an interrupt only when no higher-priority device is requesting one, e.g., when IEI is High. If the device in question requests an interrupt, it pulls down $\overline{INT}$. The CPU then responds with $\overline{INTA}$, and the interrupting device places the vector on the data bus.

In the SCC, the IP bit signals a need for interrupt servicing. When an IP bit is 1 and the IEI input is High, the $\overline{INT}$ output is pulled Low, requesting an interrupt. In the SCC, if the IE bit is not set by enabling interrupts, then the IP for that source can never be set. The IP bits are readable in RR3A.

The IUS bits signal that an interrupt request is being serviced. If an IUS is set, all interrupt sources of lower priority in the SCC and external to the SCC are prevented from requesting interrupts. The internal interrupt sources are inhibited by the state of the internal daisy chain, while lower priority devices are inhibited by the IEO output of the SCC being pulled

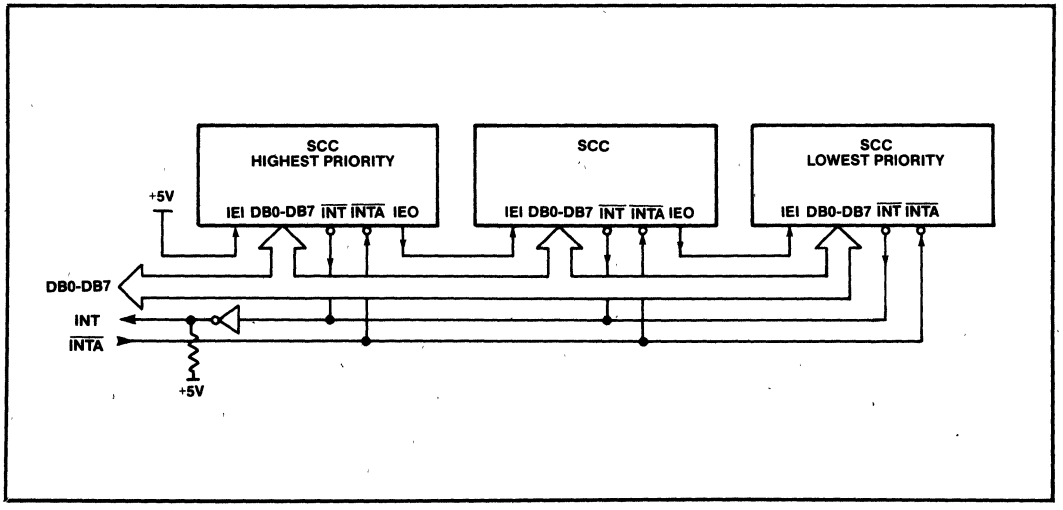


Figure 8. Daisy Chaining SCC's

Low and propagated to subsequent peripherals. An IUS bit is set during an Interrupt Acknowledge cycle if there are no higher priority devices requesting interrupts.

There are three types of interrupts: Transmit, Receive and External/Status interrupts. Each interrupt type is enabled under program control with Channel A having higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted when the transmit buffer becomes empty. (This implies that the transmitter must have had a data character written into it so that it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on First Receive Character or Special Receive condition.
- Interrupt on all Receive Characters or Special Receive condition.
- Interrupt on Special Receive condition only.

Interrupt on First Character or Special Condition and Interrupt on Special Condition Only are typically used with the Block Transfer mode. A Special Receive Condition is one of the following: receiver overrun, framing error in Asynchronous mode, End-of-Frame in SDLC mode and, optionally, a parity error. The Special Receive Condition interrupt is different from an ordinary receive character available interrupt only in the status placed in the vector during the Interrupt-Acknowledge cycle. In Interrupt on First Receive Character, an interrupt can occur from Special Receive conditions any time after the first receive character interrupt.

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{CTS}$, $\overline{CD}$, and $\overline{SYNC}$ pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition, or a zero count in the baud rate generator, or by the detection of a Break (asynchronous mode), Abort (SDLC mode) or EOP (SDLC Loop mode) sequence in the data stream. The interrupt caused by the Abort or EOP has a special feature allowing the SCC to interrupt when the Abort or EOP sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Abort condition in external logic in SDLC mode. In SDLC Loop mode this feature allows secondary stations to recognize the wishes of the primary station to regain control of the loop during a poll sequence.

CPU/DMA BLOCK TRANSFER

The SCC provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers. The Block Transfer mode uses the $\overline{READY}/\overline{REQUEST}$ output in conjunction with the $\overline{READY}/\overline{REQUEST}$ bits in WR1. The $\overline{READY}/\overline{REQUEST}$ output can be defined under software control as a $\overline{READY}$ line in the CPU Block Transfer mode (WR1; D6=0) or as a request line in the DMA Block Transfer mode (WR1; D6=1). To a DMA controller, the SCC $\overline{REQUEST}$ output indicates that the SCC is ready to transfer data to or from memory. To the CPU, the $\overline{READY}$ line indicates that the SCC is not ready to transfer data, thereby requesting that the CPU extend the I/O cycle. The $\overline{DTR}/\overline{REQUEST}$ line allows full-duplex operation under DMA control.

PROGRAMMING

Each channel has fifteen Write registers that are individually programmed from the system bus to configure the functional personality of each channel. Each channel also has eight Read registers from which the system can read Status, Baud rate, or Interrupt information.

Only the four data registers (Read, Write for channels A and B) are directly selected by a High on the D/$\overline{C}$ input and the appropriate levels on the $\overline{RD}$, $\overline{WR}$ and A/$\overline{B}$ pins. All other registers are addressed indirectly by the content of Write Register 0 in conjunction with a Low on the D/$\overline{C}$ input and the appropriate levels on the $\overline{RD}$, $\overline{WR}$ and A/$\overline{B}$ pins. If bit 4 in WW0 is 1 and bits 5 and 6 are 0 then bits 0, 1, 2 address the higher registers 8 through 15. If bits 4, 5, 6 contain a different code, bits 0, 1, 2 address the lower registers 0 through 7 as shown on Table 3.

Writing to or reading from any register except RR0, WR0 and the Data Registers thus involves two operations:

First write the appropriate code into WR0,then follow this by a write or read operation on the register thus specified. Bits 0 through 4 in WW0 are automatically cleared after this operation, so that WW0 then points to WR0 or RR0 again.

Channel A/Channel B selection is made by the A/$\overline{B}$ input (High = A, Low = B)

The system program first issues a series of commands to initialize the basic mode of operation. This is followed by other commands to qualify condi-

TABLE 3. REGISTER ADDRESSING

D/C "Point High" Code in WR0		D_2 D_1 in WR0		D_0	Write Register	Read Register
High	Either Way	X	X	X	Data	Data
Low	Not True	0	0	0	0	0
Low	Not True	0	0	1	1	1
Low	Not True	0	1	0	2	2
Low	Not True	0	1	1	3	3
Low	Not True	1	0	0	4	(0)
Low	Not True	1	0	1	5	(1)
Low	Not True	1	1	0	6	(2)
Low	Not True	1	1	1	7	(3)
Low	True	0	0	0	Data	Data
Low	True	0	0	1	9	-
Low	True	0	1	0	10	10
Low	True	0	1	1	11	(15)
Low	True	1	0	0	12	12
Low	True	1	0	1	13	13
Low	True	1	1	0	14	(10)
Low	True	1	1	1	15	15

tions within the selected mode. For example, the asynchronous mode, character length, clock rate, number of stop bits, even or odd parity might be set first. Then the interrupt mode would be set, and finally, receiver or transmitter enable.

READ REGISTERS

The SCC contains eight read registers (actually nine, counting the receive buffer (RR8) in each channel). Four of these may be read to obtain status information (RR0, RR1, RR10, and RR15). Two registers (RR12 and RR13) may be read to earn the baud rate generator time constant. RR2 contains either the unmodified interrupt vector (Channel A) or the vector modified by status information (Channel B). RR3 contains the Interrupt Pending (IP) bits (Channel A). Figure 9 shows the formats for each read register.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring: e.g. when the interrupt vector indicates a Special Receive Condition interrupt, all the appropriate error bits can be read from a single register (RR1).

WRITE REGISTERS

The SCC contains 15 write registers (16 counting WR8, the transmit buffer) in each channel. These write registers are programmed separately to configure the functional "personality" of the channels. In addition, there are two registers (WR2 and WR9) shared by the two channels that may be accessed through either of them. WR2 contains the interrupt vector for both channels, while WR9 contains the interrupt control bits. Figure 10 shows the format of each write register.

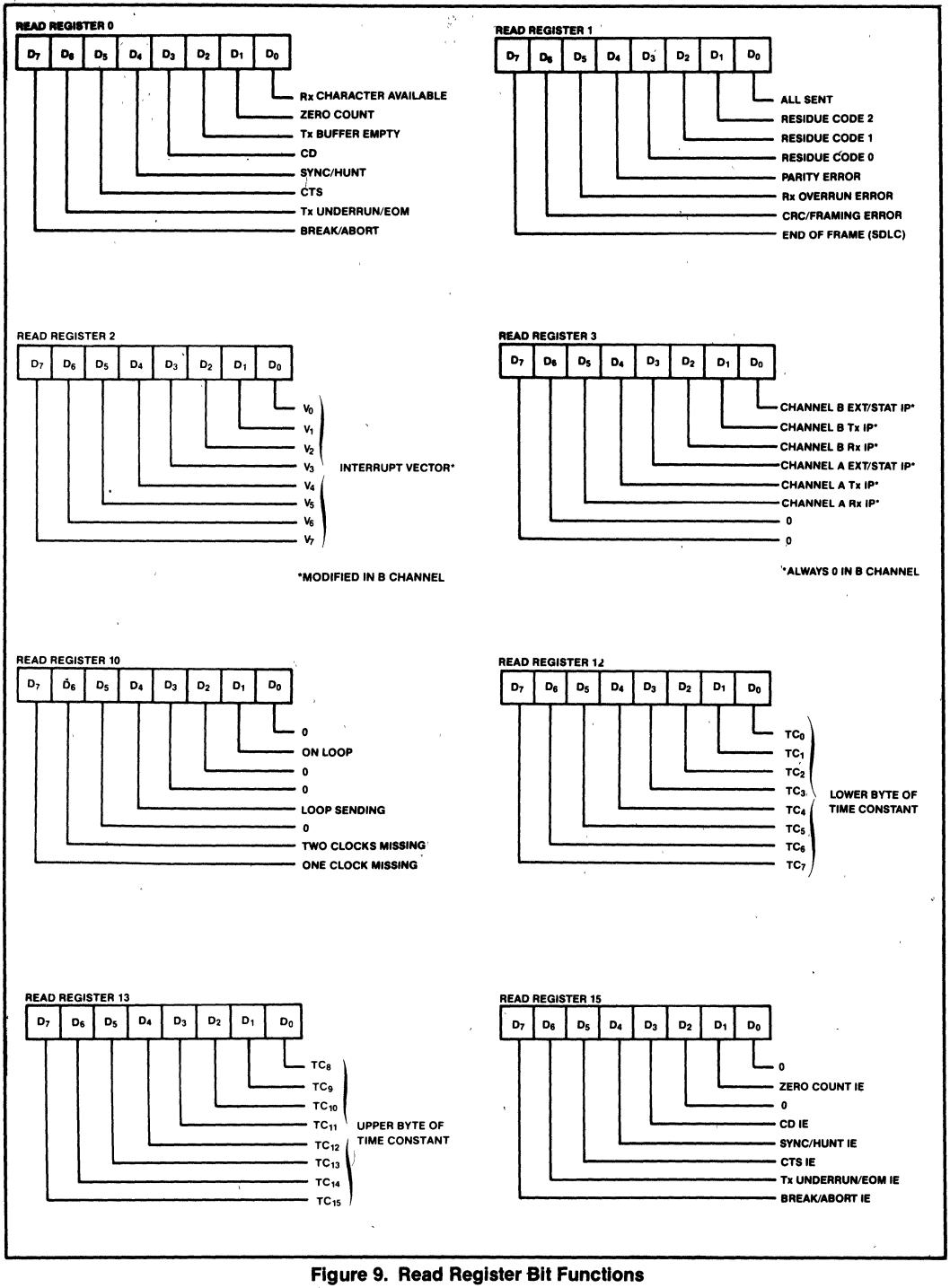


Figure 9. Read Register Bit Functions

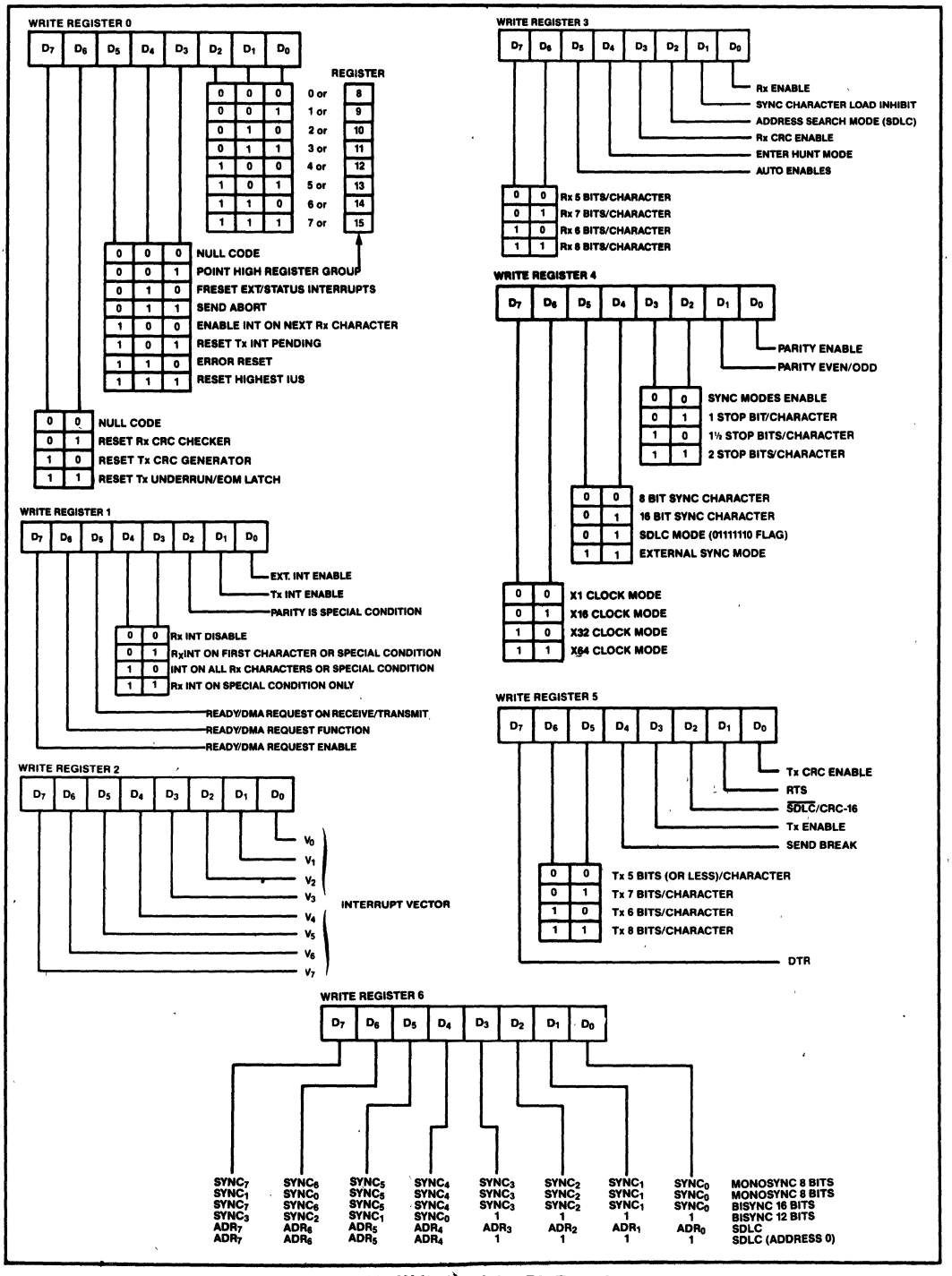


Figure 10. Write Register Bit Functions

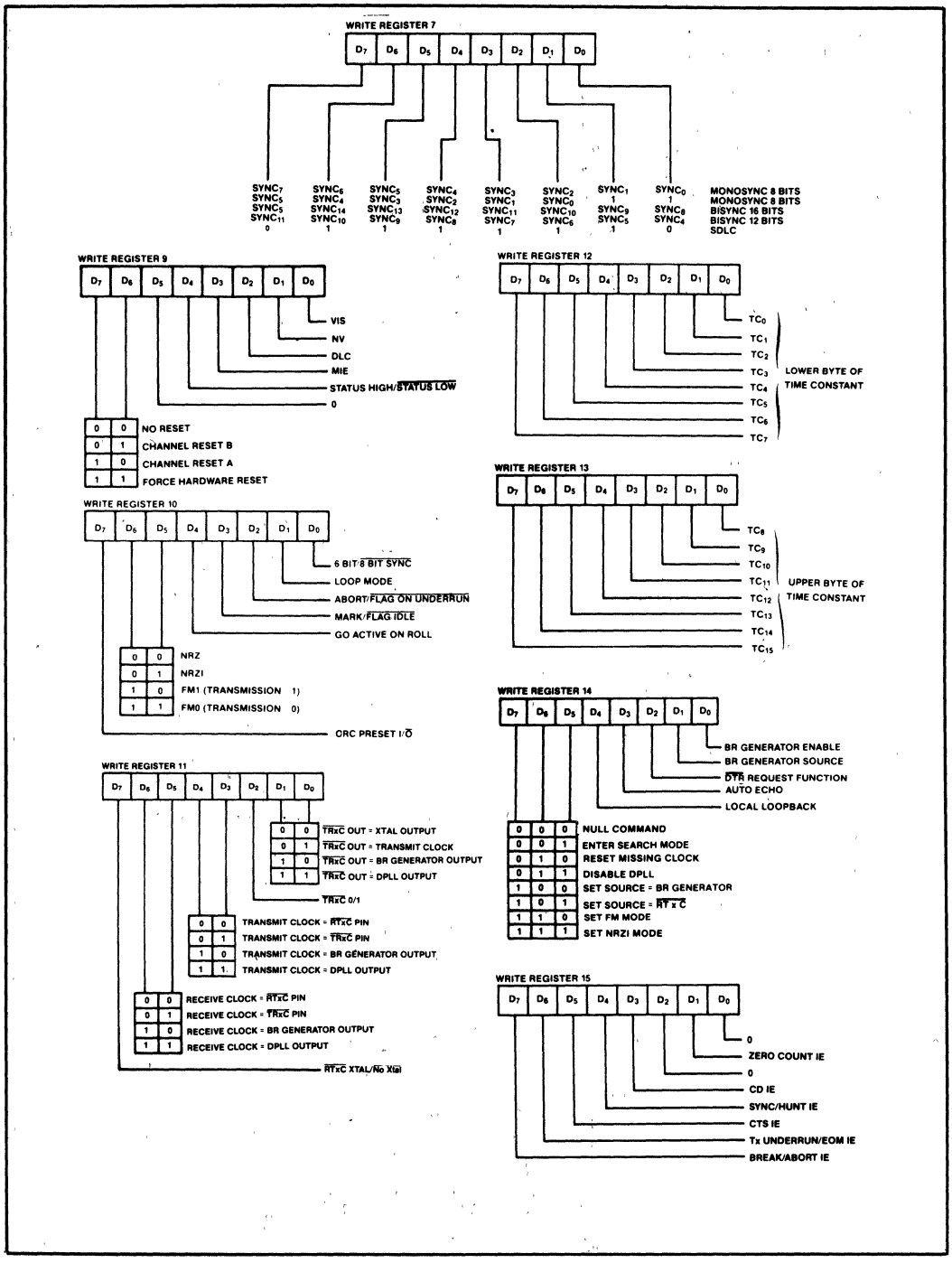


Figure 10. Write Register Bit Functions (Cont.)

82530 TIMING

The SCC generates internal control signals from $\overline{WR}$ and $\overline{RD}$ that are related to CLK. Since CLK has no phase relationship with $\overline{WR}$ and $\overline{RD}$, the circuitry generating these internal control signals must provide time for metastable conditions to disappear. This gives rise to a recovery time related to CLK. The recovery time applies only between bus transactions involving the SCC. The recovery time required for proper operaton is specified from the rising edge of $\overline{WR}$ or $\overline{RD}$ in the first transaction involving the SCC to the falling edge of $\overline{WR}$ or $\overline{RD}$ in the second transaction involving the SCC. This time must be at least 6 CLK cycles plus 200ns.

Read Cycle Timing

Figure 11 illustrates Read cycle timing. Addresses on A/$\overline{B}$ and D/$\overline{C}$ and the status on $\overline{INTA}$ must remain stable throughout the cycle. If $\overline{CS}$ falls after $\overline{RD}$ falls or if it rises before $\overline{RD}$ rises, the effective $\overline{RD}$ is shortened.

Write Cycle Timing

Figure 12 illustrates Write cycle timing. Addresses on A/$\overline{B}$ and D/$\overline{C}$ and the status on $\overline{INTA}$ must remain stable throughout the cycle. If $\overline{CS}$ falls after $\overline{WR}$ falls or if it rises before $\overline{WR}$ rises, the effective $\overline{WR}$ is shortened.

Interrupt Acknowledge Cycle Timing

Figure 13 illustrates Interrupt Acknowledge cycle timing. Between the time $\overline{INTA}$ goes Low and the falling edge of $\overline{RD}$, the internal and external IEI/IEO daisy chains settle. If there is an interrupt pending in the SCC and IEI is High when $\overline{RD}$ falls, the Acknowledge cycle is intended for the SCC. In this case, the SCC may be programmed to respond to $\overline{RD}$ Low by placing its interrupt vector on D_0-D_7 and it then sets the appropriate Interrupt-Under-Service internally.

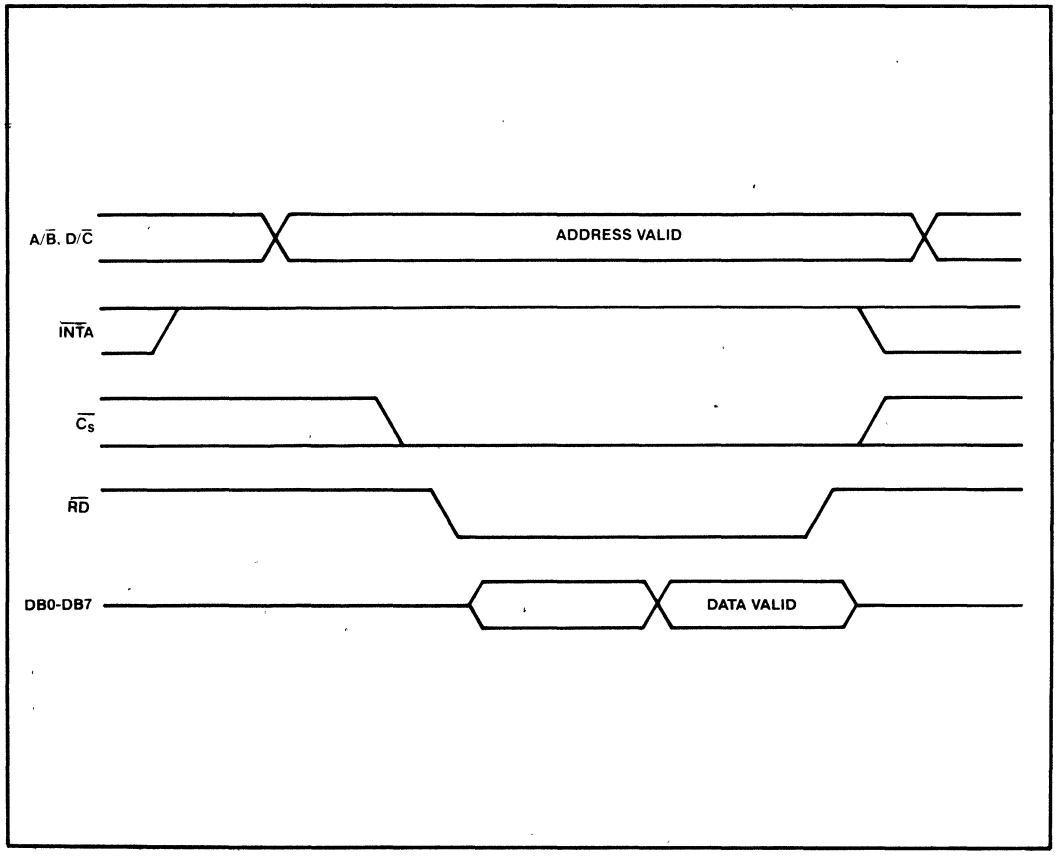


Figure 11. Read Cycle Timing

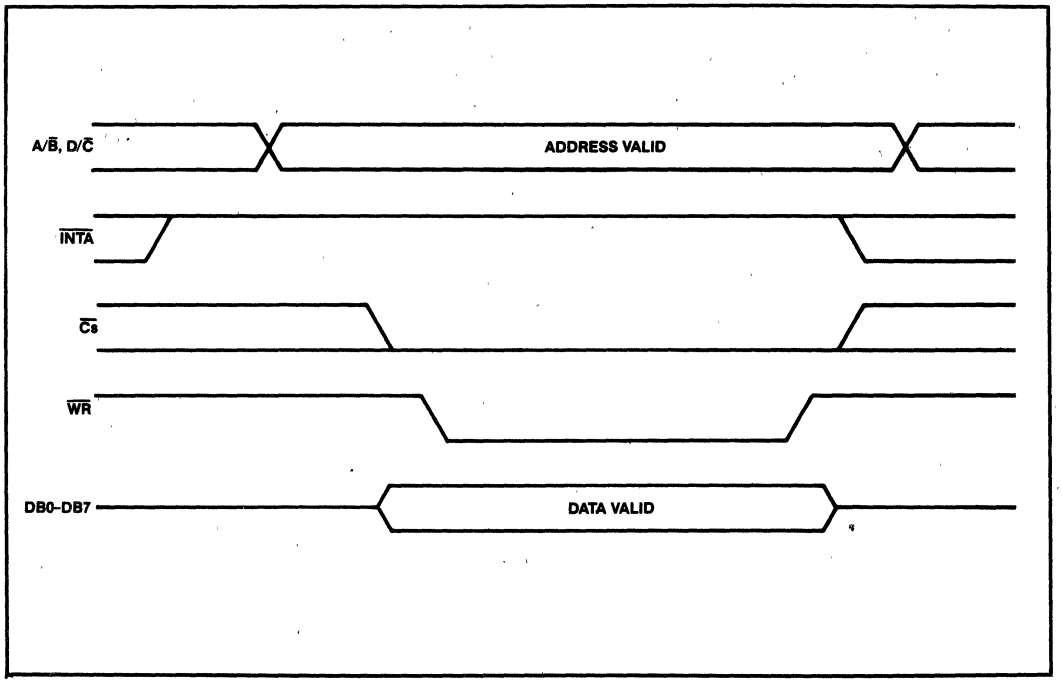


Figure 12. Write Cycle Timing

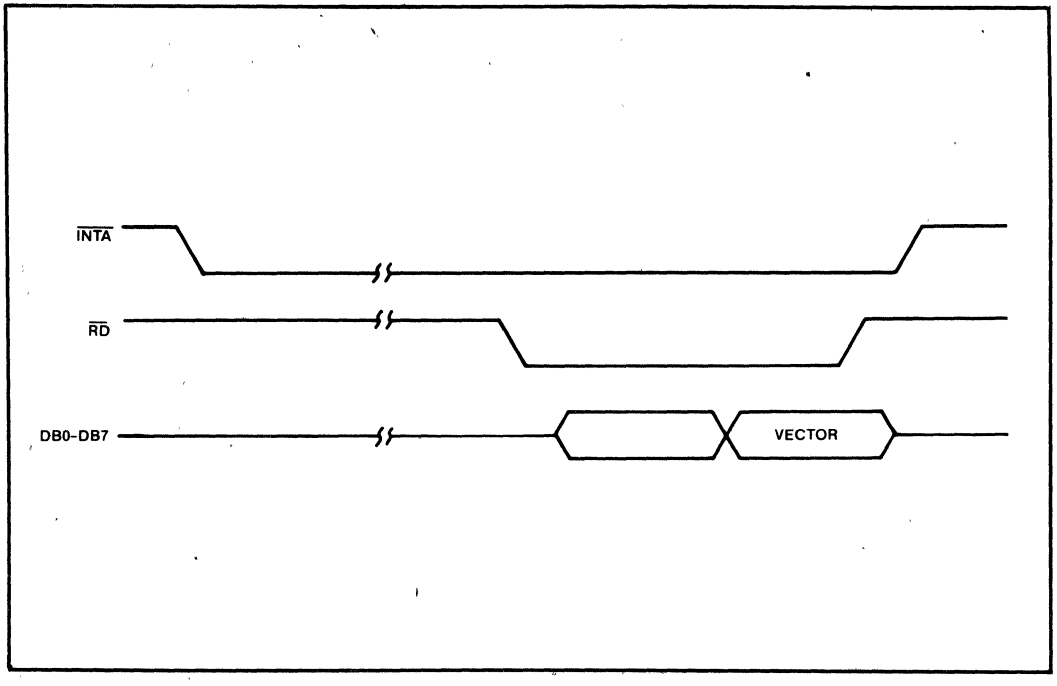


Figure 13. Interrupt Acknowledge Cycle Timing

ABSOLUTE MAXIMUM RATINGS*

Case Temperature
Under Bias ..... 0°C to +70°C
Storage Temperature
(Ceramic Package) ..... −65°C to +150°C
(Plastic Package) ..... −40°C to +125°C
Voltage On Any Pin With
Respect to Ground ..... −0.5V to +7.0V

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_C=0° C to 70° C; V_{CC}=+5V±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0 3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		+0.40	V	I_{OL} = 2 0mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = −250 μA
I_{IL}	Input Leakage Current		±10	μA	0.4 to 2.4V
I_{OL}	Output Leakage Current		±10	μA	0 4 to 2 4V
I_{CC}	V_{CC} Supply Current		250	mA	

CAPACITANCE (T_C=25° ; V_{CC}=GND=0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz,
C_{OUT}	Output Capacitance		15	pF	Unmeasured
$C_{I/O}$	Input/Output Capacitance		20	pF	pins returned to GND

A.C CHARACTERISTICS (T_C=0° C to 70° C; V_{CC}=+5V±5%)

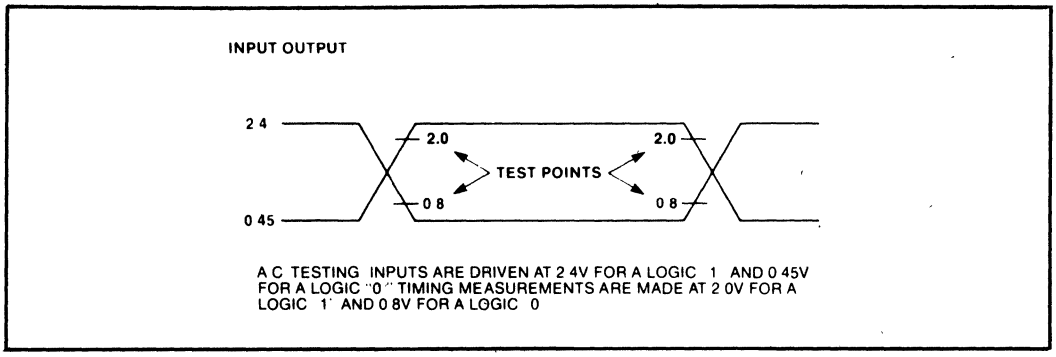
READ AND WRITE TIMING

			82530 (4MHz)		82530-6 (6 MHz)		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Units**
1	tCL	CLK Low Time	105	2000	70	1000	ns
2	tCH	CLK High Time	105	2000	70	1000	ns
3	tf	CLK Fall Time		20		10	ns
4	tr	CLK Rise Time		20		15	ns
5	tCY	CLK Cycle Time	250	4000	165	2000	ns
6	tAW	Address to $\overline{WR}$↓ Setup Time	80		80		ns
7	tWA	Address to $\overline{WR}$↑ Hold Time	0		0		ns
8	tAR	Address to $\overline{RD}$↓ Setup Time	80		80		ns
9	tRA	Address to $\overline{RD}$↑ Hold Time	0		0		ns
10	tIC	$\overline{INTA}$ to CLK↑ Setup Time	0		0		ns
11	tIW	$\overline{INTA}$ to $\overline{WR}$↓ Setup Time (Note 1)	200		200		ns
12	tWI	$\overline{INTA}$ to $\overline{WR}$↑ Hold Time	0		0		ns
13	tIR	$\overline{INTA}$ to $\overline{RD}$↓ Setup Time (Note 1)	200		200		ns
14	tRI	$\overline{INTA}$ to $\overline{RD}$↑ Hold Time	0		0		ns
15	tCI	$\overline{INTA}$ to $\overline{CLK}$↑ Hold Time	100		100		ns
16	tCLW	$\overline{CS}$ Low to $\overline{WR}$↓ Setup Time	0		0		ns
17	tWCS	$\overline{CS}$ to $\overline{WR}$↑ Hold Time	0		0		ns
18	tCHW	$\overline{CS}$ High to $\overline{WR}$↓ Setup Time	100		70		ns
19	tCLR	$\overline{CS}$ Low to $\overline{RD}$↓ Setup Time (Note 1)	0		0		ns
20	tRCS	$\overline{CS}$ to $\overline{RD}$↑ Hold Time (Note 1)	0		0		ns
21	tCHR	$\overline{CS}$ High to $\overline{RD}$↓ Setup Time (Note 1)	100		70		ns
22	tRR	$\overline{RD}$ Low Time (Note 1)	390		250		ns
23	tRDA	$\overline{RD}$↓ to Data Active Delay	0		0		ns
24	tRDI	$\overline{RD}$↑ to Data Not Valid Delay	0		0		ns
25	tRDV	$\overline{RD}$↓ to Data Valid Delay		250		180	ns
26	tDF	$\overline{RD}$↑ to Output Float Delay (Note 2)		70		45	ns

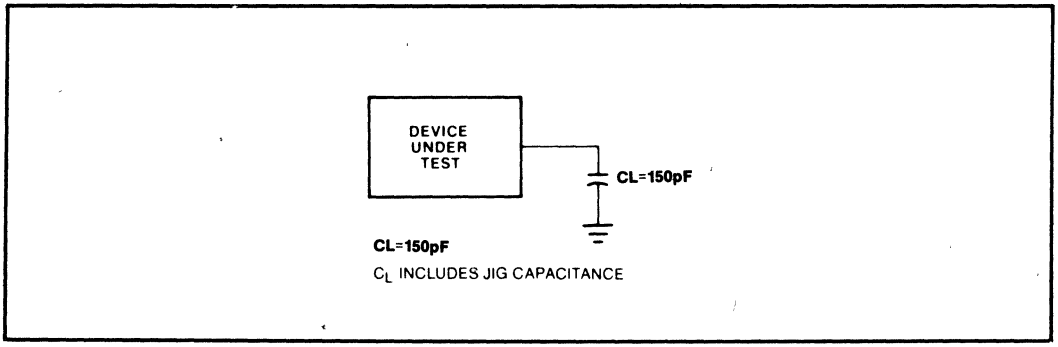
NOTES:
1. Parameter does not apply to Interrupt Acknowledge transactions.
2. Float delay is defined as the time required for a + 0.5V change in the output with a maximum D.C load and minimum A.C load.

*Timings are preliminary and subject to change.

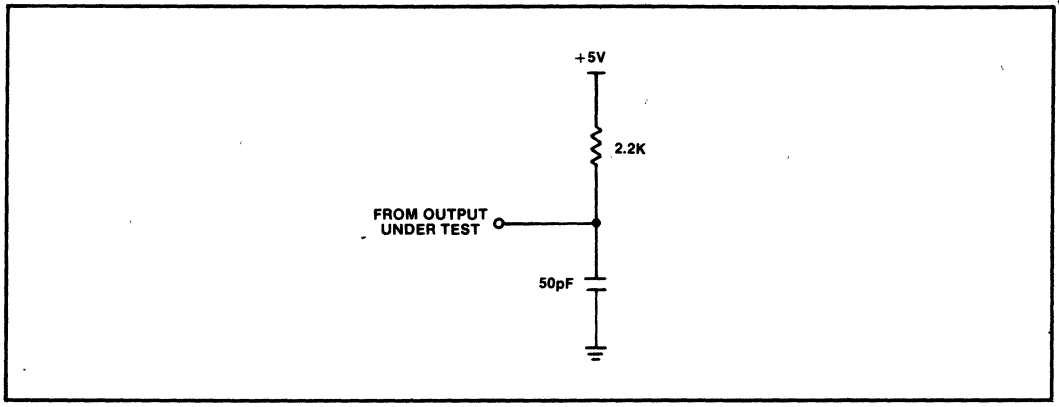
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



OPEN DRAIN TEST LOAD



230834-001

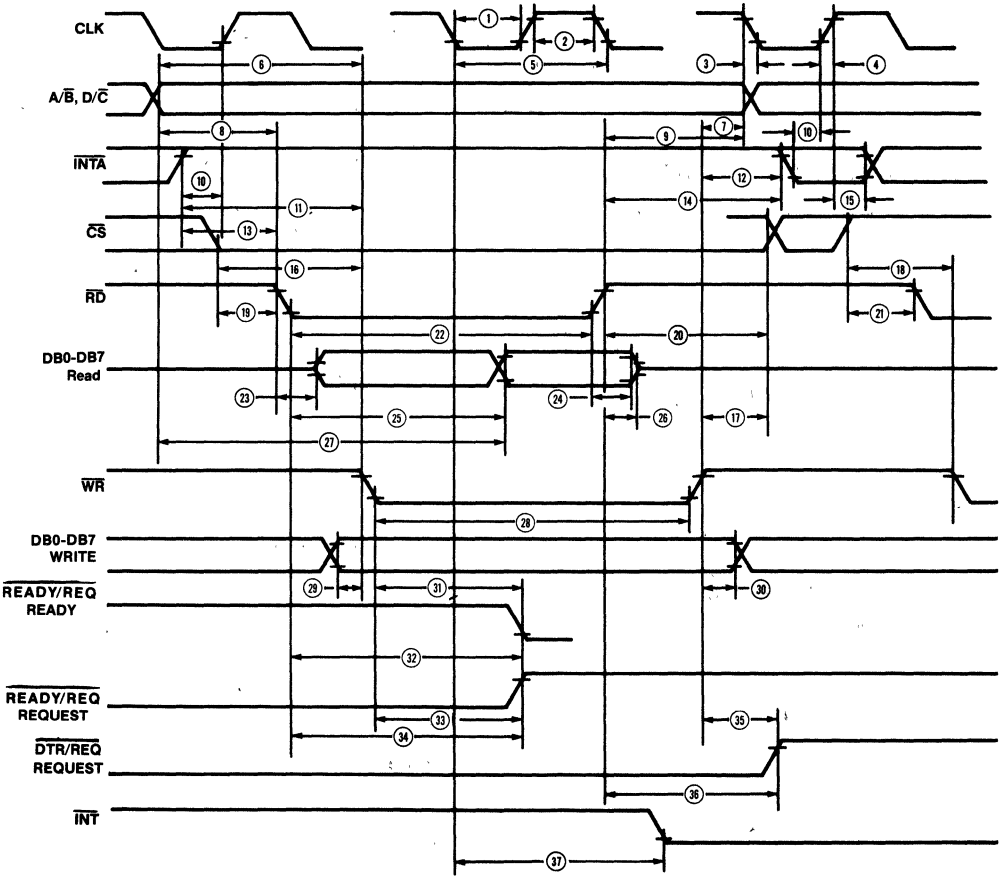


Figure 14. Read and Write Timing

INTERRUPT ACKNOWLEDGE TIMING, RESET TIMING, CYCLE TIMING

Number	Symbol	Parameter	82530 (4MHz)		82530-6 (6 MHz)		Units
			Min	Max	Min	Max	
27	tAD	Address Required Valid to Read Data Valid Delay		590		420	ns
28	TWW	$\overline{WR}$ Low Time	390		250		ns
29	tDW	Data to $\overline{WR}$↓ Setup Time	0		0		ns
30	tWD	Data to $\overline{WR}$↑ Hold Time	0		0		ns
31	tWRV	$\overline{WR}$↓ to Ready Valid Delay (Note 4)		240		200	ns
32	tRRV	$\overline{RD}$↓ to Ready Valid Delay (Note 4)		240		200	ns
33	tWRI	$\overline{WR}$↓ to $\overline{READY/REQ}$ Not Valid Delay		240		200	ns
34	tRRI	$\overline{RD}$↓ to $\overline{READY/REQ}$ Not Valid Delay		240		200	ns
35	tDWR	$\overline{WR}$↑ to $\overline{DTR/REQ}$ Not Valid Delay		5 tCY + 300		5 tCY + 250	ns
36	tDRD	$\overline{RD}$↑ to $\overline{DTR/REQ}$ Not Valid Delay		5 tCY + 300		5 tCY + 250	ns
37	tCIV	CLK↓ to $\overline{INT}$ Valid Delay (Note 4)		500		500	ns
38	tIID	$\overline{INTA}$ to $\overline{RD}$↓ (Acknowledge) Delay (Note 5)					ns
39	tII	$\overline{RD}$ (Acknowledge) Low Time	285		250		ns
40	tIDV	$\overline{RD}$↓ (Acknowledge) to Read Data Valid Delay		190		180	ns
41	tEI	IEI to $\overline{RD}$↓ (Acknowledge) Setup Time	120		100		ns
42	tIE	IEI to $\overline{RD}$↑ (Acknowledge) Hold Time	0		0		ns
43	tEIEO	IEI to IEO Delay Time		120		100	ns
44	tCEQ	CLK↑ to IEO Delay		250		250	ns
45	tRII	$\overline{RD}$↓ to $\overline{INT}$ Inactive Delay (Note 4)		500		500	ns
46	tRW	$\overline{RD}$↑ to $\overline{WR}$↓ Delay for No Reset	30		15		ns
47	tWR	$\overline{WR}$↑ to $\overline{RD}$↓ Delay for No Reset	30		30		ns
48	tRES	$\overline{WR}$ and $\overline{RD}$ Coincident Low for Reset	250		250		ns
49	tREC	Valid Access Recovery Time (Note 3)	6 tCY + 200		6 tCY + 130		ns

NOTES:
3. Parameter applies only between transactions involving the SCC.
4. Open-drain output, measured with open-drain test load.
5. Parameter is system dependent. For any SCC in the daisy chain, tIID must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for the SCC and tEIEO for each device separating them in the daisy chain.

*Timings are preliminary and subject to change.

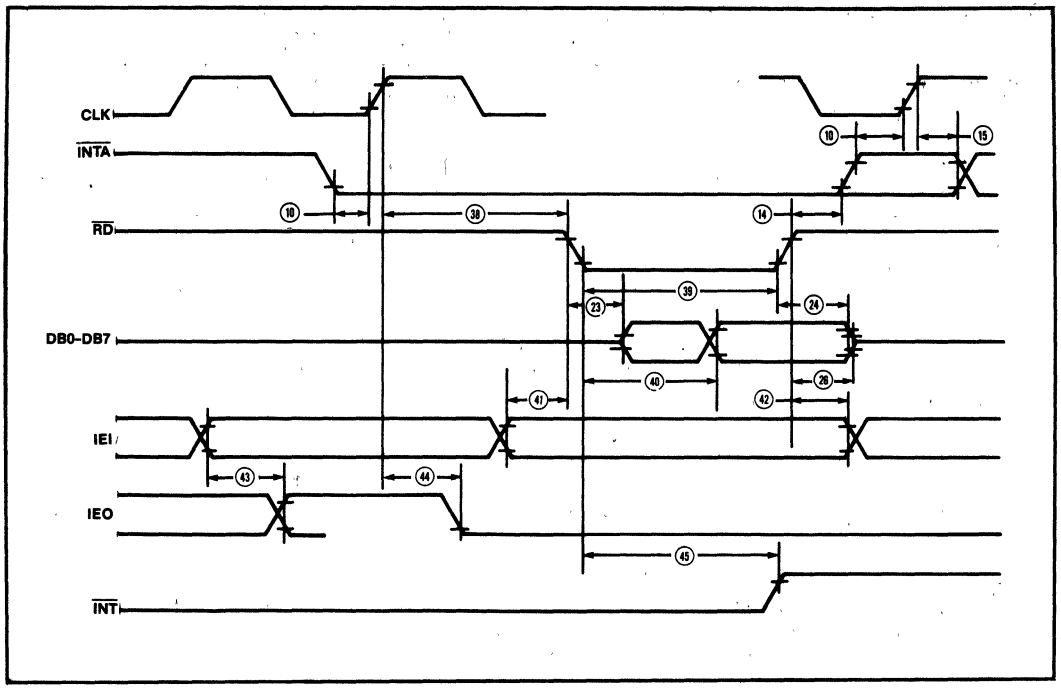


Figure 15. Interrupt Acknowledge Timing

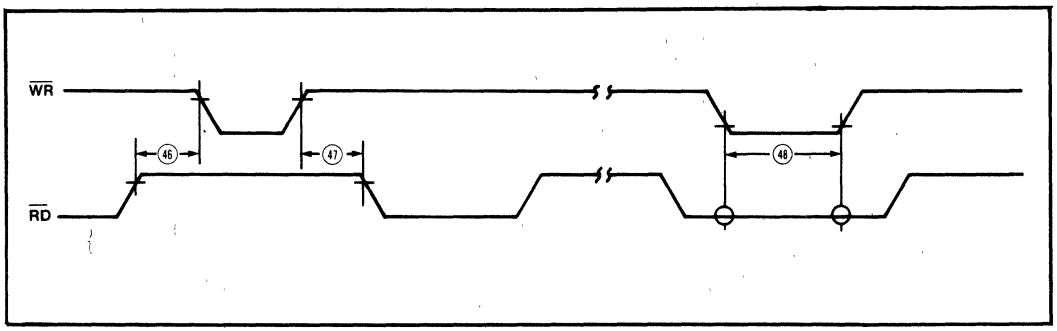


Figure 16. Reset Timing

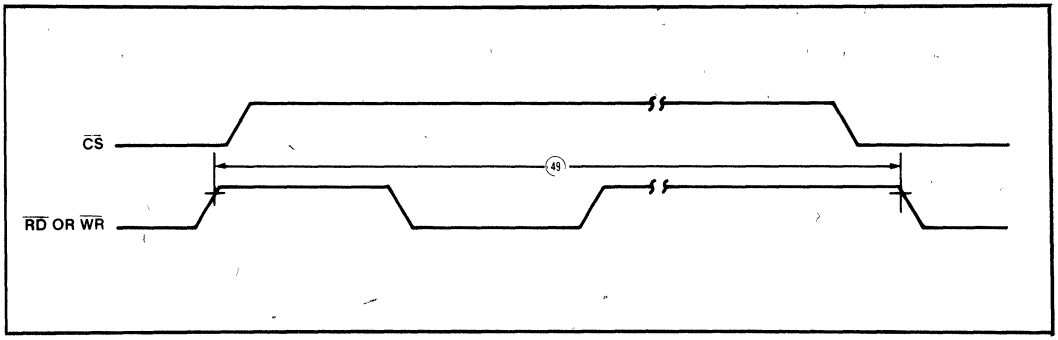


Figure 17. Cycle Timing

GENERAL TIMING

Number	Symbol	Parameter	82530 (4MHz) Min	82530 (4MHz) Max	82530-6 (6 MHz) Min	82530-6 (6 MHz) Max	Units
1	tCRV	CLK↓ to $\overline{\text{READY}}$/$\overline{\text{REQ}}$ Valid Delay		TBD		TBD	ns
2	tCRI	CLK↓ to Ready Inactive Delay		350		350	ns
3	tRCC	$\overline{\text{RxC}}$↑ to CLK↑ Setup Time (Notes 1,4)	50		50		ns
4	tRRC	RxD to $\overline{\text{RxC}}$↑ Setup Time (X1 Mode) (Note 1)	0		0		ns
5	tRCR	RxD to $\overline{\text{RxC}}$↑ Hold Time (X1 Mode) (Note 1)	150		150		ns
6	tDRC	RxD to $\overline{\text{RxC}}$↓ Setup Time (X1 Mode) (Notes 1,5)	0		0		ns
7	tRCD	RxD to $\overline{\text{RxC}}$↓ Hold Time (X1 Mode) (Notes 1,5)	150		150		ns
8	tSRC	$\overline{\text{SYNC}}$ to $\overline{\text{RxC}}$↑ Setup Time (Note 1)	-200		-200		ns
9	tRCS	$\overline{\text{SYNC}}$ to $\overline{\text{RxC}}$↑ Hold Time (Note 1)	3 tCY + 200		3 tCY + 200		ns
10	tTCC	$\overline{\text{TxC}}$↓ to CLK↑ Setup Time (Notes 2,4)	0		0		ns
11	tTCT	$\overline{\text{TxC}}$↓ to TxD Delay (X1 Mode) (Note 2)		300		300	ns
12	tTCD	$\overline{\text{TxC}}$↑ to TxD Delay (X1 Mode) (Notes 2,5)		300		300	ns
13	tTDT	TxD to $\overline{\text{TRxC}}$ Delay (Send Clock Echo)					ns
14	tDCH	$\overline{\text{RTxC}}$ High Time	180		180		ns
15	tDCL	$\overline{\text{RTxC}}$ Low Time	180		180		ns
16	tDCY	$\overline{\text{RTxC}}$ Cycle Time	400		400		ns
17	tCLCL	Crystal Oscillator Period (Note 3)	250	1000	250	1000	ns
18	tRCH	$\overline{\text{TRxC}}$ High Time	180		80		ns
19	tRCL	$\overline{\text{TRxC}}$ Low Time	180		180		ns
20	tRCY	$\overline{\text{TRxC}}$ Cycle Time	400		400		ns
21	tCC	$\overline{\text{CD}}$ or $\overline{\text{CTS}}$ Pulse Width	200		200		ns
22	tSS	$\overline{\text{SYNC}}$ Pulse Width	200		200		ns

NOTES:
1. $\overline{\text{RxC}}$ is $\overline{\text{RTxC}}$ or $\overline{\text{TRxC}}$, whichever is supplying the receive clock.
2. $\overline{\text{TxC}}$ is $\overline{\text{TRxC}}$ or $\overline{\text{RTxC}}$, whichever is supplying the transmit clock.
3. Both $\overline{\text{RTxC}}$ and $\overline{\text{SYNC}}$ have 30pF capacitors to ground connected to them.
4. Parameter applies only if the data rate is one-fourth the system clock (CLK) rate. In all other cases, no phase relationship between $\overline{\text{RxC}}$ and CLK or $\overline{\text{TxC}}$ and CLK is required.
5. Parameter applies only to FM encoding/decoding.

*Timings are preliminary and subject to change.

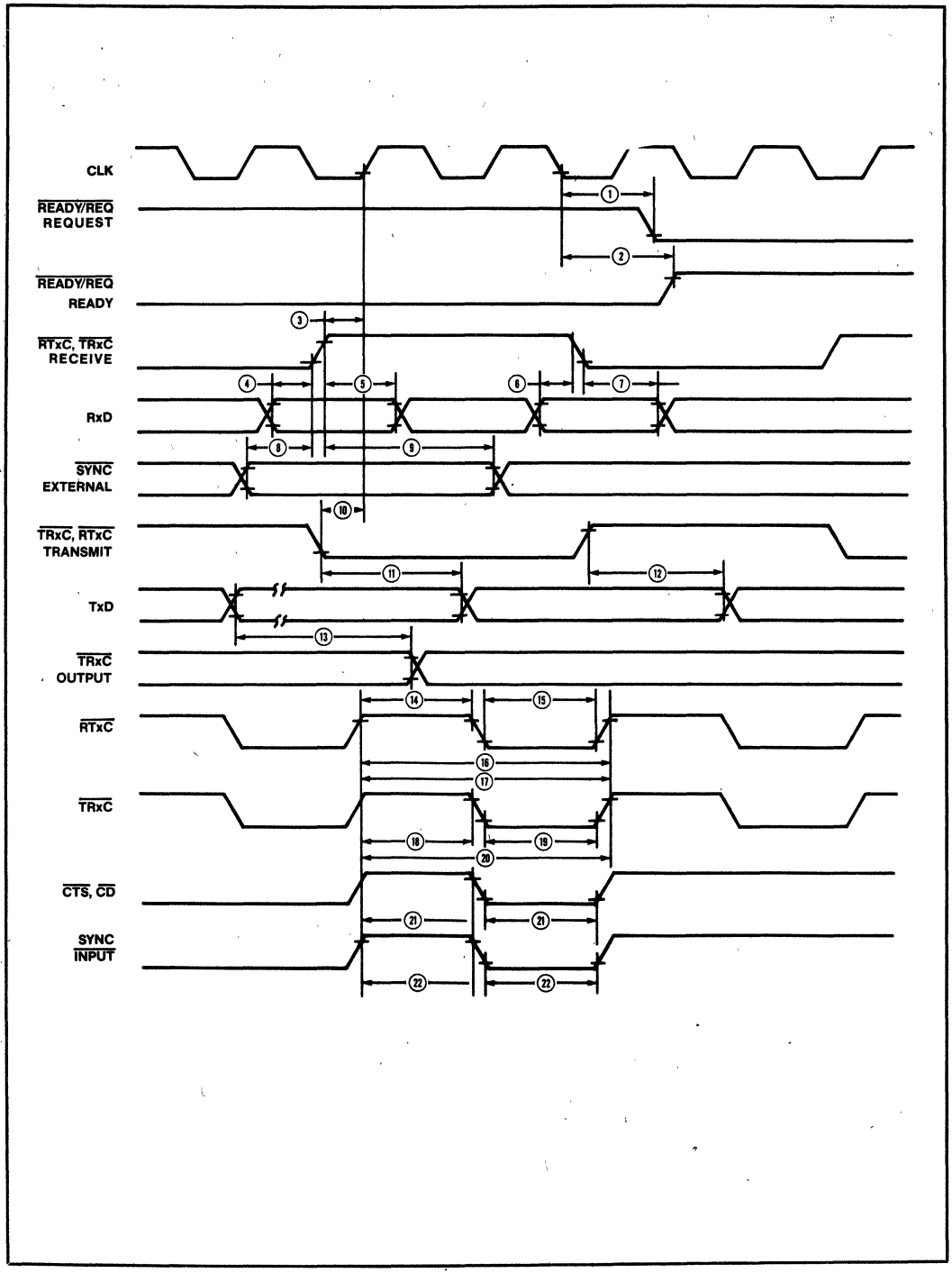


Figure 18. General Timing

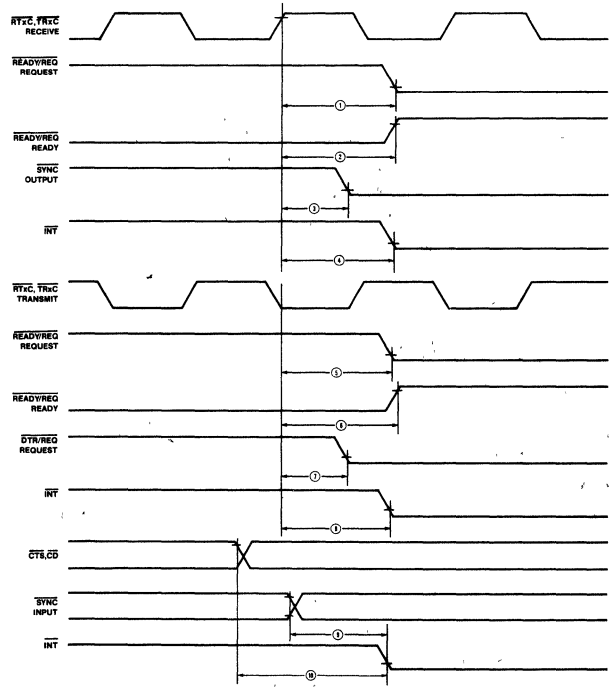
230834-001

SYSTEM TIMING

Number	Symbol	Parameter	82530 (4 MHz) Min	82530 (4 MHz) Max	82530-6 (6 MHz) Min	82530-6 (6 MHz) Max	Units
1	tRRV	$\overline{RxC}$ ↑ to $\overline{READY}/\overline{REQ}$ Valid Delay (Note 2)	TBD	TBD	TBD	TBD	tCY
2	tRWI	$\overline{RxC}$ ↑ to Ready Inactive Delay (Notes 1,2)	TBD	TBD	TBD	TBD	tCY
3	tRSC	$\overline{RxC}$ ↑ to $\overline{SYNC}$ Valid Delay (Note 2)	4	7	4	7	tCY
4	tRIV	$\overline{RxC}$ ↑ to $\overline{INT}$ Valid Delay (Notes 1,2)	TBD	TBD	TBD	TBD	tCY
5	tTRV	$\overline{TxC}$ ↓ to $\overline{READY}/\overline{REQ}$ Valid Delay (Note 3)	TBD	TBD	TBD	TBD	tCY
6	tTWI	$\overline{TxC}$ ↓ to Ready Inactive Delay (Notes 1,3)	TBD	TBD	TBD	TBD	tCY
7	tTDV	$\overline{TxC}$ ↓ to $\overline{DTR}/\overline{REQ}$ Valid Delay (Note 3)	TBD	TBD	TBD	TBD	tCY
8	tTIV	$\overline{TxC}$ ↓ to $\overline{INT}$ Valid Delay (Notes 1,3)	TBD	TBD	TBD	TBD	tCY
9	tSIV	$\overline{SYNC}$ Transition to $\overline{INT}$ Valid Delay (Note 1)	TBD	TBD	TBD	TBD	tCY
10	tCTI	$\overline{CD}$ or $\overline{CTS}$ Transition to $\overline{INT}$ Valid Delay	2	6	2	6	tCY

NOTES:
1 Open-drain output, measured with open-drain test load
2 $\overline{RxC}$ is $\overline{RTxC}$ or $\overline{TRxC}$, whichever is suppling the receive clock
3 $\overline{TxC}$ is $\overline{TRxC}$ or $\overline{RTxC}$, whichever is supplying the transmit clock

*Timings are preliminary and subject to change



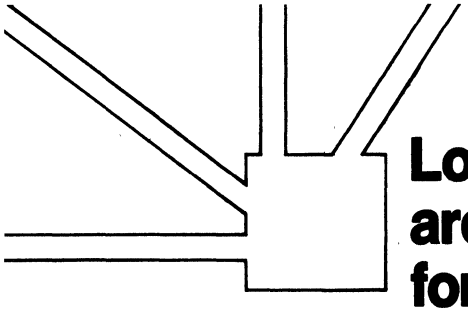
230834-001

November 1981

Local Network Architecture Proposed For Work Stations

By Robert Ryan, George Marshall, Robert Beach and Steve Kerman

Order Number - 210262-001



Local network architecture proposed for work stations

General-purpose standard compatible with Ethernet will serve many applications at a wide range of performance levels

by Robert Ryan, George Marshall, Robert Beach, and Steve Kerman, *Intel Corp., Santa Clara, Calif.*

☐ Computer-based communicating work stations and microprocessor development systems, which promise to usher in an era of electronic offices and workplaces, will need to be attached to local networks through a standardized architecture in order to be cost-effective. In response to the lack of such a standard, Intel has come up with a local network architecture that is currently geared to work stations and development systems based on Intel microprocessors. Called iLNA, the proposed network takes advantage of the work already done in association with Digital Equipment Corp. and Xerox Corp. by using the Ethernet local network design as the basis for its own data-carrying scheme.

What has been proposed is a six-layer architecture combining software and hardware that will expedite all local network functions. Its goal is simply efficient, flexible communication between users and application programs, application programs and resources, and any other combination of users, programs, and resources within the local network.

The concept of a layered architecture is not new. Indeed, IBM's well-known Systems Network Architecture is layered, as is the forthcoming International Standards Oganization and American National Institute Reference Model of Open Systems Interconnection. What is new is the fact that a network architecture has been specifically designed for Intel- and Ethernet-based equipment (see "Specifying the network," p. 122). If eventually accepted as an industry standard, the proposal will become the basis for future network architectures, and manufacturers of equipment that hooks up to local networks will want their equipment to be compatible.

In developing a local network architecture, the primary goal has to be achieving cost-competitiveness with any general-purpose network design, while at the same time equaling the efficiency and performance of a network designed for a specific application. Likewise, the network has to facilitate communication through commonly used interfaces but not be bound by any one topology or internal communication mechanism. In addition, it has to function independently of any particular computer's operating system or hardware.

The network also has to act as an error-free message-delivery medium between communication processes (programs resident in equipment attached to the network) and permit an operator or program to monitor, maintain, and modify network operations. While performing all these chores, the network must also be able to serve low-cost, low-performance equipment and incorporate future technology. In addition, the failure of any device at a work station should have minimal effect on the operation of other work stations.

To perform all these tasks, the Intel network architecture defines a set of interfaces, algorithms, and protocols by which application programs on various kinds of Intel microprocessor-based work stations can communicate. It also establishes a process-to-process communication mechanism whereby a process (any application, function, or peripheral using the network) is defined as the active element in a communicating node and the ultimate source or destination for data. Thus, for example, terminals, files, and input/output devices can communicate with one another through the use of processes. Messages are sent and received by the designated processes through what is termed a communications socket, which is a hierarchical address composed of three unique identifiers—one each for the local network, the host, and the port to a process.

Each node in the network, which may consist of one or more pieces of equipment, has a unique host identifier that distinguishes it from all other nodes installed anywhere, to ensure eventual communications between equipment on various local networks. Within each node, each process is given a local address, or port identifier. The binding of ports to processes is the responsibility of the node, and the ports remain unique within each node. Certain ports, however, may be assigned numbers in accordance with a globally consistent scheme.

Each installed local network will be given a unique identifier, its network identifier, that identifies it in multiple-network applications. In a single-network application, the network identifier is not used, but its assignment assures that an orderly progression to an internetworking environment is possible.

In designing its architecture, Intel examined applications needs of its users and chose a suitable set of

interconnect functions to serve them. These functions were then defined in a series of layers that permit the network to achieve high performance across a wide applications base. The architecture is divided into six layers (Fig. 1). The ones of interest here are the physical-link, data-link, transport, session, and network-management layers. The network layer is used when one local network must be connected to another.

The lowest-level means of sending data from one node to another, the physical-link layer, is responsible for delivering the smallest unit of data (the bit) the network handles. This layer is the one directly concerned with the transmission medium, signal type, data rate, and mechanical interconnect specifications of the network. It can be implemented using two modems, two telephone sets, and a telephone line or using coaxial cable, a baseband line driver, receiver chips, and a universal synchronous-asynchronous receiver-transmitter (Usart).

Moving from node to node

While the physical link moves data bits from one node to another, it cannot guarantee successful transmission. Electrical noise in the environment causes errors, although some transmission media are less susceptible than others. For example, the error rates generally run between 1 bit-error per 10,000 bits and 1 bit-error per 100,000 bits for transmissions over a modem-telephone network, but can be less than 1 bit in 10 million for local coaxial-cable–based networks.

Error rates can be kept quite low at the physical link level if the network designer is willing to properly locate and shield the network cabling from rf interference by other electrical utilities in, say, a building. However, the higher-level layers are a better place to reduce errors because they can exploit such multiple-bit schemes as redundancy codes and automatic repeat requests that are not available at the physical link level.

In the Ethernet physical link, data is transmitted on a 50-ohm coaxial cable that is up to 500 meters long per segment. The Manchester-encoded, baseband signal carries data at a rate of 10 megabits per second. At the start of a transmission, a 64-bit preamble is used to stabilize and synchronize the communication channel circuitry. After reception, the preamble is removed and only the Ethernet header and data are passed on.

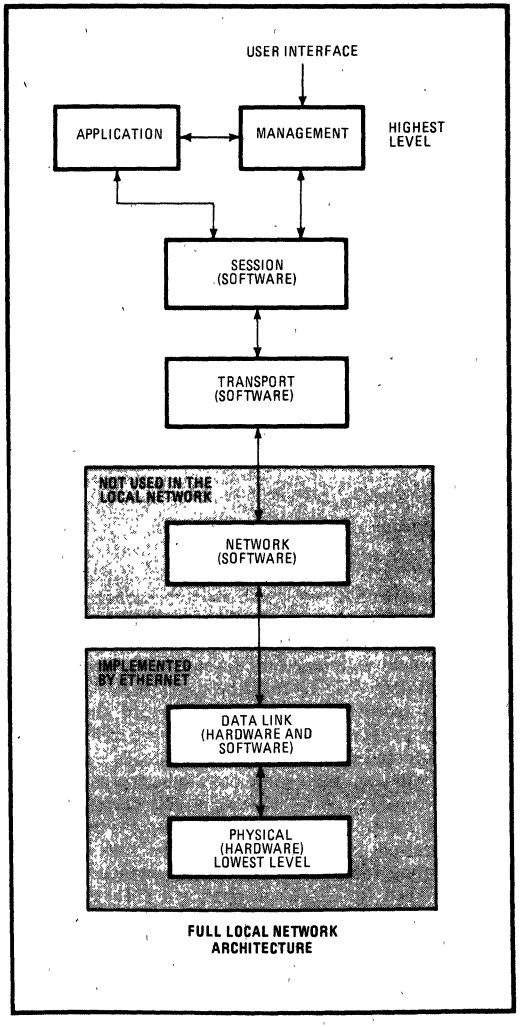
Packet-delivery service

The data-link layer makes possible a node-to-node packet delivery service. As such, it is the first step toward a process-to-process packet delivery system. The data link supplies some of the services missing from the physical link. Among others, it is responsible for framing, or the determination of where a message begins and ends; addressing, or the determination of which station should receive a message; error detection, or the determination of bit errors in the packet; and link management, which controls the access of multiple transmitters and receivers to the physical link.

A data link may deliver all the packets error-free by using various error-correcting protocols. Or it may provide, as the Ethernet data-link architecture does, a best-effort delivery service in which not all packets are delivered, but all those that are, arrive unmodified. With error-free packet delivery all packets are delivered (no lost packets), all packets are delivered just once (no duplicate packets), and all packets are received in the order sent (no nonsequential packets). However, when error-free packet delivery is required, data-link error control is necessary to perform packet sequencing and retransmission. In addition, besides having the higher-level error-coding alternatives previously noted, data-link error control also is expensive and, given the physical-link error rates, not cost-effective.

Data-link error control might provide a reliable node-

1. Layers. In the Intel local-network architecture, there are six levels of hardware and software, with the network layer omitted in strictly local (non–store-and-forward) configurations. The physical link and data link contain hardware; the others only software.



Specifying the network

The Digital Equipment–Intel–Xerox specification for the Ethernet network is the first portion of Intel's forthcoming local network architecture, and the first hardware produced for this architecture will be the Ethernet intelligent controller. The two-board set, which plugs into an Intel Multibus chassis, supplies many of the functions of the physical- and data-link layers of the network architecture.

The data-link functions performed are framing (including packet-boundary delineation and address recognition), link management (including transmission scheduling and retries in case of a collision between packets), and error detection. The physical-link functions performed are preamble generation and removal and bit encoding and decoding. The set also handles a number of system-oriented functions, such as interfacing with the system parallel bus, communicating with the central processing unit, handling data movement to and from the buffers, and interfacing with the transmitter-receiver units.

The board hardware consists of an Intel 8086 5-megahertz microprocessor with local random-access- and read-only memory, direct-memory-access channels for sending and receiving data at the required 10 megabits per second, bit-serial send-and-receive logic, packet address-recognition logic, error detection logic, and interval timers. One board contains the microprocessor, memory, timers and DMA control; the other contains the serial send-and-receive and error-detection logic.

The boards implement part of the data-link layer and also contain seven major software functions. These include the executive (or scheduler), the rest of the data-link software, transport control, session control, network management, the bootstrap, and diagnostics. Typically these software functions are implemented with programs that occupy small amounts of memory space.

to-node delivery service, but it does not ensure a reliable end-process–to–end-process delivery service. That is particularly true in any internetworking environment where two or more local networks are connected and there are multiple gateways (the physical and software connections) acting as packet forwarders. The risk of packet nondelivery then is moderate to high. In addition, end-to-end delivery retransmission (error control) would still have to be performed at the transport layer, making error control at the data-link layer redundant.

Collision insurance

The data-link software supports a large address space—up to a 48-bit destination identifier and a 48-bit source identifier—to permit flexibility in managing internetwork gateways. In operation, data-link users must supply both transmit requests and standby receive-buffers to the network. The transmit requests contain the address of the destination nodes and the data to be sent. The data link combines both into a packet that is transmitted when the line becomes idle.

Should multiple nodes transmit concurrently, they all abort their transmissions, generate a jam signal that reinforces the initial collision signal, wait a random interval before retransmission to avoid repeated collisions, and then try again. The average retransmission interval increases as a function of channel load in order to achieve channel stability under overload conditions.

On the receiving side, the intended packets are recognized by the data link, which performs a 32-bit cyclic redundancy check. If the packet is good, it is placed in an empty receive buffer. A packet that has collided is recognized as such and dropped.

As noted earlier, the data link supports framing, addressing, error detection, and link management. In the Intel Ethernet approach to framing, a carrier-sense function determines the end of a packet. When the carrier is lost, the packet is finished. The two-bit beginning-of-packet indicator at the end of the preamble actuates carrier sensing.

The address scheme permits a received packet to be accepted by any number of nodes. The data link recognizes single-host, broadcast, and multicast addresses. The first bit within the destination address distinguishes between single-host and multicast delivery, and the next 47 bits determine the multicast group identifier. Broadcast addressing is simply a special case of multicast in which the next 47 bits are all logical 1s.

The link management function controls line access when two or more nodes attempt to transmit data simultaneously through an arbitration policy called carrier-sense multiple access with collision detection. With this system, when a packet is to be sent, the link management facility determines if another carrier is present. If this is so, or if the interpacket gap time has not expired, the waiting packet is not released onto the line. When the data packet is finally transmitted, the link management function monitors the line to determine whether a collision has occurred. If a collision is detected, the random waiting period for retransmitting the packet is chosen by executing what is known as a truncated binary exponential back-off algorithm.

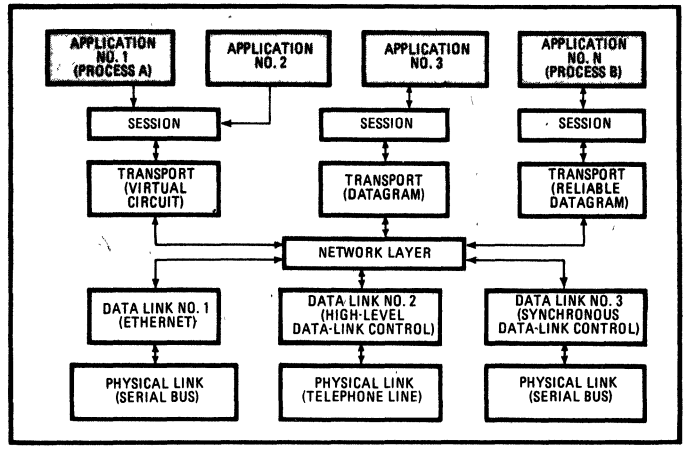
Reliable transport

The transport layer software (there is no hardware in this layer) makes possible location-independent, reliable packet transmission. Users of this layer can establish, maintain, and terminate virtual circuits, which represent full-duplex data paths between sockets.

A virtual circuit is defined by its basic properties. First, it permits multiple virtual connections to exist between processes. Second, it can be dynamically managed by the communicating processes. Third, it can accommodate message lengths that are independent of transport communication. Finally, it transmits data in a full-duplex error-controlled and flow-controlled format.

While the data-link layer makes a best-effort attempt to move individual packets from one physical node to another, the transport layer is responsible for reliably moving a user's variable-length message, such as a file transfer, from one process to another, even though the underlying packet delivery service will occasionally drop packets, duplicate packets, or deliver them out of order. A secondary responsiblity of the transport layer is to

2. Extensions. The six-layer local network architecture can be extended to include remote network configurations by simply adding the network layer and the new data links. These new configurations can be co-located or geographically dispersed.



prevent fast transmitters from swamping slow receivers. It also must ensure that the network's communication subsystem resources (primarily media bandwidth, communications processor usage, and communications buffer memory) not be wasted in frequently retransmitting packets when there is a speed mismatch. Both are accomplished by a flow control function that throttles fast transmitters when the receiver cannot keep pace.

The transport software serves several other functions as well. Since the transport layer should insulate user software from the limiting characteristics of the underlying physical network, it performs fragmentation and reassembly services that let the user software send arbitrarily long messages over the network. To accomplish this, the transmitting transport software breaks messages into packet-sized chunks and the receiving software then reassembles them.

Acknowledge and over

In order to provide its services, the transport software carefully manages the user's service requests and the packets exchanged on the data link. For example, the transport software associates a unique sequence number with every packet it sends. Likewise, the receiving transport software sends back acknowledgment packets, indicating with the sequence number which packets have been correctly received and accepted. Packets not acknowledged within a specified time are automatically retransmitted by the sender.

The transport software controls the data flow by exchanging information on the amount of receive-buffer memory that each claims to have available. The amount of buffer memory available is called a window, and a receiver that has indicated it has a large amount of receive buffer space is said to have its window wide open.

Open window

If the transmitter has several data packets, they will be delivered much faster if the receiver has sufficient buffer space and has opened its window than if the window is small and requires an exchange of window information after each packet is sent. To expedite the information exchange, the transport software uses a combined error- and flow-control algorithm that permits both functions to work at the same time. For process-to-process addressing, the transport software adheres to the standard network-address structure, which consists of the network, host, and port identifiers.

The session control software layer identifies and locates process names within the network. In order to communicate, a process using the transport layer in one node must know the socket of other processes. Since, it is unlikely that the naming convention for processes under a given computer's operating system conforms with that used in another, the session layer resolves this problem through a location-independent scheme known as a binding function, which provides users with standard-format, location-independent names for remote processes they must access.

Ties that bind

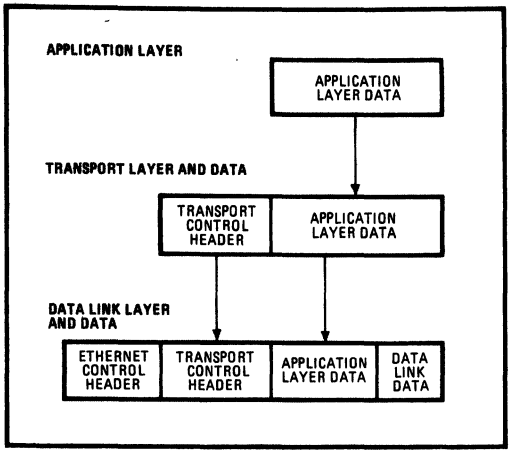
The binding function is composed of two operations—mapping and updating. Mapping is the function that, on demand from the user software, translates between process names and sockets. Updating distributes the mapping information throughout the network so that it is available when needed at each node.

The session software also supplies network status information to the application software. In turn, the transport software gives the session layer status information on its best estimate of the quality of the underlying network layers. However, the decision to abort a connection is left to the user for all but the most extreme cases, such as evidence of total equipment failure.

Network management

The network-management software layer provides the user with all those functions not required for normal operation. In addition, it includes diagnostic utilities for accessing the network components when any portion of the network fails. It also has maintenance tools that gauge the performance of various network components so users can plan for changing network demand.

Network management functions fall into one of three



3. Headers. If two processes on two different local network nodes want to communicate, the session layer software establishes a virtual circuit between them. The transport- and data-link layers add headers for identification, addressing, and control.

categories: operation, maintenance, or planning. The operation category includes all functions that are performed on a day-to-day basis as part of normal network operation. A major goal of the Intel network architecture has been eliminating the full-time network operator, and thus only the network bootstrap and the manual operations needed to add a new node to the network are included in the management layer.

The network bootstrap is the operational function used by a booting node to load its operating system from another network node. The bootstrap sequence begins when the booting node transmits a multicast packet addressed to any node that has a copy of the operating system and is willing to send it. If such a node exists on the local Ethernet data link, it will respond.

Should more than one node reply, the booting node will accept the first reply and ignore all others. If no reply is received, as would happen if either the request or reply is lost in the network because of line noise, the booting node will retransmit the request. If a reply still is not received after several retries, the bootstrap attempt will be aborted.

Preventive maintenance

The maintenance category detects failures in the network, even though it may be uncertain of exactly what the problem may be. Problem detection proceeds through three mechanisms. The first is a set of error counters made possible by the management layer; the second is an error-reporting and -logging mechanism; and the third is user observation.

The first problem detection mechanisms, the error counters, are maintained by the individual layers and record occurrences of recoverable errors. The presence of errors does not necessarily indicate a failure in that the layers are designed to operate normally in the face of a large number of errors. An excessive number of errors, however, may indicate that a problem is developing.

Since this set of counters is maintained at each node in the network, and since the nodes can be spread over a large area, the network management layer includes a remote examination function for interrogating nodes without interfering with network operations. The network management layer in a node desiring information from a remote node first sends a request to the network management layer in the remote node. The management layer in that remote node then performs the desired function and transmits a response to the requesting node.

Isolating errors

The error-counting mechanism is supplemented by an error-reporting mechanism that logs problems detected by the communication system to an error-logging file. Once a problem has been detected, it is isolated to some serviceable component through two mechanisms. First, the same error counters are used to isolate the error. Second, the management layer generates test traffic, including a loopback function within each layer, and observes the behavior of the system.

Generally, correcting the problem involves repairing or replacing hardware. Some problems, however, can be corrected simply by reinitializing a system component. In that case, the management layer can stop and reinitialize each layer.

In its planning function, the management layer supplies the network administrator with statistical information about the use of the network to help in planning network growth.

By way of example

To illustrate the operation of the software and hardware layers with a practical example, consider a case in which there are two processes, A and B, that reside on two different nodes (Fig 2). Application process A's request to communicate with process B on some remote node requires the cooperation of the communication layers of each node.

The source node's session layer first determines that process B resides at socket n, thus pinpointing process B to a specific port residing in a specific node on a specific network through the port identifier. By means of the transport interface, the session layer then attempts to create a virtual circuit between the source port and the destination port. Assuming there are no conflicts on the network, the virtual circuit is established after the two transport-layer sites exchange connection information.

The two processes can now send or receive over the virtual circuit so that data can be delivered in order, unmodified, and without duplication. The transport layer adds a transport header that includes the virtual circuit identifier and a sequence number to each piece of data it handles. It then passes the data, transport header, and application data to the Ethernet data link.

The data link adds its header (Fig. 3), consisting of the address (destination and source identifiers), framing, and error-detection bits, and it then attempts to transmit the packet. Once the data-link has established the carrier signal, the physical link is reponsible for the transmission of the bits over the serial link. □

PRINTED IN USA/T-2081/10K/1181/BL DG

intel

ARTICLE REPRINT

AR-237

October 1982

System-Level Functions Enhance Controller IC

Robert Beach and Robert Gallin
Intel Corporation
Santa Clara, California

Alex Kornhauser, Moshe Stark and Domo Van-Mierop
Intel Israel Ltd.
Haifa, Israel

"Reprinted from ELECTRONICS, October 6, 1982.

ORDER NUMBER: 210788-001

System-level functions enhance controller IC

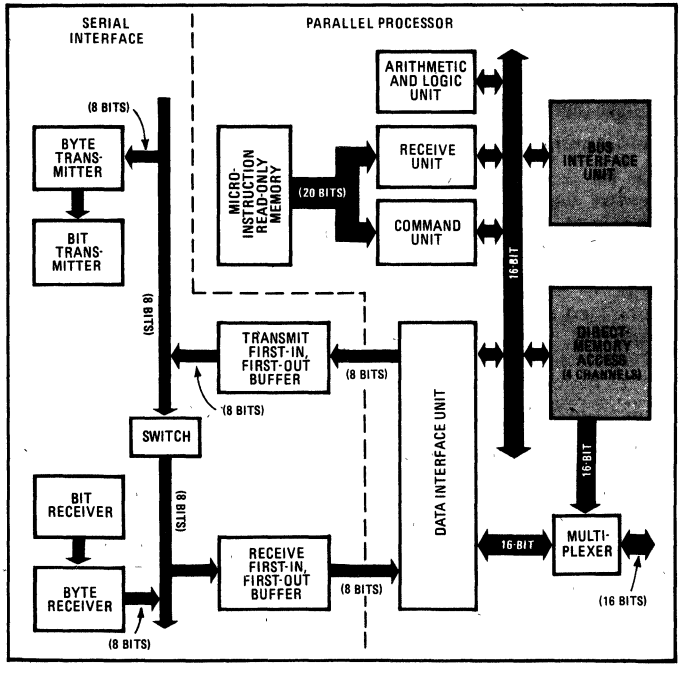
by Robert Beach and Robert Galin
Intel Corp., Santa Clara, Calif.
and Alex Kornhauser, Moshe Stark, and Dono Van-Mierop
Intel Israel Ltd., Haifa, Israel

Beyond any single new feature, it is the integration of major system-level communications functions onto a single chip that makes the 82586 local area network communications controller a true next-generation communications controller for high-speed local nets. Such functions as on-chip control of direct memory access, buffer-memory management, programmable network parameters, and diagnostics will allow designers to quickly implement cost-effective and reliable Ethernet and local nets using similar other carrier-sense, multiple-access protocols with collision detection (CSMA/CD).

Combined with the 82501 Ethernet serial interface chip and readily available transceivers, users will have a complete implementation of the Ethernet physical and data links. Although other Ethernet controller integrated circuits will also handle the fundamental implementation of these two International Standards Organization layers, the 82586 goes beyond them to offer programmable network-management capabilities that permit users to optimize the controller's operation for a variety of local networks and to gage the net's health.

In fact, Intel's goal in designing the 82586 is to serve, not only the Ethernet user, but any net that uses some form of CSMA/CD. Therefore, many of the IC's facilities are programmable for nets with different maximum lengths and data-transfer rates from those found in Ethernet (see p. 90).

A major role for the controller IC is to act as an intelligent interface with the host central processing unit, reducing its workload and saving memory space. The chip may be viewed as a parallel processor (on the right in Fig. 1), fetching and executing commands from the host at the same time it is receiving data through its serial-interface circuitry and storing it in buffer memory.



1. Peek inside. Intel's H-MOS data-link-control chip has both parallel and serial interfaces and four-channel direct-memory access. It can operate in a multiplicity of local networks because its key parameters are programmable.

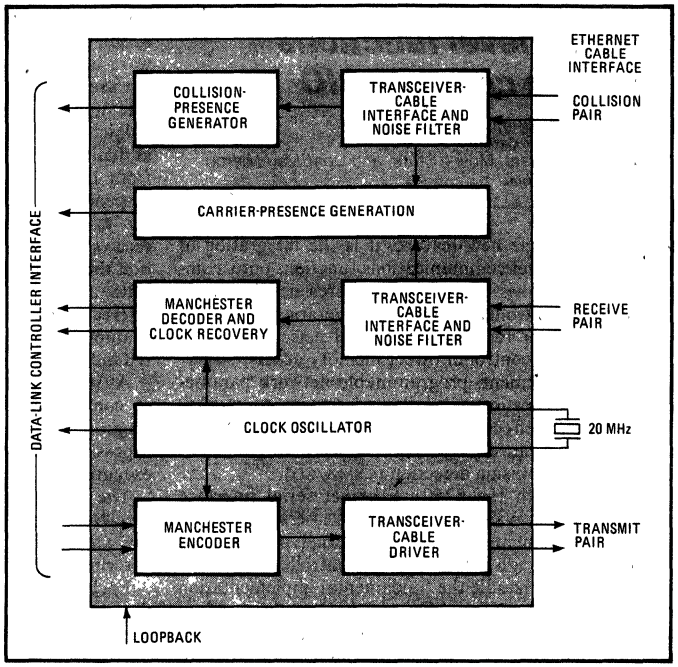
Communications between the host's CPU and the 82586 is by means of a shared memory. The only hardware interconnections are the interrupt line the controller uses to get the CPU's attention and the channel-attention line the CPU uses to get the 82586's attention.

Part of the shared memory is reserved as a bidirectional mailbox. One section of the mailbox holds instructions from the CPU to the controller, such as start, abort, suspend, and resume, plus pointers to a list of commands for execution by the parallel processor and to the received-frame area. The second section holds information the 82586 is sending to the CPU, such as status data (idle, active, no receive resources available, and so on) interrupt bits (command completed, frame received, for example), and accumulative tallies (such as cyclic-redundancy-check errors).

As well as a mailbox, the shared memory holds the list of commands prepared by the CPU that serve as the program for the 82586. The linked-list approach makes it possible to form a circular linked list used for repeated execution or a linear queue of commands.

The final section of the shared memory is the received-frame area. All the host CPU need do is identify the area by preparing two linked lists: one of frame descriptors and one of buffers with their descriptors.

Each frame descriptor has a forward pointer. The first descriptor is referenced by the mailbox and the last one is marked with an end-of-frame bit. The buffer descriptors are essentially the same for both the receive and the



2. Partners. The bipolar 82501 Ethernet serial interface chip provides Manchester encoding and decoding, noise filtering, transceiver drive signals, and collision detection as it works with the data-link control chip and the network transceiver.

transmit processes; however, the receive descriptors include a field that specifies the size of the empty buffer and an end-of-list bit.

The 82586 fills the buffers upon reception of frames and reformats the free-buffer list. Receive-buffer chaining improves memory use significantly. Without it, the host must allocate blocks of memory under the assumption that each frame will be the maximum size (1,518 bytes for Ethernet). Successive transmission may fill the buffers, even though the actual frames are far less than the maximum in size, and the controller may receive a burst of several frames but have no room. Usually, the tradeoff in buffer chaining is the processing overhead and the time for buffer switching. The 82586, however, performs the buffer chaining without CPU intervention.

Made in the high-performance MOS (H-MOS) process, the controller chip has over 56,000 devices and fits in a 48-pin dual in-line package. Besides the parallel processor, it has another major functional block, the serial interface (left in Fig. 1).

Internal architecture

On the parallel-processor side, the bus-interface unit generates bus-control signals to transfer data, commands, and status between shared memory and the 82586. The data-interface unit is a switch routing the data from the system bus to the transmit first-in, first-out buffer or the internal parallel bus and from the receive FIFO buffer to the internal parallel bus or to the system bus.

The direct-memory-access logic is an address generator that performs DMA transfers between the 82586 and the shared memory. Commands are fetched from memory by the command unit, which also writes status information to the memory. The command unit has full control over the DMA unit, loads the starting pointers and byte counts, and then triggers the DMA start.

The receive unit performs tasks for the receive memory operation similar to those that the command unit performs for the command operation. Both units fetch microinstructions from a shared read-only memory.

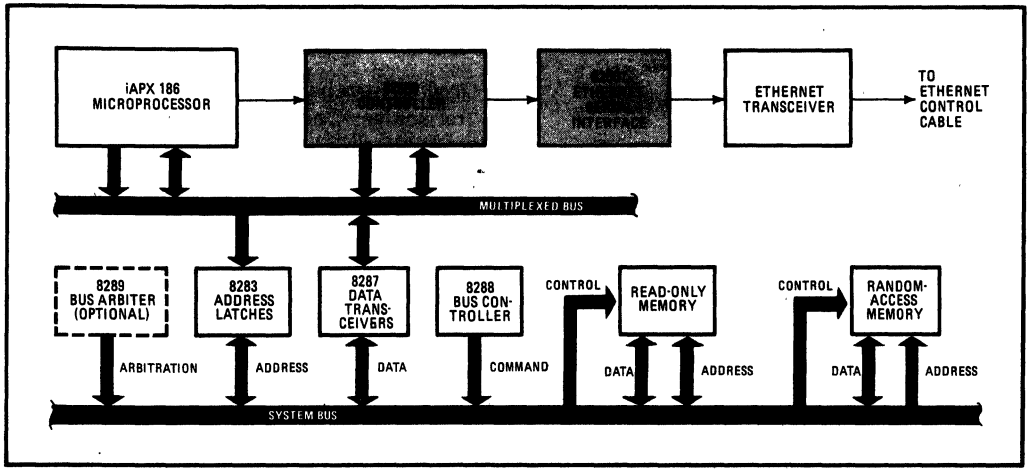
The transmit buffer regulates the traffic flowing from the parallel processor through the data-interface unit to the byte transmitter. After executing the commands coming from the transmit buffer, the byte transmitter sends status information back through the receive buffer.

The bit transmitter serializes and encodes data, generates the frame-check sequence, and transmits the data. It also controls the modem-like handshake. The bit receiver handles preamble stripping, address matching, error-flag generation, received-frame delineation, and frame-check sequence testing. It deserializes the information and delivers it in bytes to the byte receiver, which compares the destination address with the various possible address types. Then, if the address matches, it transfers the received data to the receiver buffer.

The controller interface is not complete without the 82501 Ethernet serial interface (ESI) chip. The 82501 is implemented in bipolar technology and is designed to handle the serial transmission and reception of 10-megabit-per-second packets to and from the transceiver.

The 82501 (Fig. 2) provides clock generation for itself and the 82586 controller, retiming and Manchester encoding of the transmitted data stream, driving of the transmit signal line to the transceiver, and noise filtering of the receive and collision inputs. What's more, it handles timing recovery and Manchester decoding of the received data stream and supplies receive-data, receive-clock, carrier-presence, and collision-presence signals.

Because of its four on-chip DMA channels, the 82586 can receive back-to-back bursts of frames, provided the minimum interframe spacing of 9.6 microseconds (for the 10-Mb/s Ethernet) is met. In addition, the pipelining



3. Complete system. A typical Ethernet local-network controller includes the Intel controller and encoder-decoder chips, a microprocessor, a transceiver, and auxiliary logic to connect the system to the work-station bus.

of the operation of the Ethernet interface and the host interface, plus the concurrent processing units, contribute to its performance.

The controller can operate with high-performance system buses, yet it is highly tolerant of system-bus limitations. The minimum data-transfer rate required to sustain a bit rate of 10 Mb/s is 1.25 megabytes/s. The 82586 is optimized for an 8-megahertz bus whose transfer rate is 4 megabytes/s, leaving considerable bandwidth for overhead and CPU processing.

Software diagnosis

Data-communications networks can be very complex because of their distributed and asynchronous nature, so it is hard to pinpoint a failure. The 82586 was designed with recognition of this problem and includes a set of features for improving reliability and testability.

All of these functions are performed under software control. They do not require any diagnostic hardware or any modifications. The chip offers such services as the monitoring of transmitted and received frames, support for statistics gathering and diagnostics of the entire network, diagnostic support for its node, and a means of testing its own operation.

In addition to the status information sent to the CPU after each transmission or reception, the chip also tallies the number of frames with CRC errors and alignment errors, as well as the number of frames lost due to DMA overrun or lack of empty receive buffers.

The 82586 also has mechanisms to collect statistics about the behavior of the entire network, as well as a means to locate problems in it. For example, the status of every transmitted frame provides network activity indicators, such as transmissions deferred because the channel was busy, the number of collisions experienced before the frame was transmitted, or no frame transmitted because of an excessive number of collisions.

The controller chip can be configured into a promiscuous mode, which means it captures all frames regardless of address. Such a mode is, for example, useful in implementing a monitoring station.

Each 82586 is also capable of determining whether there is a short or open circuit anywhere in the network (using time-domain reflectometry). The chip can even determine the distance of a short or open circuit from the controller, an important aid in finding the fault.

To support testing of both the software and hardware of the work station, the 82586 can be configured to an internal-loopback mode in which it is disconnected from the network and any frame transmitted is immediately re-received. This routine will indicate problems in the chip or the station.

What's more, an external loopback configuration permits users to test all the external logic between the 82586 and the link itself. This chip also checks the correct operation of the carrier-sense and collision-detect signals from the transceiver for every frame transmitted.

In order to check the operation of the chip itself, there is a dump command that causes the chip to write its internal registers to memory. For parts of the chip that cannot be checked from the outside, such as the random-number generator, a diagnose command triggers a self-test procedure that exercises any inaccessible counters.

An Ethernet node can be designed using the 82586 in conjunction with Intel's 16-bit iAPX 186 microprocessors (Fig. 3). The two chips have identical bus timing and control requirements. Thus they may share the same address latches, data latches, and bus controller.

Moreover, as an option, a bus arbiter can be used to enable designers to build a multisystem node. In this application, the 82586's system clock is driven by the iAPX 186's internally generated system-clock output.

82501 ETHERNET SERIAL INTERFACE

- **Compatible with the IEEE 802.3 Specification**
- **10-Mbs Operation**
- **Replaces 8 to 12 MSI Components**
- **Manchester Encoding/Decoding and Receive Clock Recovery**
- **10-MHz Transmit Clock Generator**
- **Driving/Receiving IEEE 802.3 Transceiver Cable**
- **Fail-Safe Watchdog Timer Circuit to Prevent Continuous Transmissions**
- **Diagnostics Loopback for Fault Detection and Isolation**
- **Directly Interfaces to the 82586 LAN Coprocessor**

The 82501 Ethernet Serial Interface (ESI) chip is designed to work directly with the 82586 LAN Coprocessor in IEEE 802.3/Ethernet and non-Ethernet 10-MBps local-area network applications. The major functions of the 82501 are to generate the 10 MHz transmit clock for the 82586, perform Manchester encoding/decoding of the transmitted/received frames, and provide the electrical interface to the Ethernet transceiver cable. Diagnostic loopback control enables the 82501 to route the signal to be transmitted from the 82586 through its Manchester encoding and decoding circuitry and back to the 82586. The combined loopback capabilities of the 82586 and 82501 result in efficient fault detection and isolation by providing sequential testing of the communications interface. An on-chip fail-safe watchdog timer circuit prevents the station from locking up in a continuous transmit mode.

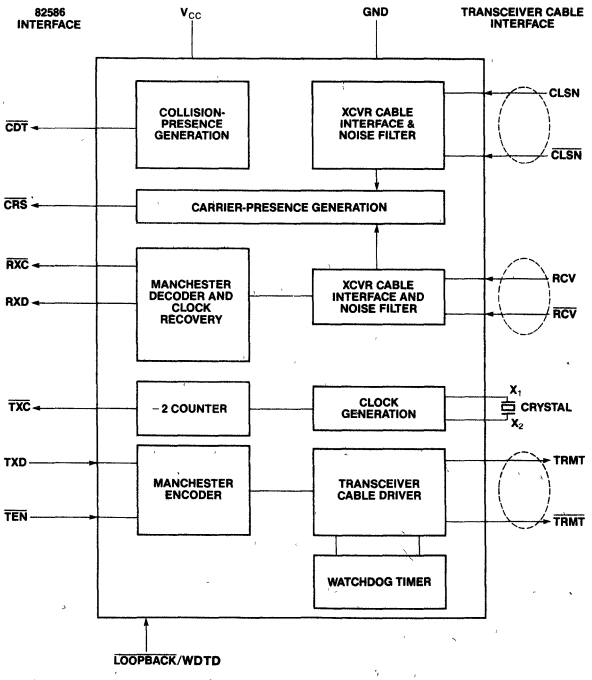


Figure 1. 82501 Functional Block Diagram

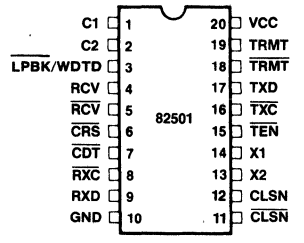


Figure 2. Pin Configuration

OCTOBER 1982
ORDER NUMBER: 210783-001

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{TXC}$	16	O	**Transmit Clock:** A 10-MHz clock output with 5 nsec rise and fall times and MOS driving levels. This clock is provided to the 82586 for serial transmission.
$\overline{TEN}$	15	I	**Transmit Enable:** An active low, TTL-level signal synchronous to $\overline{TXC}$ that enables data transmission to the transceiver cable. $\overline{TEN}$ can be driven by $\overline{RTS}$ from the 82586.
TXD	17	I	**Transmit Data:** A TTL-level input signal that is directly connected to the serial data ouput, TXD, of the 82586.
$\overline{RXC}$	8	O	**Receive Clock:** An MOS-level clock output with 5 nsec rise and fall times and 50% duty cycle. This output is connected to the 82586 receive clock input $\overline{RXC}$. There is a maximum 1.2 μsec discontinuity at the beginning of a frame reception when the phase-locked loop switches from the on-chip oscillator to the incoming data. During idle (no incoming frames) the clock frequency will be half that of the 20 MHz crystal frequency.
$\overline{CRS}$	6	O	**Carrier Sense:** A TTL-level, active low output to notify the 82586 that there is activity on the coaxial cable. This signal is asserted when valid data or a collision signal from the transceiver is present. It is deasserted at the end of a frame synchronous with $\overline{RXC}$, or when the end of the collision-presence signal (CLSN and $\overline{CLSN}$) is detected, whichever occurs later. Once deasserted, $\overline{CRS}$ will not be reasserted again for a period of 5 μsec minimum, 7 μsec maximum, regardless of any activity on the receive or collision-presence pairs.
RXD	9	O	**Receive Data:** An MOS-level output tied directly to the RXD input of the 82586 controller and sampled by the 82586 at the negative edge of $\overline{RXC}$. The bit stream received from the transceiver cable is Manchester decoded prior to being transferred to the controller. This output remains high during idle.
$\overline{CDT}$	7	O	**Collision Detect:** A TTL, active low signal which drives the $\overline{CDT}$ input of the 82586 controller. It is asserted as long as there is activity on the collision-presence pair (CLSN and $\overline{CLSN}$).
$\overline{LPBK}$/ WDTD	3	I	**Loopback:** A TTL-level control signal to enable the loopback mode. In this mode, serial data on the TXD input is routed through the 82501 internal circuits and back to the RXD output without driving the TRMT/$\overline{TRMT}$ output pair to the transceiver cable. When $\overline{LPBK}$ is asserted, the collision circuit will also be turned on at the end of each transmission to simulate the collision test. The on-chip watchdog timer can be disabled by applying a 12V level through a 4k ohm resistor to this pin.
TRMT $\overline{TRMT}$	19 18	O O	**Transmit Pair:** An output driver pair which generates the differential signal for the transmit pair of the Ethernet transceiver cable. Following the last transition, which is always positive at TRMT, the differential voltage is slowly reduced to zero volts. The output stream is Manchester encoded.
RCV $\overline{RCV}$	4 5	I I	**Receive Pair:** A differentially driven input pair which is tied to the receive pair of the Ethernet transceiver cable. The first transition on RCV will be negative-going to indicate the beginning of a frame. The last transition should be positive-going, indicating the end of a frame. The received bit stream is assumed to be Manchester encoded.
CLSN $\overline{CLSN}$	12 11	I I	**Collision Pair:** A differentially driven input pair tied to the collision-presence pair of the Ethernet transceiver cable. The collision-presence signal is a 10 MHz ±15% square wave. The first transition at CLSN is negative-going to indicate the beginning of the signal; the last transition is positive-going to indicate the end of the signal.
C1 C2	1 2	I I	**PLL Capacitor:** Phase-locked-loop capacitor inputs.
X_1 X_2	14 13	I I	**Clock Crystal:** 20-MHz crystal inputs.
V_{CC}	20		**Power:** 5 ± 10% volts.
GND	10		**Ground:** Reference.

FUNCTIONAL DESCRIPTION

Clock Generation

A 20 MHz crystal-controlled oscillator is provided as the basic clock source. This 20 MHz signal is then

divided by 2 to generate a 10 MHz ± .01% clock as required in the IEEE 802.3 specification. The oscillator requires an external parallel resonant fundamental mode, 20 MHz crystal.

Manchester Encoder and Transceiver Cable Driver

The 20 MHz clock is used to Manchester encode data on the TXD input line. The clock is also divided by 2 to produce the 10 MHz clock required by the 82586 for synchronizing its $\overline{RTS}$ and TXD signals. See Figure 3. (Note that the 82586 $\overline{RTS}$ is tied to the 82501 $\overline{TEN}$ input as shown in Figure 4.)

Data encoding and transmission begins with $\overline{TEN}$ going low. Since the first bit is a '1', the first transition on the transmit output TRMT is always negative. Transmission ends with the $\overline{TEN}$ going high. The last transition is always positive at TRMT and may occur at the center of the bit cell (last bit = 1) or at the boundary of the bit cell (last bit = 0). A one-bit delay is introduced by the 82501 between its TXD input and TRMT/$\overline{TRMT}$ output as shown in Figure 3. Following the last transition, the output $\overline{TRMT}$ is slowly brought to its high state so that zero differential voltage exists between TRMT and $\overline{TRMT}$. The undershoot for return to idle is less than 100 mV. This will eliminate DC currents in the primary of the transceiver's coupling transformer. See Figure 4.

An internal watchdog timer is started at the beginning of the frame. The duration of the watchdog timer is 25 msec ±15%. If the transmission terminates (by deasserting the $\overline{TEN}$) before the timer expires, the timer is reset (and ready for the next transmission). If the timer expires before the transmission ends, the frame is aborted. This is accomplished by disabling the output driver for the TRMT/$\overline{TRMT}$ pair and deasserting $\overline{CRS}$. RXD and $\overline{RXC}$ are not affected. The watchdog timer is reset only when the $\overline{TEN}$ is deasserted.

The cable driver is a differential gate requiring external resistors or a current sink of 20 mA (on both terminals). In addition, high-voltage protection of +16 volts maximum and short circuit to ground is provided.

Receive Section

CABLE INTERFACE AND NOISE FILTER

The 82501 input circuits can be driven directly from the Ethernet transceiver cable receive pair. In this case the cable is terminated with a pair of 39-ohm resistors in series for proper impedance matching. The 82501 has internal resistors that establish the common mode voltage. See Figure 4.

The input circuits can also be driven with ECL voltage levels. In either case, the input common mode voltage must be in the range of $V_{CC}-1.0$ to $V_{CC}-2.5$ volts to allow for a wide driver supply variation at the transceiver. The input terminals have a 15-volt maximum protection and additional clamping of low-energy, high-voltage noise signals.

A noise filter is provided at the RCV/$\overline{RCV}$ input pair to prevent spurious signals from improperly triggering the receiver circuitry. The noise filter has the following characteristics:

A negative pulse which is narrower than 15 ns or is less than −150 mV in amplitude is rejected during idle.

At the beginning of a reception, the filter is activated by the first negative pulse which is more negative than −300 mV and is wider than 30 ns.

As soon as the first valid negative pulse is recognized by the noise filter, the data threshold is lowered to 160 mV. The $\overline{CRS}$ signal is asserted to inform the 82586 controller of the beginning of a transmission, and the $\overline{RXC}$ will be held low for 1.4 μsec maximum while the internal phase-locked-loop is acquiring lock.

The filter is deactivated if no negative transition occurs within 200 ns from the last positive transition.

Immediately after the end of a reception, the filter blocks all the signals for 5 μsec minimum, 7 μsec maximum. This dead time is required to block-off spurious transitions which may occur on the coaxial cable at the end of a transmission but are not filtered out by the transceiver.

MANCHESTER DECODER AND CLOCK RECOVERY

The filtered data enters the clock recovery and decoder circuits. An analog phase-locked-loop (PLL) technique is used to extract the received clock from the data, beginning from the third negative transition of the incoming data. The PLL will acquire lock within the first 12 bit times, as seen from the RCV/$\overline{RCV}$ inputs. During that period of time, the $\overline{RXC}$ is held low. Bit cell timing distortion which can be tolerated in the incoming signal is ± 15 nsec for the preamble and ± 18 nsec for data. This distortion must have less than ± 5 ns bias distortion. The voltage-controlled oscillator (VCO) of the PLL corrects its frequency to match the incoming signal transitions.

Its VCO cycle time stays within 5% of the RXD bit cell time regardless of the time distortion allowed at the RCV/$\overline{\text{RCV}}$ input. The RCV/$\overline{\text{RCV}}$ input is decoded from Manchester to NRZ and transferred synchronously with the receive clock to the 82586 controller.

At the end of a frame, the receive clock is used to detect the absence of RCV/$\overline{\text{RCV}}$ transitions and report it to the 82586 by deasserting $\overline{\text{CRS}}$ while $\overline{\text{RXD}}$ is held high.

Collision-Presence Section

The CLSN/$\overline{\text{CLSN}}$ input signal is a 10 MHz ±15% square wave generated by the transceiver whenever two or more data frames are superimposed on the coaxial cable. The maximum asymmetry in the CLSN/$\overline{\text{CLSN}}$ signal is 60/40% for low-to-high or high-to-low levels. This signal is filtered for noise rejection in the same manner as RCV/$\overline{\text{RCV}}$. The noise filter rejects signals which are less negative than −150 mV and narrower than 15 ns during idle. It turns on at the first negative pulse which is more negative than −250 mV and wider than 30 ns. After the initial turn-on, the filter remains active indicating that a valid collision signal is present, as long as the negative CLSN/$\overline{\text{CLSN}}$ signal pulses are more negative than −250 mV. The filter returns to the "off" state if the signal becomes less negative than −150 mV, or if no negative transition occurs within 160 ns from the last positive transition. Immediately after turn-off, the collision filter is ready to be reactivated.

The common mode voltage and external termination are identical to the RCV/$\overline{\text{RCV}}$ input. (See Figure 4.) The CLSN/$\overline{\text{CLSN}}$ input also has a 15-volt maximum protection and additional clamping against low-energy, high-voltage noise signals.

A valid collision-presence signal will assert the 82501 $\overline{\text{CDT}}$ output which can be directly tied to the $\overline{\text{CDT}}$ input of the 82586 controller.

During the time that valid collision-presence transitions are present on the CLSN/$\overline{\text{CLSN}}$ input, invalid data transitions will be present on the receive data pair due to the superposition of signals from two or more stations transmitting simultaneously. It is possible for RCV/$\overline{\text{RCV}}$ to lose transitions for a few bit times due to perfect cancellation of the signals. In any case, the invalid data will not cause any discontinuity of RXC.

When a valid collision-presence signal is present the $\overline{\text{CRS}}$ signal is asserted (along with $\overline{\text{CDT}}$). However, if this collision-presence signal arrives within 6.0 ±1.0 μs from the time $\overline{\text{CRS}}$ was deasserted, only $\overline{\text{CDT}}$ is generated.

Internal Loopback

When asserted, $\overline{\text{LPBK}}$ causes the 82501 to route serial data from its TXD input, through its transmit logic (retiming and Manchester encoding), returning it through the receive logic (Manchester decoding and receive clock generation) to RXD output. The internal routing prevents the data from passing through the output drivers and onto the transmit output pair, TRMT/$\overline{\text{TRMT}}$. When in loopback mode, all of the transmit and receive circuits, including the noise filter, are tested except for the transceiver cable output driver and input receivers. Also, at the end of each frame transmitted in loopback mode, the 82501 generates a 1-μsec $\overline{\text{CDT}}$ signal within 1 μsec after the end of the frame. Thus, the collision circuits, including the noise filter, are also tested in loopback mode.

The watchdog timer remains enabled in loopback mode, terminating test frames that exceed its time-out period. The watchdog can be inhibited by placing the LPBK to a resistor connected to 12V ± 3V. The loopback feature can still be used to test the integrity of the 82501 by using the circuit shown in Figure 5.

In the normal mode ($\overline{\text{LPBK}}$ not asserted), the 82501 operates as a full duplex device, being able to transmit and receive simultaneously. This is similar to the external loopback mode of the 82586. Combining the internal and external loopback modes of the 82586 and the internal loopback and normal modes of the 82501, incremental testing of an 82586/82501-based interface can be performed under program control for systematic fault detection and fault isolation.

Interface Example

The 82501 is designed to work directly with the 82586 controller in Ethernet as well as non-Ethernet 10 Mbps LAN applications. The control and data signals connect directly between the two devices without the need for additional external logic. The complete 82586/82501/Ethernet Transceiver cable interface is shown in FIGURE 4. The 82501 provides the driver and receivers needed to directly connect to the transceiver cable, requiring only terminating resistors on each input signal pair.

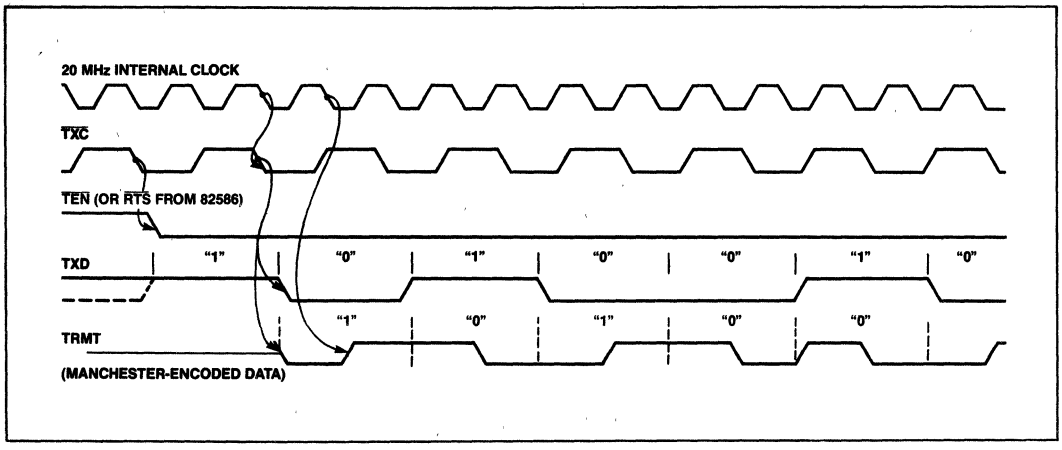


Figure 3. Start of Transmission and Manchester Encoding

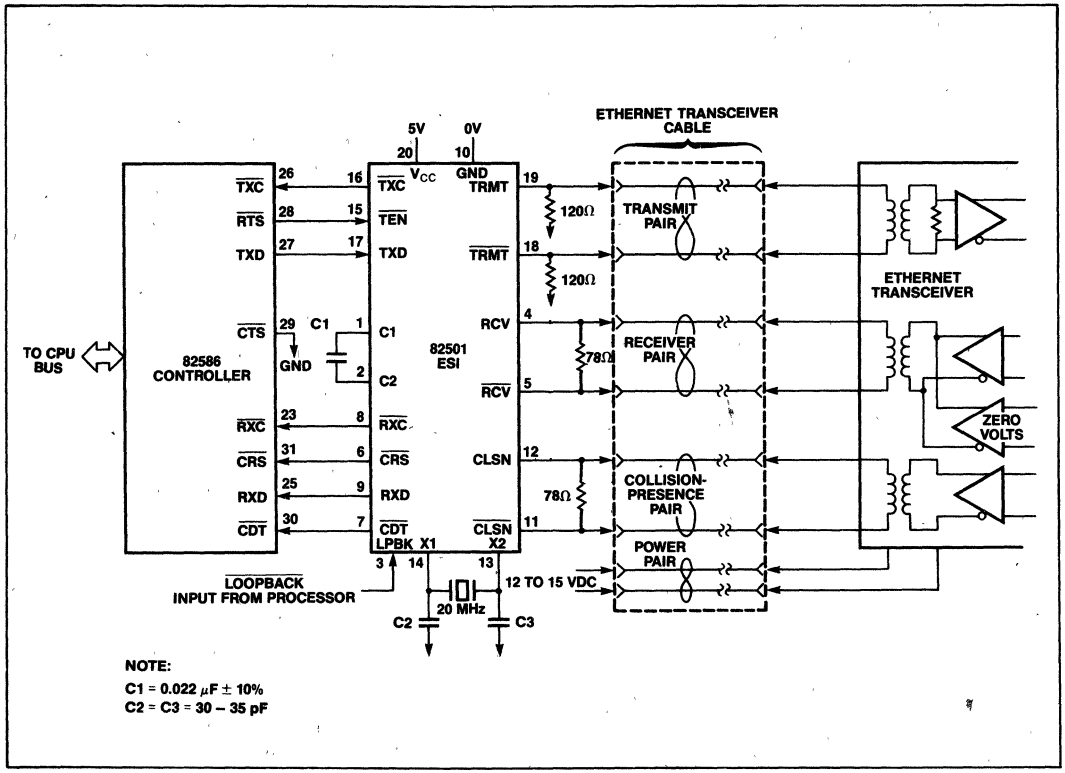


Figure 4. 82586/82501/Transceiver Cable Interface

ELECTRICAL CHARACTERISTICS

Please Note: The following specifications are preliminary values and are subject to change without notice. Contact your local Intel Sales Office for the latest specifications.

D.C. CHARACTERISTICS (T_A = 0-70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Conditions
V_{IL}	Input Low Voltage (TTL)	–0.5	+0.8	V	
V_{IH}	Input High Voltage (TTL)	2.0	V_{CC} + 0.5	V	
V_{IDF}	Input Differential Voltage	± 300	± 1500	mV	RCV and CLSN
V_{CM}	Input Common Mode Voltage	V_{CC} – 2.5	V_{CC} – 1.0	V	RCV and CLSN
V_{OL}	Output Low Voltage TTL or MOS		0.45	V	I_{OL} = 4 mA
V_{OCM}	Common Mode Output	1.0	4.5	V	R_L = 78 Ohms Differential Termination and 120Ω pulldown
V_{OH}	Output High Voltage TTL MOS	 2.4 3.9		 V V	 I_{OH} = –1.0 mA I_{OH} = –400 μA
V_{ODF}	Differential Output Swing	.6	1.1	V	R_L = 78 Ohms Differential Termination and 120Ω pulldown (TRMT)
I_{LI}	Input Leakage Current (TTL)		± 200	μA	$0 < V_{IN} < V_{CC}$ $V_L = V_{CC}$
C_{IN}	Input Capacitance		10	pF	f = 1 MHz
C_{OUT}	Output Capacitance		20	pF	f = 1 MHz
I_{CC}			200	mA	
I_F	Input Forward Current (TTL)		–500	μA	V_F = .45V

A.C. CHARACTERISTICS

A.C. Measurement Conditions

I) T_A = 0° to 70°C, V_{CC} = 5V ± 10%

II) The AC measurements are done at the following voltage levels for the various kinds of inputs and outputs

a) TTL inputs and oututs: 0.8V and 2.0V
The input voltage swing is 0.4 to 2.4V at least with 3–10 ns rise and fall times.

b) MOS outputs: The rise and fall times are measured between 0.6V and 3.6V points. The high time is measured between 3.6V points and the low time is measured between 0.6V points.

c) Differential inputs and outputs:
The 50% points of the total swing are used for delay measurements. The rise and fall times of outputs are measured at the 20 to 80% points. The differential voltage swing at the inputs is at least ± 300mV with rise and fall times of 3-15 ns measured at ±.2 volts. Once the squelch threshold has been exceeded the inputs will detect less than ±160 mV signals.

III) The AC loads for the various kind of outputs are as follows:

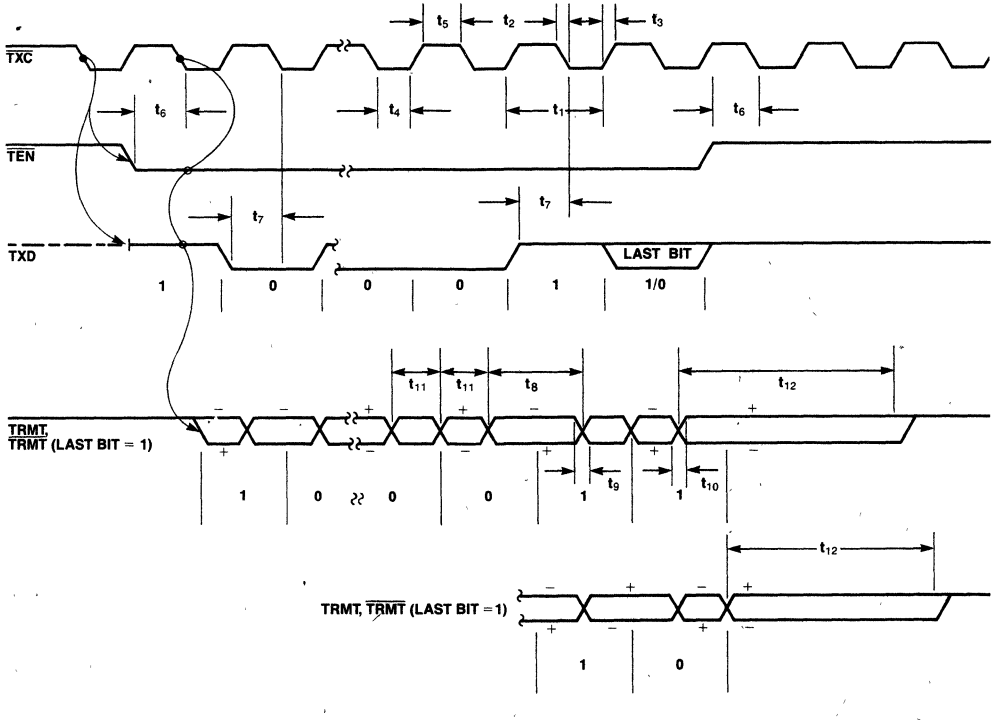
a) TTL and MOS: A 15-pF Capacitor to GND

b) Differential: A 10-pF Capacitor from each terminal to GND and a termination load resistor of 78 ohms in parallel with a 27 microhenries inductor between the two terminals.

TRANSMIT TIMING

Symbol	Parameter	Min.	Max.	Unit
t_1	$\overline{TXC}$ Cycle Time	99.99	100.01	ns
t_2	$\overline{TXC}$ Fall Time		5	ns
t_3	$\overline{TXC}$ Rise Time		5	ns
t_4	$\overline{TXC}$ Low Time	40		ns
t_5	$\overline{TXC}$ High Time	40		ns
t_6	Transmit Enable/Disable to $\overline{TXC}$ Low	50		ns
t_7	TXD Stable to $\overline{TXC}$ Low	50		ns
t_8	Bit Cell Center to Bit Cell Center of Transmit Pair Data	99.5	100.5	ns
t_9	Transmit Pair Data Fall Time [1]	1.0	5.0	ns
t_{10}	Transmit Pair Data Rise Time [1]	1.0	5.0	ns
t_{11}	Bit Cell Center to Bit Cell Boundary of Transmit Pair Data	49.5	50.5	ns
t_{12}	$\overline{TRMT}$ starts approaching its high level from Last Positive Transition of Transmit Pair Data during idle.	200	8000	ns

TRANSMIT TIMING



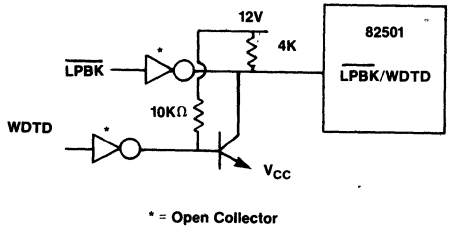
Note:
1. Measured per 802.3 Para 6.5.1.1

RECEIVE TIMING

Symbol	Parameter	Min.	Max.	Unit
t_{13}	Receive Pair Signal Pulse Width (at –.30V differential signal) of First Negative Pulse for a) Signal Rejection by Noise Filter, b) Noise Filter Turn-on in order to Begin Reception	30	15	ns ns
t_{14}	Duration which the $\overline{RXC}$ is held at low state		1400	ns
t_{15}	Receive Pair Signal Rise/Fall Time at ±.2 volt		20	ns
t_{16}[1]	Receive Pair Bit Cell Center from crossover timing distortion: In preamble In data		±20 ±20	ns ns
t_{17}[1]	Receive Pair Bit Cell Boundary allowing for timing distortion: In preamble In data		±20 ±20	ns ns
t_{18}	Receive Idle Time Before the Next Reception can Begin (as measured from the deassertion of $\overline{CRS}$)		8	μs
t_{19}	Receive Pair Signal Return to Zero Level from Last valid Positive Transition	0.20		μs
t_{20}	$\overline{CRS}$ Assertion delay from the First received valid Negative Transition of Receive Pair Signal		100	ns

Symbol	Parameter	Min.	Max.	Unit
t_{21}	$\overline{CRS}$ Deassertion delay from the Last valid positive transition received (when no Collision-Presence signal exists on the transceiver cable)		300[2]	ns
t_{24}	$\overline{RXC}$ Jitter		±5.0	ns
t_{25}	$\overline{RXC}$ Rise/Fall time			ns
t_{26}	$\overline{RXC}$ High/Low time	40		ns
t_{27}	Receive Data Stable before the Negative Edge of $\overline{RXC}$	30		ns
t_{28}	Receive Data Held valid past the Negative Edge of $\overline{RXC}$	30		ns
t_{29}	Carrier Sense deasserted before the Negative Edge of $\overline{RXC}$	10	30	ns
t_{30}	Receive data Rise/Fall time		10	ns
t_{31}	From the time $\overline{CRS}$ is deasserted until the time it can be asserted again	5	7	μs

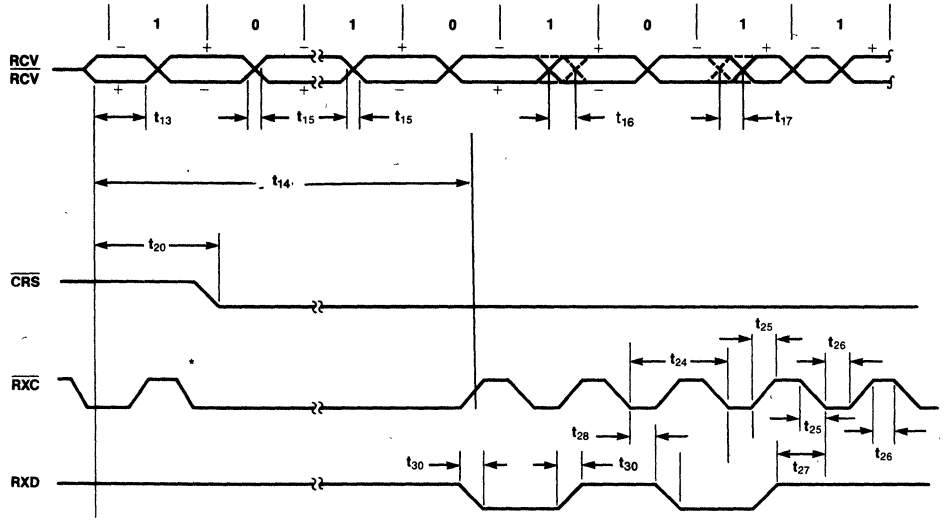
NOTES:
1. ± 5 ns of bias distortion—the remainder is random distortion.
2. $\overline{CRS}$ is deasserted synchronously with the $\overline{RXC}$. This condition is not specified in the IEEE 802.3 specification.



LPBK	WDTD	Function
1	X	LPBK mode
0	0	Normal mode
0	*1	Normal mode with watchdog timer disabled

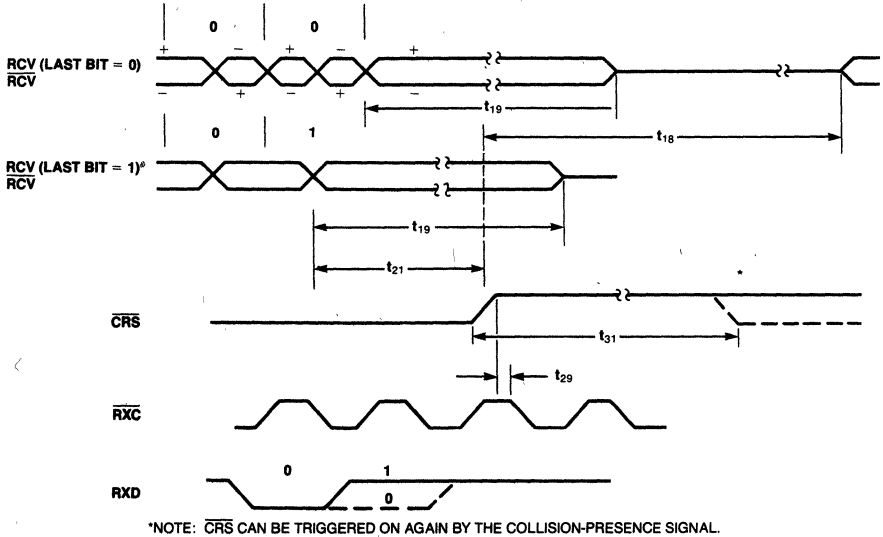
Figure 5. Watchdog Timer Disable

RECEIVE TIMING: START OF FRAME



*THIS CLOCK PULSE MAY NOT BE A VALID CLOCK PULSE

RECEIVE TIMING: END OF FRAME

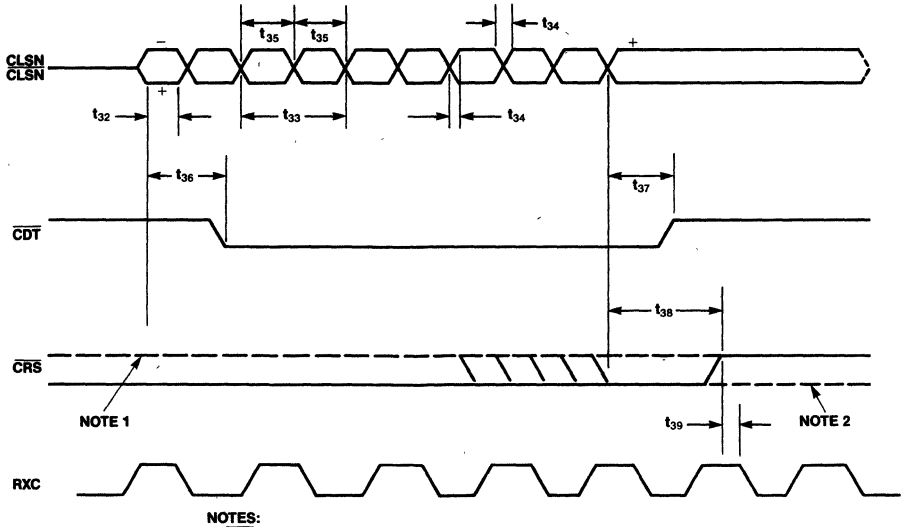


*NOTE: $\overline{CRS}$ CAN BE TRIGGERED ON AGAIN BY THE COLLISION-PRESENCE SIGNAL.

COLLISION TIMING

Symbol	Parameter	Min.	Max.	Unit
t_{32}	CLSN/CLSN Signal Pulse Width (at –.30V differential signal) of first Negative Pulse for Noise Filter Turn-on	30		ns
t_{33}	CLSN/$\overline{\text{CLSN}}$ Cycle Time	86	118	ns
t_{34}	CLSN/$\overline{\text{CLSN}}$ Rise/Fall Time at ±.2 volts		15	ns
t_{35}	CLSN/$\overline{\text{CLSN}}$ Transition Time	35	70	ns
t_{36}	$\overline{\text{CDT}}$ Assertion from the First Valid Negative Edge of Collision Pair Signal		75	ns
t_{37}	$\overline{\text{CDT}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ Signal		200	ns
t_{38}	$\overline{\text{CRS}}$ Deassertion from the Last Positive Edge of CLSN/$\overline{\text{CLSN}}$ signal (If no post-collision signal remains on the receive pair.)		350	ns
t_{39}	$\overline{\text{CRS}}$ stable before the negative edge of $\overline{\text{RXC}}$ at deassertation	10	60	ns

COLLISION TIMING



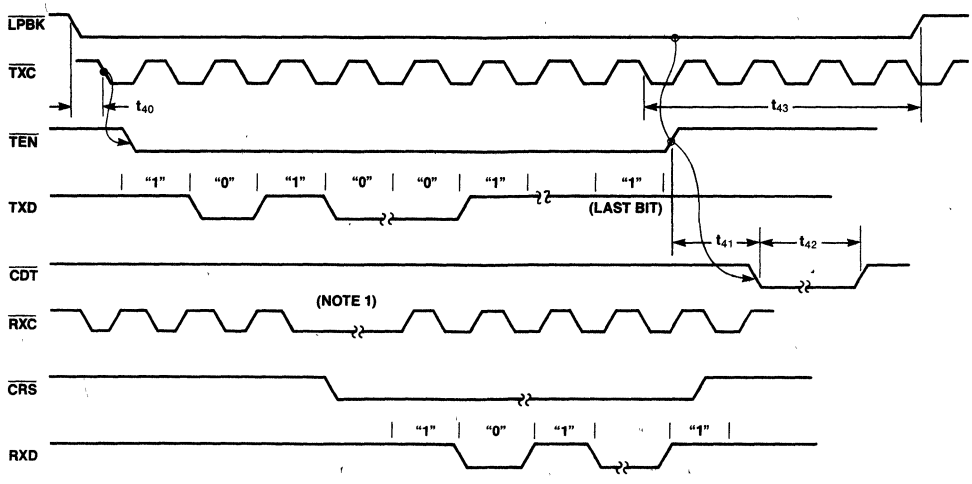
NOTES:
1. $\overline{\text{CRS}}$ WILL BE DEASSERTED FOR A PERIOD UP TO 7 μSEC MAXIMUM WHEN RCV/$\overline{\text{RCV}}$ OR CLSN/$\overline{\text{CLSN}}$ TERMINATES, WHICHEVER OCCURS LATER.
2. $\overline{\text{CRS}}$ WILL REMAIN ASSERTED AFTER THE CLSN/$\overline{\text{CLSN}}$ SIGNAL TERMINATES IF RCV/$\overline{\text{RCV}}$ SIGNALS CONTINUE.

LOOPBACK TIMING

Symbol	Parameter	Min.	Max.	Unit
t_{40}	$\overline{\text{LPBK}}$ asserted before the first attempted transmission	500		ns
t_{41}	Simulated collision test delay from the end of each attempted transmisssion	.5	1.5	μs
t_{42}	Simulated collision test duration	.5	1.0	μs
t_{43}	$\overline{\text{LPBK}}$ deasserted after the last attempted transmission	5		μs

NOTE:
In Loopback mode, $\overline{\text{RXC}}$, $\overline{\text{RXD}}$ and $\overline{\text{CRS}}$ function in the same manner as a normal Receive.

LOOPBACK TIMING



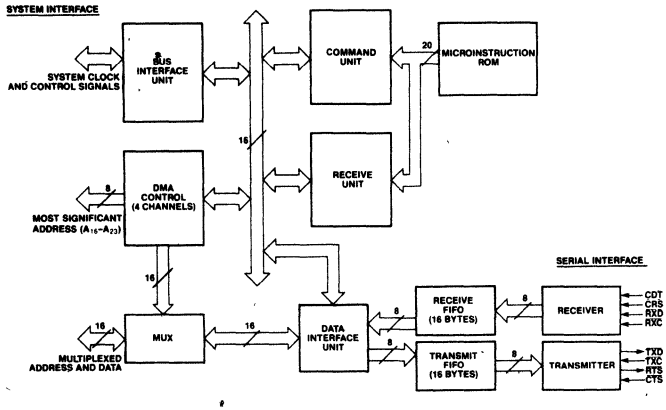
NOTE:
1. DURING LOOPBACK, THE 82501 RECEIVE CIRCUITRY USES 12 BIT TIMES WHILE THE PLL LOCKS ON THE DATA AS A RESULT, THE FIRST 12 BITS ARE LOST.

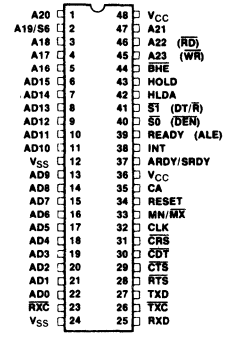
TESTABILITY

NOTES:
1. All AC parameters become valid after the PLL has stabilized: 100μs after the application of power.
2. TXC can be synchronized to tester clock by applying reset signal (12V) to the $\overline{\text{TEN}}$ pin.

82586
LOCAL AREA NETWORK COPROCESSOR

- **Fully implements the IEEE 802.3/Ethernet Data Link specifications without CPU overhead.**
- **Bus interface optimized to iAPX 186 and 188 microprocessors.**
- **On-chip DMA channels provide automatic memory management.**
- **Independent parallel bus and serial line clocks.**
- **Network diagnostics:**
  - **— Frame CRC errors**
  - **— Frame alignment errors**
  - **— Location of cable opens/shorts**
  - **— Collision tallies**
- **Self test diagnostics**
  - **— Loop back**
  - **— Register Dump**
  - **— Backoff timer check**
- **Efficient use of memory via buffer chaining.**
- **User configurable to realize broadband, short topology and 1 Mbps networks.**





NOTE THE SYMBOLS IN PARENTHESES CORRESPOND TO MINIMUM MODE.

Figure 1. 82586 Functional Block Diagram

Figure 2. 82586 Pinout

November 1983
Order Number: 210783-003

The 82586 is an intelligent, high performance Local Area Network coprocessor, implementing the CSMA/CD link access method (Carrier Sense Multiple Access with Collision Detection).

The 82586 performs a large range of link management and channel interface functions including: CSMA/CD link access, framing, preamble generation and stripping, source address generation, destination address checking, CRC generation and checking. Any data rate up to 10 Mb/s can be used.

The 82586 features a powerful host system interface. It automatically manages memory structures with command chaining and bidirectional data chaining. An on-chip DMA controller manages 4 channels transparently to the user. Buffers containing errored or collided frames can be automatically recovered. The 82586 can be configured for 8-bit or 16-bit data path, with maximum burst transfer rate of 2 or 4 Mbyte/sec, respectively. Memory address space is 16 Mbyte maximum.

The 82586 provides two independent 16 byte FIFO's, one for receiving and one for transmitting. The threshold for block transfer to/from memory is programmable, enabling the user to optimize bus overhead for a given worst case bus latency.

The 82586 provides a rich set of diagnostic and network management functions including: internal and external loopbacks, exception condition tallies, channel activity indicators, optimal capture of all frames regardless of destination address, optional capture of errored or collided frames, and time domain reflectometry for locating fault points in the cable.

The 82586 can be used in conjunction with either baseband or broadband networks. The controller can be configured for maximum network efficiency (minimum contention overhead) for any length network operating at any data rate within the 82586's range. The controller supports address field lengths of 1, 2, 3, 4, 5, or 6 bytes. It can be configured for either the IEEE 802.3/ Ethernet or HDLC method of frame delineation. Both 16-bit and 32-bit CRC are supported.

The 82586 is packaged in a 48 pin DIP and fabricated in Intel's reliable HMOS II 5 volt technology.

Table 1. 82586 Pin Description

Symbol	Pin No.	Type	Name and Function
VCC, VCC	48, 36		System Power: +5 volt power supply.
VSS, VSS	12, 24		System Ground.
RESET	34	I	RESET is an active HIGH internally synchronized signal, causing the 82586 to terminate present activity immediately. The signal must be HIGH for at least four clock cycles. The 82586 will execute RESET within ten system clock cycles starting from RESET HIGH. When RESET returns LOW, the 82586 waits for the first CA to begin the initialization sequence.
TxD	27	0	Transmitted Serial Data output signal. This signal is HIGH when not transmitting.
$\overline{\text{TxC}}$	26	I	Transmit Data Clock. This signal provides timing information to the internal serial logic, depending upon the mode of data transfer. For NRZ mode of operation, data is transferred to the TxD pin on the HIGH to LOW clock transition.
RxD	25	I	Received Data input signal.
$\overline{\text{RxC}}$	23	I	Received Data Clock. This signal provides timing information to the internal shifting logic depending upon the mode of data transfer. For NRZ data, the state of the RxD pin is sampled on the HIGH to LOW clock transition.
$\overline{\text{RTS}}$	28	0	Request To Send signal. When LOW, notifies an external interface that the 82586 has data to transmit. It is forced HIGH after a Reset and while the Transmit Serial Unit is not sending data.

Table 1. 82586 Pin Description (Cont'd.)

Symbol	Pin No.	Type	Name and Function
$\overline{CTS}$	29	I	Active LOW Clear To Send input enables the 82586 transmitter to actually send data. It is normally used as an interface handshake to $\overline{RTS}$. This signal going inactive stops transmission. It is internally synchronized. If $\overline{CTS}$ goes inactive, meeting the setup time to TxC negative edge, transmission is stopped and $\overline{RTS}$ goes inactive within, at most, two TxC cycles.
$\overline{CRS}$	31	I	Active LOW Carrier Sense input used to notify the 82586 that there is traffic on the serial link. It is used only if the 82586 is configured for external Carrier Sense. When so configured, external circuitry is required for detecting serial link traffic. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles.
$\overline{CDT}$	30	I	Active LOW Collision Detect input is used to notify the 82586 that a collision has occurred. It is used only if the 82586 is configured for external Collision Detect. External circuitry is required for detecting the collision. It is internally synchronized. To be accepted, the signal must stay active for at least two serial clock cycles. During transmission, the 82586 is able to recognize a collision one bit time after preamble transmission has begun.
INT	38	0	Active HIGH Interrupt request signal.
CLK	32	I	The system clock input from the 80186 or another symmetric clock generator.
MN/$\overline{MX}$	33	I	When HIGH, MN/$\overline{MX}$ selects $\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$, DT/$\overline{R}$ (Minimum Mode). When LOW, MN/$\overline{MX}$ selects A22, A23, READY, $\overline{S0}$, $\overline{S1}$ (Maximum Mode). Note: This pin should be static during 82586 operation.
AD0 - AD15	6-11, 13-22	I/O	These lines form the time multiplexed memory address (t1) and data (t2, t3, tW, t4) bus. When operating with an 8-bit bus, the high byte will output the address during the entire cycle. AD0-AD15 are floated after a RESET or when the bus is not acquired.
A16-A18, A20-A23	1, 3-5, 45-47	0	Used maximum mode only. These lines constitute 7 out of 8 most significant address bits for memory operation. They switch during t1 and stay valid during the entire memory cycle. The lines are floated after RESET or when the bus is not acquired.
A19/S6	2	0	During t1 it forms line 19 of the memory address. During t2 through t4 it is used as a status indicating that this is a Master peripheral cycle, and is HIGH. Its timing is identical to that of AD0 - AD15 during write operation.
HOLD	43	0	HOLD is an active HIGH signal used by the 82586 to request local bus mastership at the end of the current CPU bus transfer cycle, or at the end of the current DMA burst transfer cycle. In normal operation, HOLD goes inactive before HLDA. The 82586 can be forced off the bus by HLDA going inactive. In this case, HOLD goes inactive, at most, three bus cycles after HLDA goes inactive.
HLDA	42	I	HLDA is an active HIGH Hold Acknowledge signal indicating that the CPU has received the HOLD request and that bus control has been relinquished to the 82586. It is internally synchronized. After HOLD is detected as LOW, the processor drives HLDA LOW. Note, CONNECTING VCC TO HLDA IS NOT ALLOWED because it will cause a deadlock. Users wanting to give permanent bus access to the 82586 should connect HLDA with HOLD. If HLDA goes inactive before HOLD, the 82586 will release the bus (by HOLD going inactive), within three bus cycles at most.

Table 1. 82586 Pin Description (Cont'd.)

Symbol	Pin No.	Type	Name and Function
CA	35	I	The CA pin is a Channel Attention input used by the CPU to initiate the 82586 execution of memory resident Command Blocks. The CA signal is synchronized internally. The signal must be HIGH for at least one system clock period. It is latched internally on HIGH to LOW edge and then detected by the 82586.
$\overline{BHE}$	44	O	The Bus High Enable signal ($\overline{BHE}$) is used to enable data onto the most significant half of the data bus. Its timing is identical to that of A16-A23. With a 16-bit bus it is LOW and with an 8-bit bus it is HIGH. Note: after RESET, the 82586 is configured to 8-bit bus.
READY	39	I	This active HIGH signal is the acknowledgement from the addressed memory that the transfer cycle can be completed. While LOW, it causes wait states to be inserted. This signal must be externally synchronized with the system clock. The Ready signal internal to the 82586 is a logical OR between READY and SRDY/$\overline{ARDY}$.
SRDY/$\overline{ARDY}$	37	I	This active HIGH signal performs the same function as READY. If it is programmed at configure time to SRDY, it is identical to READY. If it is programmed to $\overline{ARDY}$, the positive edge of the Ready signal is internally synchronized. Note, the negative edge must still meet setup and hold time specifications, when in $\overline{ARDY}$ mode. The $\overline{ARDY}$ signal must be active for at least one system clock HIGH period for proper strobing. The Ready signal internal to the 82586 is a logical OR between READY (in Maximum Mode only) and SRDY/$\overline{ARDY}$. Note that following RESET, this pin assumes $\overline{ARDY}$ mode.
$\overline{S0}$, $\overline{S1}$	40,41	O	Maximum mode only. These status pins define the type of DMA transfer during the current memory cycle. They are encoded as follows: $\overline{S1}$ $\overline{S0}$ 0 0 Not Used 0 1 Read Memory 1 0 Write Memory 1 1 Passive Status is active from the middle of t4 to the end of t2. They return to the passive state during t3 or during tW when READY or ARDY is HIGH. These signals can be used by the 8288 Bus Controller to generate all memory control and timing signals. Any change from the passive state signals the 8288 to start the next t1 to t4 bus cycle. These pins are pulled HIGH and floated after a system RESET and when the bus is not acquired.
$\overline{RD}$	46	O	Used in minimum mode only. The read strobe indicates that the 82586 is performing a memory read cycle. $\overline{RD}$ is active LOW during t2, t3 and tW of any read cycle. This signal is pulled HIGH and floated after a RESET and when the bus is not acquired.
$\overline{WR}$	45	O	Used in minimum mode only. The write strobe indicates that the 82586 is performing a write memory cycle. $\overline{WR}$ is active LOW during t2, t3 and tW of any write cycle. It is pulled HIGH and floats after RESET and when the bus is not acquired.
ALE	39	O	Used in minimum mode only. Address Latch Enable is provided by the 82586 to latch the address into the 8282/8283 address latch. It is a HIGH pulse, during t1 ('clock low') of any bus cycle. Note that ALE is never floated.
$\overline{DEN}$	40	O	Used in minimum mode only. Data ENable is provided as output enable for the 8286/8287 transceivers in a stand-alone (no 8288) system. $\overline{DEN}$ is active LOW during each memory access. For a read cycle, it is active from the middle of t2 until the beginning of t4. For a write cycle, it is active from the beginning of t2 until the middle of t4. It is pulled HIGH and floats after a system RESET or when the bus is not acquired.

Table 1. 82586 Pin Description (Cont'd.)

Symbol	Pin No.	Type	Name and Function
DT/$\overline{R}$	41	O	Used in minimum mode only. DT/$\overline{R}$ is used in non-8288 systems using an 8286/8287 data bus transceiver. It controls the direction of data flow through the Transceiver. Logically, DT/$\overline{R}$ is equivalent to $\overline{S1}$. It becomes valid in the t4 preceding a bus cycle and remains valid until the final t4 of the cycle. This signal is pulled HIGH and floated after a RESET or when the bus is not acquired.

82586/HOST CPU INTERACTION

Communication between the 82586 and the host is carried out via shared memory. The 82586's direct access to memory capability allows autonomous transfer of data blocks (buffers, frames) and relieves the CPU of byte transfer overhead. The 82586 is optimized for operating with the iAPX 186, but due to the small number of hardware signals between the 82586 and the CPU, the 82586 can operate easily with other processors. In discussing 82586/Host interaction, the logical interface and the hardware bus interface are referred to separately.

The 82586 consists of two independent units: Command Unit (CU) and Receive Unit (RU). The CU executes commands from shared memory. The RU handles all activities related to frame reception. The CU and RU enable the 82586 to engage in the two activities simultaneously: the CU may be fetching and executing commands out of memory, and the RU may be storing received frames in memory. CPU intervention is only required after the CU executes a sequence of commands or the RU stores a sequence of frames.

The only hardware signals that connect the CPU and the 82586, are the INTERRUPT and CHANNEL ATTENTION, see Figure 3. Interrupt is used by the 82586 to draw the CPU's attention to a change in the SCB. The Channel Attention is used by the CPU to draw the 82586's attention.

82586 SYSTEM MEMORY STRUCTURE

The Shared Memory structure is composed of four parts: Initialization Root, System Control Block (SCB), Command List, and Receive Frame Area (RFA), see Figure 4.

The Initialization Root is at a predetermined location in the memory space, (OFFFFF6H), known to both the host the CPU and the 82586. The root is accessed at initialization and points to the System Control Block.

The System Control Block (SCB) serves as a bidirectional mailbox between the host the CPU, CU and RU. It is the central element through which the CPU and the 82586 exchange control and status

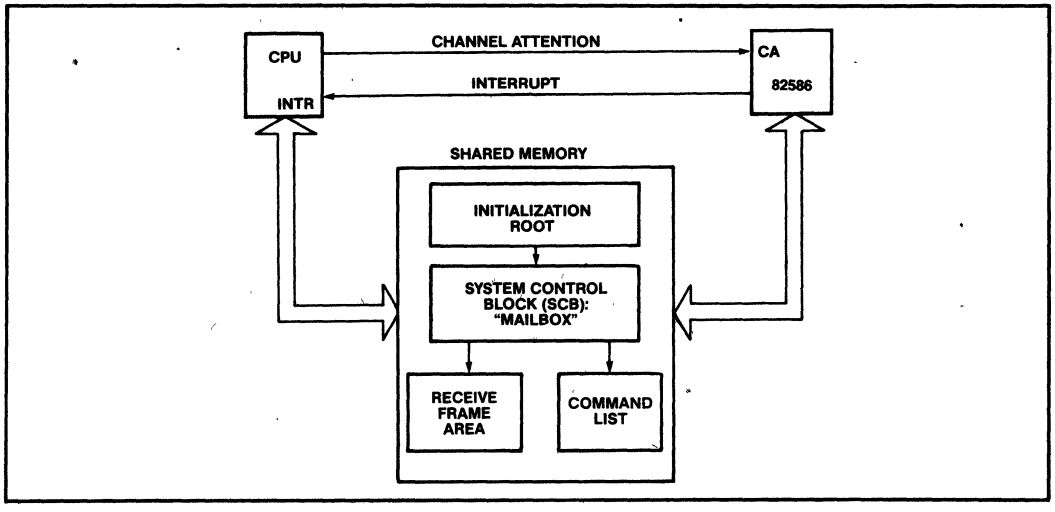


Figure 3. 82586/Host CPU Interaction

information. The SCB is composed of two parts. First, instructions from the CPU to the 82586. These include: control of the CU and RU (START, ABORT, SUSPEND, RESUME), a pointer to the list of commands for the CU, a pointer to the receive frame area, and a set of interrupt acknowledge bits. Second, information from the 82586 to the CPU that includes: state of the CU and RU (e.g. IDLE, ACTIVE READY, SUSPENDED, NO RECEIVE RESOURCES), interrupt bits (command completed, frame received, CU gone not ready, RU gone not ready), and statistics. See Figure 4.

The Command List serves as a program for the CU. Individual commands are placed in memory units called a Command Block, or CB. CB's contain command specific parameters and command specific statuses. Specifically, these high level commands are called Action Commands (e.g. Transmit, Configure).

A specific command, Transmit, causes transmission of a frame by the 82586. The Transmit command block includes Destination Address, Type Field, and a pointer to a list of linked buffers that holds the frame to be constructed from several buffers scattered in memory. The Command Unit performs,

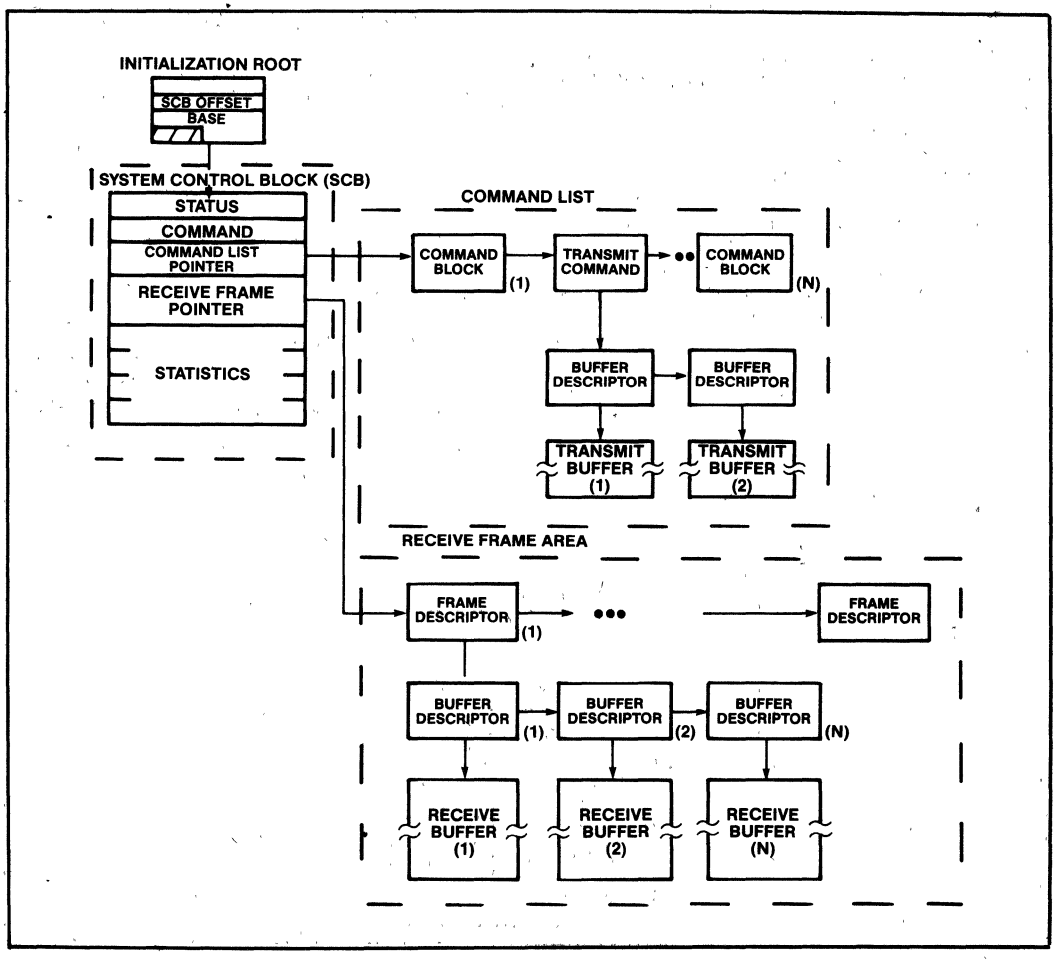


Figure 4. 82586 Shared Memory Structure

without the CPU intervention, the DMA of each buffer and the prefetching of references to new buffers in parallel. The CPU is notified only after successful transmission or retransmission.

The Receive Frame Area is a list of Free Frame Descriptors (Descriptors not yet used) and a list of buffers prepared by the user. It is conceptually distinct from the Command List. Frames arrive without being solicited by the 82586. The 82586 must be prepared to receive them even if it is engaged in other activities and to store them in the Free Frame Area. The Receive Unit fills the buffers upon frame reception and reformats the Free Buffer List into received frame structures. The frame structure is virtually identical to the format of the frame to be transmitted. The first frame descriptor is referenced by SCB, and the reclaimed and returned to the Free Buffer List, unless the chip is configured to Save Bad Frames.

Receive buffer chaining (i.e. storing incoming frames in a linked list of buffers) improves memory utilization significantly. Without buffer chaining, the user must allocate consecutive blocks of the maximum frame size (1518 bytes in Ethernet) for each frame. Taking into account that a typical frame size may be about 100 bytes, this practice is very inefficient. With buffer chaining, the user can allocate small buffers and the 82586 uses only as many as needed.

In the past, the drawback of buffer chaining was the CPU processing overhead and the time involved in the buffer switching (especially at 10 Mb/s). The 82586 overcomes this drawback by performing buffer management on its own (completely transparent to the user).

The 82586 has a 22-bit memory address range in minimum mode and 24-bit memory address range in maximum mode. All memory structures, the System Control Block, Command List, Receive Descriptor List, and all buffer descriptors must reside within one 64K-byte memory segment. The Data Buffers can be located anywhere in the memory space.

TRANSMITTING FRAMES

The 82586 executes high level or action commands from the Command List in external memory. Action commands are fetched and executed in parallel with the host CPU's operation, thereby significantly improving system performance. The general action commands format is shown in Figure 5.

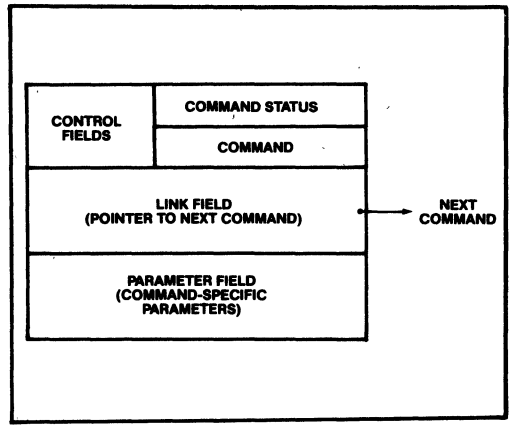


Figure 5. Action Command Format

Message transmission is accomplished by using the Transmit command. A single Transmit command contains, as part of the command-specific parameters, the destination address and type field for the transmitted frame along with a pointer to a buffer area in memory containing the data portion of the frame. (See Figure 15.) The data field is contained in a memory data structure consisting of a Buffer Descriptor (BD) and Data Buffer (or a linked list of buffer descriptors and buffers) as shown in Figure 6. The BD contains a Link Field which points to the next BD on the list and a 24-bit address pointing to the Data Buffer itself. The length of the Data Buffer is specified by the Actual Count field of the BD.

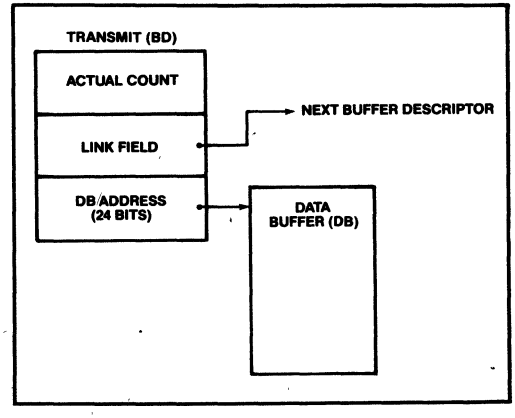


Figure 6. Data Buffer Descriptor and Data Buffer Structure

Using the BD's and Data Buffers, multiple Data Buffers can be 'chained' together. Thus, a frame with a long Data Field can be transmitted using multiple (shorter) Data Buffers chained together. This chaining technique allows the system designer to develop efficient buffer management policies.

When transmitting a frame as shown below in Figure 7:

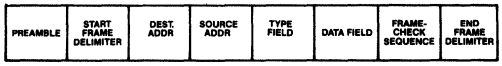


Figure 7. Frame Format

The 82586 automatically generates the preamble (alternating 1's and 0's) and start frame delimiter, fetches the destination address and type field from the Transmit command, inserts its unique address as the source address, fetches the data field from buffers pointed to by the Transmit command, and computes and appends the CRC at the end of the frame.

The 82586 can be configured to generate either the Ethernet or HDLC start and end frame delimiters. In the Ethernet mode, the start frame delimiter is two consecutive 1 bits and the end frame delimiter indicated by the lack of a signal after transmitting the last bit of the frame-check sequence field. When in the HDLC mode, the 82586 will generate the 01111110 'flag' for the start and end frame delimiters and perform the standard 'bit stuffing/stripping.' In addition, the 82586 will optionally pad frames that are shorter than the specified minimum frame length by appending the appropriate number of flags to the end of the frame.

In the event of a collision (or collisions), the 82586 manages the entire jam, random wait and retry process, reinitializing DMA pointers without CPU intervention. Multiple frames can be sent by linking the appropriate number of Transmit commands together. This is particularly useful when transmitting a message that is larger than the maximum frame size (1518 bytes for Ethernet).

RECEIVING FRAMES

In order to minimize CPU overhead, the 82586 is designed to receive frames without CPU supervision. The host CPU first sets aside an adequate amount of receive buffer space and then enables the

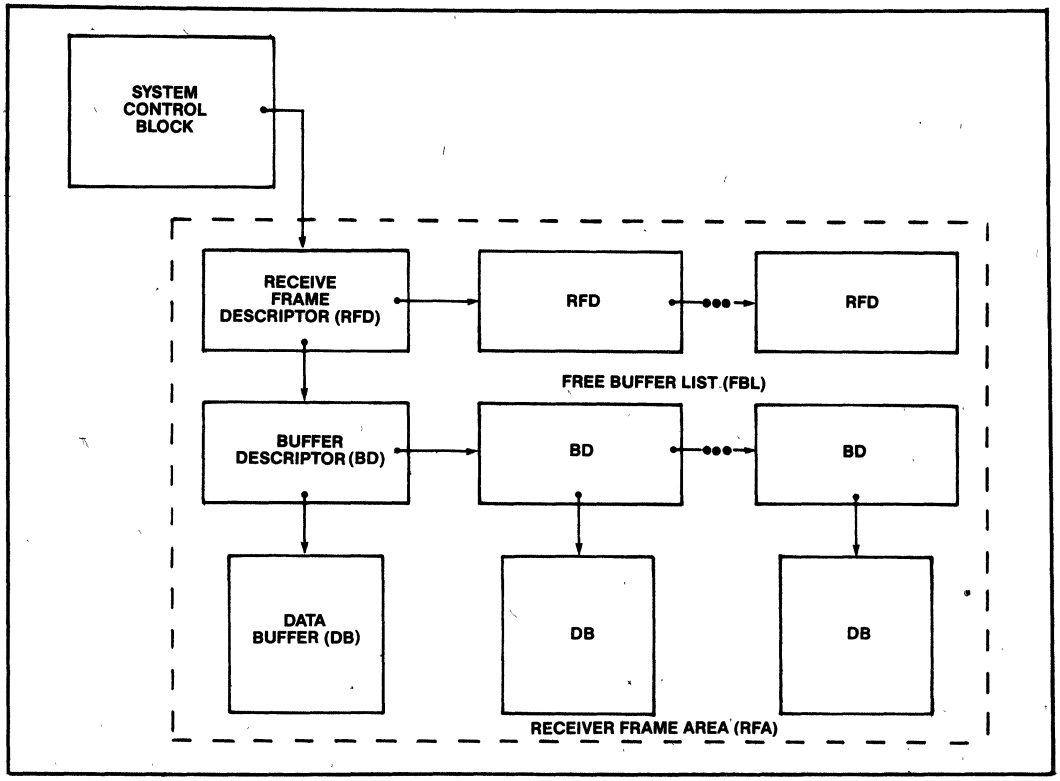


Figure 8. Receive Frame Area Diagram

82586's Receive Unit. Once enabled, the RU 'watches' for any of its frames which it automatically stores in the Receive Frame Area (RFA). The RFA consists of a Receive Descriptor List (RDL) and a list of free buffers called the Free Buffer List (FBL) as shown in Figure 8. The individual Receive Frame Descriptors that make up the RDL are used by the 82586 to store the destination and source address, type field and status of each frame that is received. (Figure 9.)

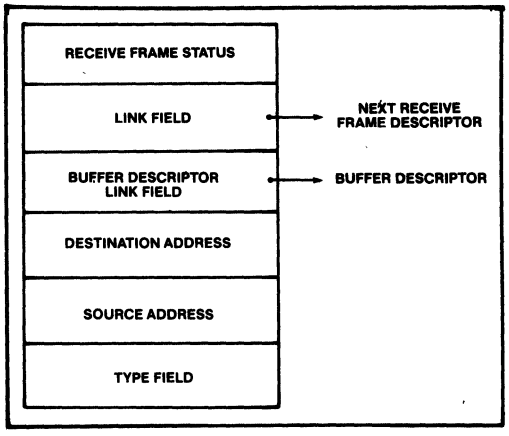


Figure 9. Receive Frame Descriptor

The 82586, once enabled, checks each passing frame for an address match. The 82586 will recognize its own unique address, one or more multicast addresses or the broadcast address. If a match occurs, it stores the destination and source address and type field in the next available RFD. It then begins filling the next free Data Buffer on the FBL (which is pointed to by the current RFD) with the data portion of the incoming frame. As one DB is filled, the 82586 automatically fetches the next DB on the FBL until the entire frame is received. This buffer chaining technique is particularly memory efficient because it allows the system designer to set aside buffers that fit a frame size that may be much shorter than the maximum allowable frame.

Once the entire frame is received without error, the 82586 performs the following housekeeping tasks:

- Updates the Actual Count field of the last Buffer Descriptor used to hold the frame just received with the number of bytes stored in its associated Data Buffer.
- Fetches the address of the next free Receive Frame Descriptor.
- Writes the address of the next free Buffer Descriptor into the next free Receive Frame Descriptor.
- Posts a 'Frame Received' interrupt status bit in the SCB.
- Interrupts the CPU.

In the event of a frame error, such as a CRC error, the 82586 automatically reinitializes its DMA pointers and reclaims any data buffers containing the bad frame. As long as Receive Frame Descriptors and data buffers are available, the 82586 will continue to receive frames without further CPU help.

82586 NETWORK MANAGEMENT AND DIAGNOSTIC FUNCTIONS

The behavior of data communication networks is typically very complex due to their distributed and asynchronous nature. It is particularly difficult to pin-point a failure when it occurs. The 82586 was designed in anticipation of these problems and includes a set of features for improving reliability and testability.

The 82586 reports on the following events after each frame transmitted:

- Transmission successful.
- Transmission unsuccessful; lost Carrier Sense.
- Transmission unsuccessful; lost Clear-to-Send.
- Transmission unsuccessful; DMA underrun because the system bus did not keep up with the transmission.
- Transmission unsuccessful; number of collisions exceeded the maximum allowed.

The 82586 checks each incoming frame and reports on the following errors, (if configured to 'Save Bad Frame'):

- CRC error: incorrect CRC in a well aligned frame.
- Alignment error: incorrect CRC in a misaligned frame.
- Frame too short: the frame is shorter than the configured value for minimum frame length.
- Overrun: the frame was not completely placed in memory because the system bus did not keep up with incoming data.
- Out of buffers: no memory resources to store the frame, so part of the frame was discarded.

NETWORK PLANNING AND MAINTENANCE

To perform proper planning, operation, and maintenance of a communication network, the network management entity must accumulate information on network behavior. The 82586 provides a rich set of network-wide diagnostics that can serve as the basis for a network management entity.

Network Activity information is provided in the status of each frame transmitted. The activity indicators are:

- Number of collisions: number of collisions the 82586 experienced in attempting to transmit this frame.
- Deferred transmission: indicates if the 82586 had to defer to traffic on the link during the first transmission attempt.

Statistics registers are updated after each received frame that passes the address filtering, and is longer than the Minimum Frame Length configuration parameter.

- CRC errors: number of frames that experienced a CRC error and were properly aligned.
- Alignment errors: number of frames that experienced a CRC error and were misaligned.
- No-resources: number of correct frames lost due to lack of memory resources.
- Overrun errors: number of frame sequences lost due to DMA overrun.

The 82586 can be configured to Promiscuous Mode. In this mode it captures all frames transmitted on the Network without checking the Destination Address. This is useful in implementing a monitoring station to capture all frames for analysis.

The 82586 is capable of determining if there is a short or open circuit anywhere in the Network using the built in Time Domain Reflectometer (TDR) mechanism.

STATION DIAGNOSTICS

The chip can be configured to External Loopback. The transmitter to receiver interconnection can be placed anywhere between the 82586 and the link to locate faults, for example: the 82586 output pins, the Serial Interface Unit, the Transceiver cable, or in the Transceiver.

The 82586 has a mechanism recognizing the transceiver 'heart beat' signal for verifying the correct operation of the Transceiver's collision detection circuitry.

82586 SELF TESTING

The 82586 can be configured to Internal Loopback. It disconnects itself from the Serial Interface Unit, and any frame transmitted is received immediately. The 82586 connects the Transmit Data to the Receive Data signal and the Transmit Clock to the Receive Clock.

The Dump Command causes the chip to write over 100 bytes of its internal registers to memory.

The Diagnose command checks the exponential Backoff random number generator internal to the 82586.

CONTROLLING THE 82586

The CPU controls operation of the 82586's Command Unit (CU) and Receive Unit (RU) of the 82586 via the System Control Block.

THE COMMAND UNIT (CU)

The Command Unit is the logical unit that executes Action Commands from a list of commands very similar to a CPU program. A Command Block (CB) is associated with each Action Command.

The CU can be modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - CU is not executing a command and is not associated with a CB on the list. This is the initial state.
- SUSPENDED - CU is not executing a command but (different from IDLE) is associated with a CB on the list.
- ACTIVE - CU is currently executing an Action Command, and points to its CB.

The CPU may affect the CU operation in two ways: issuing a CU control Command or setting bits in the COMMAND word of the Action Command.

THE RECEIVE UNIT (RU)

The Receive Unit is the logical unit that receives frames and stores them in memory.

The RU is modeled as a logical machine that takes, at any given time, one of the following states:

- IDLE - RU has no memory resources and is discarding incoming frames. This is the initial RU state.
- NO-RESOURCES - RU has no memory resources and is discarding incoming frames. This state differs from the IDLE state in that RU accumulates statistics on the number of frames it had to discard.
- SUSPENDED - RU has free memory resources to store incoming frames but discards them anyway.
- READY - RU has free memory resources and stores incoming frames.

The CPU may affect RU operation in three ways: issuing an RU Control Command, setting bits in Frame Descriptor, FD, COMMAND word of the frame currently being received, or setting EL bit of Buffer Descriptor, BD, of the buffer currently being filled.

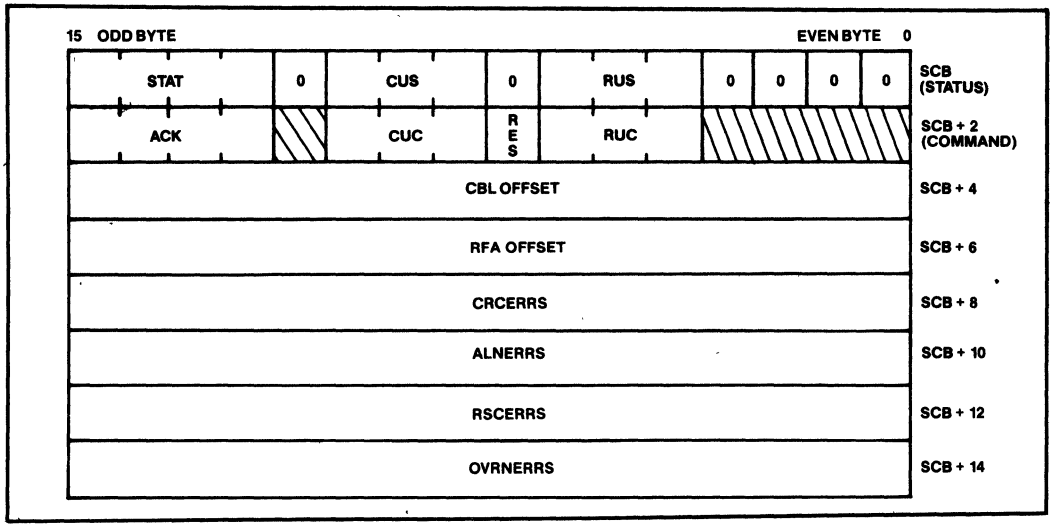


Figure 10. System Control Block (SCB) Format

SYSTEM CONTROL BLOCK (SCB)

The System Control Block is the communication mail-box between the 82586 and the host CPU. The SCB format is shown in Figure 10.

The host CPU for issuing Control Commands to the 82586 via the SCB. These commands may appear at any time during routine operation, as determined by the host CPU. After the required Control Command is setup, the CPU sends a CA signal to the 82586.

SCB is also used by the 82586 to return status information to the host CPU. After inserting the required status bits into SCB, the 82586 issues an Interrupt to the CPU.

The format is as follows:

STATUS word: Indicates the status of the 82586. This word is modified only by the 82586. Defined bits are:

CX	(Bit 15)	- A command in the CBL having its 'I' (interrupt) bit set has been executed.
FR	(Bit 14)	- A frame has been received.
CNR	(Bit 13)	- The Command Unit left the Active state.
RNR	(Bit 12)	- The Receive Unit left the Ready state.
CUS	(Bits 8-10)	- (3 bits) this field contains the status of the Command Unit. Valid values are: 0 - Idle 1 - Suspended 2 - Active 3-7 - Not used
RUS	(Bits 4-6)	- (3 bits) this field contains the status of the Receive Unit. Valid values are: 0 - Idle 1 - Suspended 2 - No Resources 3 - Not used 4 - Ready 5-7 - Not used

COMMAND word: Specifies the action to be performed as a result of the CA. This word is set by the CPU and cleared by the 82586. Defined bits are:

ACK-CX	(Bit 15)	- Acknowledges the command executed event.

ACK-FR	(Bit 14)	- Acknowledges the frame received event.
ACK-CNR	(Bit 13)	- Acknowledges that the Command Unit became not ready.
ACK-RNR	(Bit 12)	- Acknowledges that the Receive Unit became not ready.
CUC	(Bits 8-10)	- (3 bits) this field contains the command to the Command Unit. Valid values are:
	0	- NOP (doesn't affect current state of the unit).
	1	- Start execution of the first command on the CBL. If a command is in execution, then complete it before starting the new CBL. The beginnng of the CBL is in CBL OFFSET.
	2	- Resume the operation of the command unit by executing the next command. This operation assumes that the command unit has been previously suspended.
	3	- Suspend execution of commands on CBL after current command is complete.
	4	- Abort execution of commands immediately.
	5-7	- Reserved, illegal for use.
RUC	(Bits 4-6)	- (3 bits) This field contains the command to the receive unit. Valid values are:
	0	- NOP (does not alter current state of unit).
	1	- Start reception of frames. If a frame is being received, then complete reception before starting. The beginnng of the RFA is contained in the RFA OFFSET.
	2	- Resume frame receiving (only when in suspended state.)
	3	- Suspend frame receiving. If a frame is being received, then complete its reception before suspending.
	4	- Abort receiver operation immediately.
	5-7	- Reserved, illegal for use.
RESET	(Bit 7)	- Reset chip (logically the same as hardware RESET).

CBL-OFFSET:
gives the 16-bit offset address of the first command (Action Command) in the command list to be executed following CU-START. Thus, the 82586 reads this word only if the CUC field contained a CU-START Control Command.

RFA-OFFSET:
Points to the first Receive Frame Descriptor in the Receive Frame Area

CRCERRS:
CRC Errors - contains the number of properly aligned frames received with a CRC error.

ALNERRS:
Alignment Errors - contains the number of misaligned frames received with a CRC error.

RSCERRS:
Resource Errors - records the number of correct incoming frames discarded due to lack of memory resources (buffer space or received frame descriptors).

OVRNERRS:
Overrun Errors - counts the number of received frame sequences lost because the memory bus was not available in time to transfer them.

ACTION COMMANDS

The 82586 executes a 'program' that is made up of action commands in the Command List. As shown in Figure 5, each command contains the command field, status and control fields, link to the next action command in the CL, and any command-specific parameters. This command format is called the Command Block.

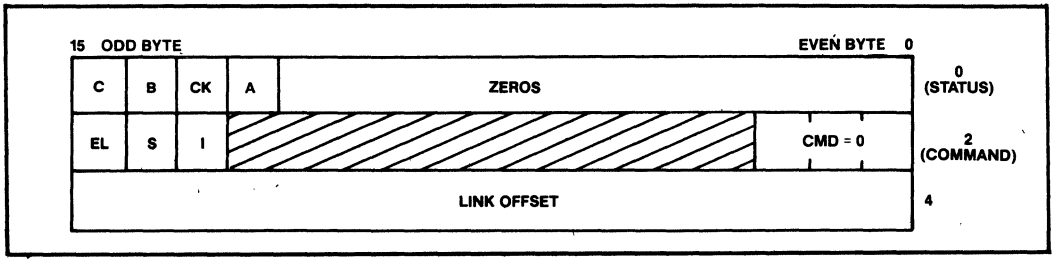


Figure 11. The NOP Command Block

The 82586 has a repertoire of 8 commands:

NOP
Setup Individual Address
Configure
Setup Multicast Address
Transmit
TDR
Diagnose
Dump

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- NOP = 0

LINK OFFSET: Address of next Command Block

NOP

This command results in no action by the 82586, except as performed in normal command processing. It is present to aid in Command List manipulation.

NOP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

IA-SETUP

This command loads the 82586 with the Individual Address. This address is used by the 82586 for recognition of Destination Address during reception and insertion of Source Address during transmission.

The IA-SETUP command includes the following fields:

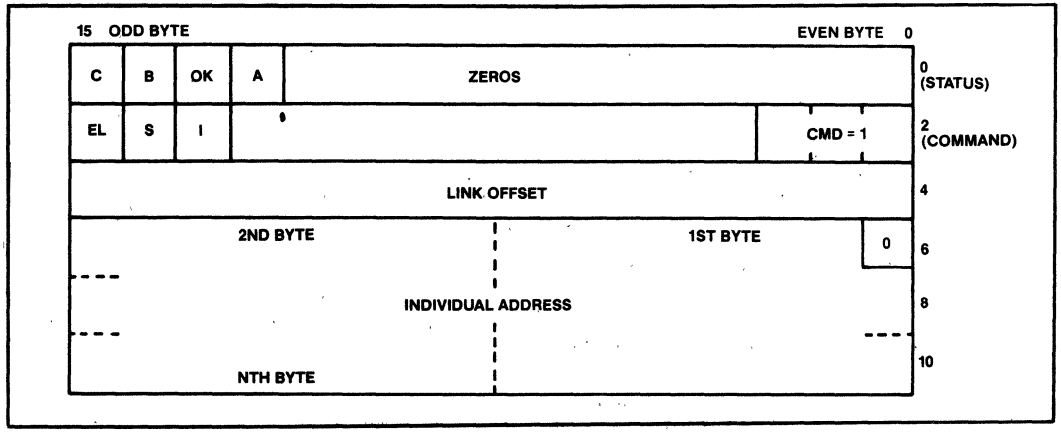


Figure 12. The IA-SETUP Command Block

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- IA-SETUP = 1

LINK OFFSET: Address of next Command Block

INDIVIDUAL ADDRESS: Individual Address parameter

The least significant bit of the Individual Address parameter must be zero for IEEE 802.3/Ethernet. However, no enforcement of 0 is provided by the 82586. Thus, an Individual Address with least significant bit 1, is possible.

CONFIGURE

The CONFIGURE command is used to update the 82586 operating parameters.

The CONFIGURE command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- Configure = 2

LINK OFFSET: Address of next Command Block

Byte 6-7:

BYTE CNT	(Bits 0-3)	- Byte Count, Number of bytes including this one, holding the parameters to be configured. A number smaller than 4 is interpreted as 4. A number greater than 12 is interpreted as 12.

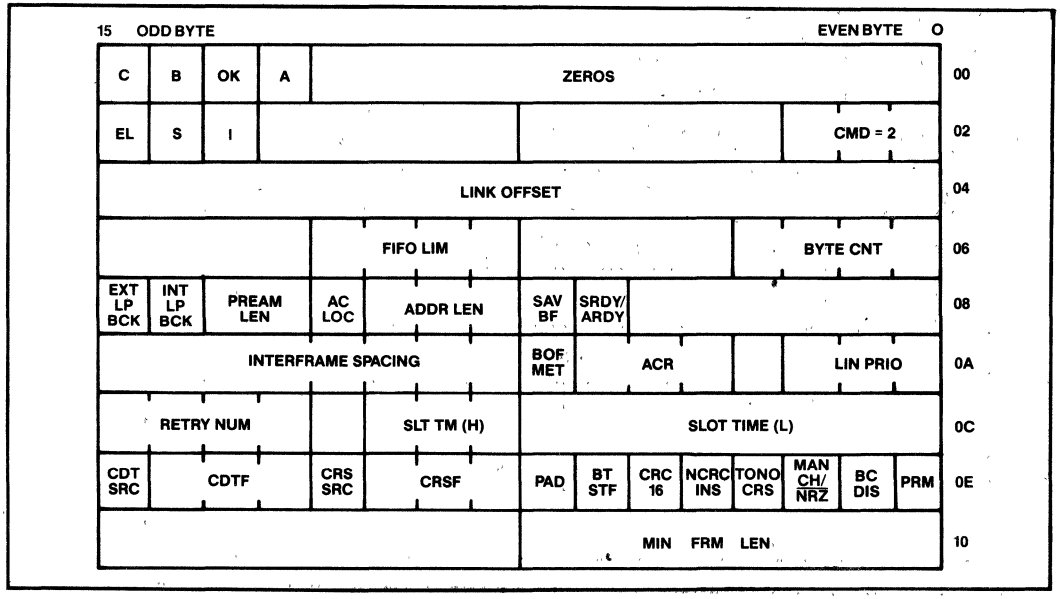


Figure 13. The CONFIGURE Command Block

FIFO-LIM	(Bits 8-11)	- Value of FIFO Threshold.

Byte 8-9:

SRDY/ARDY	(Bit 6)	
	0	- SRDY/$\overline{\text{ARDY}}$ pin operates as ARDY (internal synchronization).
	1	- SRDY/$\overline{\text{ARDY}}$ pin operates as SRDY (external synchronization).
SAV-BF	(Bit 7)	
	0	- Received bad frames are not saved in memory.
	1	- Received bad frames are saved in memory.
ADDR-LEN	(Bits 8-10)	- Number of address bytes. NOTE: 7 is interpreted as 0.
AT-LOC	(Bit 11)	
	0	- Address and Type Fields separated from data and associated with Transmit Command Block or Receive Frame Descriptor. For transmitted Frame, Source Address is inserted by the 82586.
	1	- Address and Type Fields are part of the Transmit/Receive data buffers, including Source Address (which is not inserted by the 82586).
PREAM-LEN	(Bits 12-13)	- Preamble Length including Beginning of Frame indicator: 00 - 2 bytes 01 - 4 bytes 10 - 8 bytes 11 - 16 bytes
INT-LPBCK	(Bit 14)	- Internal Loopback
EXT-LPBCK	(Bit 15)	- External Loopback. NOTE: Bits 14 and 15 configured to 1, cause Internal Loopback.

Byte 10-11:

LIN-PRIO	(Bits 0-2)	- Linear Priority
ACR	(Bits 4-6)	- Accelerated Contention Resolution (Exponential Priority)
BOF-MET	(Bit 7)	- Exponential Backoff Method 0 - IEEE 802.3/Ethernet 1 - Alternate method
INTER-FRAME SPACING	(Bits 8-15)	- Number indicating the Interframe Spacing in TxC period units

Byte 12-13:

SLOT-TIME (L)	(Bits 0-7)	- Slot Time number, low byte
SLT-TM (H)	(Bits 8-10)	- Slot Time number, high bits
RETRY-NUM	(Bits 12-15)	- Maximum number of transmission retries on collisions

Byte 14-15:

PRM	(Bit 0)	- Promiscuous Mode
BC-DIS	(Bit 1)	- Broadcast Disable
MANCH/NRZ	(Bit 2)	- Manchester or NRZ encoding/decoding
	0	- NRZ
	1	- Manchester
TONO-CRS	(Bit 3)	- Transmit on No Carrier Sense
	0	- Cease transmission if $\overline{\text{CRS}}$ goes inactive during frame transmission
	1	- Continue transmission even if no Carrier Sense
NCRC-INS	(Bit 4)	- No CRC Insertion
CRC-16	(Bit 5)	- CRC Type:
	0	- 32 bit Autodin II CRC polynomial
	1	- 16 bit CCITT CRC polynomial
BT-STF	(Bit 6)	Bitstuffing:
	0	- End of Carrier mode (Ethernet)
	1	- HDLC like Bitstuffing mode
PAD	(Bit 7)	- Padding
	0	- No Padding

	1	- Perform padding by transmitting flags for remainder of Slot Time
CRSF	(Bits 8-9)	- Carrier Sense Filter in bit times
CRS-SRC	(Bit 11)	Carrier Sense Source
	0	- External
	1	- Internal
CDTF	(Bits 12-14)	- Collision Detect Filter in bit times
CDT-SRC	(Bit 15)	- Collision Detect Source
	0	- External
	1	- Internal

Byte 16:		
MIN-FRM-LEN	(Bits 0-7)	- Minimum number of bytes in a frame

Table 2. 82586 Default Values

Preamble Length	=	2
Address Length	=	6
Broadcast Disable	=	0
CRC-16/CRC-32	=	0
No CRC Insertion	=	0
Bitstuffing/EOC	=	0
Padding	=	0
Min-Frame-Length	=	64
Interframe Spacing	=	96
Slot Time	=	512
Number of Retries	=	15
Linear Priority	=	0
Accelerated Contention Resolution	=	0
Exponential Backoff Method	=	0
Manchester/NRZ	=	0
Internal CRS	=	0
CRS Filter	=	0
Internal CDT	=	0
CDT Filter	=	0
Transmit On No CRS	=	0
FIFO THRESHOLD	=	8
SRDY/ARDY	=	0
Save Bad Frame	=	0
Address/Type Location	=	0
INT Loopback	=	0
EXT Loopback	=	0
Promiscuous Mode	=	0

CONFIGURATION DEFAULTS

The default values of the configuration parameters are compatible with the IEEE 802.3/Ethernet Standards. RESET configures the 82586 according to the defaults shown in Table 2.

MC-SETUP

This command sets up the 82586 with a set of Multicast Addresses. Subsequently, incoming frames with Destination Addresses from this set are accepted.

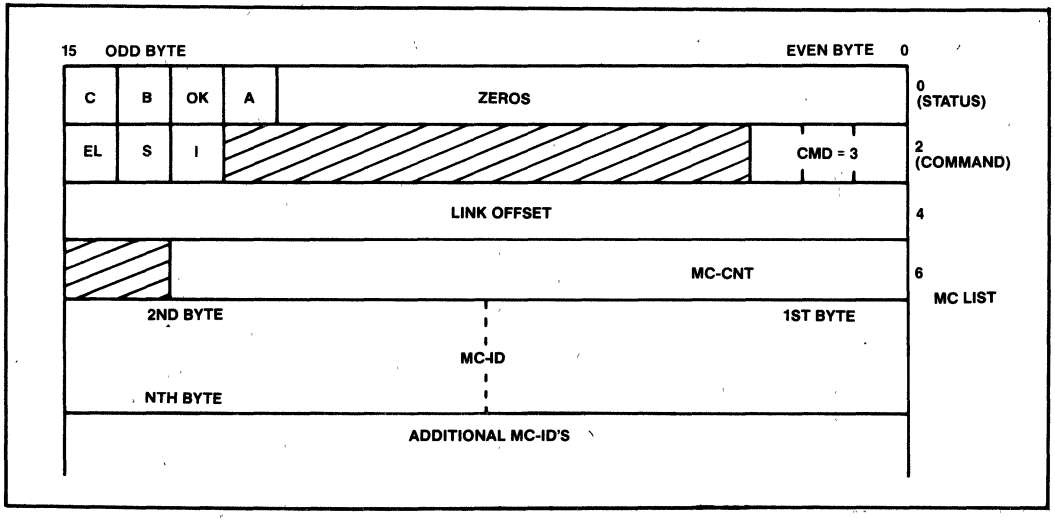


Figure 14. The MC-SETUP Command Block

The MC-SETUP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- MC-SETUP = 3

LINK OFFSET: Address of next Command Block

MC-CNT: A 14-bit field indicating the number of bytes in the MC-LIST field. MC-CNT is truncated to the nearest multiple of Address Length (in bytes). Issuing a MC-SETUP command with MC-CNT=0 disables reception of any incoming frame with a Multicast Address.

MC-LIST: A list of Multicast Addresses to be accepted by the 82586. Note that the most significant byte of an address is followed immediately by the least significant byte of the next address. Note also that the least significant bit of each Multicast Address in the set must be a one.

The Transmit-Byte-Machine maintains a 64-bit HASH table used for checking Multicast Addresses during reception.

An incoming frame is accepted if it has a Destination Address whose least significant bit is a one, and after hashing points to a bit in the HASH table whose value is one. The hash function is selecting bits 2 to 7 of the CRC register. RESET causes the HASH table to become all zeros.

After the Transmit-Byte-Machine reads a MC-SETUP command from TX-FIFO, it clears the HASH table and reads the bytes in groups whose length is determined by the ADDRESS length. Each group is hashed using CRC logic and the bit in the HASH table to which bits 2-7 of the CRC register point is set to one. A group that is not complete has no effect on the HASH table. Transmit-Byte-Machine notifies CU after completion.

TRANSMIT

The TRANSMIT command causes transmission (and if necessary retransmission) of a frame.

TRANSMIT CB includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion
A	(Bit 12)	- Command aborted
S10	(Bit 10)	- No Carrier Sense signal during transmission (between beginning of Destination Address and end of Frame Check Sequence).
S9	(Bit 9)	- Transmission unsuccessful (stopped) due to loss of Clear-to-Send signal.

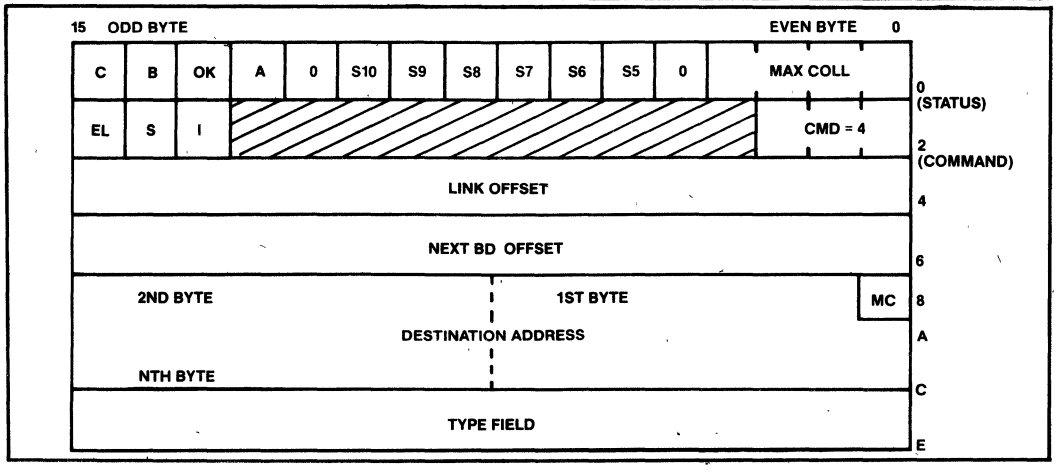


Figure 15. The Transmit Command Block

S8	(Bit 8)	- Transmission unsuccessful (stopped) due to DMA underrun, (i.e. data not supplied from the system for transmission).
S7	(Bit 7)	- Transmission had to Defer to traffic on the link.
S6	(Bit 6)	- Heart Beat, indicates that during Interframe Spacing period after the previous transmission, a pulse was detected on the Collision Detect pin.
S5	(Bit 5)	- Transmission attempt stopped due to number of collisions exceeding the maximum number of retries.
MAX-COLL	(Bits 3-0)	- Number of Collisions experienced by this frame. S5 = 1 and MAX-COLL = 0 indicates that there were 16 collisions.

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- TRANSMIT = 4

LINK OFFSET: Address of next Command Block

TBD OFFSET: Address of list of buffers holding the Information field. TBD-OFFSET = 0FFFFH indicates that there is no Information field.

DESTINATION ADDRESS: Destination Address of the frame.

TYPE FIELD: Type Field of the frame.

STATUS word:

EOF		- Indicates that this is the Buffer Descriptor of the last buffer of this frame's Information Field.
ACT-COUNT	(Bits 0-13)	- Actual number of data bytes in buffer (can be even or odd).

NEXT BD OFFSET: points to next Buffer Descriptor in list. If EOF is set, this field is meaningless.

BUFFER ADDRESS: 24-bit absolute address of buffer.

TIME DOMAIN REFLECTOMETER - TDR

This command performs a Time Domain Reflectometer test on the serial link. By performing the command, the user is able to identify shorts or opens and their location. Along with transmission of 'All Ones,' the 82586 triggers an internal timer. The timer measures the time elapsed from transmission start until 'echo' is obtained. 'Echo' is indicated by Collision Detect going active or Carrier Sense signal drop.

TDR command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

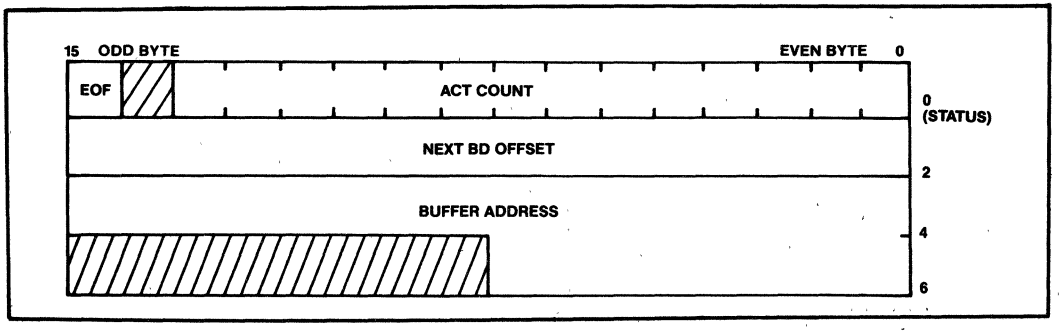


Figure 16. The Transmit Buffer Descriptor

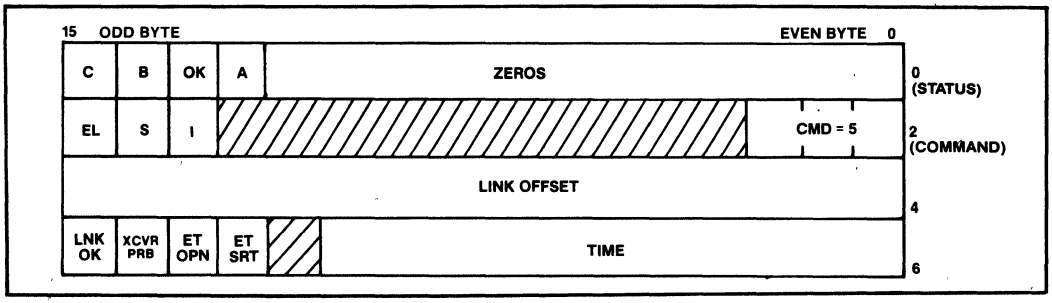


Figure 17. The TDR Command Block

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- TDR = 5

LINK OFFSET: Address of next Command Block

RESULT word:

LNK-OK	(Bit 15)	- No link problem identified
XCVR-PRB	(Bit 14)	- Transceiver Cable Problem identified (valid only in the case of a Transceiver that does not return Carrier Sense during transmission).
ET-OPN	(Bit 13)	- Open on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
ET-SRT	(Bit 12)	- Short on the link identified (valid only in the case of a Transceiver that returns Carrier Sense during transmission).
TIME	(Bits 0-10)	- Specifying the distance to a problem on the link (if one exists) in transmit clock cycles.

DUMP

This command causes the contents of over a hundred bytes of internal registers to be placed in memory. It is supplied as a self diagnostic tool, as well as to supply registers of interest to the user.

DUMP command includes the following fields:

STATUS word (written by 82586):

C	(Bit 15)	- Command completed
B	(Bit 14)	- Busy executing command
OK	(Bit 13)	- Error free completion

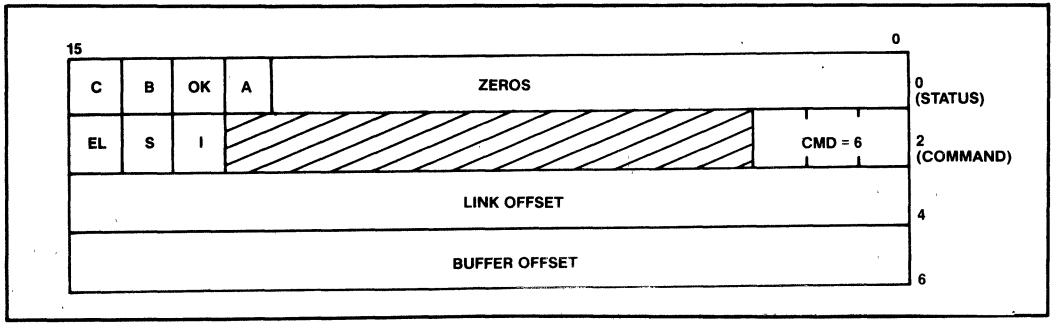


Figure 18. The DUMP Command Block

COMMAND word:

EL	(Bit 15)	- End of command list
S	(Bit 14)	- Suspend after completion
I	(Bit 13)	- Interrupt after completion
CMD	(Bits 0-2)	- DUMP = 6

LINK OFFSET: Address of next Command Block

BUFFER OFFSET: This word specifies the offset portion of the memory address which points to the top of the buffer allocated for the dumped registers contents. The length of the buffer is 170 bytes.

DUMP AREA FORMAT

Figure 18 shows the format of the DUMP area. The fields are as follows:

Bytes 00H to 0AH: These bytes correspond to the 82586 CONFIGURE command field (except bit 6 of the first word).

Bytes 0CH to 11H: The Individual Address Register content. IARO is the Individual Address least significant byte.

Bytes 12H to 13H: Status word of last command block (only bits 0-13).

Bytes 14H to 17H: Content of the Transmit CRC generator. TXCRCRO is the least significant byte. The contents are dependent on the activity before the DUMP command:

After RESET - 'All Ones.'

After successful transmission - 'All Zeros.'

After MC-SETUP command - Generated CRC value of the last MC address, on MC-LIST.

After unsuccessful transmission, depends on where it stopped.

NOTE

For 16-bit CRC only TXCRCRO and TXCRCR1 are valid.

Bytes 18H to 1BH: Contents of Receive CRC Checker. RXCRCRO is the least significant byte. The contents are dependent on the activity performed before the DUMP command:

After RESET - 'All Ones.'

After good frame reception -

1. For CRC-CCITT - 0D1F0H.
2. For CRC-Autodin-II - 7C40DD7BH.

After Bad Frame reception - corresponds to the received information.

After reception attempt, i.e. unsuccessful check for address match, corresponds to the CRC performed on the frame address.

NOTE

Any frame on the serial link modifies this register contents.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
				FIFO LIM				0	DISC MODE	0	0	0	0	0	0	00
EXT LP BCK	INT LP BCK	PREAM LEN			AT LOC	ADDR LEN		SAV BP	TRNSV AWDV	1	1	1	1	1	1	02
INTERFRAME SPACING								EBOF NET	ACR			1	LIN PRIO			04
RETRY NUM				1	SLT TM [H]			SLOT TIME [L]								06
CDT SRC	CDT F			CRS SRC	CRS F			PAD	BT STF	CRC 16	NCRC INS	TONO CRS	MAN BAC OFF	BC DIS	PRM	08
1	1	1	1	1	1	1	1	MIN PAC LEN								0A
IAR1								IAR0								0C
IAR3								IAR2								0E
IAR5								IAR4								10
RCNT COL	0	TX OK	0	0	LST CRS	LST CTS	URN	TX DEF	HRT BT	MAX COL	0	COLL NUM				12
TXCRCR 1								TXCRCR 0								14
TXCRCR 3								TXCRCR 2								16
RXCRCR 1								RXCRCR 0								18
RXCRCR 3								RXCRCR 2								1A
TEMPR 1								TEMPR 0								1C
TEMPR 3								TEMPR 2								1E
TEMPR 5								TEMPR 4								20
1	0	RX OK	1	CRC ERR	ALN ERR	0	OVRN	SHRT PKT	NO EOP	1	1	1	1	1	1	22
HASHR 1								HASHR 0								24
HASHR 3								HASHR 2								26
HASHR 5								HASHR 4								28
HASHR 7								HASHR 6								2A
LNK OK	XCVR PRB	ET OPN	ET SRT													2C
1	1	1		1	1											2E
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3A
																3C
0	0	0	0	0	0	0	0	0	0	0	0	0	ADR LEN			3E
EL				NXT RB SIZE												40

Figure 19. The DUMP Area

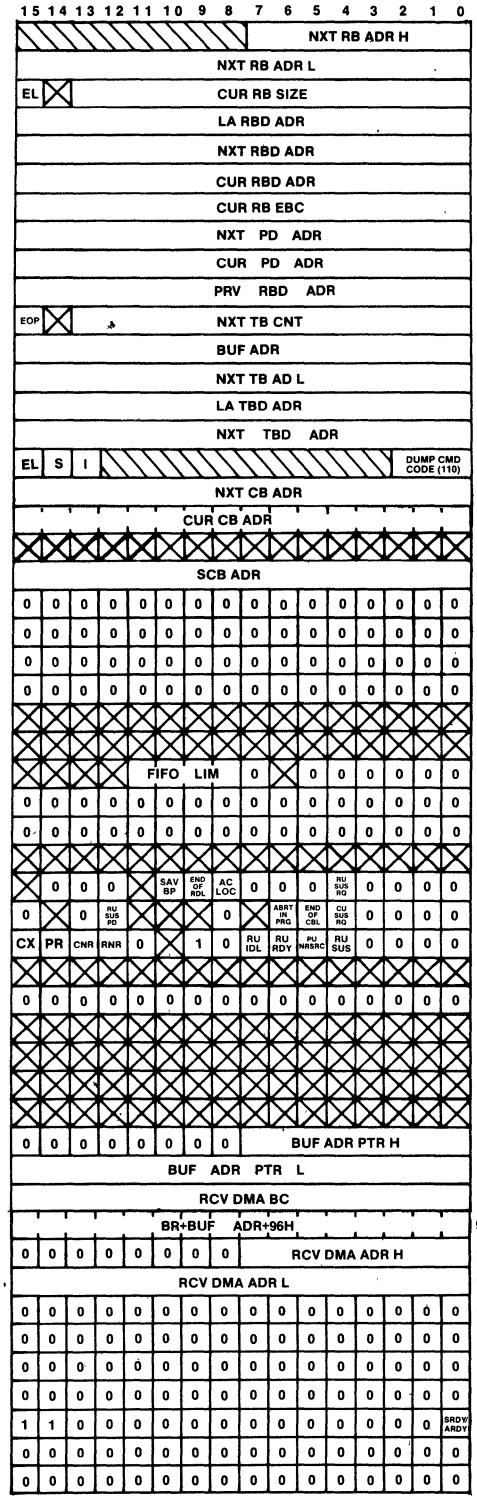


Figure 19. DUMP Area (con't)

Bytes 1CH to 21H: Temporary Registers.

Bytes 22H to 23H: Receive Status Register. Bits 6,7,8,10,11 and 13 assume the same meaning as corresponding bits in the Receive Frame Descriptor Status field.

Bytes 24H to 2BH: HASH TABLE.

Bytes 2CH to 2DH: Status bits of the last time TDR command that was performed.

NXT-RB-SIZE: Let N be the last buffer of the last received frame, then NXT-RB-SIZE is the number of bytes of available in the N+1 buffer. EL - The EL bit of the Receive Buffer Descriptor.

NXT-RB-ADR: Let N be the last Receive Buffer used, then NXT-RB-ADR is the BUFFER-ADDRESS field in the N+1 Receive-Buffer Descriptor, i.e. the pointer to the N+1 Receive Buffer.

CUR-RB-SIZE: The number of bytes in the last buffer of the last received frame. EL - The EL bit of the last buffer in the last received frame.

LA-RBD-ADR: Look Ahead Buffer Descriptor, i.e. the pointer to N+2 Receive Buffer Descriptor.

NXT-RBD-ADR: Next Receive Buffer Descriptor Address. Similar to LA-RBD-ADR but points to N+1 Receive Buffer Descriptor.

CUR-RBD-ADR: Current Receive Buffer Descriptor Address. Similar to LA-RBD-ADR, but points to Nth Receive Buffer Descriptor.

CUR-RB-EBC: Current Receive Buffer Empty Byte Count. Let N be the currently used Receive Buffer. Then CUR-RB-EBC indicates the Empty part of the buffer, i.e. the ACT-COUNT of buffer N is given by the difference between its SIZE and the CUR-RB-EBC.

NXT-FD-ADR: Next Frame Descriptor Address. Define N as the last Receive Frame Descriptor with bits C=1 and B=0, then NXT-FD-ADR is the address of N+2 Receive Frame Descriptor (with B=C=0) and is equal to the LINK-ADDRESS field in N+1 Receive Frame Descriptor.

CUR-FD-ADR: Current Frame Descriptor Address. Similar to next NXT-FD-ADR but refers to N+1 Receive Frame Descriptor (with B=1, C=0).

Bytes 54H to 55H: Temporary register.

NXT-TB-CNT: Next Transmit Buffer Count. Let N be the last transmitted buffer of the TRANSMIT command executed recently, the NXT-TB-CNT is the ACT-COUNT field in the Nth Transmit Buffer Descriptor. EOF - Corresponds to the EOF bit of the Nth Transmit Buffer Descriptor. EOF=1 indicates that the last buffer accessed by the 82586 during Transmit was the last Transmit Buffer in the data buffer chain associated with the Transmit Command.

BUF-ADR: Buffer Address. The BUF-PTR field in the DUMP-STATUS Command Block.

NXT-TB-AD-L: Next Transmit Buffer Address Low. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then NXT-TB-AD-L are the two least significant bytes of the Nth buffer address.

LA-TBD-ADR: Look Ahead Transmit Buffer Descriptor Address. Let N be the last Transmit Buffer in the transmit buffer chain of the TRANSMIT Command performed recently, then LA-TBD-ADR is the NEXT-BD-ADDRESS field of the Nth Buffer Descriptor.

NXT-TBD-ADR: Next Transmit Buffer Descriptor Address. Similar in function to LA-TBD-ADR but related to Transmit Buffer Descriptor N-1. Actually, it is the address of Transmit Buffer Descriptor N.

Bytes 60H, 61H: This is a copy of the 2nd word in the DUMP-STATUS command presently executing.

NXT-CB-ADR: Next Command Block Address. The LINK-ADDRESS field in the DUMP Command Block presently executing. Points to the next command.

CUR-CB-ADR: Current Command Block Address. The address of the DUMP Command Block currently executing.

SCB-ADR: Offset of the System Control Block (SCB).

Bytes 7EH, 7FH:

RU-SUS-RQ (Bit 4) - Receive Unit Suspend Request.

Bytes 80H, 81H:

CU-SUS-RQ (Bit 4) - Command Unit Suspend Request

END-OF-CBL (Bit 5) - End of Command Block List. If '1' indicates that DUMP-STATUS is the last command in the command chain.

ABRT-IN-PROG (Bit 6) - Command Unit Abort Request.

RU-SUS-FD (Bit 12) - Receive Unit Suspend Frame Descriptor Bit. Assume N is the Receive Frame Descriptor used recently, then RU-SUS-FD is equivalent to the S bit of N+1 Receive Frame Descriptor.

Bytes 82H, 83H:

RU-SUS (Bit 4) - Receive Unit in SUSPENDED state.

RU-NRSRC (Bit 5) - Receive Unit in NO RESOURCES state.

RU-RDY (Bit 6) - Receive Unit in READY state.

RU-IDL (Bit 7) - Receive Unit in IDLE state.

RNR (Bit 12) - RNR Interrupt In Service bit.

CNR (Bit 13) - CNR Interrupt In Service bit.

FR (Bit 14) - FR Interrupt In Service bit.

CX (Bit 15) - CX Interrupt In Service bit.

Bytes 90H to 93H:

BUF-ADR-PTR - Buffer pointer is the absolute address of the bytes following the DUMP Command block.

Bytes 94H to 95H:

RCV-DMA-BC - Receive DMA Byte Count. This field contains number of bytes to be transferred during the next Receive DMA operation. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0 then RCV-DMA-BC = (2 times ADDR-LEN plus 2) if the next Receive Frame Descriptor has already been fetched.

2. If AT-LOCation = 1 then it contains the size of the next Receive Buffer.

BR+BUF-PTR+96H - Sum of Base Address plus BUF-PTR field and 96H.

RCV-DMA-ADR - Receive DMA absolute Address. This is the next RCV-DMA start address. The value depends on AT-LOCation configuration bit.

1. If AT-LOCation = 0, then RCV-DMA-ADR is the Destination Address field located in the next Receive Frame Descriptor.

2. If AT-LOCation = 1, then RCV-DMA-ADR is the next Receive Data Buffer Address.

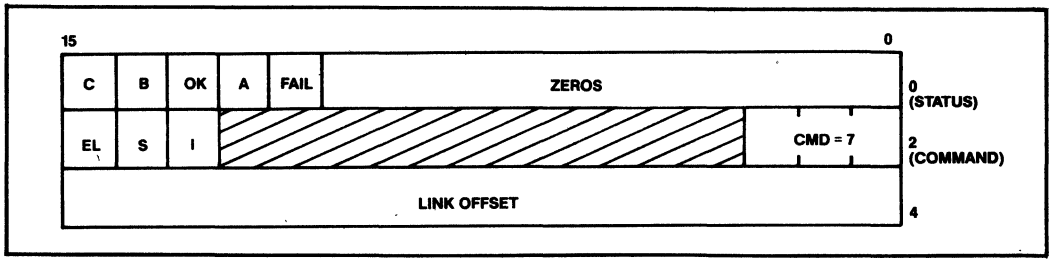


Figure 20. The DIAGNOSE Command Block

The following nomenclature has been used in the DUMP table:

0	- The 82586 writes zero in this location.
1	- The 82586 writes one in this location.
X	- The 82586 writes zero or one in this location.
///	- The 82586 copies this location from the corresponding position in the memory structure.

DIAGNOSE

The DIAGNOSE Command triggers an internal self test procedure of backoff related registers and counters.

The DIAGNOSE command includes the following:

STATUS word (written by 82586):

C	(bit 15)	- Command completed
B	(bit 14)	- Busy executing command
OK	(bit 13)	- Error free completion
FAIL	(bit 11)	- Indicates that the self test procedure failed

COMMAND word:

EL	(bit 15)	- End of command list
S	(bit 14)	- Suspend after completion
I	(bit 13)	- Interrupt after completion
CMD	(bits 0-2)	- DIAGNOSE = 7

LINK OFFSET: Address of next Command Block

RECEIVE FRAME AREA (RFA)

The Receive Frame Area, RFA, is prepared by the host CPU. data is placed into the RFA by the 82586 as frames are received. RFA consists of a list of Receive Frame Descriptors (FD), each of which is associated with a frame. RFA-OFFSET field of SCB points to the first FD of the chain; the last FD is identified by the End-of-List flag (EL). See Figure 21.

FRAME DESCRIPTOR (FD) FORMAT

The FD includes the following fields:

STATUS word (set by the 82586):

C	(bit 15)	- Completed storing frame.
B	(bit 14)	- FD was consumed by RU.
OK	(bit 13)	- Frame received successfully. If this bit is set, then all others will be reset; if it is reset, then the other bits will indicate the nature of the error.
S11	(bit 11)	- Received frame experienced CRC error.
S10	(bit 10)	- Received frame experienced an alignment error.
S9	(bit 9)	- RU ran out of resources during reception of this frame.
S8	(bit 8)	- RCV-DMA overrun.
S7	(bit 7)	- Received frame had fewer bits than configured Minimum Frame Length.
S6	(bit 6)	- No EOF flag detected (only when configured to Bitstuffing).

COMMAND word:

EL	(bit 15)	- Last FD in the list.
S	(bit 14)	- RU should be suspended after receiving this frame.

LINK OFFSET: Address of next FD in list.

RBD-OFFSET (initially prepared by the CPU and later may be updated by 82586): Address of the first RBD that represents the Information Field. RBD-OFFSET = 0FFFFH means there is no Information Field.

DESTINATION ADDRESS (written by 82586): Contains Destination Address of received frame. The length in bytes, it is determined by the Address Length configuration parameter.

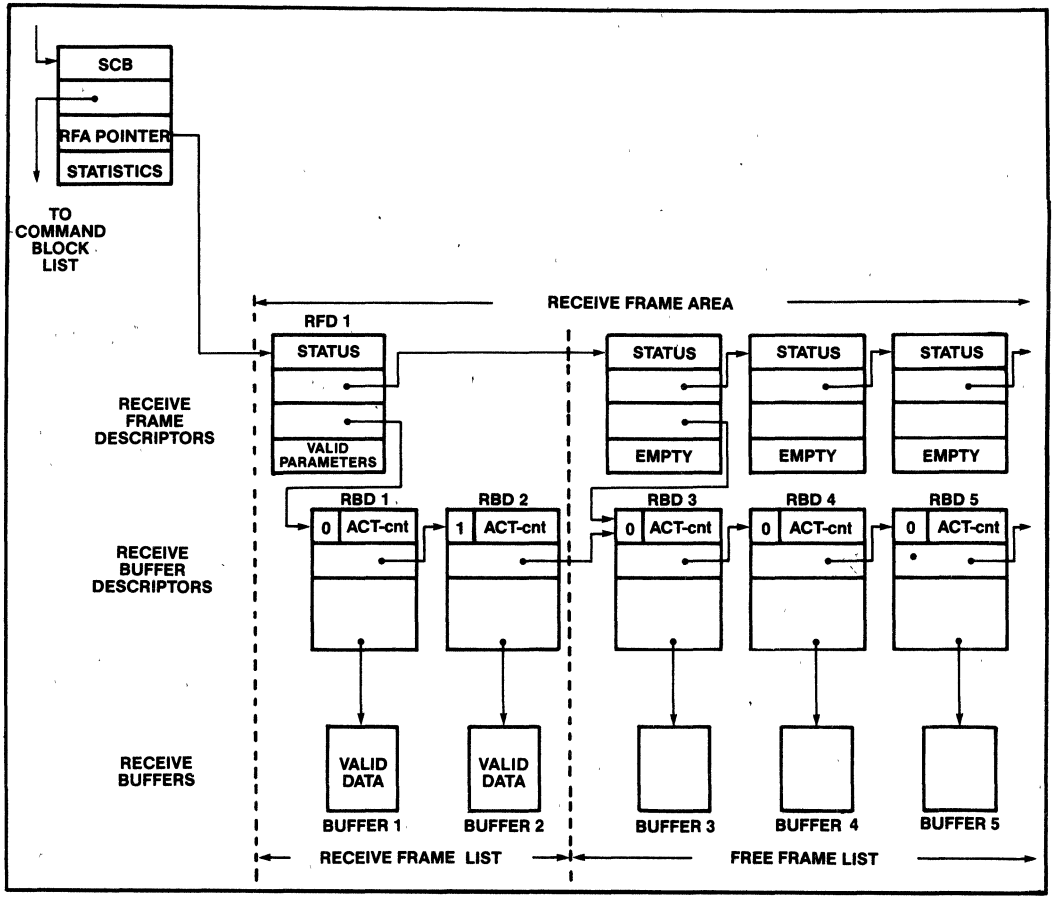


Figure 21. The Receive Frame Area

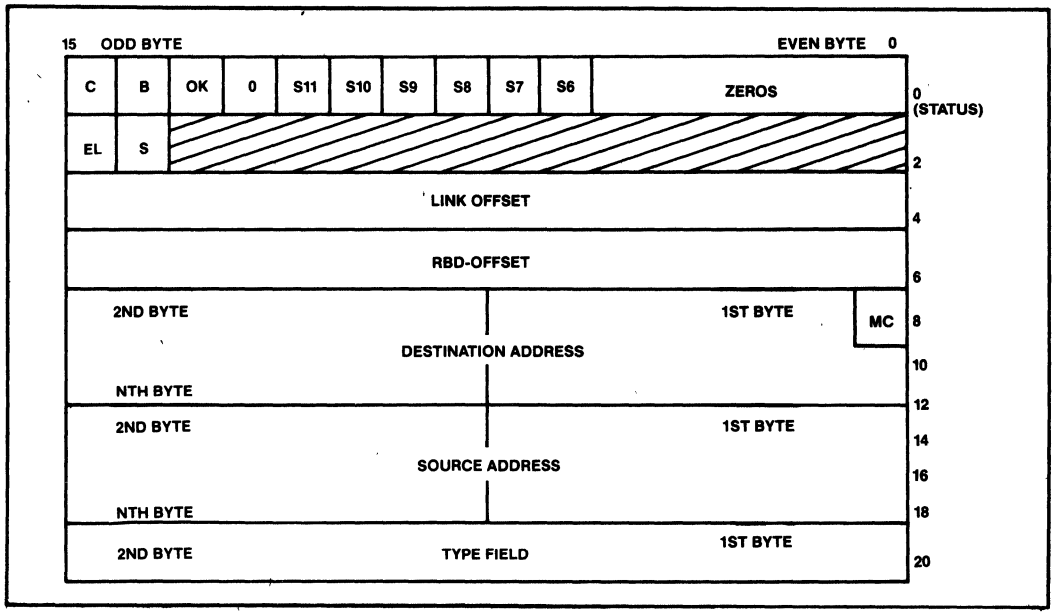


Figure 22. The Frame Descriptor (FD) Format

SOURCE ADDRESS (written by 82586): Contains Source Address of received frame. Its length is the same as DESTINATION ADDRESS

TYPE FIELD (written by 82586): Contains the 2 byte Type Field of received frame.

RECEIVE BUFFER DESCRIPTOR FORMAT

The Receive Buffer Descriptor (RBD) holds information about a buffer; size and location, and the means for forming a chain of RBDs, (forward pointer and end-of-frame indication).

The Buffer Descriptor contains the following fields:

STATUS word (written by the 82586):

EOF	(bit 15)	- Last buffer in received frame.
F	(bit 14)	- ACT COUNT field is valid.
ACT COUNT	(bits 0-13)	- Number of bytes in the buffer that are actually occupied.

NEXT RBD OFFSET: Address of next BD in list of BD's.

BUFFER ADDRESS: 24-bit absolute address of buffer.

EL/SIZE:

EL	(bit 15)	- Last BD in list.
SIZE	(bits 0-13)	- number of bytes the buffer is capable of holding.

ELECTRICAL AND TIMING CHARACTERISTICS

PLEASE NOTE:

The following specifications are preliminary values and are subject to change without notice. Contact your local Intel Sales Office for the latest specifications.

SYSTEM INTERFACE A.C. TIMING CHARACTERISTICS

T_A=0-70° C, V_{CC}=5V±10%

Figure 24 and Figure 25 define how the measurements should be done:

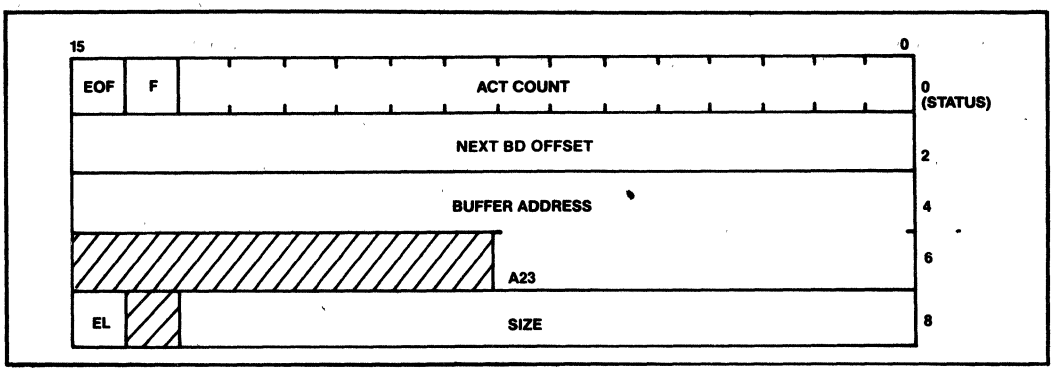


Figure 23. The Receive Buffer Descriptor (RBD) Format

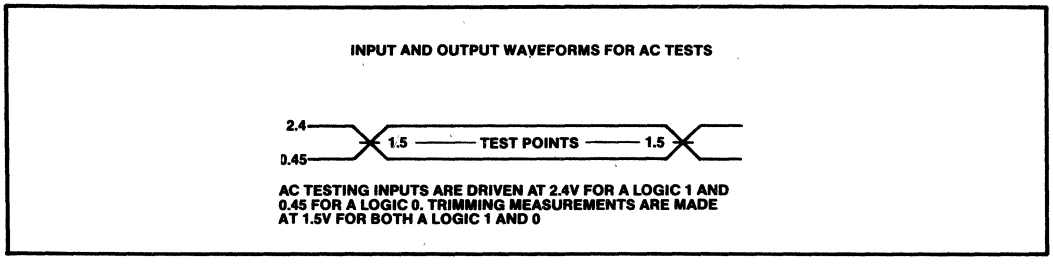


Figure 24. TTL Input/Output Voltage Levels For Timing Measurements

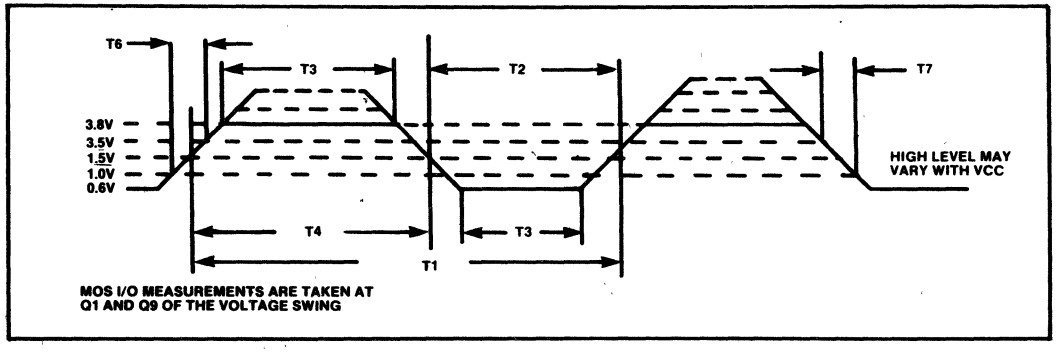


Figure 25. System Clock MOS Input Voltage Levels for Timing Measurements

D.C. CHARACTERISTICS

T_A = 0-70°C, V_{CC} = 5V ± 10% CLK, TxD, $\overline{TxC}$, RxD, $\overline{RxC}$ have MOS levels (see V_{MIL}, V_{MIH}, V_{MOL}, V_{MOH}). All other signals have TTL levels (see V_{IL}, V_{IH}, V_{OL}, V_{OH}).

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage (TTL)	-0.5	+0.8	V	
V_{IH}	Input high Voltage (TTL)	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage (TTL)		0.45	V	I_{OL}=2.5mA
V_{OH}	Output High Voltage (TTL)	2.4		V	I_{OH}=-400uA
V_{MIL}	Input Low Voltage (MOS)	-0.5	0.6	V	
V_{MIH}	Input High Voltage (MOS)	3.9	V_{CC}+0.5	V	
V_{MOL}	Output Low Voltage (MOS)		0.45	V	I_{OL}=2.5mA
V_{MOH}	Output High Voltage	V_{CC}-0.5		V	I_{OH}=-400uA
I_{LI}	Input Leakage Current		±10	uA	$0 \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	uA	$0.45 \leq V_{OUT} \leq V_{CC}$
C_{IN}	Capacitance of Input Buffer		10	pF	FC=1MHz
C_{OUT}	Capacitance of Output Buffer		20	pF	FC=1MHz
I_{CC}	Power Supply		450	mA	T_A = 0 deg. C

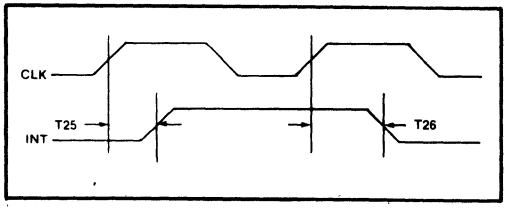


Figure 26. INT Output Timing

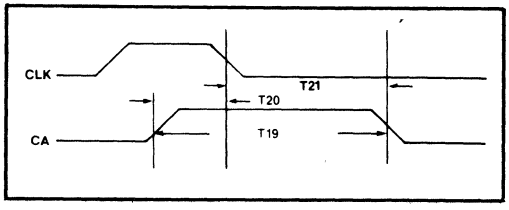


Figure 27. CA Input Timing

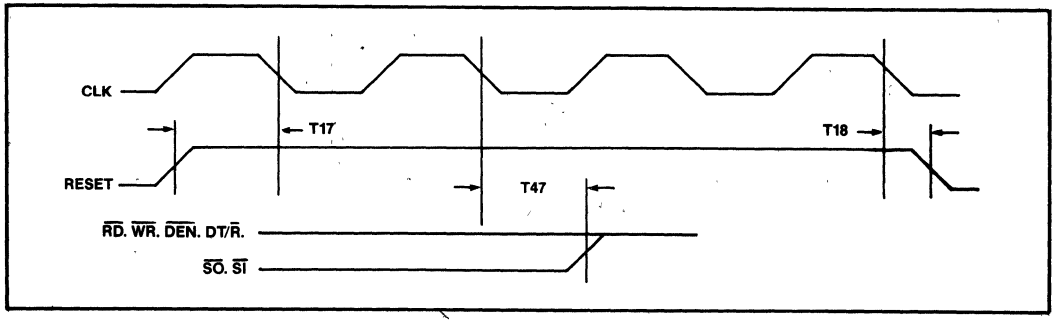


Figure 28. RESET Timing

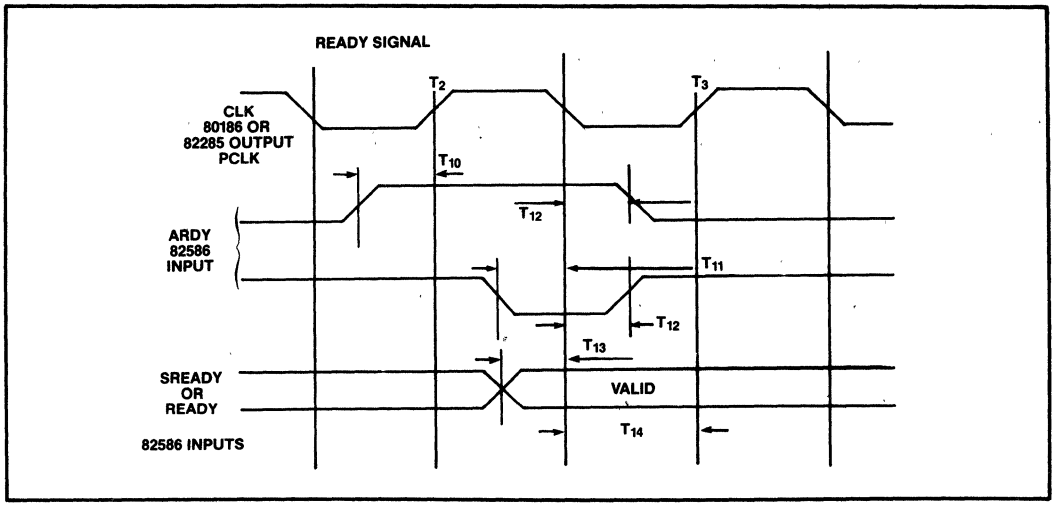


Figure 29. ARDY and SRDY Timings Relative to CLK

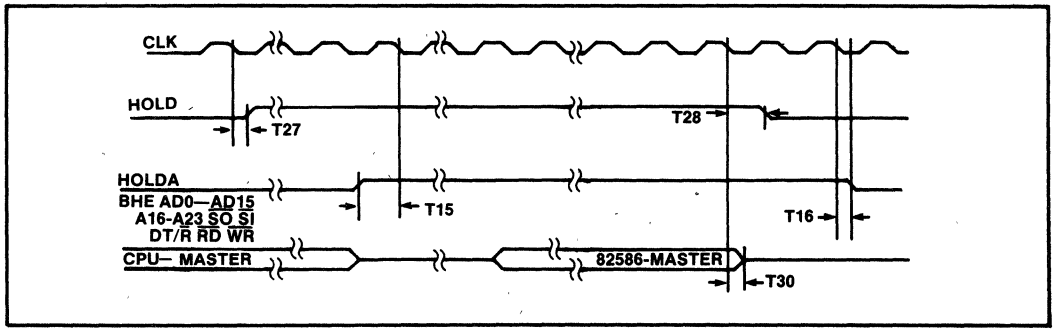


Figure 30. HOLD/HLDA Timing Relative to CLK

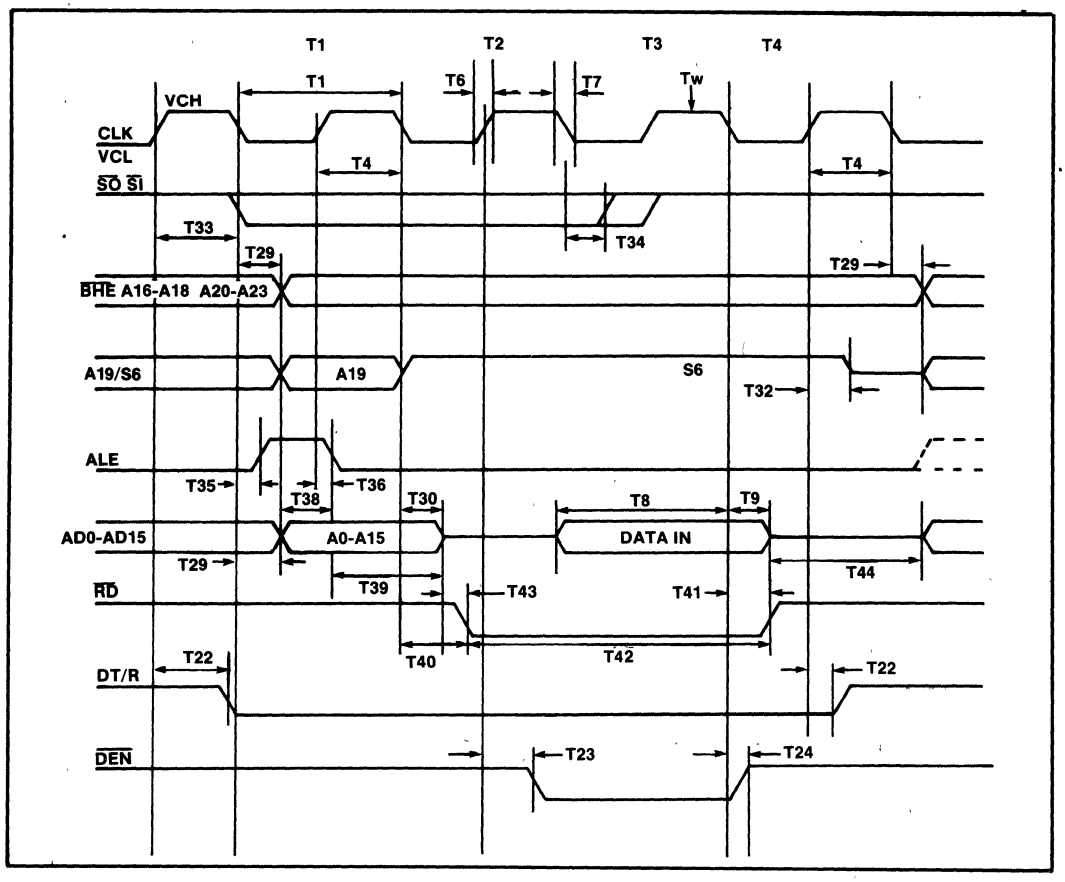


Figure 31. Read Cycle Timing

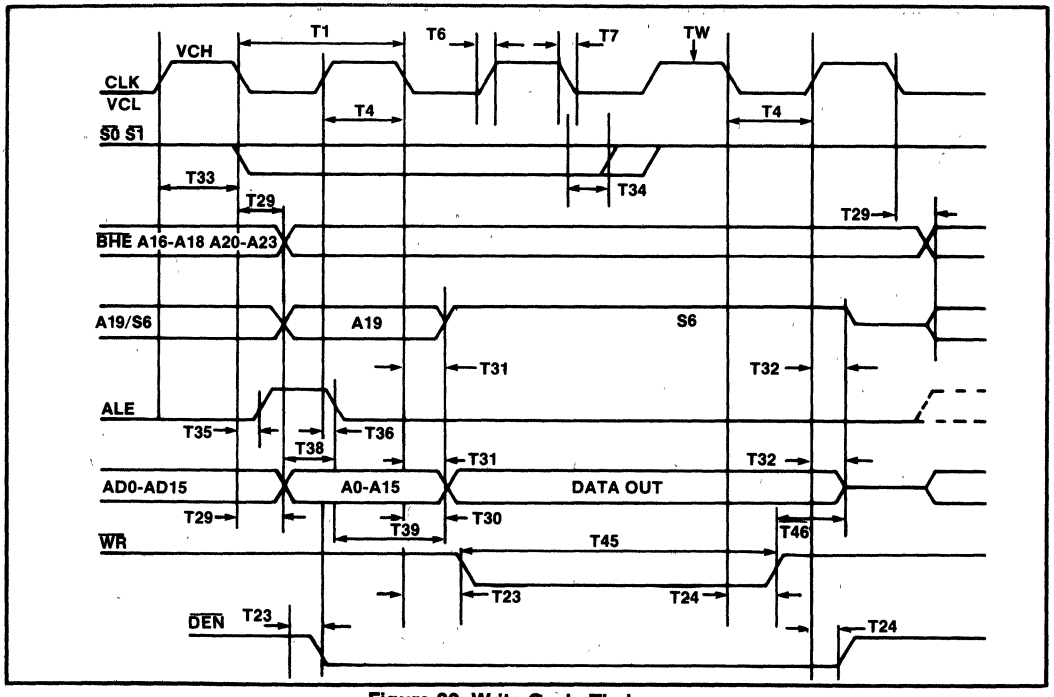


Figure 32. Write Cycle Timing

INPUT TIMING REQUIREMENTS (8MHz)*

Symbol	Parameter	Min.	Max.	Comments
T1	CLK cycle period	125	2000	
T2	CLK low time at 1.5V	53	1000	
T3	CLK low time at 0.6V	42.5	1000	
T4	CLK high time at 1.5V	53		
T5	CLK high time at 3.8V	42.5		
T6	CLK rise time		15	Note 1
T7	CLK fall time		15	Note 2
T8	Data in setup time	20		
T9	Data in hold time	10		
T10	Async RDY active setup time	20		Note 3
T11	Async RDY inactive setup time	35		Note 3
T12	Async RDY hold time	15		Note 3
T13	Synchronous ready/active setup	35		
T14	Synchronous ready hold time	0		
T15	HLDA setup time	20		Note 3
T16	HLDA hold time	10		Note 3
T17	Reset setup time	20		Note 3
T18	Reset hold time	10		Note 3
T19	CA pulse width	1 T1		
T20	CA setup time	20	15	Note 3
T21	CA hold time	10		Note 3

OUTPUT TIMINGS (8 MHz)*,

Symbol	Parameter	Min.	Max.	Comments
T22	DT/R valid delay	0	60	
T23	WR, DEN active delay	0	70	
T24	WR, DEN inactive delay	0	65	
T25	Int. active delay	0	85	Note 4
T26	Int. inactive delay	0	85	Note 4
T27	Hold active delay	0	85	Note 4
T28	Hold inactive delay	0	85	Note 4
T29	Address valid delay	0	60	
T30	Address float delay	0	50	
T31	Data valid delay	0	60	
T32	Data hold Time	0		
T33	Status active delay	0	60	
T34	Status inactive delay	0	70	
T35	ALE active delay	0	45	Note 5
T36	ALE inactive delay	0	45	Note 5
T37	ALE width	T2-10		Note 5
T38	Address valid to ALE low	T2-30		
T39	Address hold to ALE inactive	T7-10		
T40	$\overline{RD}$ active delay	0	95	
T41	$\overline{RD}$ inactive delay	0	70	
T42	$\overline{RD}$ width	2T1-50		
T43	Address float to $\overline{RD}$ active	0		
T44	$\overline{RD}$ inactive to Address active	T1-40		
T45	$\overline{WR}$ width	2T1-40		
T46	Data hold after $\overline{WR}$	T2-25		
T47	Control inactive after reset	0	60	Note 6

*All units are in ns.
**CL on all outputs is 20-200 pF unless otherwise specified.

NOTE LIST:

1 — 1.0V to 3.5V
2 — 3.5V to 1.0V
3 — to guarantee recognition at next clock
4 — CL = 50 pF
5 — CL = 100 pF
6 — Affects:
MIN MODE: $\overline{RD}$, $\overline{WR}$, DT/$\overline{R}$, $\overline{DEN}$
MAX MODE: S0, S1

SERIAL INTERFACE A.C. TIMING CHARACTERISTICS

CLOCK SPECIFICATION

Applies for $\overline{TxC}$, $\overline{RxC}$
f min = 100KHz 10 MHz ±100 ppm
f max = 10 MHz ±100 ppm
for Manchester, symmetry is needed:

$$T51, T52 = \frac{1}{2f} \pm 5\%$$

A.C. CHARACTERISTICS

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION*

Symbol	Parameter	Min.	Max.	Comments
TRANSMIT CLOCK PARAMETERS				
T48	$\overline{TxC}$ Cycle	100	1000	Notes 1, 2
T48	$\overline{TxC}$ Cycle	100		Notes 1, 3
T49	$\overline{TxC}$ Rise Time		5	Note 1
T50	$\overline{TxC}$ Fall Time		5	Note 1
T51	$\overline{TxC}$ High Time	44	1000	Note 1
T52	$\overline{TxC}$ Low Time	40		Notes 1, 4
TRANSMIT DATA PARAMETERS				
T53	TxD Rise Time		10	Notes 1, 5
T54	TxD Fall Time		10	Notes 1, 5
T55	TxD Transition - Transition	35		Notes 1, 2, 5
T56	$\overline{TxC}$ Low to TxD Valid		40	Notes 1, 3, 5
T57	$\overline{TxC}$ Low to TxD Transition		40	Notes 1, 2, 5
T58	$\overline{TxC}$ High to TxD Transition		40	Notes 1, 2, 5
T59	$\overline{TxC}$ Low to TxD High at the Transmission end		40	Notes 1, 5
REQUEST TO SEND/CLEAR TO SEND PARAMETERS				
T60	$\overline{TxC}$ Low to $\overline{RTS}$ Low. Time to Activate $\overline{RTS}$		45	Note 6
T61	$\overline{CTS}$ Valid to $\overline{TxC}$ Low. $\overline{CTS}$ Set-Up Time	45		Note 6
T62	$\overline{TxC}$ Low to $\overline{CTS}$ Invalid. $\overline{CTS}$ Hold Time	20		Notes 6, 7
T63	$\overline{TxC}$ Low to $\overline{RTS}$ High. time to deactivate $\overline{RTS}$		45	Note 6
RECEIVE CLOCK PARAMETERS				
T64	$\overline{RxC}$ Clock Cycle	100		Notes 1, 3
T65	$\overline{RxC}$ Rise Time		5	Note 1
T66	$\overline{RxC}$ Fall Time		5	Note 1
T67	$\overline{RxC}$ High Time	40	1000	Note 1
T68	$\overline{RxC}$ Low Time	44		Note 1
RECEIVE DATA PARAMETERS				
T69	RxD Setup Time	30		Note 1
T70	RxD Hold Time	30		Note 1
T71	RxD Rise Time		10	Note 1
T72	RxD Fall Time		10	Note 1

*All units are in ns.

TRANSMIT AND RECEIVE TIMING PARAMETER SPECIFICATION* (cont'd.)

Symbol	Parameter	Min.	Max.	Comments
CARRIER SENSE/COLLISION DETECT PARAMETERS				
T73	$\overline{CDT}$ Valid to $\overline{TxC}$ Low Ext. Collision Detect Setup Time	30		Note 12
T74	$\overline{TxC}$ Low to $\overline{CDT}$ Inactive. $\overline{CDT}$ Hold Time	20		Note 12
T75	$\overline{CDT}$ Low to Jamming Start			Note 8
T76	$\overline{CRS}$ Valid to $\overline{TxC}$ Low Ext. Carrier Sense Setup time	30		Note 12
T77	$\overline{TxC}$ Low to $\overline{CRS}$ Inactive.$\overline{CRS}$ Hold Time	20		Note 12
T78	$\overline{CRS}$ Low to Jamming Start			Note 9
T79	Jamming Period			Note 10
T80	$\overline{CRS}$ Inactive Setup Time to $\overline{RxC}$ High. End of Receive Pkt.	60		
T81	$\overline{CRS}$ Inactive Hold Time to $\overline{RxC}$ High. End of Receive Pkt.	10		
INTERFRAME SPACING PARAMETERS				
T82	Inter Frame Delay			Note 11

*All units are in ns.

NOTES:

1 — MOS levels.
2 — Manchester only.
3 — NRZ only.
4 — Manchester requires 50% Duty Cycle.
5 — 1 TTL Load + 50 pF.
6 — 1 TTL Load + 100 pF.
7 — Abnormal End of Transmission. $\overline{CTS}$ Expires Before $\overline{RTS}$.
8 — Programmable value:
T75 = NCDF × T48 + (12.5 to 23.5) × T48 if collision occurs after preamble.
NCDF — The Collision Detection Filter Configuration Value.
9 — Programmable value:
T78 = NCSF × T48 + (12.5 to 23.5) × T48.
NCSF — The Carrier Sense Filter Configuration Value.
TBD is a function of Internal/External Carrier Sense Bit.
10 — T79 = 32 × T48.
11 — Programmable value:
T88 = NIFS × T48.
NIFS — the IFS Configuration Value.
If NIFS is less than 31, then NIFS is enforced to 32.
*12 — To guarantee recognition on the next clock.

A.C. TIMING CHARACTERISTICS

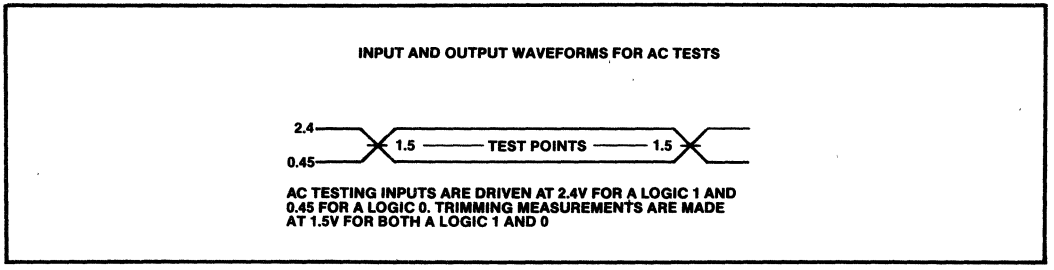


Figure 33. TTL Input/Output Voltage Levels for Timing Measurements

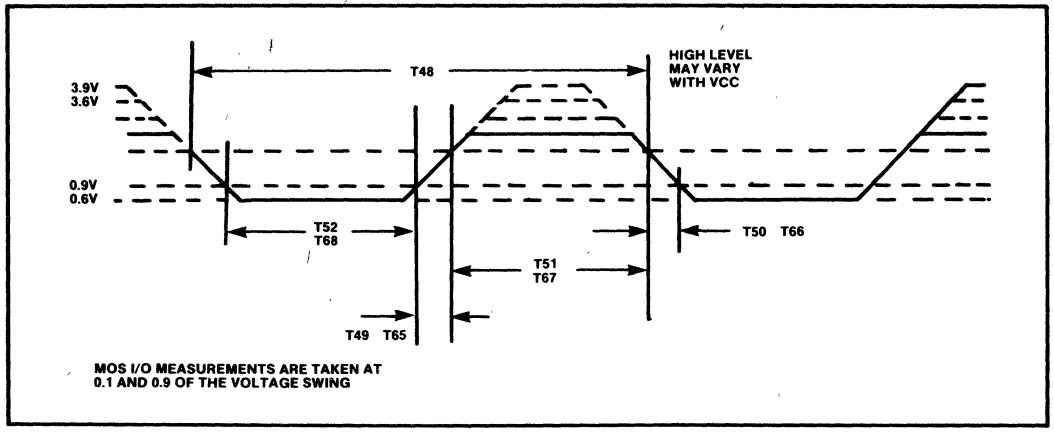


Figure 34. Serial Clock Input Voltage Levels for Timing Measurements

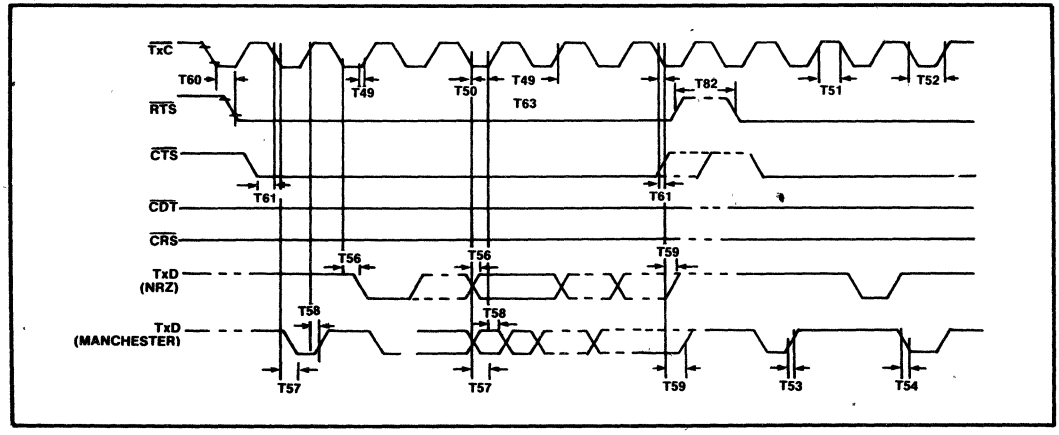


Figure 35. Transmit and Control and Data Timing

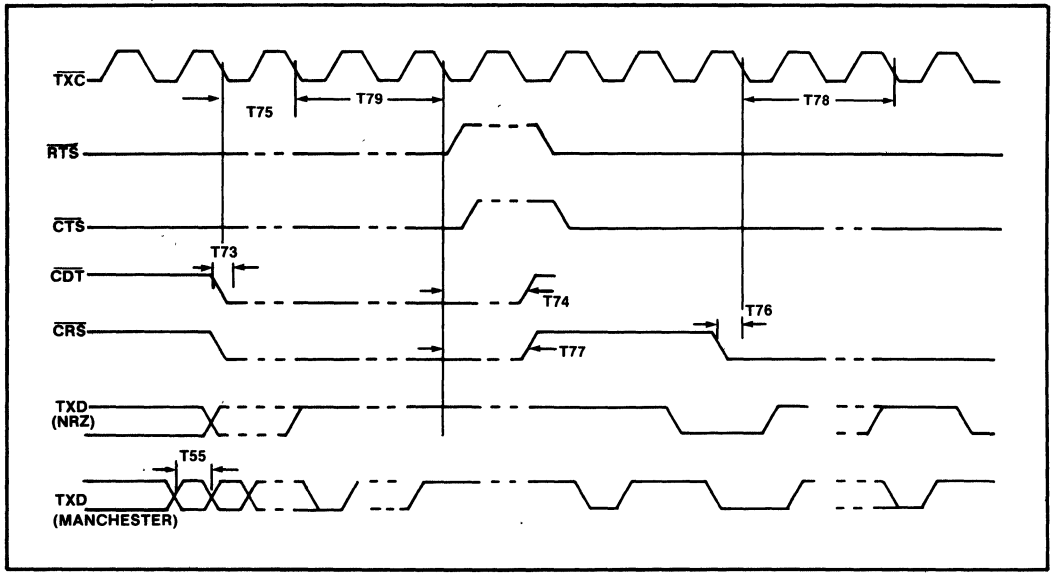


Figure 35. Transmit and Control and Data Timing (cont.)

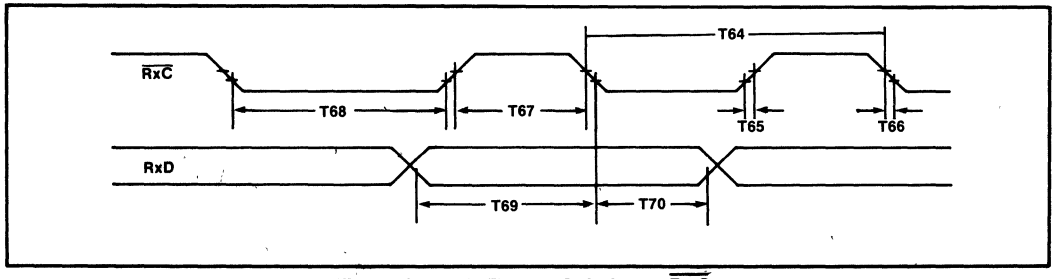


Figure 36. RxD Timing Relative to $\overline{RxC}$

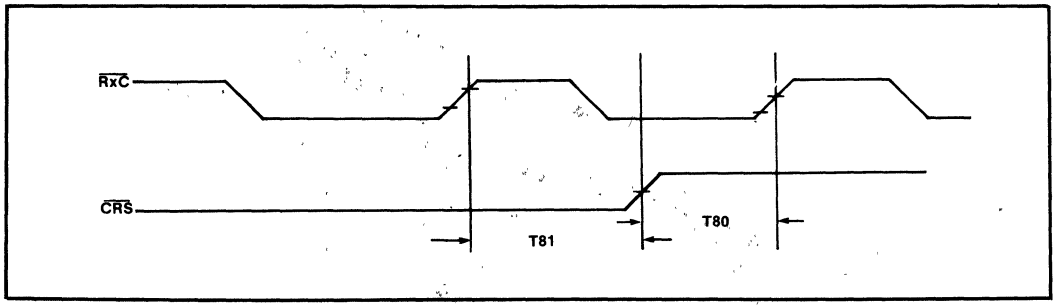


Figure 37. $\overline{CRS}$ Timing Relative to $\overline{RxC}$

intel

APPLICATION NOTE

AP-66

January 1980

Using The 8292 GPIB Controller

Tom Voll
Peripheral Components Applications

INTRODUCTION

The Intel® 8292 is a preprogrammed UPI™-41A that implements the Controller function of the IEEE Std 488-1978 (GPIB, HP-IB, IEC Bus, etc.). In order to function the 8292 must be used with the 8291 Talker/Listener and suitable interface and transceiver logic such as a pair of Intel 8293s. In this configuration the system has the potential to be a complete GPIB Controller when driven by the appropriate software. It has the following capabilities: System Controller, send IFC and Take Charge, send REN, Respond to SRQ, send Interface messages, Receive Control, Pass Control, Parallel Poll and Take Control Synchronously.

This application note will explain the 8292 only in the system context of an 8292, 8291, two 8293s and the driver software. If the reader wishes to learn more about the UPI-41A aspects of the 8292, Intel's Application Note AP-41 describes the hardware features and programming characteristics of the device. Additional information on the 8291 may be obtained in the data sheet. The 8293 is detailed in its data sheet. Both chips will be covered here in the details that relate to the GPIB controller.

The next section of this application note presents an overview of the GPIB in a tutorial, but comprehensive nature. The knowledgable reader may wish to skip this section; however, certain basic semantic concepts introduced there will be used throughout this note.

Additional sections cover the view of the 8292 from the CPU's data bus, the interaction of the 3 chip types (8291, 8292, 8293), the 8292's software protocol and the system level hardware/software protocol. A brief description of interrupts and DMA will be followed by an application example. Appendix A contains the source code for the system driver software.

GPIB/IEEE 488 OVERVIEW

DESIGN OBJECTIVES

What is the IEEE 488 (GPIB)?

The experience of designing systems for a variety of applications in the early 1970's caused Hewlett-Packard to define a standard intercommunication mechanism which would allow them to easily assemble instrumentation systems of varying degrees of complexity. In a typical situation each instrument designer designed his/her own interface from scratch. Each one was inconsistent in terms of electrical levels, pin-outs on a connector, and types of connectors. Every time they built a system they had to invent new cables and new documentation just to specify the cabling and interconnection procedures.

Based on this experience, Hewlett-Packard began to define a new interconnection scheme. They went further than that, however, for they wanted to specify the typical communication protocol for systems of instruments. So in 1972, Hewlett-Packard came out with the first version of the bus which since has been modified and standardized by a committee of several manufacturers, coordinated through the IEEE, to perfect what is now known as the IEEE 488 Interface Bus (also known as the HP-IB, the GPIB and the IEC bus). While this bus specification may not be perfect, it is a good compromise of the various desires and goals of instrumentation and computer peripheral manufacturers to produce a common interconnection mechanism. It fits most instrumentation systems in use today and also fits very well the microcomputer I/O bus requirements. The basic design objectives for the GPIB were to:

1. Specify a system that is easy to use, but has all of the terminology and the definitions related to that system precisely spelled out so that everyone uses the same language when discussing the GPIB.
2. Define all of the mechanical, electrical, and functional interface requirements of a system, yet not define any of the device aspects (they are left up to the instrument designer).
3. Permit a wide range of capabilities of instruments and computer peripherals to use a system simultaneously and not degrade each other's performance.
4. Allow different manufacturers' equipment to be connected together and work together on the same bus.
5. Define a system that is good for limited distance interconnections.
6. Define a system with minimum restrictions on performance of the devices.
7. Define a bus that allows asynchronous communication with a wide range of data rates.
8. Define a low cost system that does not require extensive and elaborate interface logic for the low cost instruments, yet provides higher capability for the higher cost instruments if desired.
9. Allow systems to exist that do not need a central controller; that is, communication directly from one instrument to another is possible.

Although the GPIB was originally designed for instrumentation systems, it became obvious that most of these systems would be controlled by a calculator or computer. With this in mind several modifications were made to the original proposal before its final adoption as an international standard. Figure 1 lists the salient characteristics of the

GPIB as both an instrumentation bus and as a computer I/O bus.

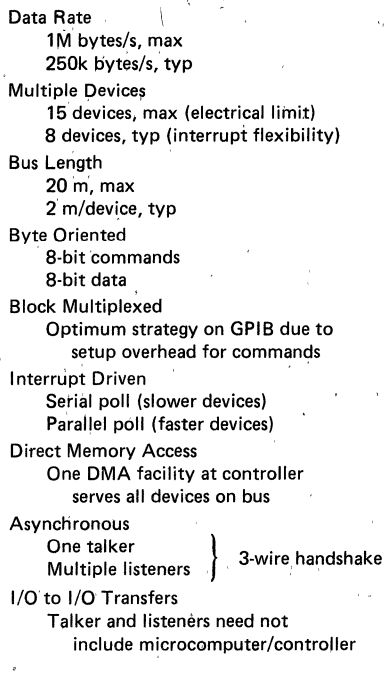


Figure 1. Major Characteristics of GPIB as Microcomputer I/O Bus

The bus can be best understood by examining each of these characteristics from the viewpoint of a general microcomputer I/O bus.

Data Rate — Most microcomputer systems utilize peripherals of differing operational rates, such as floppy discs at 31k or 62k bytes/s (single or double density), tape cassettes at 5k to 10k bytes/s, and cartridge tapes at 40k to 80k bytes/s. In general, the only devices that need high speed I/O are 0.5″ (1.3-cm) magnetic tapes and hard discs, operational at 30k to 781k bytes/s, respectively. Certainly, the 250k-bytes/s data rate that can be easily achieved by the IEEE 488 bus is sufficient for microcomputers and their peripherals, and is more than needed for typical analog instruments that take only a few readings per second. The 1M-byte/s maximum data rate is not easily achieved on the GPIB and requires special attention to considerations beyond the scope of this note. Although not required, data buffering in each device will improve the overall bus performance and allow utilization of more of the bus bandwidth.

Multiple Devices — Many microcomputer systems used as computers (not as components) service from three to seven peripherals. With the GPIB, up to 8 devices can be handled easily by 1 controller; with some slowdown in interrupt handling, up to 15 devices can work together. The limit of 8 is imposed by the number of unique parallel poll responses available; the limit of 15 is set by the electrical drive characteristics of the bus. Logically, the IEEE 488 Standard is capable of accommodating more device addresses (31 primary, each potentially with 31 secondaries).

Bus Length — Physically, the majority of microcomputer systems fit easily on a desk top or in a standard 19″ (48-cm) rack, eliminating the need for extra long cables. The GPIB is designed typically to have 2 m of length per device, which accommodates most systems. A line printer might require greater cable lengths, but this can be handled at the lower speeds involved by using extra dummy terminations.

Byte Oriented — The 8-bit byte is almost universal in I/O applications; even 16-bit and 32-bit computers use byte transfers for most peripherals. The 8-bit byte matches the ASCII code for characters and is an integral submultiple of most computer word sizes. The GPIB has an 8-bit wide data path that may be used to transfer ASCII or binary data, as well as the necessary status and control bytes.

Block Multiplexed — Many peripherals are block oriented or are used in a block mode. Bytes are transferred in a fixed or variable length group; then there is a wait before another group is sent to that device, e.g., one sector of a floppy disc, one line on a printer or tape punch, etc. The GPIB is, by nature, a block multiplexed bus due to the overhead involved in addressing various devices to talk and listen. This overhead is less bothersome if it only occurs once for a large number of data bytes (once per block). This mode of operation matches the needs of microcomputers and most of their peripherals. Because of block multiplexing, the bus works best with buffered memory devices.

Interrupt Driven — Many types of interrupt systems exist, ranging from complex, fast, vectored/priority networks to simple polling schemes. The main tradeoff is usually cost versus speed of response. The GPIB has two interrupt protocols to help span the range of applications. The first is a single service request (SRQ) line that may be asserted by all interrupting devices. The controller then polls all devices to find out which wants service. The polling mechanism is well defined and can be easily

automated. For higher performance, the parallel poll capability in the IEEE 488 allows up to eight devices to be polled at once — each device is assigned to one bit of the data bus. This mechanism provides fast recognition of an interrupting device. A drawback is the frequent need for the controller to explicitly conduct a parallel poll, since there is no equivalent of the SRQ line for this mode.

Direct Memory Access (DMA) — In many applications, no imediate processing of I/O data on a byte-by-byte basis is needed or wanted. In fact, programmed transfers slow down the data transfer rate unnecessarily in these cases, and higher speed can be obtained using DMA. With the GPIB, one DMA facility at the controller serves all devices. There is no need to incorporate complex logic in each device.

Asynchronous Transfers — An asynchronous bus is desirable so that each device can transfer at its own rate. However, there is still a strong motivation to buffer the data at each device when used in large systems in order to speed up the aggregate data rate on the bus by allowing each device to transfer at top speed. The GPIB is asynchronous and uses a special 3-wire handshake that allows data transfers from one talker to many listeners.

I/O To I/O Transfers — In practice, I/O to I/O transfers are seldom done due to the need for processing data and changing formats or due to mismatched data rates. However, the GPIB can support this mode of operation where the microcomputer is neither the talker nor one of the listeners.

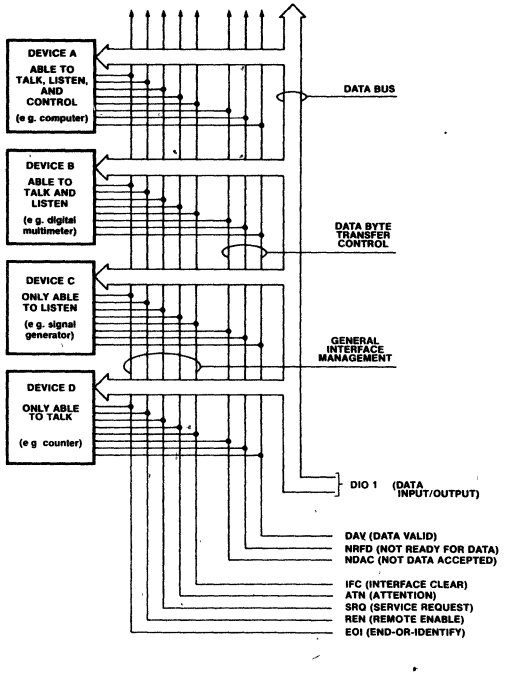


Figure 2. Interface Capabilities and Bus Structure

GPIB SIGNAL LINES

Data Bus

The lines DI01 through DI08 are used to transfer addresses, control information and data. The formats for addresses and control bytes are defined by the IEEE 488 standard (see Appendix C). Data formats are undefined and may be ASCII (with or without parity) or binary. DI01 is the Least Significant Bit (note that this will correspond to bit 0 on most computers).

Management Bus

ATN — Attention This signal is asserted by the Controller to indicate that it is placing an address or control byte on the Data Bus. ATN is de-asserted to allow the assigned Talker to place status or data on the Data Bus. The Controller regains control by re-asserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.

EOI — End or Identify This signal has two uses as its name implies. A talker may assert EOI simultaneously with the last byte of data to indicate end of data. The Controller may assert EOI along with ATN to initiate a Parallel Poll. Although many devices do not use Parallel Poll, all devices *should* use EOI to end transfers (many currently available ones do not).

SRQ — Service Request This line is like an interrupt: it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a Serial Poll at its earliest convenience. The device deasserts SRQ when polled.

IFC — Interface Clear This signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After deasserting IFC, the System Controller is the active controller of the system.

REN — Remote Enable This signal is asserted only by the System Controller. Its assertion does not place devices into Remote Control mode; REN only *enables* a device to go remote when addressed to listen. When in Remote, a device should ignore its front panel controls.

Transfer Bus

NRFD — Not Ready For Data This handshake line is asserted by a listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD deasserted (i.e., ready for data) until all devices have deasserted NRFD.

NDAC — Not Data Accepted. This handshake line is asserted by a Listener to indicate it has not yet accepted the data or control byte on the DIO lines. Note that the Controller will not see NDAC deasserted (i.e., data accepted) until all devices have deasserted NDAC.

DAV — Data Valid This handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the DIO lines and has had the minimum specified settling time.

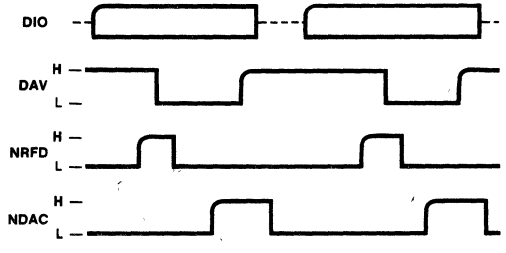


Figure 3. GPIB Handshake Sequence

GPIB INTERFACE FUNCTIONS

There are ten (10) interface functions specified by the IEEE 488 standard. Not all devices will have all functions and some may only have partial subsets. The ten functions are summarized below with the relevant section number from the IEEE document given at the beginning of each paragraph. For further information please see the IEEE standard.

1. *SH* — Source Handshake (section 2.3) This function provides a device with the ability to properly transfer data from a Talker to one or more Listeners using the three handshake lines.
2. *AH* — Acceptor Handshake (section 2.4) This function provides a device with the ability to properly receive data from the Talker using the three handshake lines. The AH function may also delay the beginning (NRFD) or end (NDAC) of any transfer.
3. *T* — Talker (section 2.5) This function allows a device to send status and data bytes when addressed to talk. An address consists of one (Primary) or two (Primary and Secondary) bytes. The latter is called an extended Talker.
4. *L* — Listener (section 2.6) This function allows a device to receive data when addressed to listen. There can be extended Listeners (analogous to extended Talkers above).
5. *SR* — Service Request (section 2.7) This function allows a device to request service (interrupt) the Controller. The SRQ line may be asserted asynchronously.
6. *RL* — Remote Local (section 2.8) This function allows a device to be operated in two modes: Remote via the GPIB or Local via the manual front panel controls.
7. *PP* — Parallel Poll (section 2.9) This function allows a device to present one bit of status to the Controller-in-charge. The device need not be addressed to talk and no handshake is required.
8. *DC* — Device Clear (section 2.10) This function allows a device to be cleared (initialized) by the Controller. Note that there is a difference between DC (*device* clear) and the IFC line (*interface* clear).
9. *DT* — Device Trigger (section 2.11) This function allows a device to have its basic operation started either individually or as part of a group. This capability is often used to synchronize several instruments.
10. *C* — Controller (section 2.12) This function allows a device to send addresses, as well as universal and addressed commands to other devices. There may be more than one controller on a system, but only one may be the controller-in-charge at any one time.

At power-on time the controller that is handwired to be the System Controller becomes the active controller-in-charge. The System Controller has several unique capabilities including the ability to send Interface Clear (IFC -- clears all device interfaces and returns control to the System Controller) and to send Remote Enable (REN — allows devices to respond to bus data once they are addressed to listen). The System Controller may optionally Pass Control to another controller, if the system software has the capability to do so.

GPIB CONNECTOR

The GPIB connector is a standard 24-pin industrial connector such as Cinch or Amphenol series 57 Micro-Ribbon. The IEEE standard specifies this connector, as well as the signal connections and the mounting hardware.

The cable has 16 signal lines and 8 ground lines. The maximum length is 20 meters with no more than two meters per device.

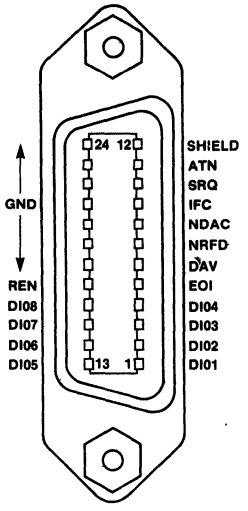


Figure 4. GPIB Connector

GPIB SIGNAL LEVELS

The GPIB signals are all TTL compatible, low true signals. A signal is asserted (true) when its electrical voltage is less than 0.5 volts and is deasserted (false) when it is greater than 2.4 volts. Be careful not to become confused with the two handshake signals, NRFD and NDAC which are also low true (i.e. > 0.5 volts implies the device is Not Ready For Data).

The Intel 8293 GPIB transceiver chips ensure that all relevant bus driver/receiver specifications are met. Detailed bus electrical specifications may be found in Section 3 of the IEEE Std 488-1978. The Standard is the ultimate reference for all GPIB questions.

GPIB MESSAGE PROTOCOLS

The GPIB is a very flexible communications medium and as such has many possible variations of protocols. To bring some order to the situation, this section will discuss a protocol similar to the one used by Ziatech's ZT80 GPIB controller for Intel's MULTIBUS™ computers. The ZT80 is a complete high-level interface processor that executes a set of high level instructions that map directly into GPIB actions. The sequences of commands, addresses and data for these instructions provide a good example of how to use the GPIB (additional information is available in the ZT80 Instruction Manual). The 'null' at the end of each instruction is for cosmetic use to remove previous information from the DIO lines.

DATA — Transfer a block of data from device A to devices B, C...
1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Universal Unlisten
3. Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   Device C Primary (Listen) Address
   etc.
4. First Data Byte
   Second Data Byte
   .
   .
   .
   Last Data Byte (EOI)
5. Null

TRIGR — Trigger devices A, B,...to take action
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Group Execute Trigger
4. Null

PSCTL — Pass control to device A
1. Device A Primary (Talk) Address
   Device A Secondary Address (if any)
2. Take Control
3. Null

CLEAR — Clear all devices
1. Device Clear
2. Null

REMAL — Remote Enable
1. Assert REN continuously

GOREM — Put devices A, B,...into Remote
1. Assert REN continuously
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Null

GOLOC — Put devices A, B,...into Local
1. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
2. Go To Local
3. Null

LOCAL — Reset all devices to Local
1. Stop asserting REN

LLKAL — Prevent all devices from returning to Local
1. Local Lock Out
2. Null

SPOLL — Conduct a serial poll of devices A, B,...
1. Serial Poll Enable
2. Universal Unlisten
3. ZT 80 Primary (Listen) Address
   ZT 80 Secondary Address
4. Device Primary (Talk) Address
   Device Secondary Address (if any)
5. Status byte from device
6. Go to Step 4 until all devices on list have been polled
7. Serial Poll Disable
8. Null

PPUAL — Unconfigure and disable Parallel Poll response from all devices
1. Parallel Poll Unconfigure
2. Null

ENAPP — Enable Parallel Poll response in devices A, B,...
1. Universal Unlisten
2. Device Primary (Listen) Address
   Device Secondary Address (if any)
3. Parallel Poll Configure
4. Parallel Poll Enable
5. Go to Step 2 until all devices on list have been configured.
6. Null

DISPP — Disable Parallel Poll response from devices A, B,...
1. Universal Unlisten
2. Device A Primary (Listen) Address
   Device A Secondary Address (if any)
   Device B Primary (Listen) Address
   Device B Secondary Address (if any)
   etc.
3. Disable Parallel Poll
4. Null

This Ap Note will detail how to implement a useful subset of these controller instructions.

HARDWARE ASPECTS OF THE SYSTEM

8291 GPIB TALKER/LISTENER

The 8291 is a custom designed chip that implements many of the non-controller GPIB functions. It provides hooks so the user's software can implement additional features to complete the set. This chip is discussed in detail in its data sheet. The major features are summarized here:

—Designed to interface microprocessors to the GPIB
—Complete Source and Acceptor Handshake
—Complete Talker and Listener Functions with extended addressing
—Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local functions
—Programmable data transfer rate
—Maskable interrupts
—On-chip primary and secondary address recognition
—1-8 MHz clock range
—16 registers (8 read, 8 write) for CPU interface
—DMA handshake provision
—Trigger output pin
—On-chip EOS (End of Sequence) recognition

The pinouts and block diagram are shown in Fig. 5. One of eight read registers is for data transfer to the CPU; the other seven allow the microprocessor to monitor the GPIB states and various bus and device conditions. One of the eight write registers is for data transfer from the CPU; the other seven control various features of the 8291.

The 8291 interface functions will be software configured in this application example to the following subsets for use with the 8292 as a controller that does not pass control. The 8291 is used only to provide the handshake logic and to send and receive data bytes. It is not acting as a normal device in this mode, as it never sees ATN asserted.

SH1	Source Handshake
AH1	Acceptor Handshake
T3	Basic Talk-only
L1	Basic Listen-only
SR0	No Service Requests
RL0	No Remote/Local
PP0	No Parallel Poll response
DC0	No Device Clear
DT0	No Device Trigger

If control is passed to another controller, the 8291 must be reconfigured to act as a talker/listener with the following subsets:

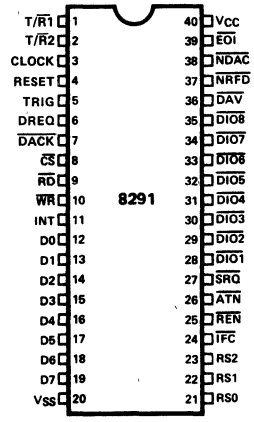
SH1	Source Handshake
AH1	Acceptor Handshake
T5	Basic Talker and Serial Poll
L3	Basic Listener
SR1	Service Requests
RL1	Remote/Local with Lockout
PP2	Preconfigured Parallel Poll
DC1	Device Clear
DT1	Device Trigger
C0	Not a Controller

Most applications do not pass control and the controller is always the system controller (see 8292 commands below).

8292 GPIB CONTROLLER

The 8292 is a preprogrammed Intel® 8041A that provides the additional functions necessary to

PIN CONFIGURATION



BLOCK DIAGRAM

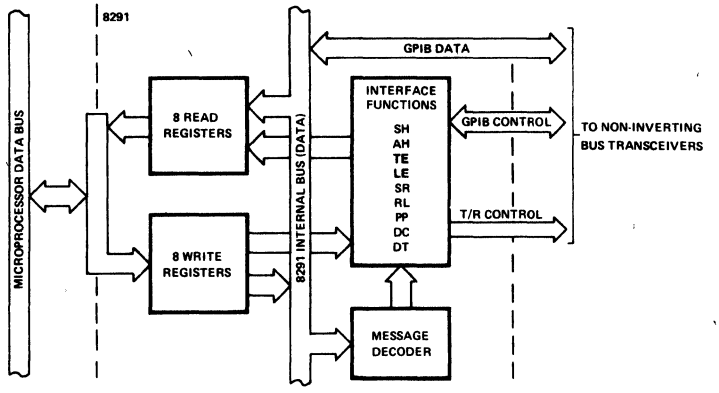


Figure 5. 8291 Pin Configuration and Block Diagram

implement a GPIB controller when used with an 8291 Talker/Listener. The 8041A is documented in both a user's manual and in AP-41. The following description will serve only as an outline to guide the later discussion.

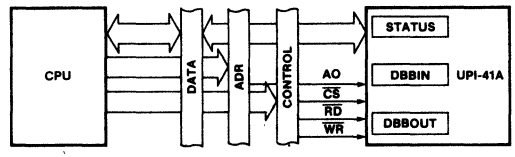
The 8292 acts as an intelligent slave processor to the main system CPU. It contains a processor, memory, I/O and is programmed to perform a variety of tasks associated with GPIB controller operation. The on-chip RAM is used to store information about the state of the Controller function, as well as a variety of local variables, the stack and certain user status information. The timer/counter may be optionally used for several time-out functions or for counting data bytes transferred. The I/O ports provide the GPIB control signals, as well as the ancillary lines necessary to make the 8291, 2, 3 work together.

The 8292 is closely coupled to the main CPU through three on-chip registers that may be independently accessed by both the master and the 8292 (UPI-41A). Figure 6 shows this Register Interface. Also refer to Figure 12.

The status register is used to pass Interrupt Status information to the master CPU (A0 = 1 on a read).

The DBBOUT register is used to pass one of five other status words to the master based on the last command written into DBBIN. DBBOUT is accessed when A0 = 0 on a Read. The five status words are Error Flag, Controller Status, GPIB Status, Event Counter Status or Time Out Status.

DBBIN receives either commands (A0 = 1 on a Write) or command related data (A0 = 0 on a write) from the master. These command related data are Interrupt Mask, Error Mask, Event Counter or Time Out.



CS	AO	RD	WR	REGISTER
0	0	0	1	READ DBBOUT
0	1	0	1	READ STATUS
0	0	1	0	WRITE DBBIN (DATA)
0	1	1	0	WRITE DBBIN (COMMAND)
1	X	X	X	NO ACTION

Figure 6. UPI-41A Registers

8293 GPIB TRANSCEIVERS

The 8293 is a multi-use HMOS chip that implements the IEEE 488 bus transceivers and contains the additional logic required to make the 8291 and 8292 work together. The two option strapping pins are used to internally configure the chip to perform the specialized gating required for use with 8291 as a device or with 8291/92 as a controller.

In this application example the two configurations used are shown in Fig. 7a and 7b. The drivers are set to open collector or three state mode as required and the special logic is enabled as required in the two modes.

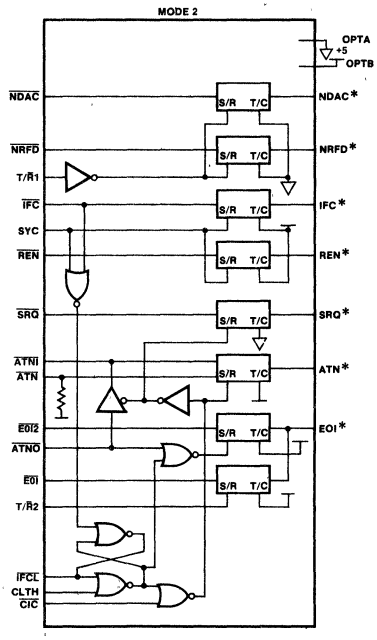


Figure 7a. 8293 Mode 2

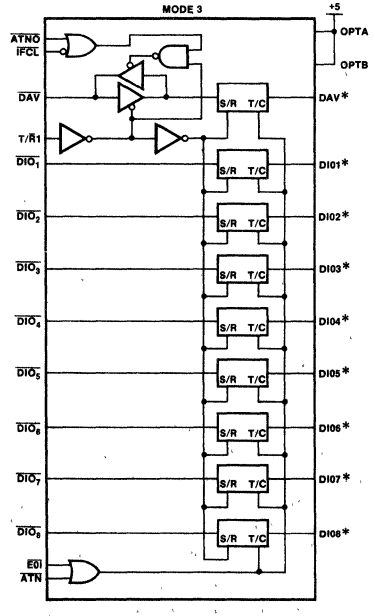


Figure 7b. 8293 Mode 3

8291/2/3 CHIP SET

Figure 8 shows the four chips interconnected with the special logic explicitly shown.

The 8291 acts only as the mechanism to put commands and addresses on the bus while the 8292 is asserting ATN. The 8291 is tricked into believing that the ATN line is not asserted by the ATN2 output of the ATN transceiver and is placed in Talk-only mode by the CPU. The 8291 then acts as though it is sending data, when in reality it is sending addresses and/or commands. When the 8292 deasserts ATN, the CPU software must place the 8291 in Talk-only, Listen-only or Idle based on the implicit knowledge of how the controller is going to participate in the data transfer. In other words, the 8291 does not respond directly to addresses or commands that it sends on the bus on behalf of the Controller. The user software, through the use of Listen-only or Talk-only, makes the 8291 behave as though it were addressed.

Although it is not a common occurrence, the GPIB specification allows the Controller to set up a data transfer between two devices and not directly participate in the exchange. The controller must know when to go active again and regain control. The chip set accomplishes this through use of the "Continuous Acceptor Handshake cycling mode" and the ability to detect EOI or EOS at the end of the transfer. See XFER in the Software Driver Outline below.

If the 8292 is not the System Controller as determined by the signal on its SYC pin, then it must be able to respond to an IFC within 100 usec. This is accomplished by the cross-coupled NORs in Fig. 7a which deassert the 8293's internal version of $\overline{\text{CIC}}$ (Not Controller-in-Charge). This condition is latched until the 8292's firmware has received the IFCL (interface clear received latch) signal by testing the $\overline{\text{IFCL}}$ input. The firmware then sets its signals to reflect the inactive condition and clears the 8293's latch.

In order for the 8292 to conduct a Parallel Poll the 8291 must be able to capture the PP response on the DIO lines. The only way to do this is to fool the 8291 by putting it into Listen-only mode and generating a DAV condition. However, the bus spec does not allow a DAV during Parallel Poll, so the back-to-back 3-state buffers (see Fig. 7b) in the 8293 isolate the bus and allow the 8292 to generate a local DAV for this purpose. Note that the 8291 cannot assert a Parallel Poll response. When the 8292 is not the controller-in-charge the 8291 may respond to PPs and the 8293 guarantees that the DIO drivers are in "open collector" mode through the OR gate (Fig. 7b).

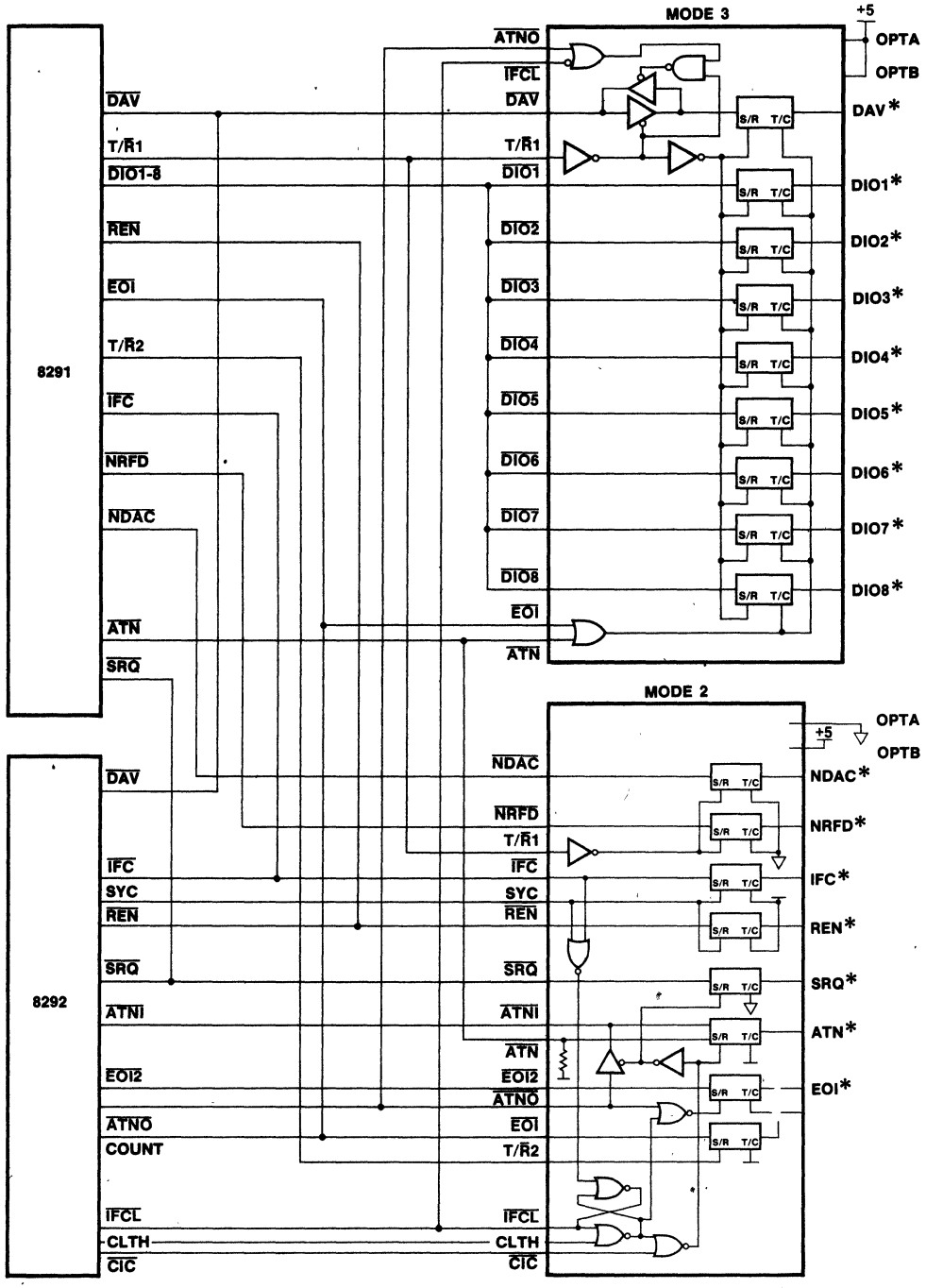


Figure 8. Talker/Listener/Controller

ZT7488/18 GPIB CONTROLLER

Ziatech's GPIB Controller, the ZT7488/18 will be used as the controller hardware in this Application Note. The controller consists of an 8291, 8292, an 8 bit input port and TTL logic equivalent to that shown in Figure 8. Figure 9 shows the card's block diagram. The ZT7488/18 plugs into the STD bus, a 56 pin 8 bit microprocessor oriented bus. An 8085 CPU card is also available on the STD bus and will be used to execute the driver software.

The 8291 uses I/O Ports 60H to 67H and the 8292 uses I/O Ports 68H and 69H. The five interrupt lines are connected to a three-state buffer at I/O Port 6FH to facilitate polling operation. This is required for the TCI, as it cannot be read internally in the 8292. The other three 8292 lines (SPI, IBF, OBF) and the 8291's INT line are also connected to minimize the number of I/O reads necessary to poll the devices.

$\overline{\text{NDAC}}$ is connected to $\overline{\text{COUNT}}$ on the 8292 to allow byte counting on data transfers. The example driver software will not use this feature, as the software is simpler and faster if an internal 8085 register is used for counting in software.

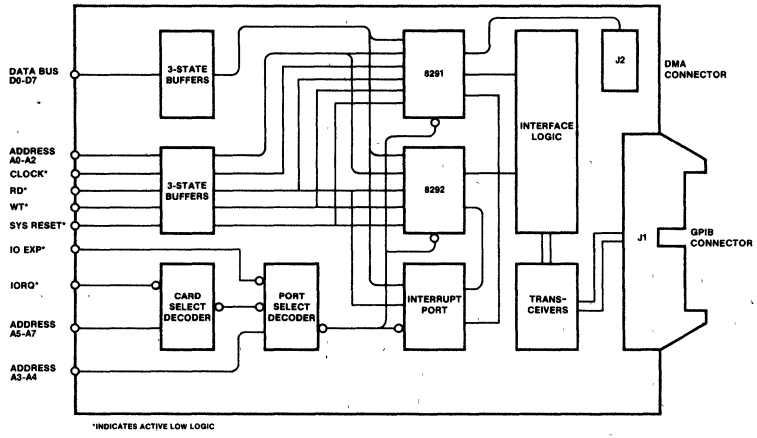


Figure 9. ZT7488/18 GPIB Controller

READ REGISTERS								PORT #	WRITE REGISTERS							
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	60H	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
DATA IN									DATA OUT							
CPT	APT	GET	END	DEC	ERR	BO	BI	61H	CPT	APT	GET	END	DEC	ERR	BO	BI
INTERRUPT STATUS 1									INTERRUPT MASK 1							
INT	SPAS	LLO	REM	SPASC	LLOC	REMC	ADSC	62H	0	0	DMAO	DMAI	SPASC	LLOC	REMC	ADSC
INTERRUPT STATUS 2									INTERRUPT MASK 2							
S8	SRQS	S6	S5	S4	S3	S2	S1	63H	S8	rsv	S6	S5	S4	S3	S2	S1
SERIAL POLL STATUS									SERIAL POLL MODE							
ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN	64H	TO	LO	0	0	0	0	ADM1	ADM0
ADDRESS STATUS									ADDRESS MODE							
CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	65H	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
COMMAND PASS THROUGH									AUX MODE							
X	DT0	DL0	AD5 0	AD4-0	AD3 0	AD2-0	AD1-0	66H	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1
ADDRESS 0									ADDRESS 0/1							
X	DT1	DL1	AD5 1	AD4-1	AD3-1	AD2 1	AD1-1	67H	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
ADDRESS 1									EOS							

Figure 10. 8291 Registers

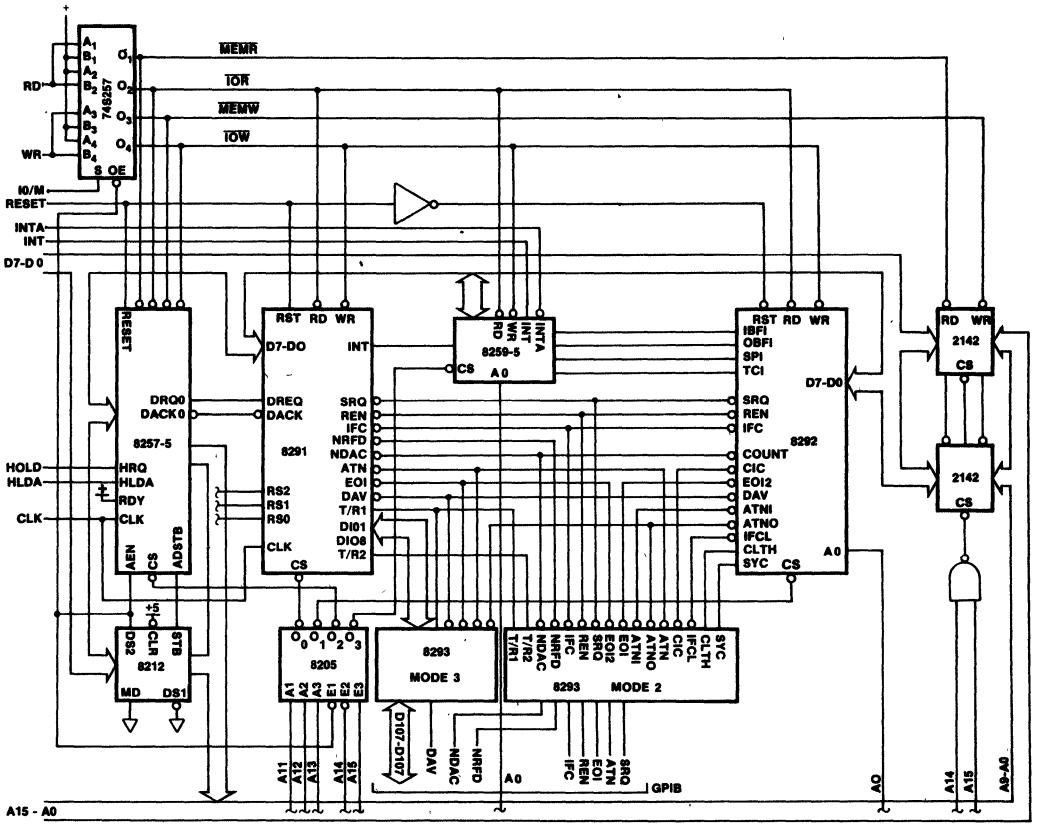


Figure 11. DMA/Interrupt GPIB Controller Block Diagram

The application example will not use DMA or interrupts; however, the Figure 11 block diagram includes these features for completeness.

The 8257-5 DMA chip can be used to transfer data between the RAM and the 8291 Talker/Listener. This mode allows a faster data rate on the GPIB and typically will depend on the 8291's EOS or EOI detection to terminate the transfer. The 8259-5 interrupt controller is used to vector the five possible interrupts for rapid software handling of the various conditions.

8292 COMMAND DESCRIPTION

This section discusses each command in detail and relates them to a particular GPIB activity. Recall that although the 8041A has only two read registers and one write register, through the magic of on-chip firmware the 8292 appears to have six read registers and five write registers. These are listed in Figure 12. Please see the 8292 data sheet for detailed definitions of each register. Note the two letter mnemonics to be used in later discussions. The CPU must not write into the 8292 while IBF (Input Buffer Full) is a one, as information will be lost.

DIRECT COMMANDS

Both the Interrupt Mask (IM) and the Error Mask (EM) register may be directly written with the LSB of the address bus (A0) a "0". The firmware uses the MSB of the data written to differentiate between IM and EM.

Load Interrupt Mask

This command loads the Interrupt Mask with D7–D0. Note that D7 must be a "1" and that interrupts are enabled by a corresponding "1" bit in this register. IFC interrupt cannot be masked off; however, when the 8292 is the System Controller, sending an ABORT command will not cause an IFC interrupt.

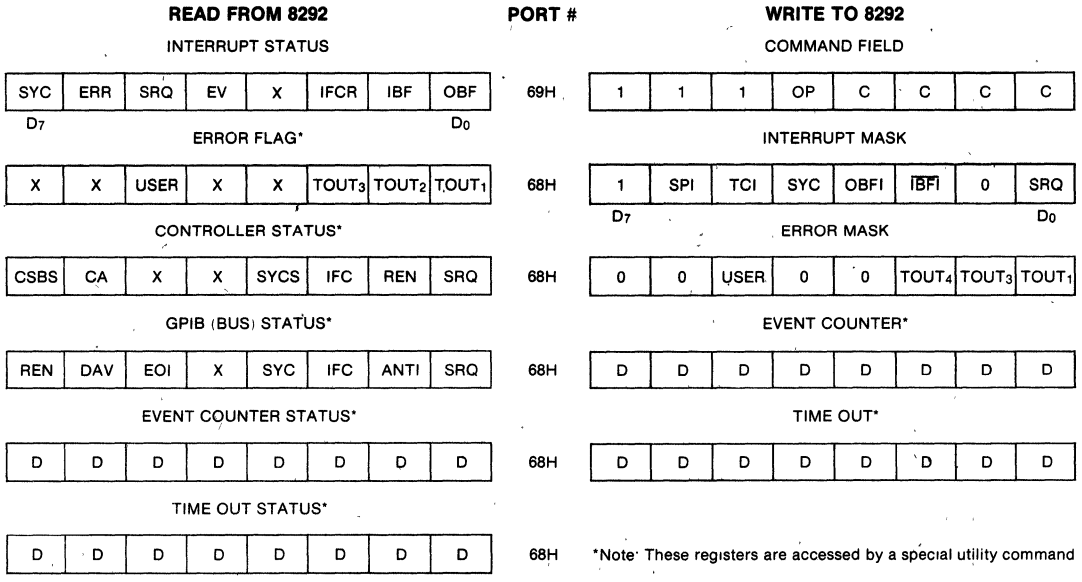


Figure 12. 8292 Registers

Load Error Mask

This command loads the Error Mask with D7–D0. Note that D7 must be a zero and that interrupts are enabled by a corresponding "1" bit in this register.

UTILITY COMMANDS

These commands are used to read or write the 8292 registers that are not directly accessible. All utility commands are written with A0 = 1, D7 = D6 = D5 = 1, D4 = 0. D3–D0 specify the particular command. For writing into registers the general sequence is:

1. wait for IBF = 0 in Interrupt Status Register
2. write the appropriate command to the 8292,
3. write the desired register value to the 8292 with A0 = 1 with no other writes intervening,
4. wait for indication of completion from 8292 (IBF = 0).

For reading a register the general sequence is:

1. wait for IBF = 0 in Interrupt Status Register
2. write the appropriate command to the 8292
3. wait for a TCI (Task Complete Interrupt)
4. Read the value of the accessed register from the 8292 with A0 = 0.

WEVC — Write to Event Counter
(Command = 0E2H)

The byte written following this command will be loaded into the event counter register and event counter status for byte counting. The internal counter is incremented on a high to low transition of the COUNT (T1) input. In this application example $\overline{\text{NDAC}}$ is connected to count. The counter is an 8 bit register and therefore can count up to 256 bytes (writing 0 to the EC implies a count of 256). If longer blocks are desired, the main CPU must handle the interrupts every 256 counts and carefully observe the timing constraints.

Because the counter has a frequency range from 0 to 133 kHz when using a 6 MHz crystal, this feature may not be usable with all devices on the GPIB. The 8291 can easily transfer data at rates up to 250 kHz and even faster with some tuning of the system. There is also a 500 ns minimum high time requirement for COUNT which can potentially be violated by the 8291 in continuous acceptor handshake mode (i.e., TNDDV1 + TDVND2-C = 350 + 350 = 700 max). When cable delays are taken into consideration, this problem will probably never occur.

When the 8292 has completed the command, IBF will become a "0" and will cause an interrupt if masked on.

WTOUT — Write to Time Out Register
(Command = 0E1H)

The byte written following this command will be used to determine the number of increments used for the time out functions. Because the register is 8 bits, the maximum time out is 256 time increments. This

is probably enough for most instruments on the GPIB but is not enough for a manually stepped operation using a GPIB logic analyzer like Ziatech's ZT488. Also, the 488 Standard does not set a lower limit on how long a device may take to do each action. Therefore, any use of a time out must be able to be overridden (this is a good general design rule for service and debugging considerations).

The time out function is implemented in the 8292's firmware and will not be an accurate time. The counter counts backwards to zero from its initial value. The function may be enabled/disabled by a bit in the Error mask register. When the command is complete IBF will be set to a "0" and will cause an interrupt if masked on.

REVC — Read Event Counter Status
(Command = 0E3H)

This command transfers the content of the Event Counter to the DBBOUT register. The firmware then sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value from the 8292 with A0 = 0.

RINM — Read Interrupt Mask Register
(Command = 0E5H)

This command transfers the content of the Interrupt Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RERM — Read Error Mask Register
(Command = 0EAH)

This command transfers the content of the Error Mask register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RCST — Read Controller Status Register
(Command = 0E6H)

This command transfers the content of the Controller Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

RTOUT — Read Time Out Status Register
(Command = 0E9H)

This command transfers the content of the Time Out Status register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

If this register is read while a time-out function is in process, the value will be the time remaining before time-out occurs. If it is read after a time-out, it will be zero. If it is read when no time-out is in process, it will be the last value reached when the previous timing occurred.

RBST — Read Bus Status Register
(Command = 0E7H)

This command causes the firmware to read the GPIB management lines, DAV and the SYC pin and place a copy in DBBOUT. TCI is set to "1" and will cause an interrupt if masked on. The CPU may read the value.

RERF — Read Error Flag Register
(Command = 0E4H)

This command transfers the content of the Error Flag register to the DBBOUT register. The firmware sets TCI = 1 and will cause an interrupt if masked on. The CPU may then read the value.

This register is also placed in DBBOUT by an IACK command if ERR remains set. TCI is set to "1" in this case also.

IACK — Interrupt Acknowledge
(Command = A1 A2 A3 A4 1 A5 1 1)

This command is used to acknowledge any combinations of the five SPI interrupts (A1–A5): SYC, ERR, SRQ, EV, and IFCR. Each bit A1–A5 is an individual acnowledgement to the corresponding bit in the Interrupt Status Register. The command clears SPI but it will be set again if all of the pending interrupts were not acknowledged.

If A2 (ERR) is "1", the Error Flag register is placed in DBBOUT and TCI is set. The CPU may then read the Error Flag without issuing an RERF command.

OPERATION COMMANDS

The following diagram (Fig. 13) is an attempt to show the interrelationships among the various 8292 Operation Commands. It is not meant to replace the complete controller state diagram in the IEEE Standard.

RST — Reset (Command = 0F2H)

This command has the same effect as an external reset applied to the chip's pin #4. The 8292's actions are:

1. All outputs go to their electrical high state. This means that SPI, TCI, OBFI, $\overline{\text{IBFI}}$, CLTH will be TRUE and all other GPIB signals will be FALSE.
2. The 8292's firmware will cause the above mentioned five signals to go FALSE after approximately 17.5 usec. (at 6 MHz).
3. These registers will be cleared: Interrupt Status, Interrupt Mask, Error Mask, Time Out, Event Counter, Error Flag.
4. If the 8292 is the System Controller (SYC is TRUE), then IFC will be sent TRUE for approximately 100 usec and the Controller function will end up in charge of the bus. If the 8292 is not the

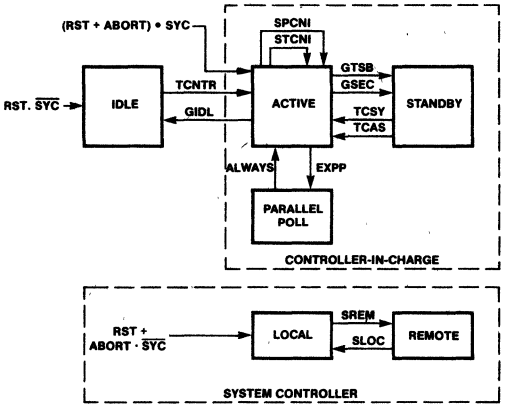


Figure 13. 8292 Command Flowchart

System Controller then it will end up in an Idle state.

5. TCI will not be set.

RSTI — Reset Interrupts (Command = 0F3)

This command clears all pending interrupts and error flags. The 8292 will stop waiting for actions to occur (e.g., waiting for ATN to go FALSE in a TCNTR command or waiting for the proper handshake state in a TCSY command). TCI will not be set.

ABORT — Abort all operations and Clear Interface (Command = 0F9H)

If the 8292 is not the System Controller this command acts like a NOP and flags a USER ERROR in the Error Flag Register. No TCI will occur.

If the 8292 is the System Controller then IFC is set TRUE for approximately 100 μsec and the 8292 becomes the Controller-in-Charge and asserts ATN. TCI will be set, only if the 8292 was NOT the CIC.

STCNI — Start Counter Interrupts (Command = 0FEH)

Enables the EV Counter Interrupt. TCI will not be set. Note that the counter must be enabled by a GSEC command.

SPCNI — Stop Counter Interrupts (Command = 0F0H)

The 8292 will not generate an EV interrupt when the counter reaches 0. Note that the counter will continue counting. TCI will not be set.

SREM — Set Interface to Remote Control (Command = 0F8H)

If the 8292 is the System Controller, it will set REN and TCI TRUE. Otherwise it only sets the User Error Flag.

SLOC — Set Interface to Local Mode (Command = 0F7H)

If the 8292 is the System Controller, it will set REN FALSE and TCI TRUE. Otherwise, it only sets the User Error Flag.

EXPP — Execute Parallel Poll (Command = 0F5H)

If not Controller-in-Charge, the 8292 will treat this as a NOP and does not set TCI. If it is the Controller-in-Charge then it sets IDY (EOI & ATN) TRUE and generates a local DAV pulse (that never reaches the GPIB because of gates in the 8293). If the 8291 is configured as a listener, it will capture the Parallel Poll Response byte in its data register. TCI is not generated, the CPU must detect the BI (Byte In) from the 8291. The 8292 will be ready to accept another command before the BI occurs; therefore the 8291's BI serves as a task complete indication.

GTSB — Go To Standby (Command = 0F6H)

If the 8292 is not the Controller-in-Charge, it will treat this command as a NOP and does not set TCI TRUE. Otherwise, it goes to Controller Standby State (CSBS), sets ATN FALSE and TCI TRUE. This command is used as part of the Send, Receive, Transfer and Serial Poll System commands (see next section) to allow the addressed talker to send data/status.

If the data transfer does not start within the specified Time-Out, the 8292 sets TOUT2 TRUE in the Error Flag Register and sets SPI (if enabled). The controller continues waiting for a new command. The CPU must decide to wait longer or to regain control and take corrective action.

GSEC — Go to Standby and Enable Counting (Command = 0F4H)

This command does the same things as GTSB but also initializes the event counter to the value previously stored in the Event Counter Register (default value is 256) and enables the counter. One may wire the count input to $\overline{\text{NDAC}}$ to count bytes. When the counter reaches zero, it sets EV (and SPI if enabled) in Interrupt Status and will set EV every 256 bytes thereafter. Note that there is a potential loss of count information if the CPU does not respond to the EV/SPI before another 256 bytes have been transferred. TCI will be set at the end of the command.

TCSY — Take Control Synchronously (Command = 0FDH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it waits

for the proper handshake state and sets ATN TRUE. The 8292 will set TOUT3 if the handshake never assumes the correct state and will remain in this command until the handshake is proper or a RSTI command is issued. If the 8292 successfully takes control, it sets TCI TRUE.

This is the normal way to regain control at the end of a Send, Receive, Transfer or Serial Poll System Command. If TCSY is not successful, then the controller must try TCAS (see warning below).

TCAS — Take Control Asynchronously (Command = 0FCH)

If the 8292 is not in Standby, it treats this command as a NOP and does not set TCI. Otherwise, it arbitrarily sets ATN TRUE and TCI TRUE. Note that this action may cause devices on the bus to lose a data byte or cause them to interpret a data byte as a command byte. Both Actions can result in anomalous behavior. TCAS should be used only in emergencies. If TCAS fails, then the System Controller will have to issue an ABORT to clean things up.

GIDL — Go to Idle (Command = 0F1H)

If the 8292 is not the Controller in Charge and Active, then it treats this command as a NOP and does not set TCI. Otherwise, it sets ATN FALSE, becomes Not Controller in Charge, and sets TCI TRUE. This command is used as part of the Pass Control System Command.

TCNTR — Take (Receive) Control (Command = 0FAH)

If the 8292 is not Idle, then it treats this command as a NOP and does not set TCI. Otherwise, it waits for the current Controller-in-Charge to set ATN FALSE. If this does not occur within the specified Time Out, the 8292 sets TOUT1 in the Error Flag Register and sets SPI (if enabled). it will not proceed until ATN goes false or it receives an RSTI command. Note that the Controller in Charge must previously have sent this controller (via the 8291's command pass through register) a Pass Control message. When ATN goes FALSE, the 8292 sets CIC, ATN and TCI TRUE and becomes Active.

SOFTWARE DRIVER OUTLINE

The set of system commands discussed below is shown in Figure 14. These commands are implemented in software routines executed by the main CPU.

The following section assumes that the Controller is the System Controller and will not Pass Control. This is a valid assumption for 99+% of all controllers. It also assumes that no DMA or Interrupts will be used. SYC (System Control Input) should not be changed after Power-on in any system — it adds unnecessary complexity to the CPU's software.

In order to use polling with the 8292 one must enable TCI but not connect the pin to the CPU's interrupt pin. TCI must be readable by some means. In this application example it is connected to bit 1 port 6FH on the ZT7488/18. In addition, the other three 8292 interrupt lines and the 8291 interrupt are also on that port (SPI-Bit 2, $\overline{\text{IBFI}}$-Bit 4, OBFI-Bit 3, 8291 INT-Bit 0).

These drivers assume that only primary addresses will be used on the GPIB. To use secondary addresses, one must modify the test for valid talk/listen addresses (range macro) to include secondaries.

INIT	INITIALIZATION
Talker/Listener	
SEND	SEND DATA
RECV	RECEIVE DATA
XFER	TRANSFER DATA
Controller	
TRIG	GROUP EXECUTE TRIGGER
DCLR	DEVICE CLEAR
SPOL	SERIAL POLL
PPEN	PARALLEL POLL ENABLE
PPDS	PARALLEL POLL DISABLE
PPUN	PARALLEL POLL UNCONFIGURE
PPOL	PARALLEL POLL
PCTL	PASS CONTROL
RCTL	RECEIVE CONTROL
SRQD	SERVICE REQUESTED
System Controller	
REME	REMOTE ENABLE
LOCL	LOCAL
IFCL	ABORT/INTERFACE CLEAR

Figure 14. Software Driver Routines

INITIALIZATION

8292 — Comes up in Controller Active State when SYC is TRUE. The only initialization needed is to enable the TCI interrupt mask. This is done by writing 0A0H to Port 68H.

8291 — Disable both the major and minor addresses because the 8291 will never see the 8292's commands/addresses (refer to earlier hardware discussion). This is done by writing 60H and 0E0H to Port 66H.

Set Address Mode to Talk-only by writing 80H to Port 64H.

Set internal counter to 3 MHz to match the clock input coming from the 8085 by writing 23H to Port 65H. High speed mode for the handshakes will not be used here even though the hardware uses three-state drivers.

No interrupts will be enabled now. Each routine will enable the ones it needs for ease of polling operation. The INT bit may be read through Port 6FH. Clear both interrupt mask registers.

Release the chip's initialization state by writing 0 to Port 65H.

```
INIT:
  Enable-8292                          ;Set up Int. pins for Port 6FH
    Enable TCI                         ;Task complete must be on
  Enable-8291
    Disable major address              ;In controller usage, the 8291
    Disable minor address              ;Is set to talk only and/or listen only
    ton                                ;Talk only is our rest state
    Clock frequency                    ;3 MHz in this ap note example
    All interrupts off
    Immediate execute pon              ;Releases 8291 from init. state
```

TALKER/LISTENER ROUTINES

Send Data

SEND <listener list pointer> <count> <EOS> <data buffer pointer>

This system command sends data from the CPU to one or more devices. The data is usually a string of ASCII characters, but may be binary or other forms as well. The data is device-specific.

My Talk Address (MTA) must be output to satisfy the GPIB requirement of only one talker at a time (any other talker will stop when MTA goes out). The MTA is not needed as far as the 8291 is concerned — it will be put into talk-only mode (ton).

This routine assumes a non-null listener list in that it always sends Universal Unlisten. If it is desired to send data to the listeners previously addressed, one could add a check for a null list and not send UNL. Count must be 255 or less due to an 8 bit register. This routine also always uses an EOS character to terminate the string output; this could easily be eliminated and rely on the count. Items in brackets ( ) are optional and will not be included in the actual code in Appendix A.

```
SEND:
  Output-to-8291 MTA, UNL              ;We will talk, nobody listen
  Put EOS into 8291                    ;End of string compare character
  While 20H ≤ listener ≤ 3EH           ;GPIB listen addresses are
    output-to-8291 listener            ;"space" thru " >" ASCII
    Increment listen list pointer      ;Address all listeners
  Output-to-8292 GTSB                  ;8292 stops asserting ATN, go to standby
  Enable-8291
    Output EOI on EOS sent             ;Send EOI along with EOS character
  If count < > 0 then
    While not (end or count = 0)       ;Wait for EOS or end of count
    (could check tout 2 here)          ;Optionally check for stuck bus-tout 2
      Output-to-8291 data              ;Output all data, one byte at a time
      Increment data buffer pointer    ;8085 CREG will count for us
      Decrement count
  Output-to-8292 TCSY                  ;8292 asserts ATN, take control sync.
  (If tout3 then take control async)    ;If unable to take control sync.
  Enable 8291                          ;Restore 8291 to standard condition
    No output EOI on EOS sent
  Return
```

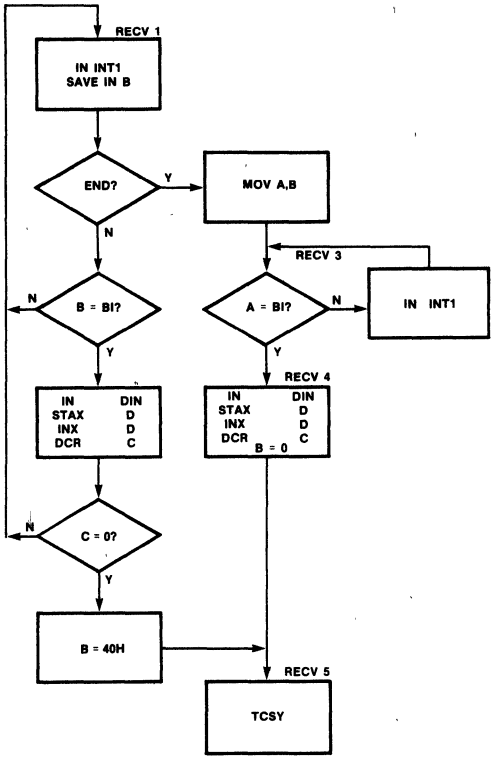


Figure 15. Flowchart For Receive Ending Conditions

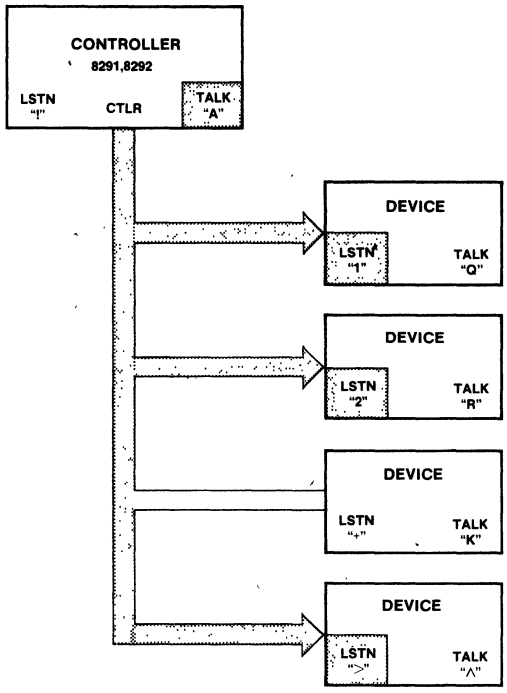


Figure 16. SEND to "1", "2", ">"; "ABCD"; EOS = "D"

Receive Data

RECV<talker> <count> <EOS> <data buffer pointer>

This system command is used to input data from a device. The data is typically a string of ASCII characters.

This routine is the dual of SEND. It assumes a new talker will be specified, a count of less than 257, and an EOS character to terminate the input. EOI received will also terminate the input. Figure 15 shows the flow chart for the RECV ending conditions. My Listen Address (MLA) is sent to keep the GPIB transactions totally regular to facilitate analysis by a GPIB logic analyzer like the Ziatech ZT488. Otherwise, the bus would appear to have no listener even though the 8291 will be listening.

Note that although the count may go to zero before the transmission ends, the talker will probably be left in a strange state and may have to be cleared by the controller. The count ending of RECV is therefore used as an error condition in most situations.

```
RECV:
  Put EOS into 8291                          ;End of string compare character
  If 40H ≤ talker ≤ 5EH then                 ;GPIB talk addresses are
    Output-to-8291 talker                    ;"@" thru "^" ASCII
  Increment talker pointer                   ;Do this for consistency's sake
  Output-to-8291 UNL, MLA                    ;Everyone except us stop listening
  Enable-8291
    Holdoff on end                           ;Stop when EOS character is
    End on EOS received                      ;Detected by 8291
    lon, reset ton                           ;Listen only (no talk)
    Immediate execute pon
  Output-to-8292 GTSB                        ;8292 stops asserting ATN, go to standby
  While not (end or count = 0 (or tout2))    ;wait for EOS or EOI or end of count
                                             ;optionally check for stuck bus-tout2
    Input-from-8291 data                     ;input data, one byte at a time
    Increment data buffer pointer
    Decrement count                          ;Use 8085 C register as counter
  (If count = 0 then error)                  ;Count should not occur before end
  Output-to-8292 TCSY                        ;8292 asserts ATN take control
  (If Tout3 then take control async.)        ;If unable to take control sync.
  Enable-8291                                ;Put 8291 back as needed for
    No holdoff on end                        ;Controller activity and
    No end on EOS received                   ;Clear holdoff due to end
    ton, reset lon
    Finish handshake                         ;Complete holdoff due to end, if any
    Immediate execute pon                    ;Needed to reset lon
  Return error-indicator
```

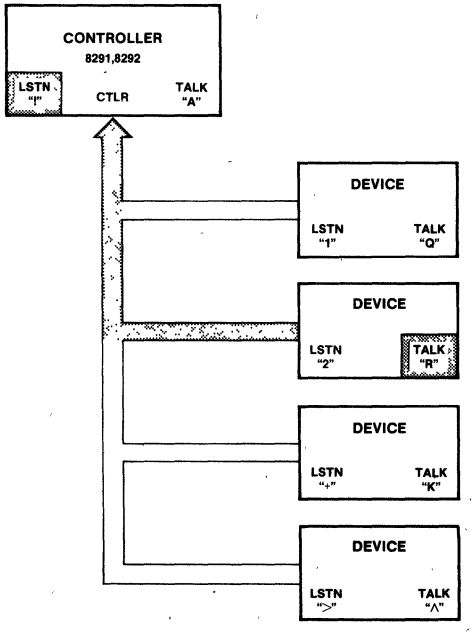


Figure 17. RECV from "R"; EOS = 0DH

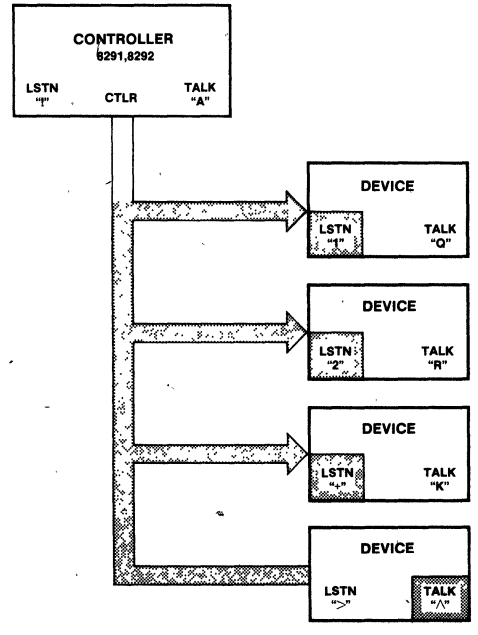


Figure 18. XFER from "^" to "1", "2", "+"; EOS = 0DH

AFN-01380A

Transfer Data

XFER < Talker > < Listener list > < EOS >

This system command is used to transfer data from a talker to one or more listeners where the controller does not participate in the transfer of the ASCII data. This is accomplished through the use of the 8291's continuous acceptor handshake mode while in listen-only.

This routine assumes a device list that has the ASCII talker address as the first byte and the string (one or more) of ASCII listener addresses following. The EOS character or an EOI will cause the controller to take control synchronously and thereby terminate the transfer.

```
XFER:
  Output-to-8291: Talker, UNL          ;Send talk address and unlisten
  While 20H ≤ listen ≤ 3EH
    Output-to-8291: Listener           ;Send listen address
    Increment listen list pointer
  Enable-8291
    lon, no ton                        ;Controller is pseudo listener
    Continuous AH mode                 ;Handshake but don't capture data
    End on EOS received                ;Capture EOS as well as EOI
    Immediate execute PON              ;Initialize the 8291
  Put EOS into 8291                    ;Set up EOS character
  Output-to-8292: GTSB                 ;Go to standby
                                       ;8292 waits for EOS or EOI and then
  Upon end (or tout2) then
    Take control synchronously         ;Regains control
  Enable-8291                          ;Go to Ready for Data
    Finish handshake
    Not continuous AH mode
    Not END on EOS received
    ton
    Immediate execute pon
  Return
```

CONTROLLER

Group Execute Trigger

TRIG < Listener list >

This system command causes a group execute trigger (GET) to be sent to all devices on the listener list. The intended use is to synchronize a number of instruments.

```
TRIG:
  Output-to-8291 UNL                   ;Everybody stop listening
  While 20H ≤ listener ≤ 3EH           ;Check for valid listen address
    Output-to-8291 Listener            ;Address each listener
    Increment listen list pointer      ;Terminate on any non-valid character
  Output-to-8291 GET                   ;Issue group execute trigger
  Return
```

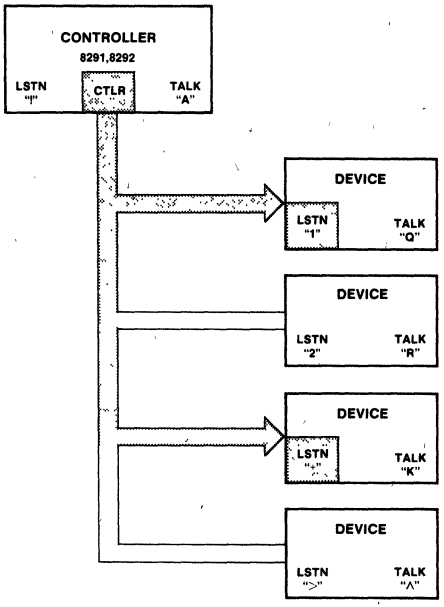


Figure 19. TRIG "1", "+"

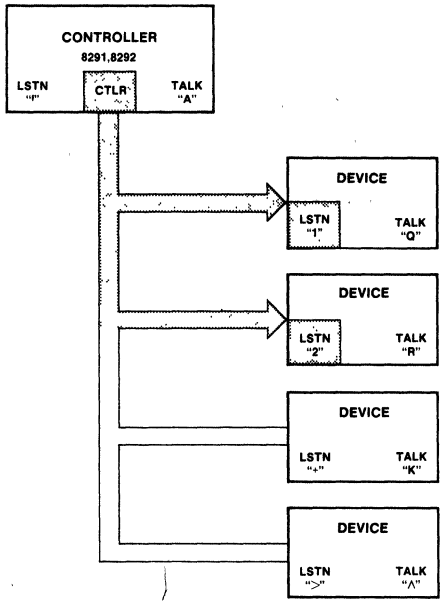


Figure 20. DCLR "1", "2"

Device Clear

DCLR < Listener list >

This system command causes a device clear (SDC) to be sent to all devices on the listener list. Note that this is not intended to clear the GPIB interface of the device, but should clear the device-specific logic.

```
DCLR:
  Output-to-8291 UNL                  ;Everybody stop listening
  While 20H ≤ Listener ≤ 3EH          ;Check for valid listen address
    Output-to-8291 listener           ;Address each listener
    Increment listen list pointer     ;Terminate on any non-valid character
  Output-to-8291 SDC                  ;Selective device clear
  Return
```

Serial Poll

SPOL < Talker list > < status buffer pointer >

This system command sequentially addresses the designated devices and receives one byte of status from each. The bytes are stored in the buffer in the same order as the devices appear on the talker list. MLA is output for completeness.

```
SPOL:
  Output-to-8291 UNL, MLA, SPE          ;Unlisten, we listen, serial poll enable
                                        ;Only one byte of serial poll
                                        ;Status wanted from each talker
  While 40H ≤ talker ≤ 5EH              ;Check for valid transfer
    Output-to-8291 talker               ;Address each device to talk
    Increment talker list pointer       ;One at a time
    Enable-8291
      lon, reset ton                    ;Listen only to get status
      Immediate execute pon             ;This resets ton
    Output-to-8292 GTSB                 ;Go to standby
    Wait for data in (BI)               ;Serial poll status byte into 8291
    Output-to-8292 TCSY                 ;Take control synchronously
    Input-from-8291 data                ;Actually get data from 8291
    Increment buffer pointer
    Enable 8291
      ton, reset lon
      Immediate execute pon             ;Resets lon
  Output-to-8291 SPD                    ;Send serial poll disable after all devices polled
  Return
```

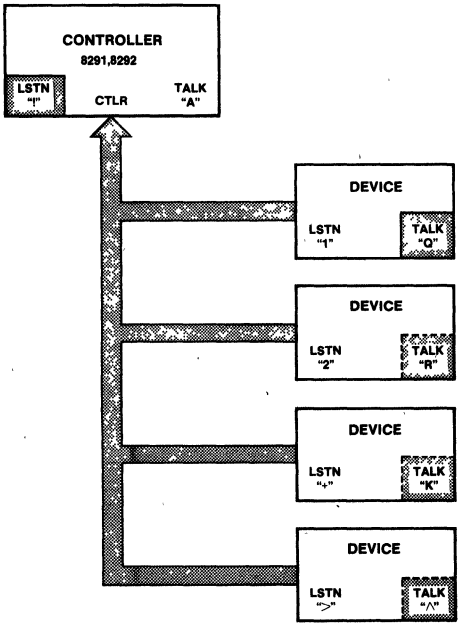


Figure 21. SPOL "Q", "R", "K", "^"

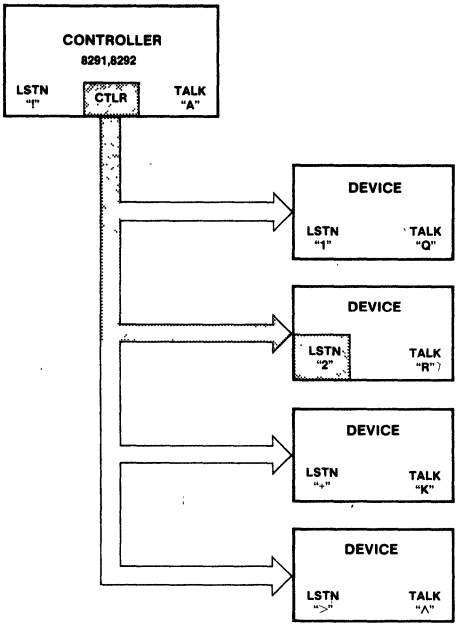


Figure 22. PPEN "2"; $iP_3P_2P_1$ = 0111B

Parallel Poll Enable

PPEN < Listener list > < Configuration Buffer pointer >

This system command configures one or more devices to respond to Parallel Poll, assuming they implement subset PP1. The configuration information is stored in a buffer with one byte per device in the same order as devices appear on the listener list. The configuration byte has the format XXXXIP3P2P1 as defined by the IEEE Std. P3P2P1 indicates the bit # to be used for a response and I indicates the assertion value. See Sec. 2.9.3.3 of the Std. for more details.

```
PPEN:
  Output-to-8291 UNL                    ;Universal unlisten
  While 20H ≤ Listener ≤ 3EH            ;Check for valid listener
    Output-to-8291 listener             ;Stop old listener, address new
    Output-to-8291 PPC, (PPE or data)   ;Send parallel poll info
    Increment listener list pointer     ;Point to next listener
    Increment buffer pointer            ;One configuration byte per listener
  Return
```

Parallel Poll Disable

PPDS < listener list >

This system command disables one or more devices from responding to a Parallel Poll by issuing a Parallel Poll Disable (PPD). It does not deconfigure the devices.

```
PPDS:
  Output-to-8291 UNL                    ;Universal Unlisten
  While 20H ≤ Listener ≤ 3EH            ;Check for valid listener
    Output-to-8291 listener             ;Address listener
    Increment listener list pointer
  Output-to-8291 PPC, PPD               ;Disable PP on all listeners
  Return
```

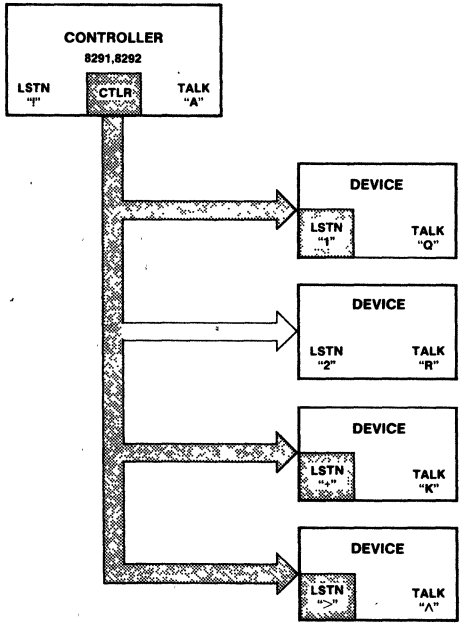


Figure 23. PPDS "1", "+", ">"

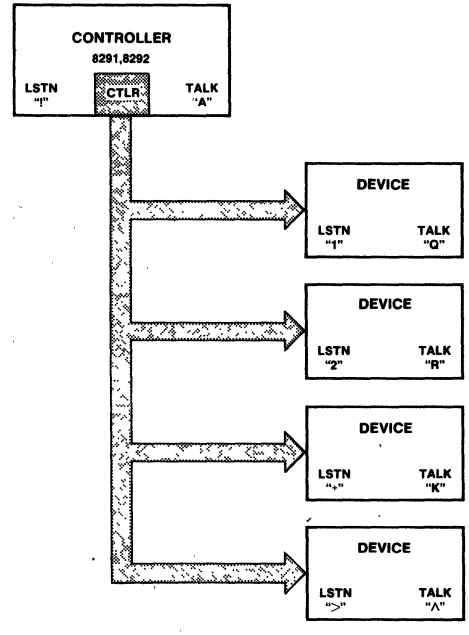


Figure 24. PPUN

Parallel Poll Unconfigure

PPUN

This system command deconfigures the Parallel Poll response of all devices by issuing a Parallel Poll Unconfigure message.

```
PPUN:
  Output-to-8291 PPU                ;Unconfigure all parallel poll
  Return
```

Conduct a Parallel Poll

PPOL

This system command causes the controller to conduct a Parallel Poll on the GPIB for approximately 12.5 usec (at 6 MHz). Note that a parallel poll does not use the handshake; therefore, to ensure that the device knows whether or not its positive response was observed by the controller, the CPU should explicitly acknowledge each device by a device-dependent data string. Otherwise, the response bit will still be set when the next Parallel Poll occurs. This command returns one byte of status.

```
PPOL:
  Enable-8291
    lon                             ;Listen only
    Immediate execute pon           ;This resets ton
  Output-to-8292 EXPP               ;Execute parallel poll
  Upon BI                           ;When byte is input
    Input-from-8291 data            ;Read it
  Enable-8291
    ton                             ;Talk only
    Immediate execute pon           ;This resets lon
  Return Data (status byte)
```

Pass Control

PCTL <talker>

This system command allows the controller to relinquish active control of the GPIB to another controller. Normally some software protocol should already have informed the controller to expect this, and under what conditions to return control. The 8291 must be set up to become a normal device and the CPU must handle all commands passed through, otherwise control cannot be returned (see Receive Control below). The controller will go idle.

```
PCTL:
  If 40H ≤ talker ≤ 5EH then
    if talker <  > MTA then         ;Cannot pass control to myself
      output-to-8291 talker, TCT    ;Take control message to talker
      Enable-8291                   ;Set up 8291 as normal device
        not ton, not lon
        Immediate execute pon       ;Reset ton and lon
        My device address, mode 1   ;Put device number in Register 6
        Undefined command pass through ;Required to receive control
        (Parallel Poll Configuration) ;Optional use of PP
      Output-to-8292 GIDL           ;Put controller in idle
  Return
```

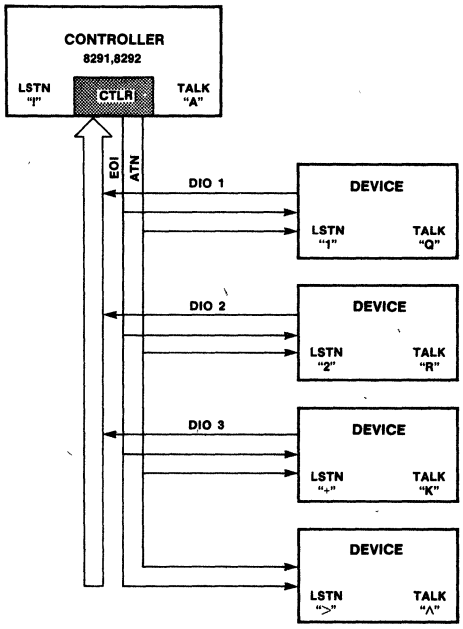


Figure 25. PPOL

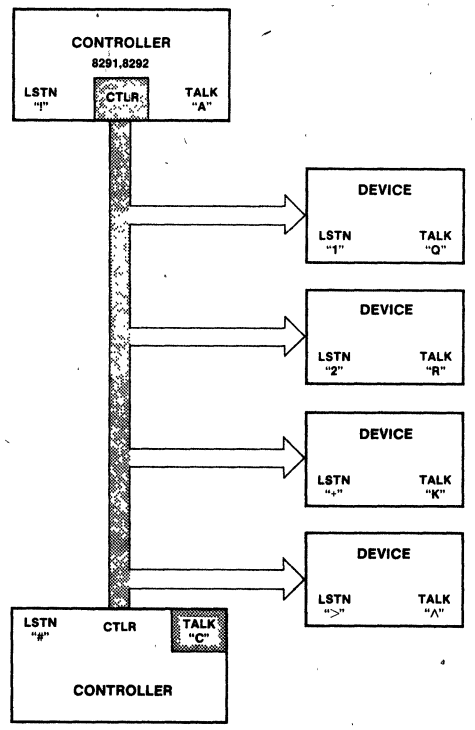


Figure 26. PCTL "C"

Receive Control

RCTL

This system command is used to get control back from the current controller-in-charge if it has passed control to this inactive controller. Most GPIB systems do not use more than one controller and therefore would not need this routine.

To make passing and receiving control a manageable event, the system designer should specify a protocol whereby the controller-in-charge sends a data message to the soon-to-be-active controller. This message should give the current state of the system, why control is being passed, what to do, and when to pass control back. Most of these issues are beyond the scope of this Ap Note.

```
RCTL:
  Upon CPT                          ;Wait for command pass through bit in 8291
    If (command=TCT) then           ;If command is take control and
      If TA then                    ;We are talker addressed
        Enable-8291

          Disable major device number   ;Controller will use ton and lon
          ton                           ;Talk only mode
          Mask off interrupts
          Immediate execute pon
```

```
        Output-to-8292 TCNTR          ;Take (receive) control
        Enable-8291
          Valid command               ;Release handshake
        Return valid
      Else
        Enable-8291
          Invalid command             ;Not talker addr. so TCT not for us
    Else
      Enable-8291
        Invalid command               ;Not TCT, so we don't care
  Return invalid
```

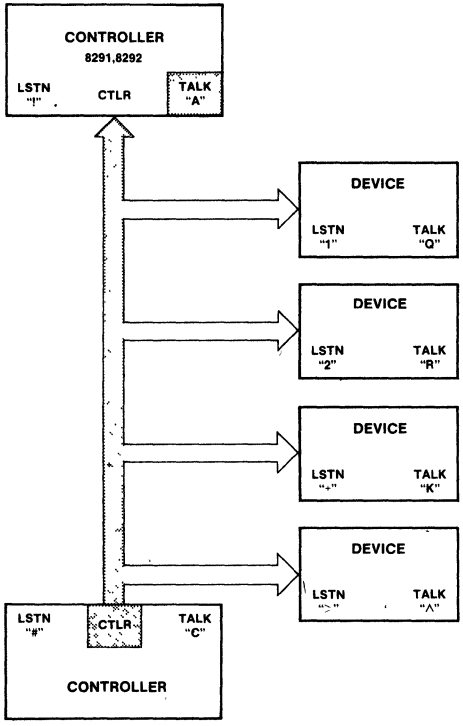


Figure 27. RCTL

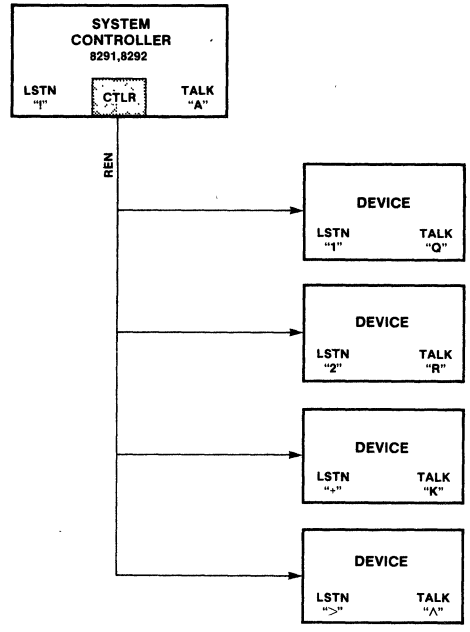


Figure 28. REME

Service Request

SRQD

This system command is used to detect the occurrence of a Service Request on the GPIB. One or more devices may assert SRQ simultaneously, and the CPU would normally conduct a Serial Poll after calling this routine to determine which devices are SRQing.

```
SRQD:
  If SRQ then                          ;Test 92 status bit
    Output-to-8292 IACK.SRQ            ;Acknowledge it
    Return SRQ
  Else return no SRQ
```

SYSTEM CONTROLLER

Remote Enable

REME

This system command asserts the Remote Enable line (REN) on the GPIB. The devices will not go remote until they are later addressed to listen by some other system command.

```
REME:
  Output-to-8292 SREM                  ;8292 asserts remote enable line
  Return
```

Local

LOCL

This system command deasserts the REN line on the GPIB. The devices will go local immediately.

```
LOCL:
  Output-to-8292 SLOC                  ;8292 stops asserting remote enable
  Return
```

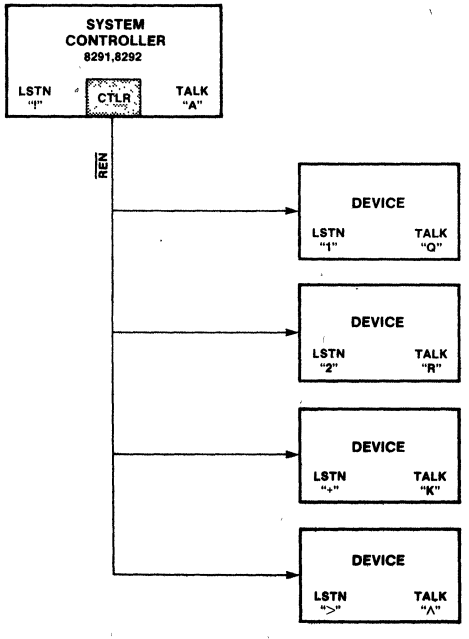


Figure 29. LOCL

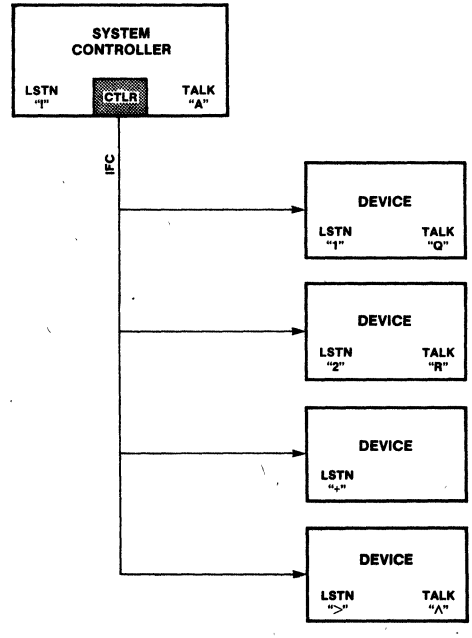


Figure 30. IFCL

Interface Clear/Abort

IFCL

This system command asserts the GPIB's Interface Clear (IFC) line for at least 100 microseconds. This causes all interface logic in all devices to go to a known state. Note that the device itself may or may not be reset, too. Most instruments do totally reset upon IFC. Some devices may require a DCLR as well as an IFCL to be completely reset. The (system) controller becomes Controller-in-Charge.

```
IFCL:
  Output-to-8292 ABORT          ;8292 asserts Interface Clear
  Return                        ;For 100 microseconds
```

INTERRUPTS AND DMA CONSIDERATIONS

The previous sections have discussed in detail how to use the 8291, 8292, 8293 chip set as a GPIB controller with the software operating in a polling mode and using programmed transfer of the data. This is the simplest mode of use, but it ties up the microprocessor for the duration of a GPIB transaction. If system design constraints do not allow this, then either Interrupts and/or DMA may be used to free up processor cycles.

The 8291 and 8292 provide sufficient interrupts that one may return to do other work while waiting for such things as 8292 Task Completion, 8291 Next Byte In, 8291 Last Byte Out, 8292 Service Request In, etc. The only difficulty lies in integrating these various interrupt sources and their matching routines into the overall system's interrupt structure. This is highly situation-specific and is beyond the scope of this Ap Note.

The strategy to follow is to replace each of the WAIT routines (see Appendix A) with a return to the main code and provide for the corresponding interrupt to bring the control back to the next section of GPIB code. For example WAITO (Wait for Byte Out of 8291) would be replaced by having the BO interrupt enabled and storing the (return) address of the next instruction in a known place. This co-routine structure will then be activated by a BO interrupt. Fig. 31 shows an example of the flow of control.

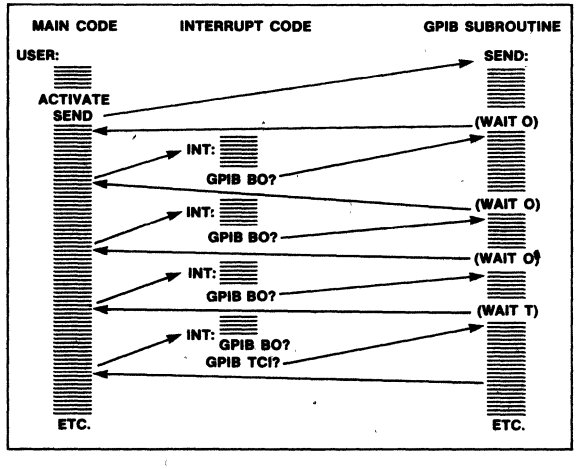


Figure 31. GPIB Interrupt & Co-Routine Flow of Control

DMA is also useful in relieving the processor if the average length of a data buffer is long enough to overcome the extra time used to set up a DMA chip. This decision will also be a function of the data rate of the instrument. The best strategy is to use the DMA to handle only the data buffer transfers on SEND and RECV and to do all the addressing and control just as shown in the driver descriptions.

Another major reason for using a DMA chip is to increase the data rate and therefore increase the overall transaction rate. In this case the limiting factor becomes the time used to do the addressing and control of the GPIB using software like that in Appendix A. The data transmission time becomes insignificant at DMA speeds unless extremely long buffers are used.

Refer to Figure 11 for a typical DMA and interrupt based design using the 8291, 8292, 8293. A system like this can achieve a 250K byte transfer rate while under DMA control.

APPLICATION EXAMPLE

This section will present the code required to operate a typical GPIB instrument set up as shown in Fig. 32. The HP5328A universal counter and the HP3325 function generator are typical of many GPIB devices; however, there are a wide variety of software protocols to be found on the GPIB. The Ziatech ZT488 GPIB analyzer is used to single step the bus to facilitate debugging the system. It also serves as a training/familiarization aid for newcomers to the bus.

This example will set up the function generator to output a specific waveform, frequency and amplitude. It will then tell the counter to measure the frequency and Request Service (SRQ) when complete. The program will then read in the data. The assembled source code will be found at the end of Appendix A.

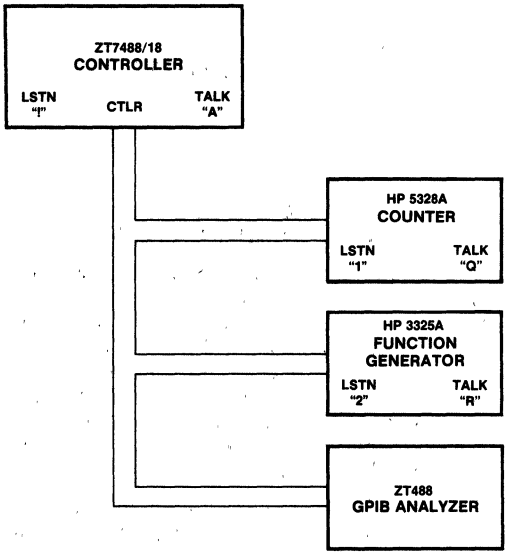


Figure 32. GPIB Example Configuration

```
SEND
  LSTN: "2", COUNT: 15, EOS: 0DH, DATA: "FU1FR37KHAM2VO (CR)"
  ;SETS UP FUNCTION GEN. TO 37 KHZ SINE, 2 VOLTS PP
  ;COUNT EQUAL TO # CHAR IN BUFFER
  ;EOS CHARACTER IS (CR) = 0DH = CARRIAGE RETURN
SEND
  LSTN: "1", COUNT: 6, EOS: "T" DATA: "PR4G7T"
  ;SETS UP COUNTER FOR P:INITIALIZE, F4: FREQ CHAN A
  ;            G7:0.1 HZ RESOLUTION, T:TRIGGER AND SRQ
  ;COUNT IS EQUAL TO # CHAR
WAIT FOR SRQ
SPOL TALK: "Q", DATA: STATUS 1
  ;CLEARS THE SRO—IN THIS EXAMPLE ONLY FREQ CTR ASSERTS SRQ
RECV TALK: "Q", COUNT: 17, EOS: 0AH,
  DATA: " +      37000.0E+0" (CR) (LF)
  ;GETS 17 BYTES OF DATA FROM COUNTER
  ;COUNT IS EXACT BUFFER LENGTH
  ;DATA SHOWN IS TYPICAL HP5328A READING THAT WOULD BE RECEIVED
```

CONCLUSION

This Application Note has shown a structured way to view the IEEE 488 bus and has given typical code sequences to make the Intel 8291, 8292, and 8293's behave as a controller of the GPIB. There are other ways to use the chip set, but whatever solution is chosen, it must be integrated into the overall system software.

The ultimate reference for GPIB questions is the IEEE Std 488, -1978 which is available from IEEE, 345 East 47th St., New York, NY, 10017. The ultimate reference for the 8292 is the source listing for it (remember it's a pre-programmed UPI-41A) which is available from INSITE, Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

APPENDIX A

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0
GPIB CONTROLLER SUBROUTINES

LOC  OBJ          LINE        SOURCE STATEMENT

                     1 $TITLE('GPIB CONTROLLER SUBROUTINES')
                     2 ;
                     3 ; GPIB CONTROLLER SUBROUTINES
                     4
                     5 ;
                     6 ;       for Intel 8291, 8292 on ZT 7488/18
                     7 ;       Bert Forbes, Ziatech Corporation
                     8 ;       2410 Broad Street
                     9 ;       San Luis Obispo, CA, USA 93401
                    10 ;
                    11 ;
                    12 ;       General Definitions & Equates
                    13 ;       8291 Control Values
                    14 ;
1000                15         ORG     1000H   ; For ZT7488/18 w/8085
                    16 ;
0060                17 PRT91   EQU     60H     ;8291 Base Port #
                    18 ;
                    19 ;       Reg #0 Data in & Data out
0060                20 DIN     EQU     PRT91+0 ;91 Data in reg
0060                21 DOUT    EQU     PRT91+0 ;91 Data out reg
                    22 ;
                    23 ;       Req # 1 Interrupt 1 Constants
0061                24 INT1    EQU     PRT91+1 ;INT Reg 1
0061                25 INTM1   EQU     PRT91+1 ;INT Mask Reg. 1
0002                26 BOM     EQU     02      ;91 BO INTRP Mask
0001                27 BIM     EQU     01      ;91 BI INTRP Mask
0010                28 ENDMK   EQU     10H     ;91 END INTRP Mask
0080                29 CPT     EQU     80H     ;91 command pass thru int bit
                    30 ;
                    31 ;       Reg #2 Interrupt 2
0062                32 INT2    EQU     PRT91+2
                    33 ;
                    34 ;       Reg #4 Address Mode Constants
0064                35 ADRMD   EQU     PRT91+4 ;91 address mode register #
0080                36 TON     EQU     80H     ;91 talk only mode & not listen only
0040                37 LON     EQU     40H     ;91 listen only & not ton
00C0                38 TLON    EQU     0C0H    ;91 talk & listen only
0001                39 MODE1   EQU     01      ;mode 1 addressing for device
                    40
                    41 ;       Reg #4          (Read)  Address Status Register
0064                42 ADRST   EQU     PRT91+4 ;reg #4
0020                43 EOIST   EQU     20H
0002                44 TA      EQU     2
0001                45 LA      EQU     1       ;listener active
                    46 ;
                    47 ;       Reg #5  (Write) Auxillary Mode Register
0065                48 AUXMD   EQU     PRT91+5 ;91 auxillary mode register #
0023                49 CLKRT   EQU     23H     ;91 3 Mhz clock input
```

```
0003              50 FNHSK   EQU     03      ;91 fininsh handshake command
0006              51 SDEOI   EQU     06      ;91 send EOI with next byte
0080              52 AXRA    EQU     80H     ;91 aux. reg A pattern
0001              53 HOHSK   EQU      1      ;91 hold off handshake on all bytes
0002              54 HOEND   EQU      2      ;91 hold off handshake on end
0003              55 CAHCY   EQU      3      ;91 continuous AH cycling
0004              56 EDEOS   EQU      4      ;91 end on EOS received
0008              57 EOIS    EQU      8      ;91 output EOI on EOS sent
000F              58 VSCMD   EQU     0FH     ;91 valid command pass through
0007              59 NVCMD   EQU     07H     ;91 invalid command pass through
00A0              60 AXRB    EQU     0A0H    ;Aux. reg. B pattern
0001              61 CPTEN   EQU     01H     ;command pass thru enable
                  62 ;
                  63 ;       Reg #5  (Read)
0065              64 CPTRG   EQU     PRT91+5
                  65 ;
                  66 ;       Reg #6  Address 0/1 reg. constants
0066              67 ADR01   EQU     PRT91+6
0060              68 DTDL1   EQU     60H     ;Disable major talker & listener
00E0              69 DTDL2   EQU     0E0H    ;Disable minor talker & listener
                  70 ;
                  71 ;       Reg #7  EOS     Character Register
0067              72 EOSR    EQU     PRT91+7
                  73 ;
                  74 ;
                  75 ;       8292    CONTROL VALUES
                  76 ;
                  77 ;
                  78 ;
0068              79 PRT92   EQU     PRT91+8 ;8292 Base Port # (CS7)
                  80 ;
0068              81 INTMR   EQU     PRT92+0 ;92 INTRP Mask Reg
00A0              82 INTM    EQU     0A0H    ;TCI
                  83 ;
0068              84 ERRM    EQU     PRT92+0 ;92 Error Mask Reg
0001              85 TOUT1   EQU     01      ;92 Time Out for Pass Control
0002              86 TOUT2   EQU     02      ;92 Time Out for Standby
0004              87 TOUT3   EQU     04      ;92 Time Out for Take Control Sync
0068              88 EVREG   EQU     PRT92+0 ;92 Event Counter Pseudo Reg
0068              89 TOREG   EQU     PRT92+0 ;92 Time Out Pseudo Reg
                  90 ;
0069              91 CMD92   EQU     PRT92+1 ;92 Command Register
                  92 ;
0069              93 INTST   EQU     PRT92+1 ;92 Interrupt Status Reg
0010              94 EVBIT   EQU     10H     ;Event Counter Bit
0002              95 IBFBT   EQU     02      ;Input Buffer Full Bit
0020              96 SRQBT   EQU     20H     ;Seq bit
                  97 ;
0068              98 ERFLG   EQU     PRT92+0 ;92 Error Flag Pseudo Reg
0068              99 CLRST   EQU     PRT92+0 ;92 Controller Status Pseudo Reg
0068             100 BUSST   EQU     PRT92+0 ;92 GPIB (Bus) Status Pseudo Reg
0068             101 EVCST   EQU     PRT92+0 ;92 Event Counter Status Pseudo Reg
0068             102 TOST    EQU     PRT92+0 ;92 Time Out Status Pseudo Reg
                 103 ;
                 104 ;       8292    OPERATION COMMANDS
                 105 ;
                 106 ;
00F0             107 SPCNI   EQU     0F0H    ;Stop Counter Interrupts
00F1             108 GIDL    EQU     0F1H    ;Go to idle
00F2             109 RSET    EQU     0F2H    ;Reset
00F3             110 RSTI    EQU     0F3H    ;Reset Interrupts
00F4             111 GSEC    EQU     0F4H    ;Goto standby, enable counting
00F5             112 EXPP    EQU     0F5H    ;Execute parallel poll
00F6             113 GTSB    EQU     0F6H    ;Goto standby
00F7             114 SLOC    EQU     0F7H    ;Set local mode
00F8             115 SREM    EQU     0F8H    ;Set interface to remote
00F9             116 ABORT   EQU     0F9H    ;Abort all operation, clear interface
00FA             117 TCNTR   EQU     0FAH    ;Take control (Receive control)
00FC             118 TCASY   EQU     0FCH    ;Take control asyncronously
00FD             119 TCSY    EQU     0FDH    ;Take control syncronously
00FE             120 STCNI   EQU     0FEH    ;Start counter interrupts
                 121 ;
                 122 ;
```

```
                    123 ;       8292    UTILITY COMMANDS
                    124 ;
                    125 ;
00E1                126 WOUT    EQU     0E1H    ;Write to timeout reg
00E2                127 WEVC    EQU     0E2H    ;Write to event counter
00E3                128 REVC    EQU     0E3H    ;Read event counter status
00E4                129 RERF    EQU     0E4H    ;Read error flag reg
00E5                130 RINM    EQU     0E5H    ;Read interrupt mask reg
00E6                131 RCST    EQU     0E6H    ;Read controller status reg
00E7                132 RBST    EQU     0E7H    ;Read GPIB Bus status reg
00E9                133 RTOUT   EQU     0E9H    ;Read timeout status reg
00EA                134 RERM    EQU     0EAH    ;Read error mask reg
000B                135 IACK    EQU     0BH     ;Interrupt Acknowledge
                    136 ;
                    137 ;
                    138 ;       PORT F BIT ASSIGNMENTS
                    139 ;
                    140 ;
                    141 ;
006F                142 PRTF    EQU     PRT91+0FH       ;ZT7488 port 6F for interrupts
0002                143 TCIF    EQU     02H     ;Task complete interrupt
0004                144 SPIF    EQU     04H     ;Special interrupt
0008                145 OBFF    EQU     08H     ;92 Output (to CPU) Buffer full
0010                146 IBFF    EQU     10H     ;92 Input (from CPU) Buffer empty
0001                147 BOF     EQU     01H     ;91 Int line (BO in this case)
                    148 ;
                    149 ;       GPIB MESSAGES (COMMANDS)
                    150 ;
0001                151 MDA     EQU     1       ;My device address is 1
0041                152 MTA     EQU     MDA+40H ;My talk address is 1 ("A")
0021                153 MLA     EQU     MDA+20H ;My listen address is 1 ("!")
003F                154 UNL     EQU     3FH     ;Universal unlisten
0008                155 GET     EQU     08      ;Group Execute Trigger
0004                156 SDC     EQU     04H     ;Device Clear
0018                157 SPE     EQU     18H     ;Serial poll enable
0019                158 SPD     EQU     19H     ;Serial poll disable
0005                159 PPC     EQU     05      ;Parallel poll configure
0070                160 PPD     EQU     70H     ;Parallel poll disable
0060                161 PPE     EQU     60H     ;Parallel poll disable
0015                162 PPU     EQU     15H     ;Parallel poll unconfigured
0009                163 TCT     EQU     09      ;Take control (pass control)
                    164
                    165 ;       MACRO DEFINITIONS
                    166 ;
                    167 ;
                    168 ;
                    169 SETF    MACRO           ;Sets flags on A register
-                   170         ORA     A
                    171         ENDM
                    172 ;
                    173 WAITO   MACRO           ;Wait for last 91 byte to be done
                    174         LOCAL   WAITL
-                   175 WAITL:  IN      INT1    ;Get Int1 status
-                   176         ANI     BOM     ;Check for byte out
-                   177         JZ      WAITL   ;If not, try again
                    178         ENDM            ;until it is
                    179 ;
                    180 ;
                    181 WAITI   MACRO           ;Wait for 91 byte to be input
                    182         LOCAL   WAITL
-                   183 WAITL:  IN      INT1    ;Get INT1 status
-                   184         MOV     B,A     ;Save status in B
-                   185         ANI     BIM     ;Check for byte in
-                   186         JZ      WAITL   ;If not, just try again
                    187         ENDM            ;until it is
                    188 ;
                    189 WAITX   MACRO           ;Wait for 92's TCI to go false
                    190         LOCAL   WAITL
-                   191 WAITL:  IN      PRTF
-                   192         ANI     TCIF
-                   193         JNZ     WAITL
                    194         ENDM
                    195 ;
```

```
                    196 WAITT   MACRO
                    197         LOCAL   WAITL
-                   198 WAITL:  IN      PRTF    ;Get task complete int,etc.
-                   199         ANI     TCIF    ;Mask it
-                   200         JZ      WAITL   ;Wait for task to be complete
                    201         ENDM
                    202
                    203 RANGE   MACRO   LOWER,UPPER,LABEL
                    204                 ;Checks for value in range
                    205                 ;branches to label if not
                    206                 ;in range. Falls through if
                    207                 ;lower <= ( (H)(L) ) <= upper.
                    208                 ;Get next byte.
-                   209         MOV     A,M
-                   210         CPI     LOWER
-                   211         JM      LABEL
-                   212         CPI     UPPER+1
-                   213         JP      LABEL
                    214         ENDM
                    215 ;
                    216 CLRA    MACRO
-                   217         XRA     A       ;A XOR A =0
                    218         ENDM
                    219 ;
                    220 ;       All of the following routines have these common
                    221 ;       assumptions about the state of the 8291 & 8292 upon entry
                    222 ;       to the routine and will exit the routine in an identical state.
                    223 ;
                    224 ;
                    225 ;       8291:   BO is or has been set,
                    226 ;               All interrupts are masked off
                    227 ;               TON mode, not LA
                    228 ;               No holdoffs in effect or enabled
                    229 ;               No holdoffs waiting for finish command
                    230 ;
                    231 ;       8292:   ATN asserted (active controller)
                    232 ;               note: RCTL is an exception--- it expects
                    233 ;               to not be active controller
                    234 ;               Any previous task is complete & 92 is
                    235 ;               ready to receive next command.
                    236 ;       8085:   Pointer registers (DE,HL) end one
                    237 ;               beyond last legal entry
                    238 ;*******************************************************
                    239 ;
                    240 ;
                    241 ;       INITIALIZATION ROUTINE
                    242 ;
                    243 ;INPUTS:        None
                    244 ;OUTPUTS:       None
                    245 ;CALLS:         None
                    246 ;DESTROYS:      A,F
                    247 ;
1000 3EA0           248 INIT:   MVI     A,INTM  ;Enable TCI
1002 D368           249         OUT     INTMR   ;Output to 92's intr. mask reg
1004 3E60           250         MVI     A,DTDL1 ;Disable major talker/listener
1006 D366           251         OUT     ADR01
1008 3EE0           252         MVI     A,DTDL2 ;Disable minor talker/listener
100A D366           253         OUT     ADR01
100C 3E80           254         MVI     A,TON   ;Talk only mode
100E D364           255         OUT     ADRMD
1010 3E23           256         MVI     A,CLKRT ;3 MHZ for delay timer
1012 D365           257         OUT     AUXMD
                    258         CLRA
1014 AF             259+        XRA     A       ;A XOR A =0
1015 D361           260         OUT     INT1
1017 D362           261         OUT     INT2    ;Disable all 91 mask bits
1019 D365           262         OUT     AUXMD   ;Immediate execute PON
101B C9             263         RET
                    264 ;
                    265 ;*******************************************************
                    266 ;
                    267 ;
                    268 ;       SEND ROUTINE
                    269 ;
```

```
                    270 ;
                    271 ;
                    272 ;       INPUTS:         HL listener list pointer
                    273 ;                       DE data buffer pointer
                    274 ;                       C  count-- 0 will cause no data to be sent
                    275 ;                       b  EOS character-- software detected
                    276 ;       OUTPUTS:        none
                    277 ;       CALLS:          none
                    278 ;       DESTROYS:       A, C, DE, HL, F
                    279 ;
                    280 ;
                    281 ;
101C 3E41           282 SEND:   MVI     A,MTA   ;Send MTA to turn off any
101E D360           283         OUT     DOUT    ;previous talker
                    284         WAITO
1020 DB61           285+??0001:  IN     INT1    ;Get Intl status
1022 E602           286+        ANI     BOM     ;Check for byte out
1024 CA2010         287+        JZ      ??0001  ;If not, try again
1027 3E3F           288         MVI     A,UNL   ;Send universal unlisten
1029 D360           289         OUT     DOUT    ;to stop previous listeners
102B 78             290         MOV     A,B     ;Get EOS character
102C D367           291         OUT     EOSR    ;Output it to 8291
                    292                         ;while listener.....
                    293 SEND1:  RANGE   20H,3EH,SEND2   ;Check next listen address
                    294+                        ;Checks for value in range
                    295+                        ;branches to label if not
                    296+                        ;in range. Falls through if
                    297+                        ;lower <= ( (H)(L) ) <= upper.
                    298+                        ;Get next byte.
102E 7E             299+        MOV     A,M
102F FE20           300+        CPI     20H
1031 FA4710         301+        JM      SEND2
1034 FE3F           302+        CPI     3EH+1
1036 F24710         303+        JP      SEND2
                    304         WAITO                   ;Wait for previous listener sent
1039 DB61           305+??0002:  IN     INT1    ;Get Intl status
103B E602           306+        ANI     BOM     ;Check for byte out
103D CA3910         307+        JZ      ??0002  ;If not, try again
1040 7E             308         MOV     A,M             ;Get this listener
1041 D360           309         OUT     DOUT            ;Output to GPIB
1043 23             310         INX     H               ;Increment listener list pointer
1044 C32E10         311         JMP     SEND1           ;Loop till non-valid listener
                    312                                 ;Enable 91 ending conditions
                    313 SEND2:  WAITO               ;Wait for lstn addr accepted
1047 DB61           314+??0003:  IN     INT1    ;Get Intl status
1049 E602           315+        ANI     BOM     ;Check for byte out
104B CA4710         316+        JZ      ??0003  ;If not, try again
                    317                         ;WAITO required for early versions
                    318                         ;of 8292 to avoid GTSB before DAC
104E 3EF6           319         MVI     A,GTSB  ;Goto standby
1050 D369           320         OUT     CMD92   ;
1052 3E88           321         MVI     A,AXRA+EOIS     ;Send EOI with EOS character
1054 D365           322         OUT     AUXMD
                    323         WAITX               ;Wait for TCI to go false
1056 DB6F           324+??0004:  IN     PRTF
1058 E602           325+        ANI     TCIF
105A C25610         326+        JNZ     ??0004
                    327         WAITT               ;Wait for TCI on GTSB
105D DB6F           328+??0005:  IN     PRTF    ;Get task complete int,etc.
105F E602           329+        ANI     TCIF    ;Mask it
1061 CA5D10         330+        JZ      ??0005  ;Wait for task to be complete
                    331
                    332 ;       delete next 3 instructions to make count of 0=256
                    333 ;
1064 79             334         MOV     A,C     ;Get count
                    335         SETF            ;Set flags
1065 B7             336+        ORA     A
1066 CA8810         337         JZ      SEND6   ;If count=0, send no data
1069 1A             338 SEND3:  LDAX    D       ;Get data byte
106A D360           339         OUT     DOUT    ;Output to GPIB
106C B8             340         CMP     B       ;Test EOS ...this is faster
                    341                         ;and uses less code than using
                    342                         ;91's END or EOI bits
```

```
106D CA7F10          343          JZ      SEND5   ;If char = EOS , go finish
                     344 SEND4:   WAITO
1070 DB61            345+??0006:   IN     INT1    ;Get Intl status
1072 E602            346+          ANI    BOM     ;Check for byte out
1074 CA7010          347+          JZ     ??0006  ;If not, try again
1077 13              348           INX    D       ;Increment buffer pointer
1078 0D              349           DCR    C       ;Decrement count
1079 C26910          350           JNZ    SEND3   ;If count < > 0, go send
107C C38810          351           JMP    SEND6   ;Else go finish
107F 13              352 SEND5:    INX    D       ;for consistency
1080 0D              353           DCR    C       ; "       "
                     354           WAITO                  ;This ensures that the standard entry
1081 DB61            355+??0007:   IN     INT1    ;Get Intl status
1083 E602            356+          ANI    BOM     ;Check for byte out
1085 CA8110          357+          JZ     ??0007  ;If not, try again
                     358                          ;assumptions for the next subroutine are met
1088 3EFD            359 SEND6:    MVI    A,TCSY  ;Take control syncronously
108A D369            360           OUT    CMD92
108C 3E80            361           MVI    A,AXRA  ;Reset send EOI on EOS
108E D365            362           OUT    AUXMD
                     363           WAITX          ;Wait for TCI false
1090 DB6F            364+??0008:   IN     PRTF
1092 E602            365+          ANI    TCIF
1094 C29010          366+          JNZ    ??0008
                     367           WAITT          ;Wait for TCI
1097 DB6F            368+??0009:   IN     PRTF    ;Get task complete int,etc.
1099 E602            369+          ANI    TCIF    ;Mask it
109B CA9710          370+          JZ     ??0009  ;Wait for task to be complete
109E C9              371           RET
                     372 ;*****************************************************************************
                     373 ;
                     374 ;         RECEIVE ROUTINE
                     375 ;
                     376 ;
                     377 ;INPUT:           HL talker pointer
                     378 ;                 DE data buffer pointer
                     379 ;                 C  count (max buffer size) 0 implies 256
                     380 ;                 B  EOS character
                     381 ;OUTPUT:          Fills buffer pointed at by DE
                     382 ;CALLS:           None
                     383 ;DESTROYS:        A, BC, DE, HL, F
                     384 ;
                     385 ;RETURNS:         A=0 normal termination--EOS detected
                     386 ;                 A=40 Error--- count overrun
                     387 ;                 A<40 or A>5EH Error--- bad talk address
                     388 ;
                     389 ;
109F 78              390 RECV:     MOV    A,B     ;Get EOS character
10A0 D367            391           OUT    EOSR    ;Output it to 91
                     392           RANGE  40H,5EH,RECV6
                     393+                         ;Checks for value in range
                     394+                         ;branches to label if not
                     395+                         ;in range. Falls through if
                     396+                         ;lower <= ( (H)(L) ) <= upper.
                     397+                         ;Get next byte.
10A2 7E              398+          MOV    A,M
10A3 FE40            399+          CPI    40H
10A5 FA3911          400+          JM     RECV6
10A8 FE5F            401+          CPI    5EH+1
10AA F23911          402+          JP     RECV6
                     403                          ;valid if 40H<= talk <=5EH
10AD D360            404           OUT    DOUT    ;Output talker to GPIB
10AF 23              405           INX    H       ;Incr pointer for consistency
                     406           WAITO
10B0 DB61            407+??0010:   IN     INT1    ;Get Intl status
10B2 E602            408+          ANI    BOM     ;Check for byte out
10B4 CAB010          409+          JZ     ??0010  ;If not, try again
10B7 3E3F            410           MVI    A,UNL   ;Stop other listeners
10B9 D360            411           OUT    DOUT
                     412           WAITO
10BB DB61            413+??0011:   IN     INT1    ;Get Intl status
10BD E602            414+          ANI    BOM     ;Check for byte out
10BF CABB10          415+          JZ     ??0011  ;If not, try again
```

```
10C2 3E21          416          MVI     A,MLA    ;For completeness
10C4 D360          417          OUT     DOUT
10C6 3E86          418          MVI     A,AXRA+HOEND+EDEOS      ;End when
10C8 D365          419          OUT     AUXMD    ;EOS or EOI & Holdoff
                   420          WAITO
10CA DB61          421+??0012:   IN     INT1     ;Get Intl status
10CC E602          422+         ANI     BOM      ;Check for byte out
10CE CACA10        423+         JZ      ??0012   ;If not, try again
10D1 3E40          424          MVI     A,LON    ;Listen only
10D3 D364          425          OUT     ADRMD
                   426          CLRA             ;Immediate XEQ PON
10D5 AF            427+         XRA     A        ;A XOR A =0
10D6 D365          428          OUT     AUXMD
10D8 3EF6          429          MVI     A,GTSB   ;Goto standby
10DA D369          430          OUT     CMD92
                   431          WAITX            ;Wait for TCI=0
10DC DB6F          432+??0013:  IN      PRTF
10DE E602          433+         ANI     TCIF
10E0 C2DC10        434+         JNZ     ??0013
                   435          WAITT            ;Wait for TCI=1
10E3 DB6F          436+??0014:  IN      PRTF     ;Get task complete int,etc.
10E5 E602          437+         ANI     TCIF     ;Mask it
10EA DB61          439 RECV1:   IN      INT1     ;Get 91 Int status (END &/or BI)
10EC 47            440          MOV     B,A      ;Save it in B for BI check later
10ED E610          441          ANI     ENDMK    ;Check for EOS or EOI
10EF C20511        442          JNZ     RECV2    ;Yes end--- go wait for BI
10F2 78            443          MOV     A,B      ;NO, retrieve status &
10F3 E601          444          ANI     BIM      ;check for BI
10F5 CAEA10        445          JZ      RECV1    ;NO, go wait for either END or BI
10F8 DB60          446          IN      DIN      ;YES, BI--- get data
10FA 12            447          STAX    D        ;Store it in buffer
10FB 13            448          INX     D        ;Increment buffer pointer
10FC 0D            449          DCR     C        ;Decrement counter
10FD C2EA10        450          JNZ     RECV1    ;If count < > 0 go back & wait
1100 0640          451          MVI     B,40H    ;Else set error indicator
1102 C31711        452          JMP     RECV5    ;And go take control
                   453 ;
1105 78            454 RECV2:   MOV     A,B      ;Retreive status
1106 E601          455 RECV3:   ANI     BIM      ;Check for BI
1108 C21011        456          JNZ     RECV4    ;If BI then go input data
110B DB61          457          IN      INT1     ;Else wait for last BI
110D C30611        458          JMP     RECV3    ;In loop
1110 DB60          459 RECV4:   IN      DIN      ;Get data byte
1112 12            460          STAX    D        ;Store it in buffer
1113 13            461          INX     D        ;Incr data pointer
1114 0D            462          DCR     C        ;Decrement count, but ignore it
1115 0600          463          MVI     B,0      ;Set normal completion indicators
                   464 ;
1117 3EFD          465 RECV5:   MVI     A,TCSY   ;Take control synchronously
1119 D369          466          OUT     CMD92
                   467          WAITX            ;Wait for TCI=0 (7 tcy)
111B DB6F          468+??0015:  IN      PRTF
111D E602          469+         ANI     TCIF
111F C21B11        470+         JNZ     ??0015
                   471          WAITT            ;Wait for TCI=1
1122 DB6F          472+??0016:  IN      PRTF     ;Get task complete int,etc.
1124 E602          473+         ANI     TCIF     ;Mask it
1126 CA2211        474+         JZ      ??0016   ;Wait for task to be complete
                   475 ;
                   476 ;if timeout 3 is to be checked, the above WAITT should
                   477 ;be omitted & the appropriate code to look for TCI or
                   478 ;TOUT3 inserted here.
                   479 ;
1129 3E80          480          MVI     A,AXRA   ;Pattern to clear 91 END conditions
112B D365          481          OUT     AUXMD    ;
112D 3E80          482          MVI     A,TON    ;This bit pattern already in "A"
112F D364          483          OUT     ADRMD    ;Output TON
1131 3E03          484          MVI     A,FNHSK  ;Finish handshake
1133 D365          485          OUT     AUXMD
                   486          CLRA
1135 AF            487+         XRA     A        ;A XOR A =0
1136 D365          488          OUT     AUXMD    ;Immediate execute PON-Reset LON
1138 78            489          MOV     A,B      ;Get completion character
1139 C9            490 RECV6:   RET
```

```
                    491 ;
                    492 ;*************************************************************
                    493 ;       XFER ROUTINE
                    494 ;
                    495 ;
                    496 ;INPUTS:         HL device list pointer
                    497 ;                B  EOS character
                    498 ;OUTPUTS:        None
                    499 ;CALLS:          None
                    500 ;DESTROYS:       A, HL, F
                    501 ;RETURNS:        A=0 normal, A < > 0 bad talker
                    502 ;
                    503 ;
                    504 ;NOTE:           XFER will not work if the talker
                    505 ;                uses EOI to terminate the transfer.
                    506 ;                Intel will be making hardware
                    507 ;                modifications to the 8291 that will
                    508 ;                correct this problem. Until that time,
                    509 ;                only EOS may be used without possible
                    510 ;                loss of the last data byte transfered.
                    511 XFER:    RANGE   40H,5EH,XFER4   ;Check for valid talker
                    512+                                ;Checks for value in range
                    513+                                ;branches to label if not
                    514+                                ;in range. Falls through if
                    515+                                ;lower <= ( (H)(L) ) <= upper.
                    516+                                ;Get next byte.
113A 7E             517+            MOV     A,M
113B FE40           518+            CPI     40H
113D FABB11         519+            JM      XFER4
1140 FE5F           520+            CPI     5EH+1
1142 F2BB11         521+            JP      XFER4
1145 D360           522             OUT     DOUT        ;Send it to GPIB
1147 23             523             INX     H           ;Incr pointer
                    524             WAITO
1148 DB61           525+??0017:      IN     INT1        ;Get Intl status
114A E602           526+            ANI     BOM         ;Check for byte out
114C CA4811         527+            JZ      ??0017      ;If not, try again
114F 3E3F           528             MVI     A,UNL       ;Universal unlisten
1151 D360           529             OUT     DOUT
                    530 XFER1:  RANGE   20H,3EH,XFER2   ;Check for valid listener
                    531+                                ;Checks for value in range
                    532+                                ;branches to label if not
                    533+                                ;in range. Falls through if
                    534+                                ;lower <= ( (H)(L) ) <= upper.
                    535+                                ;Get next byte.
1153 7E             536+            MOV     A,M
1154 FE20           537+            CPI     20H
1156 FA6C11         538+            JM      XFER2
1159 FE3F           539+            CPI     3EH+1
115B F26C11         540+            JP      XFER2
                    541             WAITO
115E DB61           542+??0018:      IN     INT1        ;Get Intl status
1160 E602           543+            ANI     BOM         ;Check for byte out
1162 CA5E11         544+            JZ      ??0018      ;If not, try again
1165 7E             545             MOV     A,M         ;Get listener
1166 D360           546             OUT     DOUT
1168 23             547             INX     H           ;Incr pointer
1169 C35311         548             JMP     XFER1       ;Loop until non-valid listener
                    549 XFER2:  WAITO
116C DB61           550+??0019:      IN     INT1        ;Get Intl status
116E E602           551+            ANI     BOM         ;Check for byte out
1170 CA6C11         552+            JZ      ??0019      ;If not, try again
1173 3E87           553             MVI     A,AXRA+CAHCY+EDEOS      ;Invisible handshake
1175 D365           554             OUT     AUXMD       ;Continuous AH mode
1177 3E40           555             MVI     A,LON       ;Listen only
1179 D364           556             OUT     ADRMD
                    557             CLRA
117B AF             558+            XRA     A           ;A XOR A =0
117C D365           559             OUT     AUXMD       ;Immed. XEQ PON
117E 78             560             MOV     A,B         ;Get EOS
117F D367           561             OUT     EOSR        ;Output it to 91
1181 3EF6           562             MVI     A,GTSB      ;Go to standby
1183 D369           563             OUT     CMD92
```

```
                    564          WAITX
1185 DB6F           565+??0020:  IN       PRTF
1187 E602           566+         ANI      TCIF
1189 C28511         567+         JNZ      ??0020
                    568          WAITT            ;Wait for TCS
118C DB6F           569+??0021:  IN       PRTF     ;Get task complete int,etc.
118E E602           570+         ANI      TCIF     ;Mask it
1190 CA8C11         571+         JZ       ??0021   ;Wait for task to be complete
1193 DB61           572 XFER3:   IN       INT1     ;Get END status bit
1195 E610           573          ANI      ENDMK    ;Mask it
1197 CA9311         574          JZ       XFER3
119A 3EFD           575          MVI      A,TCSY   ;Take control syncronously
119C D369           576          OUT      CMD92
                    577          WAITX
119E DB6F           578+??0022:  IN       PRTF
11A0 E602           579+         ANI      TCIF
11A2 C29E11         580+         JNZ      ??0022
                    581          WAITT            ;Wait for TCI
11A5 DB6F           582+??0023:  IN       PRTF     ;Get task complete int,etc.
11A7 E602           583+         ANI      TCIF     ;Mask it
11A9 CAA511         584+         JZ       ??0023   ;Wait for task to be complete
11AC 3E80           585          MVI      A,AXRA   ;Not cont AH or END on EOS
11AE D365           586          OUT      AUXMD
11B0 3E03           587          MVI      A,FNHSK  ;Finish handshake
11B2 D365           588          OUT      AUXMD
11B4 3E80           589          MVI      A,TON    ;Talk only
11B6 D364           590          OUT      ADRMD
                    591          CLRA              ;Normal return A=0
11B8 AF             592+         XRA      A        ;A XOR A =0
11B9 D365           593          OUT      AUXMD    ;Immediate XEQ PON
11BB C9             594 XFER4:   RET
                    595 ;
                    596 ;*************************************************
                    597 ;
                    598 ;
                    599 ;        TRIGGER ROUTINE
                    600 ;
                    601 ;
                    602 ;INPUTS:         HL listener list pointer
                    603 ;OUTPUTS:        None
                    604 ;CALLS:          None
                    605 ;DESTROYS:       A, HL, F
                    606 ;
                    607 ;
11BC 3E3F           608 TRIG:    MVI      A,UNL    ;
11BE D360           609          OUT      DOUT     ;Send universal unlisten
                    610 TRIG1:   RANGE    20H,3EH,TRIG2   ;Check for valid listen
                    611+                           ;Checks for value in range
                    612+                           ;branches to label if not
                    613+                           ;in range. Falls through if
                    614+                           ;lower <= ( (H)(L) ) <= upper.
                    615+                           ;Get next byte.
11C0 7E             616+         MOV      A,M
11C1 FE20           617+         CPI      20H
11C3 FAD911         618+         JM       TRIG2
11C6 FE3F           619+         CPI      3EH+1
11C8 F2D911         620+         JP       TRIG2
                    621          WAITO            ;Wait for UNL to finish
11CB DB61           622+??0024:   IN      INT1     ;Get Intl status
11CD E602           623+         ANI      BOM      ;Check for byte out
11CF CACB11         624+         JZ       ??0024   ;If not, try again
11D2 7E             625          MOV      A,M      ;Get listener
11D3 D360           626          OUT      DOUT     ;Send Listener to GPIB
11D5 23             627          INX      H        ;Incr. pointer
11D6 C3C011         628          JMP      TRIG1    ;Loop until non-valid char
                    629 TRIG2:   WAITO            ;Wait for last listen to finish
11D9 DB61           630+??0025:   IN      INT1     ;Get Intl status
11DB E602           631+         ANI      BOM      ;Check for byte out
11DD CAD911         632+         JZ       ??0025   ;If not, try again
11E0 3E08           633          MVI      A,GET    ;Send group execute trigger
11E2 D360           634          OUT      DOUT     ;to all addressed listeners
                    635          WAITO
11E4 DB61           636+??0026:   IN      INT1     ;Get Intl status
11E6 E602           637+         ANI      BOM      ;Check for byte out
```

```
11E8 CAE411        638+          JZ      ??0026  ;If not, try again
11EB C9            639           RET
                   640 ;
                   641 ;*****************************************
                   642 ;
                   643 ;DEVICE CLEAR ROUTINE
                   644 ;
                   645 ;
                   646 ;
                   647 ;INPUTS:          HL listener pointer
                   648 ;OUTPUT:          None
                   649 ;CALLS:           None
                   650 ;DESTROYS:        A, HL, F
                   651 ;
11EC 3E3F          652 DCLR:     MVI     A,UNL
11EE D360          653           OUT     DOUT
                   654 DCLR1:    RANGE   20H,3EH,DCLR2
                   655+                          ;Checks for value in range
                   656+                          ;branches to label if not
                   657+                          ;in range. Falls through if
                   658+                          ;lower <= ( (H)(L) ) <= upper.
                   659+                          ;Get next byte.
11F0 7E            660+          MOV     A,M
11F1 FE20          661+          CPI     20H
11F3 FA0912        662+          JM      DCLR2
11F6 FE3F          663+          CPI     3EH+1
11F8 F20912        664+          JP      DCLR2
                   665           WAITO
11FB DB61          666+??0027:    IN     INT1    ;Get Intl status
11FD E602          667+          ANI     BOM     ;Check for byte out
11FF CAFB11        668+          JZ      ??0027  ;If not, try again
1202 7E            669           MOV     A,M
1203 D360          670           OUT     DOUT    ;Send listener to GPIB
1205 23            671           INX     H
1206 C3F011        672           JMP     DCLR1
                   673 DCLR2:    WAITO
1209 DB61          674+??0028:    IN     INT1    ;Get Intl status
120B E602          675+          ANI     BOM     ;Check for byte out
120D CA0912        676+          JZ      ??0028  ;If not, try again
1210 3E04          677           MVI     A,SDC   ;Send device clear
1212 D360          678           OUT     DOUT    ;To all addressed listeners
                   679           WAITO
1214 DB61          680+??0029:    IN     INT1    ;Get Intl status
1216 E602          681+          ANI     BOM     ;Check for byte out
1218 CA1412        682+          JZ      ??0029  ;If not, try again
121B C9            683           RET
                   684 ;
                   685 ;*************************************************
                   686 ;
                   687 ;         SERIAL POLL ROUTINE
                   688 ;
                   689 ;INPUTS:          HL talker list pointer
                   690 ;                 DE status buffer pointer
                   691 ;OUTPUTS:         Fills buffer pointed to by DE
                   692 ;CALLS:           None
                   693 ;DESTROYS:        A, BC, DE, HL, F
                   694 ;
121C 3E3F          695 SPOL:     MVI     A,UNL   ;Universal unlisten
121E D360          696           OUT     DOUT
                   697           WAITO
1220 DB61          698+??0030:    IN     INT1    ;Get Intl status
1222 E602          699+          ANI     BOM     ;Check for byte out
1224 CA2012        700+          JZ      ??0030  ;If not, try again
1227 3E21          701           MVI     A,MLA   ;My listen address
1229 D360          702           OUT     DOUT
                   703           WAITO
122B DB61          704+??0031:    IN     INT1    ;Get Intl status
122D E602          705+          ANI     BOM     ;Check for byte out
122F CA2B12        706+          JZ      ??0031  ;If not, try again
1232 3E18          707           MVI     A,SPE   ;Serial poll enable
1234 D360          708           OUT     DOUT    ;To be formal about it
                   709           WAITO
1236 DB61          710+??0032:    IN     INT1    ;Get Intl status
```

```
1238 E602          711+          ANI     BOM      ;Check for byte out
123A CA3512        712+          JZ      ??0032   ;If not, try again
                   713 SPOL1:    RANGE   40H,5EH,SPOL2   ;Check for valid talker
                   714+                          ;Checks for value in range
                   715+                          ;branches to label if not
                   716+                          ;in range. Falls through if
                   717+                          ;lower <= ( (H)(L) ) <= upper.
                   718+                          ;Get next byte.
123D 7E            719+          MOV     A,M
123E FE40          720+          CPI     40H
1240 FA9412        721+          JM      SPOL2
1243 FE5F          722+          CPI     5EH+1
1245 F29412        723+          JP      SPOL2
1248 7E            724           MOV     A,M      ;Get talker
1249 D360          725           OUT     DOUT     ;Send to GPIB
124B 23            726           INX     H        ;Incr talker list pointer
124C 3E40          727           MVI     A,LON    ;Listen only
124E D364          728           OUT     ADRMD
                   729           WAITO            ;Wait for talk address to complete
1250 DB61          730+??0033:    IN     INT1     ;Get Intl status
1252 E602          731+          ANI     BOM      ;Check for byte out
1254 CA5012        732+          JZ      ??0033   ;If not, try again
                   733           CLRA             ;Pattern for immediate XEQ PON
1257 AF            734+          XRA     A        ;A XOR A =0
1258 D365          735           OUT     AUXMD
125A 3EF6          736           MVI     A,GTSB   ;Goto standby
125C D369          737           OUT     CMD92
                   738           WAITX            ;Wait for TCI false
125E DB6F          739+??0034:   IN      PRTF
1260 E602          740+          ANI     TCIF
1262 C25E12        741+          JNZ     ??0034
                   742           WAITT            ;Wait for TCI
1265 DB6F          743+??0035:   IN      PRTF     ;Get task complete int,etc.
1267 E602          744+          ANI     TCIF     ;Mask it
1269 CA6512        745+          JZ      ??0035   ;Wait for task to be complete
                   746           WAITI            ;Wait for status byte input
126C DB61          747+??0036:   IN      INT1     ;Get INT1 status
126E 47            748+          MOV     B,A      ;Save status in B
126F E601          749+          ANI     BIM      ;Check for byte in
1271 CA6C12        750+          JZ      ??0036   ;If not, just try again
1274 3EFD          751           MVI     A,TCSY   ;Take control sync
1276 D369          752           OUT     CMD92
                   753           WAITX            ;Wait for TCI false
1278 DB6F          754+??0037:   IN      PRTF
127A E602          755+          ANI     TCIF
127C C27812        756+          JNZ     ??0037
                   757           WAITT            ;Wait for TCI
127F DB6F          758+??0038:   IN      PRTF     ;Get task complete int,etc.
1281 E602          759+          ANI     TCIF     ;Mask it
1283 CA7F12        760+          JZ      ??0038   ;Wait for task to be complete
1286 DB60          761           IN      DIN      ;Get serial poll status byte
1288 12            762           STAX    D        ;Store it in buffer
1289 13            763           INX     D        ;Incr pointer
128A 3E80          764           MVI     A,TON    ;Talk only for controller
128C D364          765           OUT     ADRMD    ;
                   766           CLRA
128E AF            767+          XRA     A        ;A XOR A =0
128F D365          768           OUT     AUXMD    ;Immeditate XEQ PON
                   769                            ;CLR LA
1291 C33D12        770           JMP     SPOL1    ;Go on to next device on list
                   771 ;
1294 3E19          772 SPOL2:    MVI     A,SPD    ;Serial poll disable
1296 D360          773           OUT     DOUT     ;We know BO was set (WAITO above)
                   774           WAITO
1298 DB61          775+??0039:    IN     INT1     ;Get Intl status
129A E602          776+          ANI     BOM      ;Check for byte out
129C CA9812        777+          JZ      ??0039   ;If not, try again
                   778           CLRA
129F AF            779+          XRA     A        ;A XOR A =0
12A0 D365          780           OUT     AUXMD    ;Immediate XEQ PON to clear LA
12A2 C9            781           RET
                   782 ;
                   783 ;*****************************************************
                   784 ;
```

```
                        785 ;       PARALLEL POLL ENABLE ROUTINE
                        786 ;
                        787 ;INPUTS:         HL listener list pointer
                        788 ;                DE configuration byte pointer
                        789 ;OUTPUTS:        None
                        790 ;CALLS:          None
                        791 ;DESTROYS:       A, DE, HL, F
                        792 ;
                        793 ;
12A3 3E3F               794 PPEN:   MVI     A,UNL   ;Universal unlisten
12A5 D360               795         OUT     DOUT
                        796 PPEN1:  RANGE   20H,3EH,PPEN2   ;Check for valid listener
                        797+                        ;Checks for value in range
                        798+                        ;branches to label if not
                        799+                        ;in range. Falls through if
                        800+                        ;lower <= ( (H)(L) ) <= upper.
                        801+                        ;Get next byte.
12A7 7E                 802+        MOV     A,M
12A8 FE20               803+        CPI     20H
12AA FAD812             804+        JM      PPEN2
12AD FE3F               805+        CPI     3EH+1
12AF F2D812             806+        JP      PPEN2
                        807         WAITO           ;Valid wait 91 data out req
12B2 DB61               808+??0040:  IN     INT1    ;Get Intl status
12B4 E602               809+        ANI     BOM     ;Check for byte out
12B6 CAB212             810+        JZ      ??0040  ;If not, try again
12B9 7E                 811         MOV     A,M     ;Get listener
12BA D360               812         OUT     DOUT
                        813         WAITO
12BC DB61               814+??0041:  IN     INT1    ;Get Intl status
12BE E602               815+        ANI     BOM     ;Check for byte out
12C0 CABC12             816+        JZ      ??0041  ;If not, try again
12C3 3E05               817         MVI     A,PPC   ;Parallel poll configure
12C5 D360               818         OUT     DOUT
                        819         WAITO
12C7 DB61               820+??0042:  IN     INT1    ;Get Intl status
12C9 E602               821+        ANI     BOM     ;Check for byte out
12CB CAC712             822+        JZ      ??0042  ;If not, try again
12CE 1A                 823         LDAX    D       ;Get matching configuration byte
12CF F660               824         ORI     PPE     ;Merge with parallel poll enable
12D1 D360               825         OUT     DOUT
12D3 23                 826         INX     H       ;Incr pointers
12D4 13                 827         INX     D
12D5 C3A712             828         JMP     PPEN1   ;Loop until invalid listener char
                        829 PPEN2:  WAITO
12D8 DB61               830+??0043:  IN     INT1    ;Get Intl status
12DA E602               831+        ANI     BOM     ;Check for byte out
12DC CAD812             832+        JZ      ??0043  ;If not, try again
12DF C9                 833         RET
                        834 ;
                        835 ;PARALLEL POLL DISABLE ROUTINE
                        836 ;
                        837 ;INPUTS:         HL listener list pointer
                        838 ;OUTPUTS:        None
                        839 ;CALLS:          None
                        840 ;DESTROYS:       A, HL, F
                        841 ;
12E0 3E3F               842 PPDS:   MVI     A,UNL   ;Universal unlisten
12E2 D360               843         OUT     DOUT
                        844 PPDS1:  RANGE   20H,3EH,PPDS2   ;Check for valid listener
                        845+                        ;Checks for value in range
                        846+                        ;branches to label if not
                        847+                        ;in range. Falls through if
                        848+                        ;lower <= ( (H)(L) ) <= upper.
                        849+                        ;Get next byte.
12E4 7E                 850+        MOV     A,M
12E5 FE20               851+        CPI     20H
12E7 FAFD12             852+        JM      PPDS2
12EA FE3F               853+        CPI     3EH+1
12EC F2FD12             854+        JP      PPDS2
                        855         WAITO
12EF DB61               856+??0044:  IN     INT1    ;Get Intl status
12F1 E602               857+        ANI     BOM     ;Check for byte out
12F3 CAEF12             858+        JZ      ??0044  ;If not, try again
```

```
12F6 7E          859          MOV     A,M      ;Get listener
12F7 D360        860          OUT     DOUT
12F9 23          861          INX     H        ;Incr pointer
12FA C3E412      862          JMP     PPDS1    ;Loop until invalid listener
                 863 PPDS2:   WAITO
12FD DB61        864+??0045:   IN     INT1     ;Get Intl status
12FF E602        865+         ANI     BOM      ;Check for byte out
1301 CAFD12      866+         JZ      ??0045   ;If not, try again
1304 3E05        867          MVI     A,PPC    ;Parallel poll configure
1306 D360        868          OUT     DOUT
                 869          WAITO
1308 DB61        870+??0046:   IN     INT1     ;Get Intl status
130A E602        871+         ANI     BOM      ;Check for byte out
130C CA0813      872+         JZ      ??0046   ;If not, try again
130F 3E70        873          MVI     A,PPD    ;Parallel poll disable
1311 D360        874          OUT     DOUT
                 875          WAITO
1313 DB61        876+??0047:   IN     INT1     ;Get Intl status
1315 E602        877+         ANI     BOM      ;Check for byte out
1317 CA1313      878+         JZ      ??0047   ;If not, try again
131A C9          879          RET
                 880 ;
                 881 ;        PARALLEL POLL UNCONFIGURE ALL ROUTINE
                 882 ;
                 883 ;
                 884 ;INPUTS:          None
                 885 ;OUTPUTS:         None
                 886 ;CALLS:           None
                 887 ;DESTROYS:        A, F
                 888 ;
131B 3E15        889 PPUN:    MVI     A,PPU    ;Parallel poll unconfigure
131D D360        890          OUT     DOUT
                 891          WAITO
131F DB61        892+??0048:   IN     INT1     ;Get Intl status
1321 E602        893+         ANI     BOM      ;Check for byte out
1323 CA1F13      894+         JZ      ??0048   ;If not, try again
1326 C9          895          RET
                 896 ;
                 897 ;************************************************
                 898 ;
                 899 ;CONDUCT A PARALLEL POLL
                 900 ;
                 901 ;
                 902 ;INPUTS:          None
                 903 ;OUTPUTS:         None
                 904 ;CALLS:           None
                 905 ;DESTROYS:        A, B, F
                 906 ;RETURNS:         A= parallel poll status byte
                 907 ;
1327 3E40        908 PPOL:    MVI     A,LON    ;Listen only
1329 D364        909          OUT     ADRMD
                 910          CLRA             ;Immediate XEQ PON
132B AF          911+         XRA     A        ;A XOR A =0
132C D365        912          OUT     AUXMD    ;Reset TON
132E 3EF5        913          MVI     A,EXPP   ;Execute parallel poll
1330 D369        914          OUT     CMD92
                 915          WAITI            ;Wait for completion= BI on 91
1332 DB61        916+??0049:   IN     INT1     ;Get INTl status
1334 47          917+         MOV     B,A      ;Save status in B
1335 E601        918+         ANI     BIM      ;Check for byte in
1337 CA3213      919+         JZ      ??0049   ;If not, just try again
133A 3E80        920          MVI     A,TON    ;Talk only
133C D364        921          OUT     ADRMD
                 922          CLRA             ;Immediate XEQ PON
133E AF          923+         XRA     A        ;A XOR A =0
133F D365        924          OUT     AUXMD    ;Reset LON
1341 DB60        925          IN      DIN      ;Get PP byte
1343 C9          926          RET
                 927 ;
                 928 ;************************************************
                 929 ;PASS CONTROL ROUTINE
                 930 ;
                 931 ;INPUTS:          HL pointer to talker
                 932 ;OUTPUTS:         None
```

```
                933 ;CALLS:          None
                934 ;DESTROYS:       A, HL, F
                935 PCTL:    RANGE   40H,5EH,PCTL1   ;Is it a valid talker ?
                936+                                 ;Checks for value in range
                937+                                 ;branches to label if not
                938+                                 ;in range. Falls through if
                939+                                 ;lower <= ( (H)(L) ) <= upper.
                940+                                 ;Get next byte.
1344 7E         941+         MOV     A,M
1345 FE40       942+         CPI     40H
1347 FA8A13     943+         JM      PCTL1
134A FE5F       944+         CPI     5EH+1
134C F28A13     945+         JP      PCTL1
134F FE41       946          CPI     MTA        ;Is it my talker address
1351 CA8A13     947          JZ      PCTL1      ;Yes, just return
1354 D360       948          OUT     DOUT       ;Send on GPIB
                949          WAITO
1356 DB61       950+??0050:   IN      INT1       ;Get Intl status
1358 E602       951+         ANI     BOM        ;Check for byte out
135A CA5613     952+         JZ      ??0050     ;If not, try again
135D 3E09       953          MVI     A,TCT      ;Take control message
135F D360       954          OUT     DOUT
                955          WAITO
1361 DB61       956+??0051:   IN      INT1       ;Get Intl status
1363 E602       957+         ANI     BOM        ;Check for byte out
1365 CA6113     958+         JZ      ??0051     ;If not, try again
1368 3E01       959          MVI     A,MODE1    ;Not talk only or listen only
136A D364       960          OUT     ADRMD      ;Enable 91 address mode 1
                961          CLRA
136C AF         962+         XRA     A          ;A XOR A =0
136D D365       963          OUT     AUXMD      ;Immediate XEQ PON
136F 3E01       964          MVI     A,MDA      ;My device address
1371 D366       965          OUT     ADR01      ;enabled to talk and listen
1373 3EA1       966          MVI     A,AXRB+CPTEN    ;Command pass thru enable
1375 D365       967          OUT     AUXMD
                968 ;*******optional PP configuration goes here********
1377 3EF1       969          MVI     A,GIDL     ;92 go idle command
1379 D369       970          OUT     CMD92
                971          WAITX
137B DB6F       972+??0052:  IN      PRTF
137D E602       973+         ANI     TCIF
137F C27B13     974+         JNZ     ??0052
                975          WAITT               ;Wait for TCI
1382 DB6F       976+??0053:  IN      PRTF       ;Get task complete int,etc.
1384 E602       977+         ANI     TCIF       ;Mask it
1385 CA8213     978+         JZ      ??0053     ;Wait for task to be complete
1389 23         979          INX     H
138A C9         980 PCTL1:   RET
                981 ;
                982 ;
                983 ;****************************************
                984 ;
                985 ;RECEIVE CONTROL ROUTINE
                986 ;
                987 ;INPUTS:          None
                988 ;OUTPUTS:         None
                989 ;CALLS:           None
                990 ;DESTROYS:        A, F
                991 ;RETURNS:         0= invalid (not take control to us or CPT bit not on)
                992 ;                 < > 0 = valid take control-- 92 will now be in control
                993 ;NOTE:            THIS CODE MUST BE TIGHTLY INTEGRATED INTO ANY USER
                994 ;                 SOFTWARE THAT FUNCTIONS WITH THE 8291 AS  A DEVICE.
                995 ;                 NORMALLY SOME ADVANCE WARNING OF IMPENDING PASS
                996 ;                 CONTROL SHOULD BE GIVEN TO US BY THE CONTROLLER
                997 ;                 WITH OTHER USEFUL INFO. THIS PROTOCOL IS SITUATION
                998 ;                 SPECIFIC AND WILL NOT BE COVERED HERE.
                999 ;
               1000 ;
138B DB61      1001 RCTL:    IN      INT1       ;Get INT1 req (i.e. CPT etc.)
138D E680      1002          ANI     CPT        ;Is command pass thru on ?
138F CACF13    1003          JZ      RCTL2      ;No, invalid-- go return
1392 DB65      1004          IN      CPTRG      ;Get command
1394 FE09      1005          CPI     TCT        ;Is it take control ?
```

```
1396 C2CA13      1006          JNZ     RCTL1   ;No, go return invalid
1399 DB64        1007          IN      ADRST   ;Get address status
139B E602        1008          ANI     TA      ;Is TA on ?
139D CACA13      1009          JZ      RCTL1   ;No -- go return invalid
13A0 3E60        1010          MVI     A,DTDL1 ;Disable talker listener
13A2 D366        1011          OUT     ADR01
13A4 3E80        1012          MVI     A,TON   ;Talk only
13A6 D364        1013          OUT     ADRMD
                 1014          CLRA
13A8 AF          1015+         XRA     A       ;A XOR A =0
13A9 D361        1016          OUT     INT1    ;Mask off INT bits
13AB D362        1017          OUT     INT2
13AD D365        1018          OUT     AUXMD
13AF 3EFA        1019          MVI     A,TCNTR ;Take (receive) control 92 command
13B1 D369        1020          OUT     CMD92
13B3 3E0F        1021          MVI     A,VSCMD ;Valid command pattern for 91
13B5 D365        1022          OUT     AUXMD
                 1023 ;******** optional TOUT1 check could be put here ********
                 1024          WAITX
13B7 DB6F        1025+??0054:  IN      PRTF
13B9 E602        1026+         ANI     TCIF
13BB C2B713      1027+         JNZ     ??0054
                 1028          WAITT           ;Wait for TCI
13BE DB6F        1029+??0055:  IN      PRTF    ;Get task complete int,etc.
13C0 E602        1030+         ANI     TCIF    ;Mask it
13C2 CABE13      1031+         JZ      ??0055  ;Wait for task to be complete
13C5 3E09        1032          MVI     A,TCT   ;Valid return pattern
13C7 C3CF13      1033          JMP     RCTL2   ;Only one return per routine
13CA 3E0F        1034 RCTL1:   MVI     A,VSCMD ;Acknowledge CPT
13CC D365        1035          OUT     AUXMD
                 1036          CLRA            ;Error return pattern
13CE AF          1037+         XRA     A       ;A XOR A =0
13CF C9          1038 RCTL2:   RET
                 1039 ;
                 1040 ;************************************************
                 1041 ;
                 1042 ;        SRQ ROUTINE
                 1043 ;
                 1044 ;INPUTS:         None
                 1045 ;OUTPUTS:        None
                 1046 ;CALLS:          None
                 1047 ;RETURNS:        A= 0 no SRQ
                 1048 ;                A < > 0 SRQ occured
                 1049 ;
                 1050 ;
13D0 DB69        1051 SRQD:    IN      INTST   ;Get 92's INTRQ status
13D2 E620        1052          ANI     SRQBT   ;Mask off SRQ
13D4 CAE213      1053          JZ      SRQD2   ;Not set--- go return
13D7 F608        1054          ORI     IACK    ;Set--- must clear it with IACK
13D9 D369        1055          OUT     CMD92
13DB DB69        1056 SRQD1:   IN      INTST   ;Get IBF
13DD E602        1057          ANI     IBFBT   ;Mask it
13DF CADB13      1058          JZ      SRQD1   ;Wait if not set
13E2 C9          1059 SRQD2:   RET
                 1060 ;
                 1061 ;*******************************************
                 1062 ;
                 1063 ;REMOTE ENABLE ROUTINE
                 1064 ;
                 1065 ;INPUTS:         None
                 1066 ;OUTPUTS:        None
                 1067 ;CALLS:          NONE
                 1068 ;DESTROYS:       A, F
                 1069 ;
13E3 3EF8        1070 REME:    MVI     A,SREM
13E5 D369        1071          OUT     CMD92   ;92 asserts remote enable
                 1072          WAITX           ;Wait for TCI = 0
13E7 DB6F        1073+??0056:  IN      PRTF
13E9 E602        1074+         ANI     TCIF
13EB C2E713      1075+         JNZ     ??0056
                 1076          WAITT           ;Wait for TCI
13EE DB6F        1077+??0057:  IN      PRTF    ;Get task complete int,etc.
13F0 E602        1078+         ANI     TCIF    ;Mask it
13F2 CAEE13      1079+         JZ      ??0057  ;Wait for task to be complete
```

```
13F5 C9          1080          RET
                 1081 ;
                 1082 ;****************************************
                 1083 ;
                 1084 ;LOCAL ROUTINE
                 1085 ;
                 1086 ;
                 1087 ;INPUTS:          None
                 1088 ;OUTPUTS:         None
                 1089 ;CALLS:           None
                 1090 ;DESTROYS:        A, F
                 1091 ;
13F6 3EF7        1092 LOCL:    MVI      A,SLOC
13F8 D369        1093          OUT      CMD92    ;92 stops asserting remote enable
                 1094          WAITX              ;Wait for TCI =0
13FA DB6F        1095+??0058: IN        PRTF
13FC E602        1096+         ANI      TCIF
13FE C2FA13      1097+         JNZ      ??0058
                 1098          WAITT              ;Wait for TCI
1401 DB6F        1099+??0059: IN        PRTF     ;Get task complete int,etc.
1403 E602        1100+         ANI      TCIF     ;Mask it
1405 CA0114      1101+         JZ       ??0059   ;Wait for task to be complete
1408 C9          1102          RET
                 1103 ;
                 1104 ;****************************************
                 1105 ;
                 1106 ;INTERFACE CLEAR / ABORT ROUTINE
                 1107 ;
                 1108 ;
                 1109 ;INPUTS:                 None
                 1110 ;OUTPUTS:                None
                 1111 ;CALLS:                 None
                 1112 ;DESTROYS:              A, F
                 1113 ;
                 1114 ;
1409 3EF9        1115 IFCL:    MVI      A,ABORT
140B D369        1116          OUT      CMD92    ;Send IFC
                 1117          WAITX              ;Wait for TCI =0
140D DB6F        1118+??0060: IN        PRTF
140F E602        1119+         ANI      TCIF
1411 C20D14      1120+         JNZ      ??0060
                 1121          WAITT              ;Wait for TCI
1414 DB6F        1122+??0061: IN        PRTF     ;Get task complete int,etc.
1416 E602        1123+         ANI      TCIF     ;Mask it
1418 CA1414      1124+         JZ       ??0061   ;Wait for task to be complete
                 1125 ;Delete both WAITX & WAITT if this routine
                 1126 ;is to be called while the 9292 is
                 1127 ;Controller-in-Charge. If not C.I.C. then
                 1128 ;TCI is set, else nothing is set (IFC is sent)
                 1129 ;and the WAIT'S will hang forever
141B C9          1130          RET
                 1132 ;
```

```
                    1133 ;APPLICATION EXAMPLE CODE FOR 8085
                    1134 ;
0032                1135 FGDNL   EQU     '2'     ;Func gen device num "2" ASCII,lstn
0031                1136 FCDNL   EQU     '1'     ;Freq ctr device num "1" ASCII,lstn
0051                1137 FCDNT   EQU     'Q'     ;Freq ctr talk address
000D                1138 CR      EQU     0DH     ;ASCII carriage return
000A                1139 LF      EQU     0AH     ;ASCII line feed
00FF                1140 LEND    EQU     0FFH    ;List end for Talk/Listen lists
0040                1141 SRQM    EQU     40H     ;Bit indicating device sent SRQ
                    1142 ;
141C 46553146       1143 FGDATA: DB      'FU1FR37KHAM2VO',CR     ;Data to set up func. gen
1420 5233374B
1424 48414D32
1428 564F
142A 0D
000F                1144 LIM1    EQU     15                      ;Buffer length
142B 50463447       1145 FCDATA: DB      'PF4G7T'                ;Data to set up freq ctr
142F 3754
0006                1146 LIM2    EQU     6                       ;Buffer length
1431 31             1147 LL1:    DB      FCDNL,LEND              ;Listen list for freq ctr
1432 FF
1433 32             1148 LL2:    DB      FGDNL,LEND              ;Listen list for func. gen
1434 FF
1435 51             1149 TL1:    DB      FCDNT,LEND              ;Talk list for freq ctr
1436 FF
                    1150 ;
                    1151 ;SETUP FUNCTION GENERATOR
1437 060D           1152         MVI     B,CR    ;EOS
1439 0E0F           1153         MVI     C,LIM1  ;Count
143B 111C14         1154         LXI     D,FGDATA        ;Data pointer
143E 213314         1155         LXI     H,LL2   ;Listen list pointer
1441 CD1C10         1156         CALL    SEND
                    1157 ;
                    1158 ;SETUP FREQ COUNTER
                    1159 ;
1444 0654           1160         MVI     B,'T'   ;EOS
1446 0E06           1161         MVI     C,LIM2  ;Count
1448 112814         1162         LXI     D,FCDATA        ;Data pointer
144B 213114         1163         LXI     H,LL1   ;Listen list pointer
144E CD1C10         1164         CALL    SEND
                    1165 ;
                    1166 ;WAIT FOR SRQ FROM FREQ CTR
                    1167 ;
1451 CDD013         1168 LOOP:   CALL    SRQD    ;Has SRQ occurred ?
1454 CA5114         1169         JZ      LOOP    ;No, wait for it
                    1170 ;
                    1171 ;SERIAL POLL TO CLEAR SRQ
                    1172 ;
1457 11003C         1173         LXI     D,SPBYTE        ;Buffer pointer
145A 213514         1174         LXI     H,TL1           ;Talk list pointer
145D CD1C12         1175         CALL    SPOL
1460 1B             1176         DCX     D       ;Backup buffer pointer to ctr byte
1461 1A             1177         LDAX    D       ;Get status byte
1462 E640           1178         ANI     SRQM    ;Did ctr assert SRQ ?
1464 CA7714         1179         JZ      ERROR   ;Ctr should have said yes
                    1180 ;
                    1181 ;RECEIVE READING FROM COUNTER
                    1182 ;
1467 060A           1183         MVI     B,LF    ;EOS
1469 0E11           1184         MVI     C,LIM3  ;Count
146B 213514         1185         LXI     H,TL1   ;Talk list pointer
146E 11013C         1186         LXI     D,FCDATI        ;Data in buffer pointer
1471 CD9F10         1187         CALL    RECV
1474 C27714         1188         JNZ     ERROR
                    1189 ;
                    1190 ;******* rest of user processing goes here *****
                    1191 ;
                    1192 ;
1477 00             1193 ERROR:  NOP             ;User dependant error handling
                    1194 ;       ETC.
3C00                1195 ORG     3C00H
3C00                1196 SPBYTE: DS      1       ;Location for serial poll byte
0011                1197 LIM3    EQU     17      ;Max freq counter input
```

```
3C01                1198 FCDATI: DS      LIM3    ;Freq ctr input buffer
                    1199         END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS


USER SYMBOLS
ABORT  A 00F9   ADR01  A 0056   ADRMD  A 0054   ADRST  A 0054   AUXMD  A 0055   AXRA   A 0080   AXRB   A 00A0
BIM    A 0001   BOF    A 0001   BOM    A 0002   BUSST  A 0058   CAHCY  A 0003   CLKRT  A 0023   CLRA   + 0007
CLRST  A 0068   CMD92  A 0069   CPT    A 0080   CPTEN  A 0001   CPTRG  A 0065   CR     A 000D   DCL    A 0014
DCLR   A 11EC   DCLR1  A 11F0   DCLR2  A 1209   DIN    A 0060   DOUT   A 0050   DTDL1  A 0050   DTDL2  A 00E0
EDEOS  A 0004   ENDMK  A 0010   EOIS   A 0008   EOIST  A 0020   EOSR   A 0057   ERFLG  A 0068   ERRM   A 0068
ERROR  A 1477   EVBIT  A 0010   EVCST  A 0068   EVREG  A 0068   EXPP   A 00F5   FCDATA A 1428   FCDATI A 3C01
FCDNL  A 0031   FCDNT  A 0051   FGDATA A 141C   FGDNL  A 0032   FNHSK  A 0003   GET    A 0008   GIDL   A 00F1
GSEC   A 00F4   GTS8   A 00F6   HOEND  A 0002   HOHSK  A 0001   IACK   A 000B   IBFBT  A 0002   IBFF   A 0010
IFCL   A 1409   INIT   A 1000   INT1   A 0061   INT2   A 0062   INTM   A 00A0   INTM1  A 0061   INTMR  A 0068
INTST  A 0069   LA     A 0001   LEND   A 00FF   LF     A 000A   LIM1   A 000F   LIM2   A 0006   LIM3   A 0011
LL1    A 1431   LL2    A 1433   LOCL   A 13F6   LON    A 0040   LOOP   A 1451   MDA    A 0001   MLA    A 0021
MODE1  A 0001   MTA    A 0041   NVCMD  A 0007   OBFF   A 0008   PCTL   A 1344   PCTL1  A 138A   PPC    A 0005
PPD    A 0070   PPDS   A 12E0   PPDS1  A 12E4   PPDS2  A 12FD   PPE    A 0060   PPEN   A 12A3   PPEN1  A 12A7
PPEN2  A 12D8   PPOL   A 1327   PPU    A 0015   PPUN   A 131B   PRT91  A 0050   PRT92  A 0068   PRTF   A 006F
RANGE  + 0005   RBST   A 00E7   RCST   A 00E5   RCTL   A 1388   RCTL1  A 13CA   RCTL2  A 13CF   RECV   A 109F
RECV1  A 10EA   RECV2  A 1105   RECV3  A 1106   RECV4  A 1110   RECV5  A 1117   RECV6  A 1139   REME   A 13E3
RERF   A 00E4   RERM   A 00EA   REVC   A 00E3   RINM   A 00E5   RSET   A 00F2   RSTI   A 00F3   RTOUT  A 00E9
SDEOI  A 0006   SEND   A 101C   SEND1  A 102E   SEND2  A 1047   SEND3  A 1069   SEND4  A 1070   SEND5  A 107F
SEND6  A 1088   SETF   + 0000   SLOC   A 00F7   SPBYTE A 3C00   SPCNI  A 00F0   SPD    A 0019   SPE    A 0018
SPIF   A 0004   SPOL   A 121C   SPOL1  A 123D   SPOL2  A 1294   SREM   A 00F8   SRQBT  A 0020   SRQD   A 13D0
SRQD1  A 13DB   SRQD2  A 13E2   SRQM   A 0040   STCNI  A 00FE   TA     A 0002   TCASY  A 00FC   TCIF   A 0002
TCNTR  A 00FA   TCSY   A 00FD   TCT    A 0009   TL1    A 1435   TLON   A 00C0   TON    A 0090   TOREG  A 0068
TOST   A 0068   TOUT1  A 0001   TOUT2  A 0002   TOUT3  A 0004   TRIG   A 11BC   TRIG1  A 11C0   TRIG2  A 11D9
UNL    A 003F   VSCMD  A 000F   WAITI  + 0002   WAITO  + 0001   WAITT  + 0004   WAITX  + 0003   WEVC   A 00E2
WOUT   A 00E1   XFER   A 113A   XFER1  A 1153   XFER2  A 116C   XFER3  A 1193   XFER4  A 11B8

ASSEMBLY COMPLETE,    NO ERRORS
```

APPENDIX B

TEST CASES FOR THE SOFTWARE DRIVERS

The following test cases were used to exercise the software routines and to check their action. To provide another device/controller on the GPIB a ZT488 GPIB Analyzer was used. This analyzer acted as a talker, listener or another controller as needed to execute the tests. The sequence of outputs are shown with each test. All numbers are hexadecimal.

SEND TEST CASES

B =	44	44	44
C =	30	2	0
DE =	3E80	3E80	3E80
HL =	3E70	3E70	3E70
3E70:	20 30 3E 3F		
3E80:	11 44		
GPIB output:	41 ATN	41 ATN	41 ATN
	3F ATN	3F ATN	3F ATN
	20 ATN	20 ATN	20 ATN
	30 ATN	30 ATN	30 ATN
	3E ATN	3E ATN	3E ATN
	11	11	
	44 EOI	44 EOI	
Ending B =	44	44	44
Ending C =	2E	0	0
Ending DE =	3E82	3E82	3E80
Ending HL =	3E73	3E73	3E73

RECEIVE TEST CASES

B =	44	44	44	44	44	44	44
C =	30	30	30	30	4	4	0=256
DE =	3E80	3E80	3E80	3E80	3E80	3E80	3E80
HL =	3E70	3E70	3E70	3E70	3E70	3E70	3E70
3E70:	40	50	5E	5F	40	40	40
GPIB output:	40 ATN	50 ATN	5E ATN		40 ATN	40 ATN	40 ATN
	3F ATN	3F ATN	3F ATN		3F ATN	3F ATN	3F ATN
	21 ATN	21 ATN	21 ATN		21 ATN	21 ATN	21 ATN
ZT488 Data	1	1	1		1	11	1
In	2	2	2		2	22	2
	3	3	3		3	33	3
	4	4	44,EOI		4	44	44
	44	5,EOI					
Ending A =	0	0	0	5F	40	0	0
Ending B =	0	0	0	44	40	0	0
Ending C =	2B	2B	2C	30	0	0	FC
Ending DE =	3E85	3E85	3E84	3E80	3E84	3E84	3E84
Ending HL =	3E71	3E71	3E71	3E70	3E71	3E71	3E71

SERIAL POLL TEST CASES

C =	30	C =	30
DE =	3E80	DE =	3E80
HL =	3E70	HL =	3E70
3E70:	40	3E70:	5F
	50	GPIB output:	3F ATN
	5E		21 ATN
	5F		18 ATN

```
GPIB output:  3F ATN                          19 ATN
      output:  21 ATN          Ending C  = 30
      output:  18 ATN          Ending DE = 3E80
      output:  40 ATN          Ending HL = 3E70
       input*: 00
      output:  50 ATN
       input*: 41
      output:  5E ATN
       input*: 7F
      output:  19 ATN
*NOTE: leave ZT488 in single step mode even on input
  Ending C  = 30
  Ending DE = 3E83
  Ending HL = 3E73
Ending 3E80:  00  41  7F
```

PASS CONTROL TEST CASES

```
          HL = 3E70        3E70        3E70
       3E70:  40           41(MTA)     5F
GPIB output:  40 ATN
              09 ATN
              — ATN
  Ending HL = 3E71        3E70        3E70
  Ending A  = 02          41(MTA)     5F
```

RECEIVE CONTROL TEST CASES

```
GPIB input                10 ATN      40 ATN      41 ATN
                          ATN         09 ATN      09 ATN
Run Receive Control
GPIB Input                            ATN         ATN
Ending A =                0           0           09
```

PARALLEL POLL ENABLE TEST CASES

```
          DE = 3E80                   3E80
          HL = 3E70                   3E70
       3E70:  20  30  3E  3F          3F
       3E80:  01  02  03

GPIB output:  3F ATN                  3F ATN
              20 ATN
              05 ATN
              61 ATN
              30 ATN
              05 ATN
              62 ATN
              3E ATN
              05 ATN
              63 ATN

  Ending DE = 3E83                    3E80
  Ending HL = 3E73                    3E70
```

PARALLEL POLL DISABLE TEST CASES

```
          HL = 3E70                   3E70
       3E70:  20  30  3E  3F          3F
```

GPIB output:	3F ATN	3F ATN
	20 ATN	05 ATN
	30 ATN	70 ATN
	3E ATN	
	05 ATN	
	70 ATN	
Ending HL =	3E73	3E70

PARALLEL POLL UNCONFIGURE TEST CASE

GPIB output: 15 ATN

PARALLEL POLL TEST CASES

Set DIO #	1	2	3	4	5	6	7	8	None
Ending A	1	2	4	8	10	20	40	80	0

SRQ TEST

	Set SRQ momentarily	Reset SRQ
Ending A	= 02	00

TRIGGER TEST

HL = 3E70
DE = 3E80
BC = 4430
3E70: 20 30 3E 3F
GPIB output: 3F ATN
20 ATN
30 ATN
3E ATN
08 ATN
Ending HL = 3E73
DE = 3E80
BC = 4430

DEVICE CLEAR TEST

HL = 3E70
DE = 3E80
BC = 4430
3E70: 20 30 3E 3F
GPIB output: 3F ATN
20 ATN
30 ATN
3E ATN
14 ATN
Ending HL = 3E73
DE = 3E80
RC = 4430

XFER TEST

```
          B = 44
         HL = 3E70
      3E70:  40  20  30  3E  3F
GPIB output: 40 ATN
             3F ATN
             20 ATN
             30 ATN
             3E ATN
 GPIB input: 0
             1
             2
             3
             44
  Ending A = 0
          B = 44
         HL = 3E74
```

APPLICATION EXAMPLE
GPIB OUTPUT/INPUT

```
GPIB output: 41 ATN
             3F ATN
             32 ATN
             46
             55
             31
             46
             52
             33
             37
             4B
             48
             41
             4D
             32
             56
             4F
             0D EOI
             41 ATN
             3F ATN
             31 ATN
             50
             46
             34
             47
             37
             54 EOI
 GPIB input: SRQ
GPIB output: 3F ATN
             21 ATN
             18 ATN
             51 ATN
 GPIB input: 40 SRQ
GPIB output: 19 ATN
             51 ATN
```

	3F ATN
	21 ATN
GPIB input:	20
	2B
	20
	20
	20
	33
	37
	30
	30
	30
	2E
	30
	45
	2B
	30
	0D
	0A
GPIB output:	XX ATN

APPENDIX C

REMOTE MESSAGE CODING

Mnemonic	Message Name		Type	Class	Bus Signal Line(s) and Coding That Asserts the True Value of the Message: DIO 8	7	6	5	4	3	2	1	DAV	NRFD	NDAC	ATN	EOI	SRQ	IFC	REN
ACG	addressed command group		M	AC	Y	Ø	Ø	Ø	X	X	X	X	XXX	1	X	X	X	X	X	X
ATN	attention		U	UC	X	X	X	X	X	X	X	X	XXX	1	X	X	X	X	X	X
DAB	data byte	(Notes 1, 9)	M	DD	D8	D7	D6	D5	D4	D3	D2	D1	XXX	Ø	X	X	X	X	X	X
DAC	data accepted		U	HS	X	X	X	X	X	X	X	X	XXØ	X	X	X	X	X	X	X
DAV	data valid		U	HS	X	X	X	X	X	X	X	X	1XX	X	X	X	X	X	X	X
DCL	device clear		M	UC	Y	Ø	Ø	1	Ø	1	Ø	Ø	XXX	1	X	X	X	X	X	X
END	end		U	ST	X	X	X	X	X	X	X	X	XXX	Ø	1	X	X	X	X	X
EOS	end of string	(Notes 2, 9)	M	DD	E8	E7	E6	E5	E4	E3	E2	E1	XXX	Ø	X	X	X	X	X	X
GET	group execute trigger		M	AC	Y	Ø	Ø	Ø	1	Ø	Ø	Ø	XXX	1	X	X	X	X	X	X
GTL	go to local		M	AC	Y	Ø	Ø	Ø	Ø	Ø	Ø	1	XXX	1	X	X	X	X	X	X
IDY	identify		U	UC	X	X	X	X	X	X	X	X	XXX	X	1	X	X	X	X	X
IFC	interface clear		U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	1	X	X	X
LAG	listen address group		M	AD	Y	Ø	1	X	X	X	X	X	XXX	1	X	X	X	X	X	X
LLO	local lock out		M	UC	Y	Ø	Ø	1	Ø	Ø	Ø	1	XXX	1	X	X	X	X	X	X
MLA	my listen address	(Note 3)	M	AD	Y	Ø	1	L5	L4	L3	L2	L1	XXX	1	X	X	X	X	X	X
MTA	my talk address	(Note 4)	M	AD	Y	1	Ø	T5	T4	T3	T2	T1	XXX	1	X	X	X	X	X	X
MSA	my secondary address	(Note 5)	M	SE	Y	1	1	S5	S4	S3	S2	S1	XXX	1	X	X	X	X	X	X

Mnemonic	Message Name	Type	Class	DIO 8	7	6	5	4	3	2	1	DIO	DRD	AFA	ATN	EOI	SRQ	IFC	REN
				Bus Signal Line(s) and Coding That Asserts the True Value of the Message															
NUL	null byte	M	DD	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	XXX	X	X	X	X	X		
OSA	other secondary address	M	SE	(OSA = SCG ∧ $\overline{\text{MSA}}$)															
OTA	other talk address	M	AD	(OTA = TAG ∧ $\overline{\text{MTA}}$)															
PCG	primary command group	M	——	(PCG = ACG ∨ UCG ∨ LAG ∨ TAG)															
PPC	parallel poll configure	M	AC	Y	Ø	Ø	Ø	Ø	1	Ø	1	XXX	1	X	X	X	X		
PPE	parallel poll enable	M	SE	Y	1	1	Ø	S	P3	P2	P1	XXX	1	X	X	X	X		
PPD	parallel poll disable	M	SE	Y	1	1	1	D4	D3	D2	D1	XXX	1	X	X	X	X		
PPR1	parallel poll response 1	U	ST	X	X	X	X	X	X	X	1	XXX	1	1	X	X	X		
PPR2	parallel poll response 2	U	ST	X	X	X	X	X	X	1	X	XXX	1	1	X	X	X		
PPR3	parallel poll response 3	U	ST	X	X	X	X	X	1	X	X	XXX	1	1	X	X	X		
PPR4	parallel poll response 4	U	ST	X	X	X	X	1	X	X	X	XXX	1	1	X	X	X		
PPR5	parallel poll response 5	U	ST	X	X	X	1	X	X	X	X	XXX	1	1	X	X	X		
PPR6	parallel poll response 6	U	ST	X	X	1	X	X	X	X	X	XXX	1	1	X	X	X		
PPR7	parallel poll response 7	U	ST	X	1	X	X	X	X	X	X	XXX	1	1	X	X	X		
PPR8	parallel poll response 8	U	ST	1	X	X	X	X	X	X	X	XXX	1	1	X	X	X		
PPU	parallel poll unconfigure	M	UC	Y	Ø	Ø	1	Ø	1	Ø	1	XXX	1	X	X	X	X		
REN	remote enable	U	UC	X	X	X	X	X	X	X	X	XXX	X	X	X	X	1		
RFD	ready for data	U	HS	X	X	X	X	X	X	X	X	XØX	X	X	X	X	X		
RQS	request service	U	ST	X	1	X	X	X	X	X	X	XXX	Ø	X	X	X	X		
SCG	secondary command group	M	SE	Y	1	1	X	X	X	X	X	XXX	1	X	X	X	X		
SDC	selected device clear	M	AC	Y	Ø	Ø	Ø	Ø	1	Ø	Ø	XXX	1	X	X	X	X		
SPD	serial poll disable	M	UC	Y	Ø	Ø	1	1	Ø	Ø	1	XXX	1	X	X	X	X		
SPE	serial poll enable	M	UC	Y	Ø	Ø	1	1	Ø	Ø	Ø	XXX	1	X	X	X	X		
SRQ	service request	U	ST	X	X	X	X	X	X	X	X	XXX	X	X	1	X	X		
STB	status byte	M	ST	S8	X	S6	S5	S4	S3	S2	S1	XXX	Ø	X	X	X	X		
TCT	take control	M	AC	Y	Ø	Ø	Ø	1	Ø	Ø	1	XXX	1	X	X	X	X		
TAG	talk address group	M	AD	Y	1	Ø	X	X	X	X	X	XXX	1	X	X	X	X		
UCG	universal command group	M	UC	Y	Ø	Ø	1	X	X	X	X	XXX	1	X	X	X	X		
UNL	unlisten	M	AD	Y	Ø	1	1	1	1	1	1	XXX	1	X	X	X	X		
UNT	untalk	M	AD	Y	1	Ø	1	1	1	1	1	XXX	1	X	X	X	X		

The 1/Ø coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

(1) D1-D8 specify the device dependent data bits.

(2) E1-E8 specify the device dependent code used to indicate the EOS message.

(3) L1-L5 specify the device dependent bits of the device's listen address.

(4) T1-T5 specify the device dependent bits of the device's talk address.

(5) S1-S5 specify the device dependent bits of the device's secondary address.

(6) S specifies the sense of the PPR.

S	Response
Ø	Ø
1	1

P1-P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
Ø	Ø	Ø	PPR1
1	1	1	PPR8

(7) D1-D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.

(8) S1-S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)

(9) The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function

(10) The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.

(11) This code is provided for system use, see 6.3.

intel

APPLICATION NOTE

AP-166

October 1983

USING THE 8291A GPIB TALKER/LISTENER

ORDER NUMBER: 230832-001

INTRODUCTION

This application note explains the Intel® 8291A GPIB (General Purpose Interface Bus) Talker/Listener as a component, and shows its use in GPIB interface design tasks.

The first section of this note presents an overview of IEEE 488 (GPIB). The second section introduces the Intel® GPIB component family. A detailed explanation of the 8291A follows. Finally, some application examples using the component family are presented.

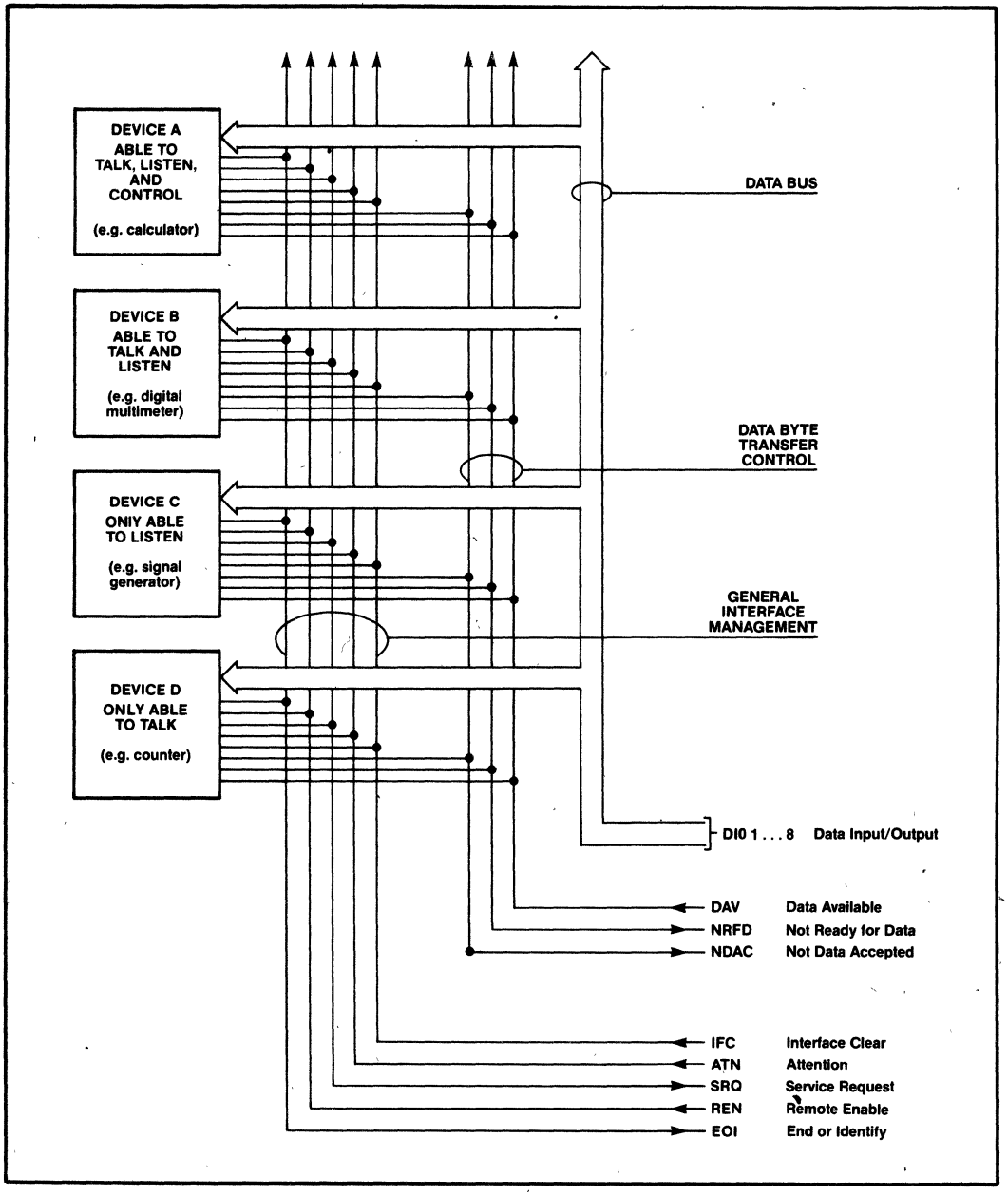


Figure 1. Interface Capabilities and Bus Structure

OVERVIEW OF IEEE 488/GPIB

The GPIB is a parallel interface bus with an asynchronous interlocking data exchange handshake mechanism. It is designed to provide a common communication interface among devices over a maximium distance of 20 meters at a maximum speed of 1 Mbps. Up to 15 devices may be connected together. The asynchronous interlocking handshake dispenses with a common synchronization clock, and allows intercommunication among devices capable of running at different speeds. During any transaction, the data transfer occurs at the speed of the slowest device involved.

The GPIB finds use in a diversity of applications requiring communication among digital devices over short distances. Common examples are: programmable instrumentation systems, computer to peripherals, etc.

The interface is completely defined in the IEEE Std.-488-1978.

A typical implementation consists of logical devices which talk (talker), listen (listeners), and control GPIB activity (controllers).

Interface Functions

The interface between any device and the bus may have a combination of several different capabilities (called 'functions'). Among a total of ten functions defined, the Talker, Listener, Source Handshake, Acceptor Handshake and Controller are the more common examples. The Talker function allows a device to transmit data. The Listener function allows reception. The Source and Acceptor Handshakes, synchronized with the Talker and Listener functions respectively, exchange the handshake signals that coordinate data transfer. The Controller function allows a device to activate the interface functions of the various devices through commands. Other interface functions are: Service request, Remote local, Parallel poll, Device clear and Device trigger. Each interface may not contain all these functions. Further, most of these functions may be implemented to various levels (called 'subsets') of capability. Thus, the overall capability of an interface may be tailored to the needs of the communicating device.

Electrical Signal Lines

As shown in Figure 1, the GPIB is composed of eight data lines (D08-D01), five interface management lines (IFC, ATN, SRQ, REN, EOI), and three transfer control lines (DAV, NRFD, NDAC).

The eight data lines are used to transfer data and commands from one device to another with the help of the management and control lines. Each of the five interface management lines has a specific function.

ATN (attention) is used by the Controller to indicate that it (the controller) has access to the GPIB and that its output on the data lines is to be interpreted as a command. ATN is also used by the controller along with EOI to indicate a parallel poll.

SRQ (service request) is used by a device to request service from the controller.

REN (remote enable) is used by the controller to specify the command source of a device. A device can be issued commands either locally through its front panel or by the controller.

EOI (end or identify) may be used by the controller as well as a talker. A controller uses EOI along with ATN to demand a parallel poll. Used by a talker, EOI indicates the last byte of a data block.

IFC (interface clear) forces a complete GPIB interface to the idle state. This could be considered the GPIB's "interface reset." GPIB architecture allows for more than one controller to be connected to the bus simultaneously. Only one of these controllers may be in command at any given time. This device is known as the controller-in-charge. Control can be passed from one controller to another. Only one among all the controllers present on a bus can be the system controller. The system controller is the only device allowed to drive IFC.

Transfer Control Lines

The transfer control lines conduct the asynchronous interlocking three-wire handshake.

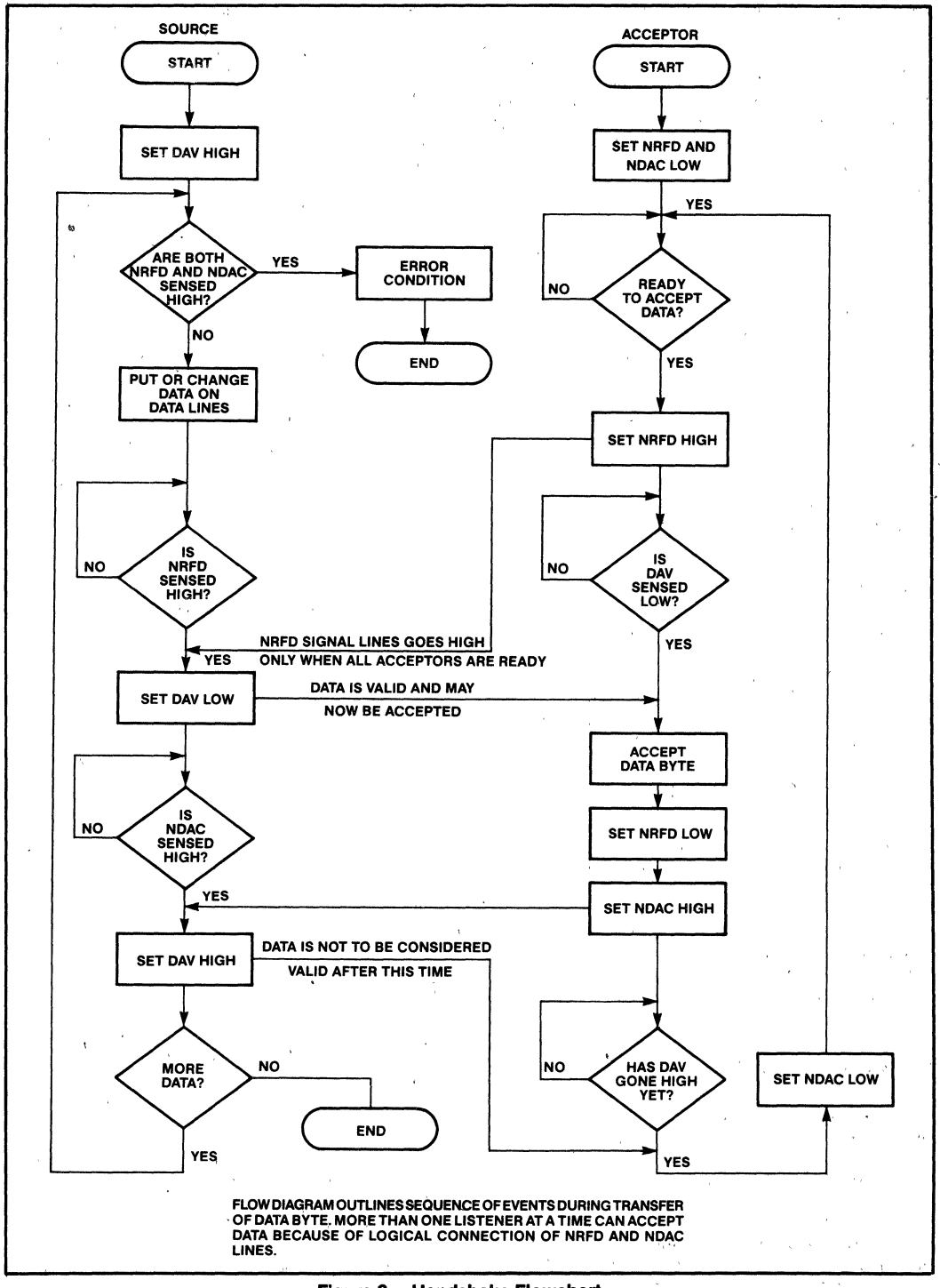
DAV (data valid) is driven by a talker and indicates that valid data is on the bus.

NRFD (not ready for data) is driven by the listeners and indicates that not all listeners are ready for more data.

NDAC (not data accepted) is used by the listeners to indicate that not all listeners have read the GPIB data lines yet.

The asynchronous 3-wire handshake flowchart is shown in Figure 2. This is a concept fundamental to the asynchronous nature of the GPIB and is reviewed in the following paragraphs.

Assume that a talker is ready to start a data transfer. At the beginning of the handshake, NRFD is false indicating that the listener(s) is ready for data. NDAC is true indicating that the listener(s) has not accepted the data, since no data has been sent yet. The talker places data on the data lines, waits for the required settling time, and then indicates valid data by driving DAV true. All active listeners drive NRFD true indicating that they are not



FLOW DIAGRAM OUTLINES SEQUENCE OF EVENTS DURING TRANSFER OF DATA BYTE. MORE THAN ONE LISTENER AT A TIME CAN ACCEPT DATA BECAUSE OF LOGICAL CONNECTION OF NRFD AND NDAC LINES.

Figure 2. Handshake Flowchart

ready for more data. They then read the data and drive NDAC false to indicate acceptance. The talker responds by deasserting DAV and readies itself to transfer the next byte. The listeners respond to DAV false by driving NDAC true. The talker can now drive the data lines with a new data byte and wait for NRFD to be false to start the next handshake cycle.

Bus Commands

When ATN and DAV are true data patterns which have been placed by the controller on the GPIB, they are interpreted as commands by the other devices on the interface. The GPIB standard contains a repertory of commands such as MTA (My Talk Address), MSA (My Secondary Address), SPE (Serial Poll Enable), etc. All other patterns in conjunction with ATN and DAV are classified as undefined commands and their meaning is user-dependent.

Addressing Techniques

To allow the controller to issue commands selectively to specific devices, three types of addressing exist on the GPIB: talk only/listen only (ton/lon), primary, and secondary.

Ton/lon is a method where the ability of the GPIB interface to talk or listen is determined by the device and not by the GPIB controller. With this method, fixed roles can be easily designated in simple systems where reassignment is not necessary. This is appropriate and convenient for certain applications. For example, a logic analyzer might be interfaced via the GPIB to a line printer in order to document some type of failure. In this case, the line printer simply listens to the logic analyzer, which is a talker.

The controller addresses devices through three commands, MTA (my talk address), MLA (my listen address), and MSA (my secondary address). The device address is imbedded in the command bit pattern. The device whose address matches the imbedded pattern is enabled. Some devices may have the same logical talk and listen addresses. This is allowable since the talker and listener are separate functions. However, two of the same functions cannot have the same address.

In primary addressing, a device is enabled to talk (listen) by receiving the MTA (MLA) message.

Secondary addressing extends the address field from 5 to 10 bits by allowing an additional byte. This additional byte is passed via the MSA message. Secondary addressing can also be used to logically divide devices into various subgroups. The MSA message applies only to the device(s) whose primary address immediately preceed it.

INTEL'S® GPIB COMPONENTS

The logic designer implementing a GPIB interface has, in the past, been faced with a difficult and complex discrete logic design. Advances in LSI technology have produced sophisticated microprocessor and peripheral devices which combine to reduce this once complex interface task to a system consisting of a small set of integrated circuits and some software drivers. A microprocessor hardware/software solution and a high-level language source code provide an additional benefit in end-product maintenance. Product changes are a simple matter of revising the product software. Field changes are as easy as exchanging EPROMS.

Intel® has provided an LSI solution to GPIB interfacing with a talker/listener device (8291A), a controller device (8292), and a transceiver (8293). An interface with all capabilities except for the controller function can be built with an 8291A and a pair of 8293's. The addition of the 8292 produces a complete interface. Since most devices in a GPIB system will not have the controller function capability, this modular approach provides the least cost to the majority of interface designs.

Overview of the 8291A GPIB Talker/Listener

The Intel® 8291A GPIB Talker/Listener operates over a clock range of 1 to 8 MHz and is compatible with the MCS-85, iAPX-86, and 8051 families of microprocessors.

A detailed description of the 8291A is given in the data sheet.

The 8291A implements the following functions: Source Handshake (SH), Acceptor Handshake (AH), Talker Extended (TE), Service Request (SRQ), Listener Extended (LE), Remote/Local (RL), Parallel Poll (PP2), Device Clear (DC), and Device Trigger (DT).

Current states of the 8291A can be determined by examining the device's status read registers. In addition, the 8291A contains 8 write registers. These registers are shown in Figure 3. The three register select pins RS3-RS0 are used to select the desired register.

The data — in register moves data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. When the 8291A is addressed to talk, it uses the data - out register to move data onto the GPIB. The serial poll mode and status registers are used to request service and program the serial poll status byte.

A detailed description of each of the registers, along with state diagrams can be found in the 8291A data sheet.

READ REGISTERS								RS2	RS1	RS0	WRITE REGISTERS							
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	0	0	0	D07	D06	D05	D04	D03	D02	D01	D00
DATA IN											DATA OUT							
CPT	APT	GET	END	DEC	ERR	BO	BI	0	0	1	CPT	APT	GET	END	DEC	ERR	BO	BI
INTERRUPT STATUS 1											INTERRUPT ENABLE 1							
INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC	0	1	0	0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC
INTERRUPT STATUS 2											INTERRUPT ENABLE 2							
S8	SRQS	S6	S5	S4	S3	S2	S1	0	1	1	S8	RSV	S6	S5	S4	S3	S2	S1
SERIAL POLL STATUS 2											SERIAL POLL MODE							
ton	Lon	EOI	LPAS	TPAS	LA	TA	MJMN	1	0	0	TO	LO	0	0	0	0	ADM1	ADM0
ADDRESS STATUS											ADDRESS MODE							
CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	1	0	1	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
COMMAND PASS THROUGH											AUX MODE							
INT	DTO	DLO	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0	1	1	0	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1
ADDRESS 0											ADDRESS 0/1							
X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1	1	1	1	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
ADDRESS 1											EOS							

Figure 3. 8291A REGISTERS

Address Mode

The address mode and status registers are used to program the addressing modes and track addressing states. The auxiliary mode register is used to select a variety of functions. The command pass through register is used for undefined commands and extended addresses. The address 0/1 register is used to program the addresses to which the 8291A will respond. The address 0 and address 1 registers allow reading of these programmed addresses plus trading of the interrupt bit. The EOS register is used to program the end of sequence character.

Detailed descriptions of the addressing modes available with the 8291A are described in the 8291A data sheet. Examples of how to program these modes are shown below.

1. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0010 0001	Major is Talking. Address = 01H
Address 0/1 Register	1100 0010	Minor is Listener. Address = 02H

2. MODE: — Talker has single address of 01H
   — Listener has single address of 02H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Select Mode 1 Addressing
Address 0/1 Register	0100 0010	Major is Listener. Address = 02H
Address 0/1 Register	1010 0001	Minor is Talking. Address = 01H

Note that in both of the above examples, the listener will respond to a MLA message with five least significant bits equal to 02H and the talker to a 01H.

3. MODE: — Talker and listener both share a single address of 03H.

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0001	Selects Mode 1 Addressing
Address 0/1 Register	0000 0011	Talker and Listener Address = 03
Address 0/1 Register	1110 0000	Minor Address is disabled

4. MODE: — Talker and listener have a primary address of 04H and a secondary address of 05H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0010	Selects Mode 2 Addressing
Address 0/1 Register	0000 0100	Primary Address = 04H
Address 0/1 Register	1000 0101	Minor Address is disabled

5. MODE: — Talker has a primary address of 06H. Listener has a primary address of 07H

CPU WRITES TO:	PATTERN	COMMENT
Address Mode Register	0000 0011	Select Mode 3
Address 0/1 Register	0010 0110	Talker Address = 06
Address 0/1 Register	1100 0111	Listener Primary = 07

The CPU will verify the secondary addresses which could be the same or different.

APPLICATION OF THE 8291A

This phase of the application note will examine programming of the 8291A, corresponding bus commands and responses, CPU interruption, etc. for a variety of GPIB activities. This should provide the reader with a clear understanding of the role the 8291A performs in a GPIB system. The talker function, listener function, remote message handling, and remote/local operations including local lockout, are discussed.

Talker Functions

TALK-ONLY (ton). In talk only mode the 8291A will not respond to the MTA message from a controller. Generally, ton is used in an environment which does not have a controller. Ton is also employed in an interface that includes the controller function.

When the 8291A is used with the 8292, the sequence of events for initialization are as follows:

1) The Interrupt/Enable registers are programmed.
2) Ton is selected.
3) Settling time is selected.
4) EOS character is loaded.
5) "Pon" local message is sent.
6) CPU waits for Byte Out (BO) and sends a byte to the data out register.

Addressed Talker (Via MTA Message)

The GPIB controller will direct the 8291A to talk by sending a My Talk Address (MTA) message containing the 8291A's talk address. The sequence of events is as follows:

1) The interrupt enable and serial poll mode registers are programmed.
2) Mode 1 is selected.
3) Settling time is selected.
4) Talker and listener addresses are programmed.
5) Power on (pon) local message is sent.
6) CPU waits for an interrupt. When the controller has sent the MTA message for the 8291A an interrupt will be generated if enabled and the ADSC bit will be set.
7) CPU reads the Address Status register to determine if the 8291A has been addressed to talk (TA = 1).
8) CPU waits for an interrupt from either BO or ADSC.
9) When BO is set, the CPU writes the data byte to the data out register.
10) CPU continues to poll the status registers.
11) When unaddressed ADSC, will be set and TA reset.

LISTENER FUNCTIONS

LISTEN-ONLY (lon). In listen-only mode the 8291A will not respond to the My Listen Address (MLA) message from the controller. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) Lon is selected.
3) EOS characted is programmed.
4) "Pon" local message is sent.
5) CPU waits for BI and reads the byte from the data-in register.

Note that enabling both ton and lon can create an internal loopback as long as another listener exists.

Addressed Listening (Via the MLA Message)

The GPIB controller will direct the 8291A to listen by sending a MLA message containing the 8291A's listen address. The sequence of events is as follows:

1) The Interrupt Enable registers are programmed.
2) The serial poll mode register is loaded as desired.
3) Talker and listener addresses are loaded.
4) "Pon" local message is sent
5) The CPU waits for an interrupt. When the controller has sent the MLA message for the 8291A, the ADSC bit will be set.
6) The CPU reads the Address Status Register to determine if the 8291A has been addressed to listen (LA = 1).
7) CPU waits for an interrupt for BI or ADSC.
8) When BI is set, the CPU reads the data byte from the data-in register.
9) The CPU continues to poll the status registers.
10) When unaddressed, ADSC will be set and LA reset.

Remote/Local and Lockout

Remote and local refer to the source of control of a device connected to the GPIB. Remote refers to control from the GPIB controller-in-charge. Local refers to control from the device's own system. Reference should be made to the RL state diagram in the 8291A data sheet.

Upon "pon" the 8291A is in the local state. In this state the REM bit in Interrupt Status 1 Register is reset. When the GPIB controller takes control of the bus it will drive the REN (remote enable) line true. This will cause the REM bit and REMC (remote/local change) bit to be set. The distinction between remote and local modes is necessary in that some types of devices will have local controls which have functions which are also controlled by remote messages.

In the local state the device is allowed to store, but not respond to, remote messages which control functions which are also controlled by local messages. A device

which has been addressed to listen will exit the local state and go the the remote state if the REN message is true and the local rtl (return to local) message is false. The state of the "rtl" local message is ignored and the device is "locked" into the local state if the LLO remote message is true. In the Remote state the device is not allowed to respond to local message which control function that are also controlled by remote messages. A device wil exit the remote state and enter the local state when REN goes false. It will also enter the local state if the GTL (go to local) remote message is true and the device has been addressed to listen. It will also enter the local state if the rtl message is true and the LLO message is false or ACDS is inactive.

A device will exit the remote state and enter RWLS (remote with lockout state) if the LLO (local lockout) message is true and ACDS is active. In this mode, those local message which control functions which are also controlled by remote messages are ignored. In other words, the "rtl": message is ignored. A device will exit RWLS and to to the local state if REN goes false. The device will exit RWLS and go to LWLS if the GTL message is true and the device is addressed to listen.

Polling

The IEEE-488 standard specifies two methods for a slave device to let the controller know that it needs service.

These two methods are called Serial and Parallel Poll. The controller performs one of these two polling methods after a slave device requests service. As implied in the name, a Serial Poll is when the controller sequentially asks each device if it requested service. In a Parallel Poll the controller asks all of the devices on the GPIB if they requested service, and they reply in parallel.

Serial Poll

When the controller performs a Serial Poll, each slave device sends back to the controller a Serial Poll Status Byte. One of the bits in the Serial Poll Status Byte indicates whether this device requested service or not. The remaining 7 bits are user defined, and they are used to indicate what type of service is required. The IEEE-488 spec only defines the service request bit, however HP has defined a few more bits in the Serial Poll Status Byte. This can be seen in figure 4.

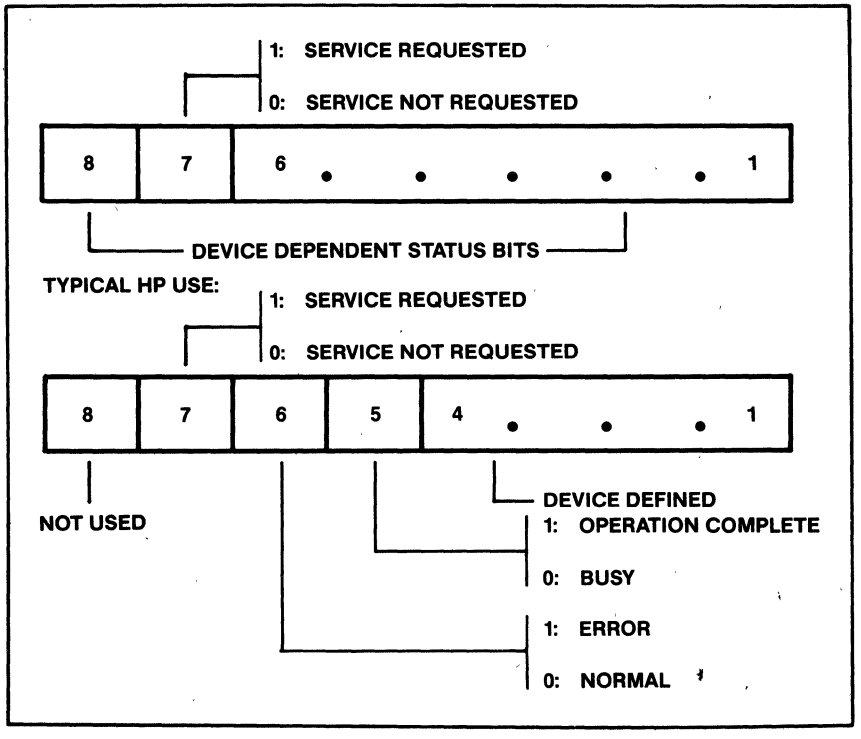


Figure 4. The Serial Poll Status Byte

When a slave device needs service it drives the SRQ line on the GPIB bus true (low). For the 8291A this is done by setting bit 7 in the Serial Poll Status Byte. The CPU in the controller may be interrupted by SRQ or it may poll a register to determine the state of SRQ. Using the 8292 one could either poll the interrupt status register for the SRQ interrupt status bit, or enable SRQ to interrupt the CPU. After the controller recognizes a service request, it goes into the serial poll routine.

The first thing the controller does in the serial poll routine is assert ATN. When ATN is asserted true the controller takes control of the GPIB, and all slave devices on the bus must listen. All bytes sent over the bus while ATN is true are commands. After the controller takes control, it sends out a Universal Unlisten (UNL), which tells all previously addressed listeners to stop listening. The controller then sends out a byte called SPE (Serial Poll Enable). This command notifies all of the slaves on the bus that the controller has put the GPIB in the Serial Poll Mode State (SPMS). Now the controller addresses the first slave device to TALK and puts itself in the listen mode. When the controller resets ATN the device addressed to talk transmits to the controller its Serial Poll Status Byte. If the device just polled was the one requesting service, the SRQ line on the GPIB goes false, and bit 7 in the serial poll status byte of the 8291A is reset. If more than one device is requesting service, SRQ remains low until all of the devices requesting service have been polled, since SRQ is wire-ored. To continue the Serial Poll, the controller asserts ATN, addresses the next device to talk then reads the Serial Poll Status Byte. When the controller is finished polling it asserts ATN, sends the universal untalk command (UNT), then sends the Serial Poll Disable command (SPD). The flow of the serial poll can be seen from the example in figure 5.

0. DEVICE A REQUESTS SERVICE (SRQ)
1. ASSERT ATN
2. UNIVERSAL UNLISTEN (UNL)
3. SERIAL POLL ENABLE (SPE)
4. DEVICE A TALK ADDRESS (MTA)
5. RELEASE ATN
6. DEVICE A STATUS BYTE (STB) (RQS SET)
7. ASSERT ATN
8. DEVICE B TALK ADDRESS (MTA)
9. RELEASE ATN
10. DEVICE B STATUS BYTE (STB) (RQS CLEAR)
11. ASSERT ATN
12. DEVICE C TALK ADDRESS (MTA)
13. RELEASE ATN
14. DEVICE C STATUS BYTE (STB) (RQS CLEAR)
15. ASSERT ATN
16. UNIVESAL UNTALK (UNT)
17. SERIAL POLL DISABLE (SPD)
18. GO PROCESS SERVICE REQUEST

Figure 5. Serial Polling

The following section describes the events which happen in a serial poll when 8291A and 8292 are the controller, and another 8291A is the slave device. While going through this section the reader should refer to the register diagrams for the 8291A and 8292.

A. DEVICE A REQUESTS SERVICE (SRQ BECOMES TRUE)

The slave devices rsv bit in the 8291A's serial poll mode register is set.

B. CONTROLLER RECOGNIZES SRQ AND ASSERTS ATN

Thf 8292's SPI pin 33 interrupts the CPU. The CPU reads the 8292's Interrupt status register and finds the SRQ bit set. The CPU tells the 8292 to 'Take Control Synchronously' by writing a OFDH to the 8292's command register.

C. THE CONTROLLER SENDS OUT THE FOLLOWING COMMANDS: UNIVERSAL UNLISTEN (UNL), SERIAL POLL ENABLE (SPE), MY TALK ADDRESS (MTA).

(MTA is a command which tells one of the devices on the bus to talk.)

The CPU in the controller waits for a BO (byte out) interrupt in the 8291A's interrupt status 1 register before it writes to the Data Out register a 3FH (UNL), 18H (SPE), 010XXXXX (MTA). The X represents the programmable address of a device on the GPIB. When the 8291A in the slave device receives its talk address, the ADSC bit in the Interrupt Status register 2 is set, and in the Address Status Register TA and TPAS bits are set.

D. CONTROLLER RECONFIGURES ITSELF TO LISTEN AND RESETS ATN

The CPU in the controller puts the 8291A in the listen only mode by writing a 40H to the Address Mode register of the 8291A, and then a OOH to the Aux Mode register. The second write is an 'Immediate Execute pon' which must be used when switching addressing modes such as talk only to listen only. To reset ATN the CPU tells the 8292 to 'Go To Standby' by writing a 0F6H to the command register. The moment ATN is reset, the 8291A in the slave device sets SPAS in Interrupt Status 2 register, and transmits the serial poll status byte. SRQS in the Serial Poll Status byte of the 8291A slave device is reset, and the SRQ line on the GPIB bus becomes false.

E. THE CONTROLLER READS THE SERIAL POLL STATUS BYTE, SETS ATN, THEN RECONFIGURES ITSELF TO TALK

The CPU in the controller waits for the Byte In bit (BI) in the 8291A's Interrupt Status 1 register. When this bit is set the CPU reads the Data In register to receive the Serial Poll Status Byte. Since bit 7 is set, this was the device which requested service. The CPU in the controller tells the 8292 to 'Take Control Synchronously' which asserts ATN. The moment ATN is asserted true the 8291A in the slave device resets SPAS, and sets the Serial Poll Com-

plete (SPC) bit in the Interrupt Status 2 register. The controller reconfigures itself to talk by setting the TO bit in the Address Mode register and then writing a OOH to the Aux Mode register.

F. THE CONTROLLER SENDS THE COMMANDS UNIVERSAL UNTALK (UNT), AND SERIAL POLL DISABLE (SPD) THEN RESETS THE SRQ BIT IN THE 8292 INTERRUPT STATUS REGISTER

The CPU in the controller waits for the BO Interrupt status bit to be set in the Interrupt Status 1 register of the 8291A before it writes 5FH (UNT) and 19H (SPD) to the Data Out register. The CPU then writes a 2BH to the 8292's command register to reset the SRQ status bit in the Interrupt Status register. When the 8291A in the slave device receives the UNT command the ADSC bit in the Interrupt Status 2 register is set, and the TA and TPAS bits in the Address Status register will be reset. At this point the controller can service the slave device's request.

Note that in the software listing of AP-66 (USING THE 8292 GPIB CONTROLLER) there is a bug in the serial poll routines. In the 'SRQ ROUTINE' when the CPU finds that the SRQ bit in the interrupt status register is set, it immediately writes the interrupt Acknowledge command to the 8292 to reset this bit. However the SRQ GPIB line will still be driven true until the slave device driving SRQ has been polled. Therefore, the SRQ status bit in the 8292 will become set and latched again, and as a result the SRQ status bit in the 8292 will still be set after the serial poll. The proper time to reset the SRQ bit in the 8292 is after SRQ on the GPIB becomes false.

Parallel Poll

The 8291A supports an additional method for obtaining status from devices known as parallel poll (PPOL). This method limits the controller to a maximum of 8 devices at a time since each device will produce a single bit response on the GPIB data lines. As shown in the state diagrams, there are three basic parallel poll sates: PPIS (parallel poll idle state), PPSS (parallel poll standby state), and PPAS (parallel poll active state).

In PPIS, the device's parallel poll function is in the idle state and will not respond to a parallel poll. PPSS is the standby state, a state in which the device will respond to a parallel poll from the controller. The response is inititated by the controller driving both ATN and EOI true simultaneously.

The 8291A state diagram shows a transition from PPIS to PPSS with the "lpe" message. This is a PP2 implementation for a parallel poll. This "lpe" (local poll enable) local message is achieved by writing $011USP_3P_2P_1$ to the Aux Mode Register with U=0. The S bit is the sense bit. If the "ist" (individual status) local message value matches the sense bit, then the 8291A will give a true response to a parallel poll. Bits P_3-P_1 identify which data line is used for a response.

For example, assume the programmer decides that the system containing the 8291A shall participate in parallel poll. The programmer, upon system initialization would write to the Aux Mode Register and reset the U bit and set the S bit plus identify a data line (P_3-P_1 bits). At "pon," the 8291A would not resond true to a parallel poll unless the parallel poll flag is set (via Aux Mode Register command).

When a status condition in the user system occurs and the programmer decides that this condition warrants a true response, then programmers software should set the parallel poll flag. Since the S bit value matches the "ist" (set) condition a true response will be given to all parallel polls.

An additional method of parallel polling reading exists known as a PP1 implementation. In this case the controller sends a PPE (parallel poll enable) message. PPE contains a bit pattern similar to the bit pattern used to program the "lpe" local message. The 8291A will receive this as an undefined command and use it to generate an "lpe" message. Thus the controller is specifying the sense bits and data lines for a response. A PPD (parallel poll disable) message exists which clears the bits $SP_3P_2P_1$ and sets the U bit. This also will be received by the 8291A and used to generate an "lpe" false local message.

The actual sequence of events is as follows. The controller sends a PPC (parallel poll configure) message. This is an undefined command which is received in the CPT register and the handshake is held off. The local CPU reads this bit pattern, decodes it, and sends a VSCMD message to the Aux Mode Register. The controller then sends a ppe message which is also recieved as a undefined command in the CPT register. The local CPU reads this, decodes it clears the MSB, and writes this to the Aux Mode Register generating the "lpe" message.

The controller then sends ATN and EOI true and the 8291A drives the appropriate data line if the "ist" (parallel poll flag) is true. The controller will then send a PPD (parallel poll disable) message (again, an undefined command). The CPU reads this from the CPT register and uses it to write a new "lpe" message (this "lpe" message will be false). The controller then sends a PPU (parallel poll unconfigure) message. Since this is also an undefined command, it goes into the CPT register. When the local CPU decodes this, the CPU should clear the "ist" (parallel poll flag).

APPLICATION EXAMPLES

In the course of developing this application note, two complete and identical GPIB systems were built. The

schematics and block diagrams are contained in Appendix 1. These systems feature an 8088 CPU, 8237 DMA controller,serial I/O (8251A and 8253),RAM, EPROM, and a complete GPIB talker/listener controller. Jumper switches were provided to select between a controller function and a talker/listener function. This system design is based on the design of Intel's SDK-86 prototyping kit and thus shares the same I/O and memory addresses. This system uses the same download software to transfer object files from Intel development systems.

Two Software Drivers

Two software drivers were developed to demonstrate a ton/lon environment. These two programs (BOARD 1 and BOARD 2) are contained in Appendix 2.

In this example, one of the systems (BOARD 1) initially is programmed in talk-only mode and synchronization is achieved by waiting for the listening board to become active. This is sensed by the lack of a GPIB error since a condition of no active listener produces an ERR status condition. Board 1 upon detecting the presence of an active listener transmitts a block of 100 bytes from a PROM memory across the bus. The second system (BOARD 2) receives this data and stores it in a buffer, EOI is sent true by the talker (BOARD 1) with the last byte of data. Upon detection of EOI, BOARD 2 switches to the talk only mode while BOARD 1 upon terminal count switches to the listen only mode. BOARD 2 then detects the presence of an active listener and transmitts the contents of its buffer back to BOARD 1 which stores this data in the buffer. EOI again is sent with the last byte and BOARD 2 switches back to listen-only. BOARD 1 upon detecting EOI then compares the contents of its buffer with the contents of its PROM to ensure that no data transmission errors occured. The process then repeats itself.

8291A with HP 9835A

An example of the 8291A used in conjunction with a bus controller is also included in this application note. In this example, the 8291A system used in previous experiments was connected via the GPIB to a Hewlett-Packard 9835A desktop computer. This computer contains, in addition to a GPIB interface, a black and white CRT, keyboard, tape drive for high quality data cassettes, and a calculator type printer. The software for the HP 9835A is shown in Appendix 3. The user should refer to the operation manuals for the HP 9835A for information on the features and programming methods for the HP 9835A.

In this example, the 8292 was removed from its socket and the OPTA and OPTB pins of the two 8293 transceiver reconfigured to modes 0 and 1. Optionally, the mode pins could have been left wired for modes 2 and 3 and the 8292 left in its socket with its SYC pin wired to ground. This would have produced the same effect.

The first action performed is sending IFC. Generally, this is done when a controller first comes on line. This pulse is at least 100 us in duration as specified by the IEEE-488 standard.

The software checks to see if active listeners are on line. For demonstration purposes, the HP 9835A will flag the operator to indicate that listeners are on line.

The HP 9835A then configures and performs a parallel poll (PPOL). The parallel poll indicates 1 bit of status of each device in a group of up to 8 devices. Such information could be used by an application program to determine whether optional devices are part of a system configuration. Such optional devices might include mass storage devices, printers, etc. where the application software for the controller might need to format data to match each type of device. Once the PPOL sequence is finished, the HP 9835A offers the user the opportunity to execute user commands from the keyboard. At this time the HP 9835A sits in a loop waiting for an SRQ condition. When the operator hits a key on the keyboard, the HP 9835A processor is interrupted and vectors to a service routine where the key is read and the appropriate routine is executed. The HP 9835A will then return to the loop checking for SRQ true. For this application, the valid keys are G,D,R,H,and X. Pressing the "G" key causes the GET command to be sent across the bus. A message to this effect is printed in the CRT and the HP 9835A returns. The "D" key causes the SDC message to be sent with the 8291A being the addressed device. Again, an appropriate message is output on th HP 9835A CRT. The "R" key causes the GTL message to be sent. The CRT displays "REMOTE MESSAGE SENT." The "H" key causes a menu to be displayed on the HP 9835A CRT screen. This menu lists the allowed commands and their functions. NO GPIB commands are sent. The "X" key allows the operator to send one line of data across the bus. The line of data is terminated by a carriage return and line feed produced by pressing the "CONTINUE" key on the HP 9835A.

The characters are stored in the sequence entered into a buffer whose maximum size is 80 characters. Pressing the "CONTINUE" key terminates storing characters in the array and all characters including the carriage return and line feed are sent. EOI is then sent true with a false byte of OOH. This false byte is due to the 1975 standard which allows asyncronous sending and reception of EOI. (The 8291A supports the later 1978 standard which eliminates this false byte).

After any key command is serviced control returns to the loop which checks for SRQ active. Should SRQ be active, then the keyboard interrupt is disabled and a message printed to indicate that SRQ has been received true.

The controller then performs a parallel poll.

This is an example of how parallel poll may be used to

quickly check which group of devices contains a device sending SRQ. The eight devices in a group would, of course, have software drivers which allow a true response to a PPOL if that device is currently driving SRQ true. This would be a valuable method of isolation of the SRQ source in a system with a large number of devices. In this application program, only the response from the 8291A is of concern and only the 8291A's response is considered. It does, however, demonstrate the technique employed. If a true response from the 8291A is detected, then a message to this effect is printed on the HP 9835A CRT screen. From this process, the controller has identified the device requesting service and will use a serial poll(SPOL) to determine the reason for the service request. This method of using PPOL is not specifically defined by the IEEE-488 standard but is a use of the resources provided.

The controller software then prints a message to indicate that it is about to perform a serial poll. This serial poll will return to the controller the current status of the 8291A and clear the service request. The status byte received is then printed on the CRT screen of the HP 9835A. One of the 8291A status bits indicates that the 8291A system has a field (on line or less) of data to transfer to the HP 9835A. If this bit is set, then the HP 9835A addresses the 8291A system to talk. The data is sent by the 8291A system is then printed on the CRT screen of the HP 9835A. The HP 9835 then enables the keyboard interrupts and goes into its SRQ checking loop.

Appendix 4 contains the software for the 8291A system which is connected to the HP 9835A via the GPIB. This software throws away the first byte of data it receives since this transfer was used by the HP 9835A to test when the 8291A system came on line.

Next, both status registers are read and stored in the two variable STAT 1 and STAT 2. It is necessary to store the status since reading the status registers clears the status bits.

Initially, six status bits are evaluated (END, GET, CPT, DEC, REMC, ADSC). Some of these conditions require that additional status bits be evaluated.

If END is true, then the 8291A system has received a block from the HP 9835A and the contents of a buffer is printed on the CRT screen. Next, the CPT bit is checked. PPC and PPE are the only valid undefined commands in this example.

Next, the GET bit is examined and if true, the CRT screen connected to the serial channel on the 8291A system prints a message to indicate that the trigger command has been received. A similar process occurs with the DEC and REMC status bits.

Address Status Change (ADSC) is checked to see if the 8291A has been addressed or unaddressed by the controller. If ADSC is false, then the software checks the keyboard at the CRT terminal. If ADSC is set, then the TA and LA bits are read and evaluated to determine whether the 8291A has been addressed to talk or listen. The DMA controller is set to start transfers at the start of the character buffer and the type of transfer is determined by whether the 8291A is in TADS or LADS. We only need to set up the DMA controller since the transfers will be transparent to the system processor. The keyboard from the CRT terminal is then checked. If a key as been hit, then this character is stored in the character buffer and the buffer printer set to the next character location. This process repeats until the received character is a line feed. The line feed is echoed to the CRT, the serial poll status byte updated and the SRQ line driven true. This allows the 8291A system to store up to one line of characters before requesting a transfer to the controller. Recall that upon receiving an SRQ, the controller will perform a serial poll and subsequently address the 8291A to talk. The 8291A system then goes back to reading the status register thus repeating the process.

CONCLUSION

This application note has shown a basic method to view the IEEE 488 bus, when used in conjunction with Intel's® 8291A.

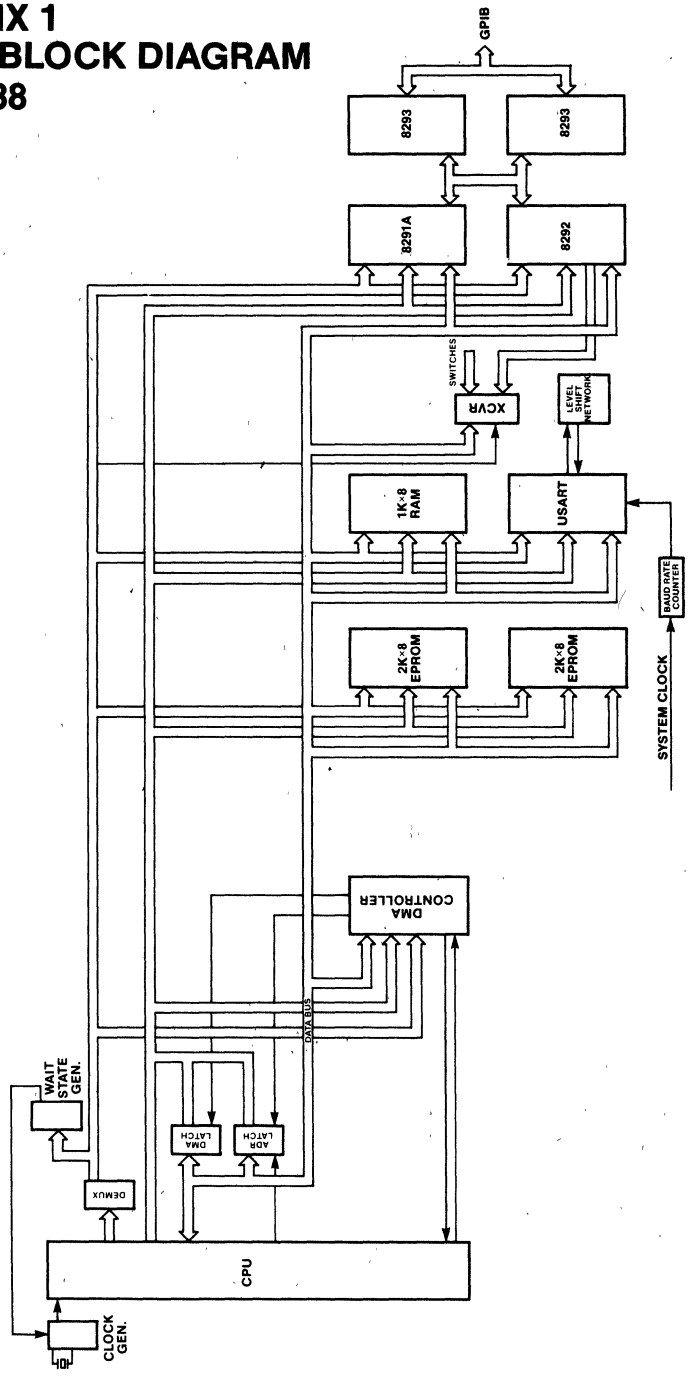
The main reference for GPIB questions is the IEEE Standard 488 - 1978. Reference 8291A's data sheet for detailed information on it.

Additional Intel® GPIB products include iSBX-488, which is a multimode board consisting of the 8291A, 8292, and 8293.

REFERENCES

8291A Data Sheet
8292 Data Sheet
8293 Data Sheet
Application Note #66 "Using the 8292 GPIB Controller"
PLM-86 User Manual
HP 9835A User's Manual
IEEE—488—1978 Standard

APPENDIX 1
SYSTEM BLOCK DIAGRAM WITH 8088



230832-001

APPENDIX 2
SOFTWARE DRIVERS FOR BLOCK DATA TRANSFER

```
PL/M-86 COMPILER     BOARD 1

ISIS-II PL/M-86  V1.1  COMPILATION OF MODULE BOARD 1
OBJECT MODULE PLACED IN   F1   BRD1   OBJ
COMPILER INVOKED BY.     PLM86.  F1   BRD1.  SRC SYMBOLS MEDIUM

          /*   BOARD 1 TPT PROGRAM          */
          /*   THIS BOARD TALKS TO THE OTHER BOARD BY  */
          /*   TRANSFERRING A BLOCK OF DATA VIA THE 8237  */
          /*   COUPLED WITH THE 8291A     THE 8291A IS PROGRAM- */
          /*   MED TO SEND EOI WHEN RECOGNIZING THE LAST     */
          /*   DATA BYTE'S BIT PATTERN    WHILE DATA IS BEING */
          /*   TRANSFERRED, THE PROCESSOR PERFORMS  I/O READS */
          /*   OF THE 8237 COUNT REGISTERS TO SIMULATE BUS    */
          /*   ACTIVITY, AND TO DETERMINE WHEN TO TURN THE    */
          /*   LINE AROUND.  AFTER THE 8237 HAS REACHED  */
          /*   TERMINAL COUNT, THE 8291A IS PROGRAMMED TO      */
          /*   THE LISTENER STATE AND WAITS FOR THE BLOCK    */
          /*   TO BE TRANSMITTED BACK FROM THE SECOND BOARD.  */
          /*   THIS DATA IS PLACED IN A SECOND BUFFER AND     */
          /*   ITS CONTENTS COMPARED WITH THE ORIGINAL DATA  */
          /*   TO CHECK FOR INTERFACE INTEGRITY.   */

1         BOARD1:

          DO;
               /* PROCEDURES */

2    1         CO:  PROCEDURE (XXX) ;
3    2              DECLARE XXX BYTE,
                    SER$STAT  LITERALLY  'OFFF2H',
                    SER$DATA  LITERALLY  'OFFF0H',
                    TXRDY     LITERALLY  '01H',
4    2              DO WHILE (INPUT  (SER$STAT) AND TXRDY) <>     TXRDY;
5    3              END;
6    2              OUTPUT (SER$DATA)      = XXX;
7    2         END CO;

                    /*   SETUP BUFFERS */
8    1              DECLARE BUFF2 (100)  BYTE;    /*  RAM STORAGE AREA */
9    1                   DECLARE BUFF1 (100) BYTE DATA

                    (1,2,3,4,5,6,7,8,9,10H,
                    11H, 12H, 13H, 14H, 15H, 16H, 17H, 18H, 19H, 20H,
                    21H, 22H, 23H, 24H, 25H, 26H, 27H, 28H, 29H, 30H,
                    31H, 32H, 33H, 34H, 35H, 36H, 37H, 38H, 39H, 40H,
                    41H, 42H, 43H, 44H, 45H, 46H, 47H, 48H, 49H, 50H,
                    51H, 52H, 53H, 54H, 55H, 56H, 57H, 58H, 59H, 60H,
                    61H, 62H, 63H, 64H, 65H, 66H, 67H, 68H, 69H, 70H,
                    71H, 72H, 73H, 74H, 75H, 76H, 77H, 78H, 79H, 80H,
                    81H. 82H, 83H, 84H, 85H, 86H, 87H, 88H, 89H, 90H,
```

```
PL/M-86  COMPILER  BOARD1

                   91H, 92H, 93H, 94H, 95H, 96H, 97H, 98H, 99H, 0DH);
10  1              DECLARE BUFF3(17)  BYTE DATA
                   (0DH, 0AH, 'COMPARE ERROR', 0DH, 0AH);   /* ROM STORAGE AREA */

                   /* 8237 PORT ADDRESSES  */

11  1              DECLARE

                   CLEAR$FF          LITERALLY '0FFDDH', /* MASTER CLEAR */
                   START$0$LO        LITERALLY '0FFD0H',
                   START$0$HI         LITERALLY '0FFD0H',
                   0$COUNT$LO        LITERALLY '0FFD1H',
                   0$COUNT$HI        LITERALLY '0FFD1H',
                   SET$MODE          LITERALLY '0FFD8H',
                   CMD$37            LITERALLY '0FFD8H',
                   SET$MASK          LITERALLY '0FFDFH',

                   /* 8237 COMMAND  - DATA BYTES */
12  1              DECLARE           DMA$ADR$TALK    POINTER;
13  1              DECLARE           DMA$ADR$LSTN    POINTER;

14  1              DECLARE

                   RD$TRANSFER       LITERALLY '48H',
                   WR$TRANSFER       LITERALLY '44H',
                   NORM$TIME         LITERALLY '20H',
                   TC$LO1            LITERALLY '0FFH',
                   TC$HI1            LITERALLY '00H',
                   TC$LO2            LITERALLY '99D',          /* 100 XFERS  */
                   TC          LITERALLY '01H',
                   I                 BYTE;

15  1              DECLARE

                   DMA$WRD$TALK (2)  WORD    AT    (@DMA$ADR$TALK),
                   DMA$WRD$LSTN(2)   WORD    AT    (@DMA$ADR$LSTN);

                   /* 8291A PORT ADDRESSES */

16  1              DECLARE

                   PORT$OUT          LITERALLY '0FFC0H', /* DATA OUT*/
                   PORT$IN           LITERALLY '0FFC0H'
                   STATUS$1          LITERALLY '0FFC1H', /*INTR STAT 2*/
                   STATUS$2          LITERALLY '0FFC2H', /* INTR STAT 2 */
                   ADDR$STATUS       LITERALLY '0FFC4H',
                   COMMAND$MOD       LITERALLY '0FFC5H', /*CMD PASS THRU */
                   ADDR$0            LITERALLY '0FFC6H',
                   EOS$REG           LITERALLY '0FFC7H', /*  EOS REGISTER */
```

```
                /* 8291A COMMAND  - DATA BYTES */

PL/M-86 COMPILER    BOARD1

17  1                   DECLARE

                        END$EOI    LITERALLY        '88H',
                        DNE   LITERALLY        '10H',
                        PON   LITERALLY        '00H',
                        RESET      LITERALLY        '02H',
                        CLEAR      LITERALLY        '00H',
                        DMA$REQ$L LITERALLY        '10H',
                        DMA$REQ$T LITERALLY        '20H',
                        MOD1$TO    LITERALLY        '80H',
                        MOD1$LO    LITERALLY        '40H',
                        EOS   LITERALLY        'ODH',
                        PRESCALER LITERALLY        '23H',
                        HIGH$SPEED LITERALLY            '0A4H',
                        OKAY       LITERALLY        'OFFFFH',
                        XYZ   BYTE,
                        MATCH      WORD,
                        BO    LITERALLY        '02H',
                        BI    LITERALLY        '01H',
                        ERR   LITERALLY        '04H',

          /* CODE BEGINS */

18  1     START91:

                        OUTPUT  (STATUS$2)  =CLEAR;   /* SHUT-OFF DMA REQ BITS TO */
                                                      /* PREVENT EXTRA DMA REQS  */
                                                       /*FROM 8291A                */

                        /* MANIPULATE DMA ADDRESS VARIABLES */

19  1                   DMA$ADR$TALK    =(@BUFF1);
20  1                   DMA$ADR$LSTN    =(@BUFF2);
21  1                   DMA$WRD$TALK(1)=SHL (DMA$WRD$TALK(1), 4);
22  1                   DMA$WRD$TALK(0)=DMA$WRD$TALK (0) + DMA$WRD$TALK (1);
23  1                   DMA$WRD$LSTN(1)=SHL (DMA$WRD$LSTN (1), 4);
24  1                   DMA$WRD$LSTN(0)=DMA$WRD$LSTN (0) +DMA$WRD$LSTN (1);

25  1           INIT37I:
                 /* INIT 8237 FOR TALKER FUNCTIONS */

26  1                   OUTPUT     (CLEAR$FF)      =CLEAR;/* TOGGLE MASTER CLEAR */
27  1                   OUTPUT     (CMD$37)        =NORM$TIME;
28  1                   OUTPUT     (SET$MODE)      =RD$TRANSFER;
                        OUTPUT     (SET$MASK)      =CLEAR;
29  1                   OUTPUT     (START$0$LO)       =DMA$WRD$TALK (0);
30  1                   DMA$WRD$TALK (0)           =SHR (DMA$WRD$TALK (0), 8);
31  1                   OUTPUT     (START$0$HI)       =DMA$WRD$TALK (0);
32  1                   OUTPUT     (0$COUNT$LO)       =TC$LO2;
33  1                   OUTPUT     (0$COUNT$HI)       =TC$HI2;
                        /* INIT 8291A FOR TALKER FUNCTIONS */

PL/M-86 COMPILER      BOARD1
```

```
34  1                OUTPUT  (EOS$REG)     =EOS;
35  1                OUTPUT  (COMMAND$MOD)     =END$EOI; /*  EOI ON EOS SENT */
36  1                OUTPUT  (ADDR$STATUS)     =MOD1$TO; /* TALK ONLY */
37  1                OUTPUT  (COMMAND$MOD)     =PRESCALER,
38  1                OUTPUT  (COMMAND$MOD)     =HIGH$SPEED,
39  1                OUTPUT  (COMMAND$MOD)     =PON;

40  1                DO WHILE (INPUT  (STATUS$1) AND BO)  =0;
41  2                END;     /*  WAIT FOR BO INTR  */
42  1                OUTPUT  (PORT$OUT)  = 0AAH;

43  1                DO WHILE  (INPUT  (STATUS$1) AND ERR) = ERR;
44  2                   DO WHILE  (INPUT  (STATUS$1)  AND BO) = 0;
45  3                   END;  /*  WAIT FOR BO INTR  */
46  2                   OUTPUT  (PORT$OUT)  =0AAH;
47  2                END;

48  1                OUTPUT  (STATUS$2)  =DMA$REQ$T,  /*  ENABLE DMA REQS */

49  1                    DO WHILE  (INPUT  (CMD$37)  AND TC) <>         TC;
                      /* WAIT FOR TC  = 0 */
50  2                    END;

51  1                INIT37L;

                     OUTPUT  (STATUS$2)  =CLEAR,  /*  DISABLE DMA REQS */

                     /* INIT 8237 FOR LISTENER FUNCTIONS */

52  1                OUTPUT  (CLEAR$FF) 0=CLEAR; /*  TOGGLE MASTER RESET */
53  2                OUTPUT  (CMD$37)     =NORM$TIME;
54  1                OUTPUT  (SET$MODE)  =WR$TRANSFER,
55  1                OUTPUT  (SET$MASK)  =CLEAR;
56  1                OUTPUT  (START$0$LO)    =DMA$WRD$LSTN (0);
57  1                DMA$WRD$LSTN (0)     =SHR (DMA$WRD$LSTN (0), 8);
58  1
    OUTPUT       (START $0$HI)      =DMA$WRD$LSTN (0);
59  1                OUTPUT  (0$COUNT$LO)      =TC$LO1;
60  1                OUTPUT  (0$COUNT$HI)      =TC$HI1;

                     /* INIT 8291A FOR LISTENER FUNCTIONS  */

61  1                OUTPUT  (COMMAND$MOD)     =RESET,
62  1                OUTPUT  (ADDR$STATUS)     =MOD1$LO; /*  LISTEN ONLY */
63  1                OUTPUT  (COMMAND$MOD)     =PON,

64  1                DO WHILE (INPUT  (STATUS$1) AND BI)  =0,
65  2                END;  /*  WAIT FOR BI INTR */
66  1                XYZ  = INPUT (PORT$IN);

67  1                OUTPUT  (STATUS$2)  =DMA$REQ$L,  /*  ENABLE DMA REQS  */

68  1                DO WHILE  (INPUT  (STATUS$1)  AND DNE)<>          DNE;
                     /*  WAIT FOR EOI RECEIVED  */
```

```
PL/M-86 COMPILER    BOARD 1

     70   1     CMPBLKS

                /* COMPARE THE TWO BUFFERS CONTENTS */

                MATCH=CMPB  (@BUFF1, @BUFF2, 100);

     71   1     IF MATCH = OKAY THEN GOTO START91;

                /*  SEND ERROR MESSAGE IN BUFFER 3  */

     73   1     DO I=0 TO 16;
     74   2        CALL CO  (BUFF 3  (I)  );
     75   2     END;

     76   1     GOTO START91;

     77   1     END;


MODULE INFORMATION:

     CODE AREA SIZE              =01DBH     475D
     CONSTANT AREA SIZE          =0075H     117D
     VARIABLE AREA SIZE          =0070H     112D
     MAXIMUM STACK SIZE          =0006H       6D
     243 LINES READ
     0 PROGRAM ERROR (S)


END OF PL/M-86  COMPILATION
```

```
PL/M-86 COMPILER    BOARD2

ISIS-II PL/M-86  V1.1 COMPILATION OF MODULE BOARD2
OBJECT MODULE PLACED IN : F1: BRD2, OBJ
COMPILER INVOKED BY:     PLM86  :F1:  BRD2, SRC


                    /* BOARD 2 TPT PROGRAM        */
                    /*                            */
                    /* THIS BOARD LISTENS TO THE OTHER BOARD (1)   */
                    /* AND DMA'S DATA INTO A BUFFER, WHILE WAITING */
                    /* FOR THE END INTERRUPT BIT TO BECOME ACTIVE  */
                    /* UPON END ACTIVE, THE DATA IN THE BUFFER IS  */
                    /* SENT BACK TO THE FIRST BOARD VIA THE GPID   */
                    /* WHEN THE BLOCK IS FINISHED THE 8291A IS  */
                    /* PROGRAMMED BACK INTO THE LISTENER MODE    */

    1          BOARD2

               DO;
                    /* 8237 PORT ADDRESSES */

2   1               DECLARE

                    CLEAR$FF        LITERALLY       'OFFDDH',     /*MASTER CLEAR */
                    START$0$Lo      LITERALLY       'OFFD0H',
                    START$0$HI      LITERALLY       'OFFD0H',
                    0$COUNT$LO      LITERALLY       'OFFD1H',
                    0$COUNT$HI      LITERALLY       'OFFD1H',
                    SET$MODE        LITERALLY       'OFFD8H',
                    CMD$37          LITERALLY       'OFFD8H',
                    SET$MASK        LITERALLY       'OFFDFH',

                    /* 8237 COMMAND  - DATA BYTES  */

3   1               DECLARE

                    RD$TRANSFER     LITERALLY       '48H',
                    WR$TRANSFER     LITERALLY       '44H',
                    ADDR$1A         LITERALLY       '00H',
                    ADDR$1B         LITERALLY       '01H',
                    NORM$TIME       LITERALLY       '20H',
                    TC$LO1          LITERALLY       'OFFH',
                    TC$HI1          LITERALLY       '00H',
                    TC$LO2          LITERALLY       '99D',
                    TC$HI2          LITERALLY       '00H',
                    TC          LITERALLY       '01H',

                    /* 8291A  PORT ADDRESSES */

4   1               DECLARE

                    PORT$OUT        LITERALLY       'OFFC0H',
                    PORT$IN         LITERALLY       'OFFC0H',/* DATA IN */
                    STATUS$1        LITERALLY       'OFFC1H',  /* INTR STAT 1 */
                    STATUS$2        LITERALLY       'OFFC2H',  /* INTR STAT 2 */
                    ADDR$STATUS     LITERALLY       'OFFC4H',  /* ADDR STAT   */
                    COMMAND$MOD     LITERALLY       'OFFC5H',  /* CMD PASS THRU */
```

```
PL/M-86 COMPILER  BOARD2

                    ADDR$0     LITERALLY       'OFFC6H',
                    EOS$REG    LITERALLY       'OFFC7H', /*  EOS REGISTER */

                    /* 8291A  COMMAND - DATA BYTES */

5   1               DECLARE

                    END$EOI    LITERALLY       '88H',
                    DNE  LITERALLY       '10H',
                    PON  LITERALLY       '00H',
                    RESET      LITERALLY       '02H',
                    CLEAR      LITERALLY       '00H',
                    DMA$REQ$L LITERALLY        '10H',
                    DMA$REQ$T LITERALLY        '20H',
                    MOD1$TO    LITERALLY       '80H',
                    MOD1$LO    LITERALLY       '40',
                    EOS  LITERALLY       'ODH',
                    PRESCALER LITERALLY        '23H',
                    HIGH$SPEED LITERALLY       'A4H',
                    XYZ        BYTE,
                    BO         LITERALLY '02H',
                    BI         LITERALLY '01H',
                    ERR        LITERALLY '04H',

6   1           START91;

                    OUTPUT  (STATUS$2)  =CLEAR   /*  END INITILIZATION STATE */

                    /* INIT 8237 FOR LISTENER FUNCTION  */

7   1           INIT37L;

                    OUTPUT  (CLEAR$FF)  =CLEAR; /*  TOGGLE MASTER RESET  */
8   1               OUTPUT  (CMD$37)    =NORM$TIME,
9   1               OUTPUT  (SET$MODE)  =WR$TRANSFER,  /*  BLOCK XFER MODE  */
10  1               OUTPUT  (SET$MASK)  =CLEAR,
11  1               OUTPUT  (START$O$LO)      =ADDR$1A;
12  1               OUTPUT  (START$O$HI)      =ADDR$1B;
13  1               OUTPUT  (O$COUNT$LO)      =TC$LO1;
14  1               OUTPLUT (O$COUNT$HI)      =TC$HI1;

                    /*  INIT 8291A  FOR LISTENER FUNCTIONS   */

15  1               OUTPUT  (COMMAND$MOD)     =RESET;
16  1               OUTPUT  (ADDR$STATUS)     =MOD1$LO,
17  1               OUTPUT  (COMMAND$MOD)     =PON;
18  1               DO WHILE (INPUT  (STATUS$1)  AND BI)  =O;
19  2               END;  /*  WAIT FOR BI INTR */
20  1               XYZ= INPUT  (PORT$IN);
21  1               OUTPUT   (STATUS$2)     =DMA$REQ$L;

                    /* WAIT UNTIL EOI RCVD AND END INTR-BIT SET  */

22  1                  DO WHILE  (INPUT  (STATUS$1)  AND DNE ) <>   DNE;
```

```
PL/M-86  COMPILER  BOARD2

23  1               END;

24  1           INIT37T;
                  /* INIT 8237 FOR TALKER FUNCTION  */

                OUTPUT  (STATUS$2)  =CLEAR;    /* CLEAR  8291A  DRQ  */
25  1           OUTPUT  (CLEAR$FF)  =CLEAR;
26  1           OUTPUT  (CMD$37)    =NORM$TIME;
27  1           OUTPUT  (SET$MODE)  =RD$TRANSFER,   /*  BLOCK XFER MODE */
28  1           OUTPUT  (SET$MASK)  =CLEAR;
29  1           OUTPUT  (START$0$LO)      =ADDR$1A;
30  1           OUTPUT  (START$0$HI)      =ADDR$1B;
31  1           OUTPUT  (0$COUNT$LO)      =TC$LO2;
32  1           OUTPUT  (0$COUNT$HI)      =TC$HI2;

                /*  INIT 8291A  FOR TALKER FUNCTION  */

33  1           OUTPUT  (EOS$REG)     =EOS;
34  1           OUTPUT  (COMMAND$MOD)         =END$EOI; /* EOI ON EOS SENT  */
35  1           OUTPUT  (ADDR$STATUS)         =MOD1$TO; /* TALK ONLY  */
36  1           OUTPUT  (COMMAND$MOD)         =PRESCALER;
37  1           OUTPUT  (COMMAND$MOD)         =HIGH$SPEED;
38  1           OUTPUT  (COMMAND$MOD)         =PON;

39  1           DO WHILE  (INPUT  (STATUS$1)   AND BO)   =0;
40  2           END;     /* WAIT FOR BO INTR  */
41  1           OUTPUT  (PORT$OUT)    =0AAH;

42  1           DO WHILE  (INPUT  (STATUS$1)  AND ERR)   =ERR;
43  2             DO WHILE   (INPUT  (STATUS$1)   AND BO)   =0;
44  3             END;     /* WAIT FOR BO INTR  */
45  2             OUTPUT  (PORT$OUT)  =0AAH;
46  2           END;

47  1           OUTPUT  (STATUS$2)     =DMA$REG$T;
                /*  WAIT FOR TC=0  */

48  1               DO WHILE   (INPUT   (CMD$37)  AND TC) <>  TC;
49  2               END;

50  1           GOTO START91;

51  1    END;


MODULE INFORMATION

    CODE AREA SIZE       =0122H    290D
    CONSTANT AREA SIZE   =0000H      0D
    VARIABLE AREA SIZE   =0001H      1D
    MAXIMUM STACK SIZE   =0000H      0D
    152 LINES READ
    0 PROGRAM ERROR (S)
```

APPENDIX 3
SOFTWARE FOR HP 9835A

```
10    REM SEND IN
TERFACE CLEAR
20    ABORTIO 7
30    REM FORCE E
RRORS UNTIL LIST
ENERS ACTIVE
40 Frcerr:    OUT
PUT 704 USING "#
,K";"B"
50 Chkstst:    ST
ATUS 7;Stat1,Sta
t2,Stat3,Stat4
60    Err=Stat2 A
ND 1
70    IF Err=1 TH
EN GOTO Frcerr
80    PRINT CHR$(
12),"LISTENERS A
RE ON LINE "
90    REM CONFIGU
RE PPOLL
100   PPOLL CONFI
GURE 704;"0000
00"
110

    ! response on
 bit 4
120   PRINT CHR$(
12),"PARALLEL PO
LL CONFIGURED"
130   REM ENABLE
KEYBOARD INTERRU
PT
140   PRINT "COMM
AND = ?      (HIT
 'H' FOR LIST)"
150 Keyen:   ON K
BD GOSUB 610
160   STATUS 7;St
at1,Stat2,Stat3,
Stat4
170   Srq=BINAND(
Stat1,128)
180   IF Srq=0 TH
EN GOTO Keyen
190   OFF KBD
200   PRINT CHR$(
12),"SRQ RECEIVE
D"
210   PRINT "SEND
ING PARALLEL POL
L RESPONSE MESSA
GE"
220   REM EXECUTI
NG PARALLEL POLL
230   Ppollbyte=P
POLL(7)
240   PRINT "PARA
LLEL POLL BYTE =
 ";Ppollbyte
250   PRINT "----
----------------
------"
260   Ppollbyte=B
INAND(Ppollbyte,
8)
270   IF Ppollbyt
e=0 THEN GOTO PS
291
280   PRINT "SRQ
NOT FROM 8291"
281   PRINT "COMM
AND = ?      (HIT
'H' FOR LIST)"
290   GOTO Keyen
300 P8291:   PRIN
T "SRQ IS FROM N
CC 8291 ... THE
ENTERPRISE"
310   PRINT "PERF
```

```
ORMING SERIAL PO
LL TO GET STATUS
 "
320  STATUS 704;
Stat
330  PRINT CHR$(
12),"Status = ";
Stat
340  Dxfer=BINAN
D(Stat,1)
520  IF Dxfer>0
THEN GOTO Rcvr
530  GOTO Keyen
531 Rcvr:  REM R
EADY TO RCV CHAR
S FROM GPIB
540  DIM G$[80]
550  ENTER 704 U
SING "%,T";G$
560  PRINT CHR$(
12),G$
570  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
580  GOTO Keyen
590  REM INTERRU
PT SERVICE ROUTI
NES
600  REM GET KEY
BOARD DATA
610 Whatkey:  DI
M K$[80]
620  K$=KBD$
630  IF K$="G" T
HEN GOTO Get
640  IF K$="D" T
HEN GOTO Dec
650  IF K$="R" T
HEN GOTO Rem
660  IF K$="H" T
HEN GOTO Help
670  IF K$="X" T
HEN GOTO Xmit
680 Get:  TRIGGE
R 704
690  PRINT CHR$(
12),"GROUP EXECU
TE TRIGGER SENT"
700  PRINT " "
710  PRINT "COMM
AND = ?     (HIT
'H' FOR LIST)"
720  RETURN
730 Dec:  RESET
704
740  PRINT CHR$(
12),"SELECTIVE D
EVICE CLEAR SENT
"
750  PRINT " "
760  PRINT "COMM
AND = ?     (HIT
'H' FOR LIST)"
770  RETURN
780 Rem:  LOCAL
704
790  PRINT CHR$(
12),"REMOTE MESS
AGE SENT"
800  PRINT " "
810  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
820  RETURN
830 Help:  PRINT
 CHR$(12)
840  PRINT " @@@
@ OPERATOR ALLOW
ABLE COMMANDS @@
@@ "
850  PRINT " hit
 key   result"
860  PRINT "   G
      Send GET m
essage"
870  PRINT "   D
      Send DEC m
essage"
```

```
880  PRINT "   R
     Send REM/L
OC message"
890  PRINT "   X
     Xmits keyb
oard input to 82
91"
900  PRINT "   H
     Prints thi
s table"
910  PRINT " "
920  PRINT "...
so ahead, TRY IT
 !"
930  RETURN
940 Xmit:  DIM A
$[80]
950  PRINT CHR$(
12),"Enter data
to send and hit
CONTINUE"
960  INPUT A$
970  OUTPUT 704;
A$
971  EOI 7;0
980  PRINT "COMM
AND = ?     (HIT
 'H' FOR LIST)"
990  RETURN
1000 END
```

APPENDIX 4
SOFTWARE FOR HP 8088/HP 9835A VIA GPIB

```
PL/M-86 COMPILER     HPIB


ISIS-II PL/M-86 V1.1 COMPILATION OF MODULE HPIB
OBJECT MODULE PLACED IN :F1:HPIB.OBJ
COMPILER INVOKED BY:  PLM86 :F1:HPIB.SRC LARGE


   1            HPIB:
                /*

                PARAMETER DECLARATIONS
                */

                DO;

   2    1       DECLARE

                ADDR$HI         LITERALLY    '01H',
                ADDR$LO         LITERALLY    '00H',
                ADSC            LITERALLY    '01H',
                BI          LITERALLY    '01H',
                BO          LITERALLY    '02H',
                CHAR$COUNT   BYTE,
                CHAR            BYTE,
                CHARS(80)    BYTE,
                CLEAR           LITERALLY    '00H',
                CPT         LITERALLY    '80H',
                CRLF            LITERALLY    '0AH',
                DEC         LITERALLY    '08H',
                DMA$ADR$LSTN     POINTER,
                DMA$ADR$TALK     POINTER,
                DMA$WRD$LSTN(2) WORD AT      (@DMA$ADR$LSTN),
                DMA$WRD$TALK(2) WORD AT      (@DMA$ADR$TALK),
                DMA$REG$L    LITERALLY    '10H',
                DMA$REG$T    LITERALLY    '20H',
                DNE         LITERALLY    '10H',
                END$EOI         LITERALLY    '88H',
                EOS         LITERALLY    '0DH',
                ERR         LITERALLY    '04H',
                GET         LITERALLY    '20H',
                I           BYTE,
                LISTEN          LITERALLY    '04H',
                MLA         LITERALLY    '04H',
                MODE$1          LITERALLY    '01H',
                NO$DMA          LITERALLY    '00H',
                NO$RSV          LITERALLY    '00H',
                NORM$TIME    LITERALLY    '20H',
                PON         LITERALLY    '00H',
                PPC         LITERALLY    '05H',
                PPE$MASK        LITERALLY    '60H',
                PPOLL$CNFG$FLAG LITERALLY    '01H',
                PPOLL$EN$BYTE    BYTE,
                PRI$BUF(80) BYTE AT      (@CHARS),
                RD$XFER         LITERALLY    '48H',
                RESET           LITERALLY    '02H',
                REMC            LITERALLY    '02H',
                RSV         LITERALLY    '40H',
                RXRDY           LITERALLY    '02H',
```

```
PL/M-86 COMPILER    HPIB

                SRQS            LITERALLY   '40H',
                STAT1           BYTE,
                STAT2           BYTE,
                TALK            LITERALLY   '02H',
                TA$OR$LA        BYTE,
                TRG         LITERALLY   '41H',
                TC          LITERALLY   '01H',
                TC$HI           LITERALLY   '00H',
                TC$LO           LITERALLY   'OFFH',
                TXRDY           LITERALLY   '01H',
                UDC         BYTE,
                WR$XFER         LITERALLY   '44H',
                XYZ         BYTE;

                /*

                PORT DECLARATIONS

                */

  3   1         DECLARE

                ADDR$0          LITERALLY   'OFFC6H',
                ADDR$STATUS     LITERALLY   'OFFC4H',
                CLEAR$FF        LITERALLY   'OFFDDH',
                CMD$37          LITERALLY   'OFFD8H',
                COMMAND$MOD     LITERALLY   'OFFC5H',
                COUNT$HI        LITERALLY   'OFFD1H',
                COUNT$LO        LITERALLY   'OFFD1H',
                CPT$REG         LITERALLY   'OFFC5H',
                EOS$REG         LITERALLY   'OFFC7H',
                PORT$IN         LITERALLY   'OFFC0H',
                PORT$OUT        LITERALLY   'OFFC0H',
                SER$DATA        LITERALLY   'OFFF0H',
                SER$STAT        LITERALLY   'OFFF2H',
                SET$MASK        LITERALLY   'OFFDFH',
                SET$MODE        LITERALLY   'OFFDBH',
                SPOLL$STAT      LITERALLY   'OFFC3H',
                START$HI        LITERALLY   'OFFD0H',
                START$LO        LITERALLY   'OFFD0H',
                STATUS$1        LITERALLY   'OFFC1H',
                STATUS$2        LITERALLY   'OFFC2H';

                /* crt  messages list */

  4   1         DECLARE GET$MSG(11) BYTE DATA (0DH,0AH,'TRIGGER',0AH,0DH);
  5   1         DECLARE DEC$MSG(16) BYTE DATA (0DH,0AH,'DEVICE CLEAR',0AH,0DH);
  6   1         DECLARE REMC$MSG(10) BYTE DATA (0DH,0AH,'REMOTE',0DH,0AH);
  7   1         DECLARE CPT$MSG(22) BYTE DATA (0DH,0AH,'UNDEF CMD RECEIVED',0AH,0DH);
  8   1         DECLARE HUH$MSG(11) BYTE DATA (0DH,0AH,'HUH ???',0DH,0AH);


                    /* called procedures */

  9   1         REQSER:     PROCEDURE;
```

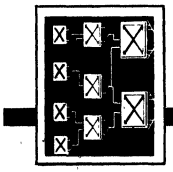
```
PL/M-86 COMPILER     HPIB

 10   2              OUTPUT (SPOLL$STAT)=TRG;

 11   2              DO WHILE (INPUT (SPOLL$STAT) AND SRQS)=SRQS;
 12   3              END;

 13   2              OUTPUT (SPOLL$STAT)=NO$RSV;

 14   2              END REQSER;

 15   1      CO: PROCEDURE(XXX);
 16   2              DECLARE
                      XXX         BYTE;

 17   2              DO WHILE (INPUT (SER$STAT) AND TXRDY)<>TXRDY;
 18   3              END;
 19   2              OUTPUT (SER$DATA)=XXX;
 20   2          END CO;
 21   1      HUH:        PROCEDURE;
 22   2                  DO I=0 TO 10;
 23   3                   CALL CO (HUH$MSG(I));
 24   3                  END;
 25   2              END HUH;

 26   1      CI:     PROCEDURE;

 27   2              IF (INPUT (SER$STAT) AND RXRDY)=RXRDY THEN
 28   2                  DO;
 29   3                      I=0;
 30   3                      CHAR$COUNT=0;
 31   3      STORE$CHAR:         CHAR=(INPUT (SER$DATA) AND 7FH);
 32   3                      CHAR$COUNT=CHAR$COUNT+1;
 33   3                      CALL CO (CHAR);
 34   3                      CHARS(I)=CHAR;
 35   3                      I=I+1;
 36   3                      IF CHAR <> CRLF THEN
 37   3                          DO;
 38   4                           DO WHILE (INPUT (SER$STAT) AND RXRDY) <>RXRDY;
 39   5                           END;
 40   4                           GOTO STORE$CHAR;
 41   4                          END;
 42   3                      CALL REQSER;
 43   3                  END;
 44   2              END CI;

 45   1      TALK$EXEC:  PROCEDURE;

 46   2              OUTPUT (STATUS$2)=CLEAR;

                      /*
                      manipulate address bits for DMA controller
                      */

 47   2              DMA$ADR$TALK=(@CHARS);
 48   2              DMA$WRD$TALK(1)=SHL(DMA$WRD$TALK(1),4);
 49   2              DMA$WRD$TALK(0)=DMA$WRD$TALK(0)+DMA$WRD$TALK(1);

 50   2              OUTPUT (CLEAR$FF)=CLEAR;
```

```
PL/M-86 COMPILER     HPIB

  51   2              OUTPUT (CMD37)=NORM$TIME;
  52   2              OUTPUT (SET$MODE)=RD$XFER;
  53   2              OUTPUT (SET$MASK)=CLEAR;
  54   2              OUTPUT (START$LO)=DMA$WRD$TALK(0);
  55   2              DMA$WRD$TALK(0)=SHR(DMA$WRD$TALK(0),8);
  56   2              OUTPUT (START$HI)=DMA$WRD$TALK(0);
  57   2              OUTPUT (COUNT$LO)=CHAR$COUNT;
  58   2              OUTPUT (COUNT$HI)=0;


  59   2              OUTPUT (EOS$REG)=EOS;
  60   2              OUTPUT (COMMAND$MOD)=END$EOI;

  61   2              DO WHILE (INPUT (STATUS$1) AND B0)=0;
  62   3              END;
  63   2              OUTPUT (PORT$OUT)=0AAH;

  64   2              DO WHILE (INPUT (STATUS$1) AND ERR)=ERR;
  65   3                  DO WHILE (INPUT (STATUS$1) AND B0)=0;
  66   4                  END;
  67   3                  OUTPUT (PORT$OUT)=0AAH;
  68   3              END;
  69   2              OUTPUT (STATUS$2)=DMA$REG$T;

  70   2              END TALK$EXEC;

  71   1      LISTEN$EXEC:    PROCEDURE;

  72   2              OUTPUT (STATUS$2)=CLEAR;
  73   2              OUTPUT (CLEAR$FF)=CLEAR;
  74   2              OUTPUT (CMD$37)=NORM$TIME;
  75   2              OUTPUT (SET$MODE)=WR$XFER;
  76   2              OUTPUT (SET$MASK)=CLEAR;
  77   2              DMA$ADR$LSTN=(@CHARS);
  78   2              DMA$WRD$LSTN(1)=SHL(DMA$WRD$LSTN(1),4);
  79   2              DMA$WRD$LSTN(0)=DMA$WRD$LSTN(0)+DMA$WRD$LSTN(1);
  80   2              OUTPUT (START$LO)=DMA$WRD$LSTN(0);
  81   2              DMA$WRD$LSTN(0)=SHR(DMA$WRD$LSTN(0),8);
  82   2              OUTPUT (START$HI)=DMA$WRD$LSTN(0);
  83   2              OUTPUT (COUNT$LO)=TC$LO;
  84   2              OUTPUT (COUNT$HI)=TC$HI;
  85   2              OUTPUT (STATUS$2)=DMA$REG$L;

  86   2              END LISTEN$EXEC;

  87   1      PRINTER:    PROCEDURE;

  88   2              I=0;

  89   2              DO WHILE PRI$BUF(I) <>CRLF;
  90   3                  CALL CO (PRI$BUF(I));
  91   3                  I=I+1;
  92   3              END;
  93   2              CALL CO (PRI$BUF(I));

  94   2              END PRINTER;
```

```
PL/M-86 COMPILER    HPIB

 95   1      ADSC$EXEC:  PROCEDURE;

 96   2              TA$OR$LA=INPUT (ADDR$STATUS);

 97   2              IF (TA$OR$LA AND TALK)=TALK THEN
 98   2                  CALL TALK$EXEC;
 99   2              IF (TA$OR$LA AND LISTEN)=LISTEN THEN
100   2                  CALL LISTEN$EXEC;

101   2              END ADSC$EXEC;

102   1      GET$EXEC:   PROCEDURE;
103   2                  DO I=0 TO 10;
104   3                   CALL CO (GET$MSG(I));
105   3                  END;
106   2              END GET$EXEC;

107   1      DEC$EXEC:   PROCEDURE;
108   2                  DO I=0 TO 15;
109   3                   CALL CO (DEC$MSG(I));
110   3                  END;
111   2              END DEC$EXEC;

112   1      REMC$EXEC:  PROCEDURE;
113   2                  DO I=0 TO 9;
114   3                   CALL CO (REMC$MSG(I));
115   3                  END;
116   2              END REMC$EXEC;

117   1      PPOLL$CON:  PROCEDURE;

118   2              OUTPUT (COMMAND$MOD)=PPOLL$CNFG$FLAG;

119   2              END PPOLL$CON;

120   1      PPOLL$EN:   PROCEDURE;

121   2              PPOLL$EN$BYTE=(UDC AND 6FH);
122   2              OUTPUT (COMMAND$MOD)=PPOLL$EN$BYTE;

123   2              END PPOLL$EN;

124   1      CPT$EXEC:   PROCEDURE;
125   2                  DO I=0 TO 21;
126   3                   CALL CO (CPT$MSG(I));
127   3                  END;

128   2                  UDC=INPUT (CPT$REG);
129   2                  UDC=(UDC AND 7FH);
130   2                  IF (UDC AND PPC)=PPC THEN
131   2                   CALL PPOLL$CON;

132   2                  IF (UDC AND PPE$MASK)=PPE$MASK THEN
133   2                   CALL PPOLL$EN;
```

```
PL/M-86 COMPILER    HPIB


 134   2                END CPT$EXEC;
                  /*
                  BEGIN CODE
                  */

 135   1          INIT:

                      OUTPUT  (CLEAR$FF)   =CLEAR;
 136   1                OUTPUT  (COMMAND$MOD)    =RESET;
 137   1                OUTPUT  (ADDR$STATUS)    =MODE$1;
 138   1                OUTPUT  (ADDR$0)     =MLA;
 139   1                OUTPUT  (STATUS$2)   =NO$DMA;
 140   1                OUTPUT  (COMMAND$MOD)    =PON;

 141   1          LISTENERS:

                      /* response to listeners check */


                      DO WHILE (INPUT (STATUS$1) AND BI)=0;
 142   2                END;

 143   1                XYZ=INPUT (PORT$IN);
 144   1                XYZ=INPUT (STATUS$2);

 145   1          CMD:

                  RDSTAT:
                      /* read status registers and interpret command */

                      STAT1=INPUT (STATUS$1);
 146   1                STAT2=INPUT (STATUS$2);

 147   1                IF (STAT1 AND DNE)=DNE THEN
 148   1                   CALL PRINTER;
 149   1                IF (STAT1 AND CPT)=CPT THEN
 150   1                    DO;
 151   2                    CALL CPT$EXEC;
 152   2                    STAT2=(STAT2 AND OFEH);
 153   2                    END;
 154   1                IF (STAT1 AND GET)=GET THEN
 155   1                    DO;
 156   2                    CALL GET$EXEC;
 157   2                    STAT2=(STAT2 AND OFEH);
 158   2                    END;
 159   1                IF (STAT1 AND DEC)=DEC THEN
 160   1                    DO;
 161   2                    CALL DEC$EXEC;
 162   2                    STAT2=(STAT2 AND OFEH);
 163   2                    END;
 164   1                IF (STAT2 AND REMC)=REMC THEN
 165   1                    DO;
 166   2                    CALL REMC$EXEC;
 167   2                    STAT2=(STAT2 AND OFEH);
 168   2                    END;
 169   1                IF (STAT2 AND ADSC)=ADSC THEN
```

```
PL/M-86 COMPILER     HPIB


 170   1                DO;
 171   2                CALL ADSC$EXEC;
 172   2                STAT2=(STAT2 AND OFEH);
 173   2                END;

 174   1            CALL CI;

 175   1            GOTO CMD;



 176   1        END;


MODULE INFORMATION:

     CODE AREA SIZE      = 0475H   1141D
     CONSTANT AREA SIZE  = 0000H      0D
     VARIABLE AREA SIZE  = 0061H     97D
     MAXIMUM STACK SIZE  = 000AH     10D
     349 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-86 COMPILATION
```



LSI TRANSCEIVER CHIPS COMPLETE GPIB INTERFACE

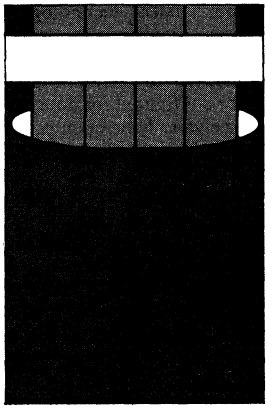
A GPIB interface meeting IEEE 488 standards can be built with only three or four chips!

by Pradip Madan and Jim Frederick

The decision to support the IEEE 488 standard with integrated circuits was based on the potential popularity of the interface standard and its applications potential. Although a serial interface supports many system throughput requirements, a parallel interface over short distances can provide much higher data transfer rates, yet remain economical despite the extra interconnection copper required.

The IEEE 488 standard is for a parallel interface designed to operate over a limited distance. Its general purpose nature makes the general purpose interface bus (GPIB) attractive for a variety of systems, and also allows manufacturers to design their equipment interfaces to a common standard. As a result, users can mix equipment from different manufacturers without having to adapt the interfaces for compatibility. To date the GPIB has been incorporated in computer peripherals, such as printers, but the most applications have been in programmable instrumentation systems. Other GPIB applications include camera control in computer controlled studios, electronic surveillance, peripheral control, modular add-ons to photocopiers, and so forth.

Integration benefit



Shortly after the IEEE committee had put the final touches on its standard specifications, engineers began building GPIB interface subsystems. Because the standard had just been defined, there were no large scale integration (LSI) chips available. Therefore, the first GPIB implementations were board level designs replete with small scale integration (SSI) and medium scale integration (MSI) logic chips. A typical effort included four or five rows of ten chips each.

With the advent of integrated circuit GPIB chips, chip counts dropped dramatically, reliability improved, and space requirements shrank. Consequently, the price range of systems for which GPIB had become practical began to decrease. A fully functional GPIB subsystem can now be constructed with less than one-tenth the number of chips formerly required. In fact, the complete

Pradip Madan is the product manager for microprocessor peripheral components at Intel Corp, 2625 Walsh Ave, Santa Clara, CA 95051, where he has been employed for 2 years. He has a BSEE, an MS in computer science, and an MBA in finance.

Jim Frederick is a microcomputer design engineer at Intel Corp. Since joining the company in 1974, he has been involved in several different projects. Mr Frederick has studied at the College of San Mateo, and the University of Santa Clara.

talker/listener/controller mode logic resides in four LSI chips: one Intel 8291A talker/listener, one 8292 controller, and two 8293 transceivers. All these LSI, including the transceiver, are implemented in metal oxide semiconductor (MOS) technology.

Unlike the controller or talker/listener functions which could be integrated routinely in N-channel MOS (NMOS) technology, the transceiver posed special problems in MOS integration.

The chip's size includes a 7-mil ground line and two ground pads in order to handle the 432-mA current.

The standard calls for the transceiver circuitry to be able to drive each of the 16 bus lines with a nominal 48 mA of current. In addition, it specifies a minimum required input hysteresis and places a limit on propagation delays. Driving relatively high currents quickly was not a familiar province of MOS technology. Certainly the garden variety NMOS lacked the necessary speed-power product to handle the task.

However, progress in NMOS technology has produced the high speed, densely integrated high performance MOS (HMOS) technology which has the necessary characteristics to meet the current drive and propagation speed requisites.

Designing the 8293 transceiver

Although the 48-mA drive required by the 16 GPIB lines had only been implemented with bipolar technology before, HMOS technology—with its reduced gate lengths, smaller size, and lower parasitic capacitance—looked like it could handle the job. Architecturally, the 8293 contains nine transceiver circuits which can be configured for data or interface management line transceivers. Nine open collector or 3-state line drivers that could sink 48 mA, in addition to twelve Schmitt-type line receivers, were used to implement the nine transceivers. Fig 1 is a schematic representation of one of these 3-state drivers.

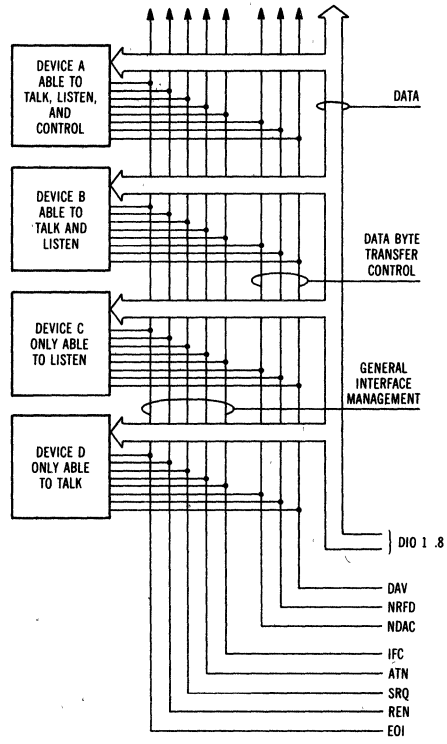
Additional logic was added for decoding the transmit/receive mode control of each of the transceivers. The 8293 was conceived as operating in four distinct modes: talker/listener control transceiver, talker/listener/controller control transceiver, talker/listener data transceiver, and/or talker/listener/controller data transceiver. Thus, a 2-pin select scheme allows a user to select the desired operating mode.

Choosing appropriate active devices

All of the 8293's functional elements required only four different types of active field effect transistors (FETs). Low threshold enhancement type devices show good high output voltage characteristics, and were used as output pullup devices in push-pull 3-state drivers. Enhancement type FETs were also used for fast switching and low leakage: depletion type devices were used for resistive pullup in buffering. Depletion type FETs also played an important role in meeting the hysteresis specifications of the IEEE 488 standard. Finally, higher threshold depletion type devices were used to prevent the bus lines from being disturbed on power-up and power-down.

A conventional MOS transistor capable of supplying 48 mA at 0.5 V would have been physically too large. HMOS technology, however, permits such a device to be fabricated in an area of less than 150 mil^2 (97 mm^2). Furthermore, the low speed/power product of HMOS allowed a multi-stage design so that, like transistor-transistor logic (TTL) circuitry, natural hysteresis could be built into the receivers.

IEEE 488 interface standard



The IEEE 488 interface standard specifies an 8-bit parallel, bidirectional data bus with eight additional lines for data-byte transfer control and general interface management. The three data-byte transfer lines are data valid (DAV), not ready for data (NRFD), and not data accepted (NDAC). States of these three lines determine when data on the 8-bit data bus are valid, ready to be received, and received, respectively. General interface mangement lines are interface clear (IFC), attention (ATN), service request (SRQ), remote enable (REN), and end or identify (EOI). These lines are used to clear the bus and establish control, initiate polling, pass control from a controller to another controller or the front panel, and indicate the end of a transfer sequence.

Special layout techniques

The transceiver was implemented using new layout techniques aimed at reducing the series resistance in the polysilicon gate structures of the large transistors, and routing ac signal paths over metal interconnects in order to reduce capacitance and series resistance. Chip size, 188 x 156 mils (5 x 4 mm), includes a 7-mil (0.2-mm) ground line and two ground pads in order to handle the 432-mA current generated when all drivers are on. Power consumption is 300 mW, typically, with driver or receiver speeds of 20 ns under light loads and speeds of 85 ns under the maximum load of 4500 pF.

Signaling a new trend?

Until the advent of the 8293, complex MOS chips relied on bipolar drivers to handle the heavy bus loading found in complex systems. The 8293 could point the way to future microprocessors and controllers that include their own MOS drivers. Such a scheme would significantly reduce the time lost by going through external buffers. It would also provide all the other benefits of system integration.

The 8293 is essentially a non inverting buffer chip capable of driving high currents. The 8291A talker/listener chip and 8292 GPIB controller chip are designed to interface with the 8080, 8085, iAPX 86, iAPX 88, and 8048/8051 microprocessors and single-chip microcomputers. However, the 8291A and 8292 cannot electrically drive a standard IEEE 488 bus by themselves. Thus, the 8293 was designed to interface between the GPIB and a single 8291A or a combination of the 8291A and 8292. (See Fig 2.)

The chip is divided into nine distinct transceivers. Each one's characteristics, such as 3-state or open-collector outputs, and transmit or receive modes of operation, are determined by internal logic control. (See Fig 3.) Thus, in mode 0 talker/listener control configuration the attention (ATN) transceiver is forced into an input-only mode with respect to the bus's ATN line. The end or identify ($\overline{EOI}$) transceiver, on the other hand, is either a transmitter or receiver depending on the state of the transmit/receive (T/$\overline{R}$2) line. Its interface to the GPIB is 3-state because of the fixed 5 V logic on the $\overline{EOI}$ transceiver's output control. In mode 1, the talker/listener data configuration, the 8293 is a true transceiver with its operations mode controlled by the state of the T/$\overline{R}$1 line and its output characteristics (3-state or open-collector) determined by the states of the $\overline{ATN}$ and $\overline{EOI}$ lines. (See Fig 3.)

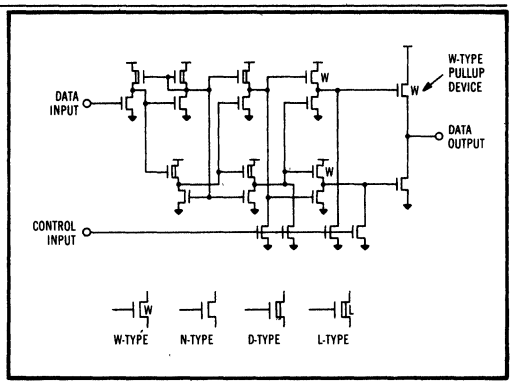


Fig 1 3-state driver schematic. Nine such open collector drivers are used in the interface.

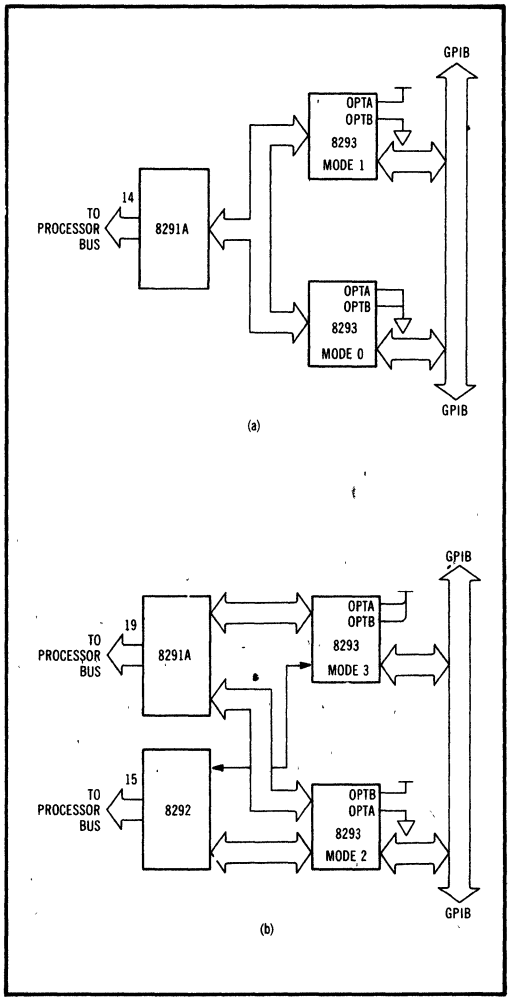


Fig 2 8239 is designed for use in talker/listener implementation (a), or for talker/listener/controller interface (b).

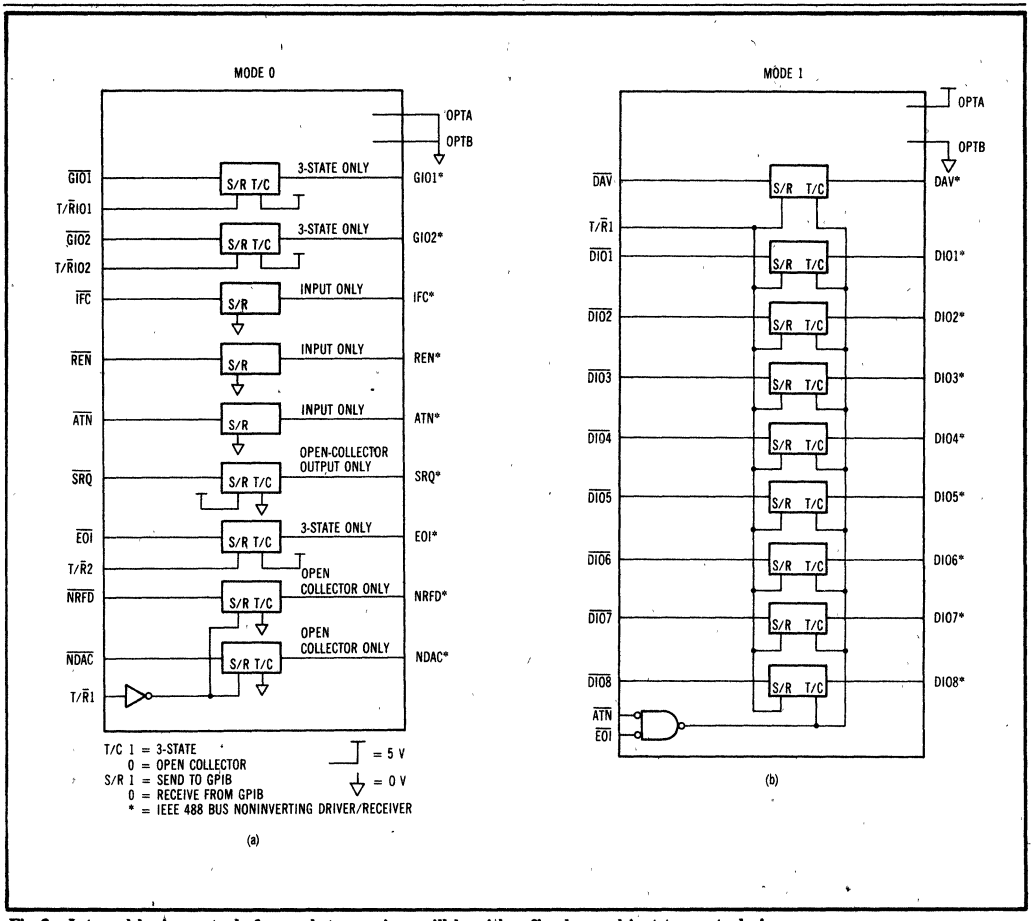


Fig 3 Internal logic controls for each transceiver will be either fixed or subject to control via external logic. In mode 0, chip is set up for control, thus some transceivers are fixed in transmit or receive mode only. In mode 1, chip is configured as true transceiver—all nine transceivers can transmit or receive depending on state of T/R̄1 pin. In (a) is talker/listener control configuration, and in (b), talker/listener data configuration.

The talker/listener/controller control configuration, mode 2, is a full transceiver mode but the operation mode of the transceivers is determined by more complex combinational logic. (See Fig 4.) The fourth mode (mode 3), which is the talker/listener/controller data configuration, is again a true transceiver whose mode of operation is controlled by the state of the T/R̄1 line. In this mode, some additional interval combinational logic is enabled to permit the 8293 to support the 8292 in taking bus control synchronously.

...complete talker/listener/controller mode logic resides in four LSI chips.

The 8293's overall mode (mode 0, 1, 2, or 3) is determined by the state on the option pins 26 and 27. For example, if both pins are tied low (0 V), the chip is in mode 0. If both are high (5 V) it is in mode 3. The particular state of these pins will determine the characteristics of the other 26 pins. (See the Table, "8293 Mode Selection Pin Mapping.")

Talker/listener only

If the IEEE 488 is to be implemented in a system that is able to talk and listen (eg, a digital multimeter), only talk (eg, a counter), or only listen (eg, a signal generator),

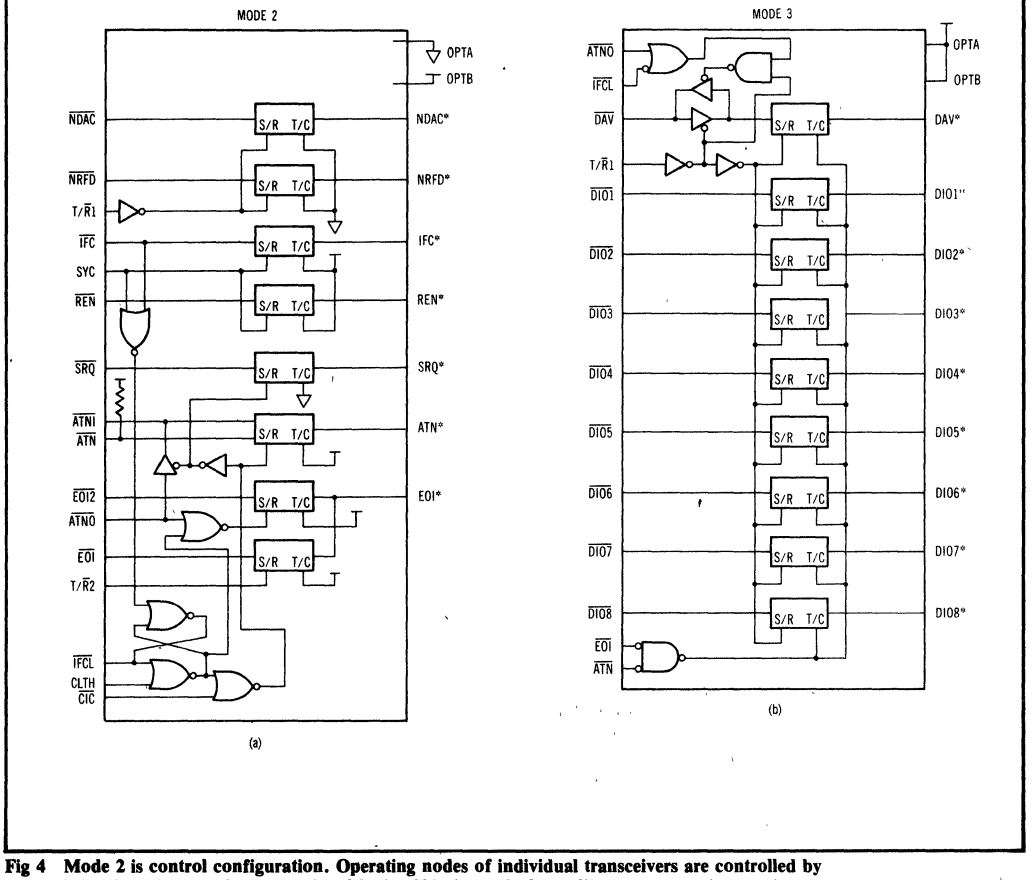


Fig 4 Mode 2 is control configuration. Operating nodes of individual transceivers are controlled by external signals and internal combinational logic. Chip in mode 3 acts like true transceiver, as in mode 1, except some extra functions have been included in order to support controller function. In (a), talker/listener/controller configuration is for control, and in (b), for data.

then the entire interface can be built with a single 8291A and a pair of 8293s. (See Fig 5.) In this configuration, one 8293 handles the eight data lines DIO1 to DIO8 and the other handles the data-byte transfer handshake lines and general interface management lines. Both transceivers are connected to the 8291A's $\overline{ATN}$, and $\overline{EOI}$, and T/$\overline{R1}$ lines.

Talker/listener/controller

For an IEEE 488 controller (like the HP 85 or Tektronix 4051), the system must be able to take control of the bus or delegate it to another controller. Such an interface scheme can be implemented using an 8291A, an 8292, and a pair of 8293s. (See Fig 6.) The arrangement is similar to that of a talker/listener interface; one 8293 handles the DIO1 through DIO8 bus data lines and the other handles the data byte transfer handshake and general interface management lines. The difference is that pins 26 and 27 have been selected for modes 2 and 3 and several additional control functions have been added. The attention in (ATNI) lines and attention out (ATNO) lines permit the 8292 to monitor the GPIB's ATN line and take control of the bus. In conjunction with the ATN line, the EOI2 line is used by the 8292 to initiate a polling sequence.

The chip is divided into nine distinct transceivers and each one's characteristics are determined by internal logic.

Lastly, the system controller line (SYC) enables the control function. If it is low, the 8292 is prevented from acting as a controller. If it is switched high, the 8292 can act as a controller. In essence, the SYC controls the direction of the interface clear (IFC) and remote enable (REN) signals.

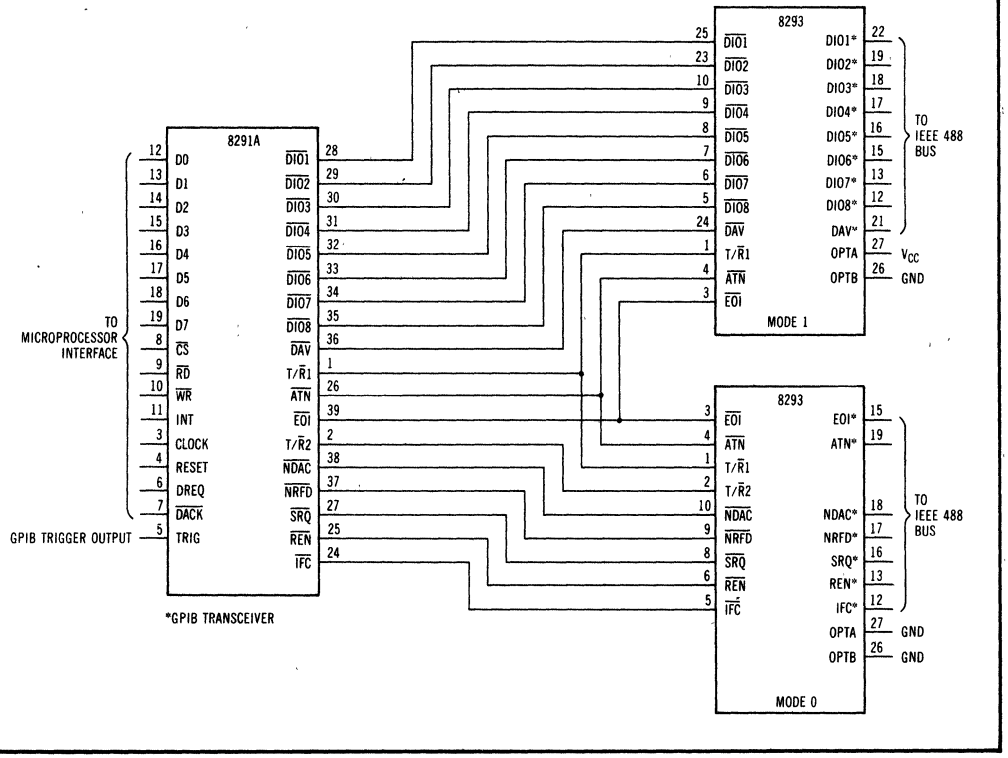
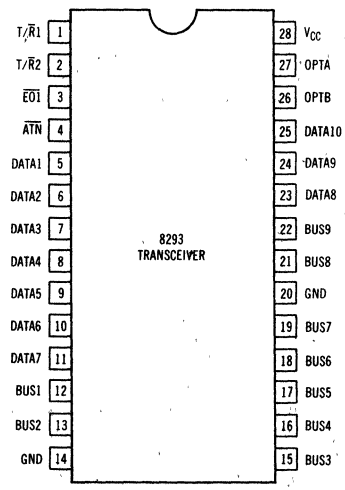


Fig 5 Talker/listener only implementation can be built using just three chips—single 8291A and a pair of 8293s. First (upper) transceiver chip is used for bidirectional data flow on DIO1 to DIO8 data lines. Lower 8293 handles some of data byte transfer control lines and general interface management lines.

8293 MODE SELECTION PIN MAPPING

PIN NAME	PIN NO	IEEE IMPLEMENTATION NAME			
		MODE 0	MODE 1	MODE 2	MODE 3
OPTA	27	0	1	0	1
OPTB	26	0	0	1	1
DATA1	5	IFC	DIO8	IFC	DIO8
BUS1	12	IFC*	DIO8*	IFC*	DIO8*
DATA2	6	REN	DIO7	REN	DIO7
BUS2	13	REN*	DIO7*	REN*	DIO7*
DATA3	7	NC	DIO6	EOI2	DIO6
BUS3	15	EOI*	DIO6*	EOI*	DIO6*
DATA4	8	SRQ	DIO5	SRQ	DIO5
BUS4	16	SRQ*	DIO5*	SRQ*	DIO5*
DATA5	9	NRFD	DIO4	NRFD	DIO4
BUS5	17	NRFD*	DIO4*	NRFD*	DIO4*
DATA6	10	NDAC	DIO3	NDAC	DIO3
BUS6	18	NDAC*	DIO3*	NDAC*	DIO3*
DATA7	11	T/R1O1	NC	ATNI	ATNO
DATA8	23	T/R1O2	DIO2	ATNO	DIO2
BUS7	19	ATN*	DIO2*	ATN*	DIO2*
DATA9	24	GIO1	DAV	CIC	DAV
BUS8	21	GIO1*	DAV*	CLTH	DAV*
DATA10	25	GIO2	DIO1	IFCL	DIO1
BUS9	22	GIO2*	DIO1*	SYC	DIO1*
T/R1	1	T/R1	T/R1	T/R1	T/R1
T/R2	2	T/R2	NC	T/R2	IFCL
EOI	3	EOI	EOI	EOI	EOI
ATN	4	ATN	ATN	ATN	ATN

*These pins are the IEEE 488 bus noninverting driver/receivers. They include all the bus terminations required by the standard, and connect directly to the GPIB connector



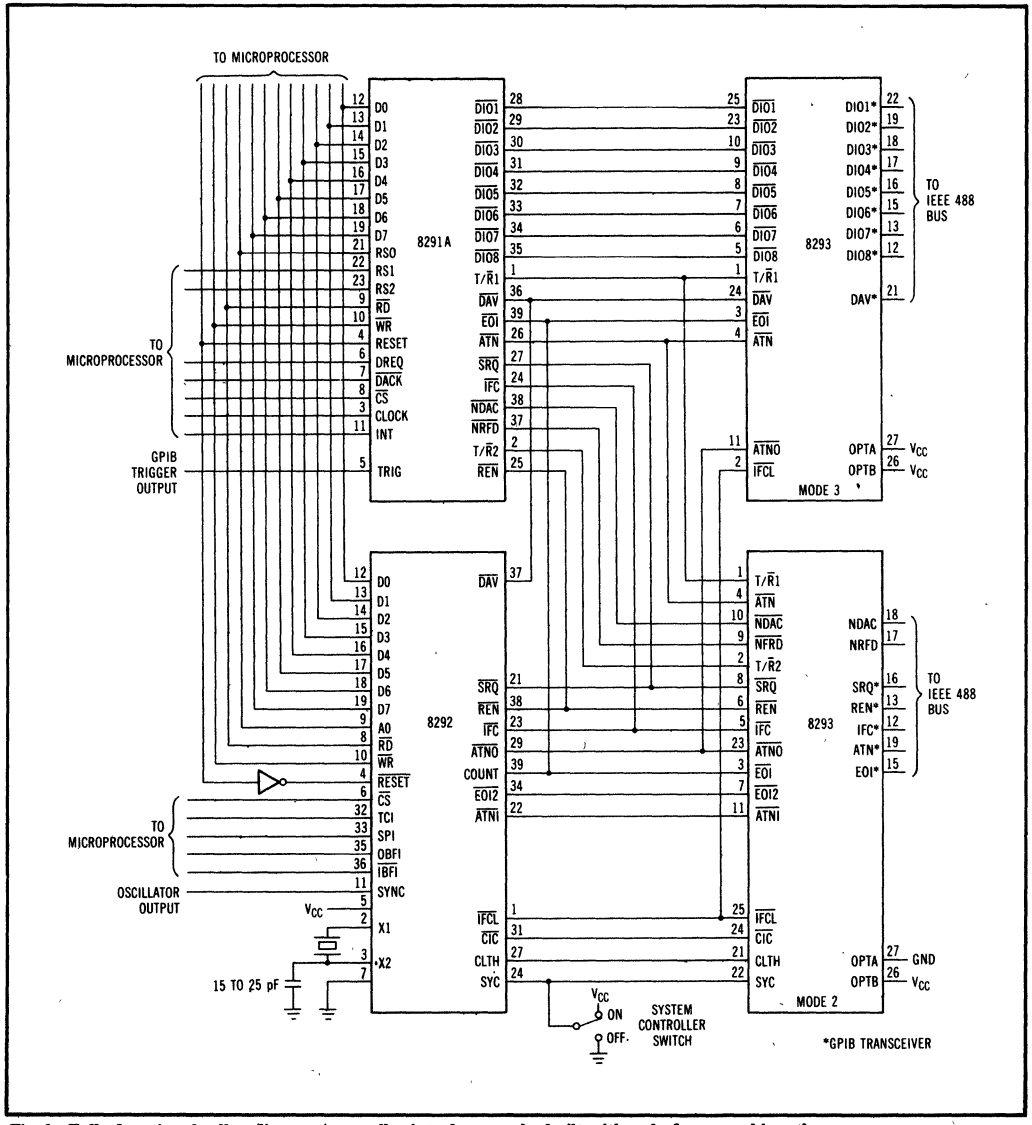


Fig 6 Fully functional talker/listener/controller interface can be built with only four LSI chips; the 8291A, 8292, and a pair of 8293s. Like simpler talker/listener only case, one 8293 handles data transceiver functions while other handles data byte transfer control and general interface management. There are additional control lines enabled which support the controller (8292) activity.

Summary

Before the advent of integrated solutions for IEEE 488 implementation, it usually took forty to fifty SSI and MSI chips to build this interface. A large portion of those were eliminated by controllers and interface chips like the 8291A and 8292. Now, with the last part of the interface available in LSI, a fully functional interface can be built using only four LSI chips. The cost of the original design was typically $400 to $500. A set of the three chips, the 8291A, and two 8293s (for a talker/listener function) allows a 15-fold reduction in cost. The power dissipation of a 40-chip interface was in the vicinity of 10 W. The power dissipation of the 4-chip approach is a mere 1.5 W. The size of the PC board is considerably smaller, too, and that lowers the manufacturing costs and improves reliability.

Please rate the value of this article to you by circling the appropriate number in the "Editorial Score Box" on the Inquiry Card.

High 704 *Average 705* *Low 706*

intel®

ARTICLE REPRINT

AR-113

January 1980

LSI Chips Ease Standard 488 Bus Interfacing

Ronald M. Williams
Intel Corporation, Santa Clara, California

Reprinted with permission of *Computer Design Magazine*, October 1979 issue.

LSI CHIPS EASE STANDARD 488 BUS INTERFACING

Time and cost disadvantages of interfacing to the IEEE Std 488 bus are overcome with a dedicated LSI chip set that incorporates most of its functional and electrical specifications

Ronald M. Williams Intel Corporation, Santa Clara, California

Historically, interface techniques proliferated as designers evolved customized links among instruments, controllers, and processors for realtime test measurements or data communications, resulting in excessive and expensive codes, formats, signal levels, and timing factors. Obviously, interface standardization was mandatory to save design costs for engineers, development costs for manufacturers, and system integration costs for users. Thus, IEEE Standard 488-1978 (a revision of ANSI/IEEE Std 488-1975) offers a universal instrumentation system approach to automatic operating measurement configurations that provides compatibility, versatility, and flexibility. This system approach establishes a suitable standard bus for interfacing programmable devices from different manufacturers. Outstanding advantages of the standard bus include byte serial, bit parallel digital data handling, synchronized communication among devices at varying data rates, and hardware interchangeability and interconnection in daisy-chained fashion. However, some restrictive disadvantages that have hindered implementation are highly complex logic protocol, time consuming design analysis, and lack of low cost components to perform the intricate logic control functions. To overcome these drawbacks, a large scale integrated (LSI) chip set has been designed with built-in IEEE Std 488 logic controls. Thus,

interfacing has been significantly simplified for properly connecting processor buses and programming system protocols.

Interface Overview

The IEEE Standard 488-1978 bus interface includes electrical, mechanical, and functional specifications* for interconnecting both programmable and nonprogrammable electronic measuring apparatus with other apparatus and accessories necessary to assemble instrumentation systems. The functional specifications occupy about 80% of the document and involve a proportional amount of system design time to implement. Bus functions encompass 16 active signal lines, 10 interface functions, the protocol by which interface functions send and receive messages, and logical and timing relationships between signal states.

Functional requirements of the standard can be incorporated in either hardware, software, or a combination of both. Some designers have chosen the hardware approach to incorporate all the interface functions, using about 200 medium scale integrated (MSI) and small scale integrated (SSI) packages. This technique costs about $1000 for a complete interface board. As a result, many cost sensitive implementations of the bus interface use only a subset of its functions custom tailored to the requirements of the devices involved, thereby reducing package count and expense by curtailing the interchangeability advantages.

Other designers have selected the software approach to implement the bus interface. One disadvantage of this approach is that programming is an expensive and extended project; another is that a subroutine has to be executed with each transferred byte. This overhead not only burdens the microprocessor within a device, but also reduces the overall speed of the bus. This approach costs about $200 for the interfacing functions.

*This article deals with the functional aspects (interface signals that exist on the physical bus) of IEEE Std 488-1978, and is not intended as a complete dissertation on the major elements of the standard. For detailed definitions of the mechanical (physical cable connections), electrical (timing, voltages, and currents), and operational (application software routines) technicalities, interested readers should consult the *IEEE Standard Digital Interface for Programmable Instrumentation,* IEEE Std 488-1978, Institute of Electrical and Electronics Engineers, Inc, New York, NY 10017, Nov 30, 1978—**Ed.**

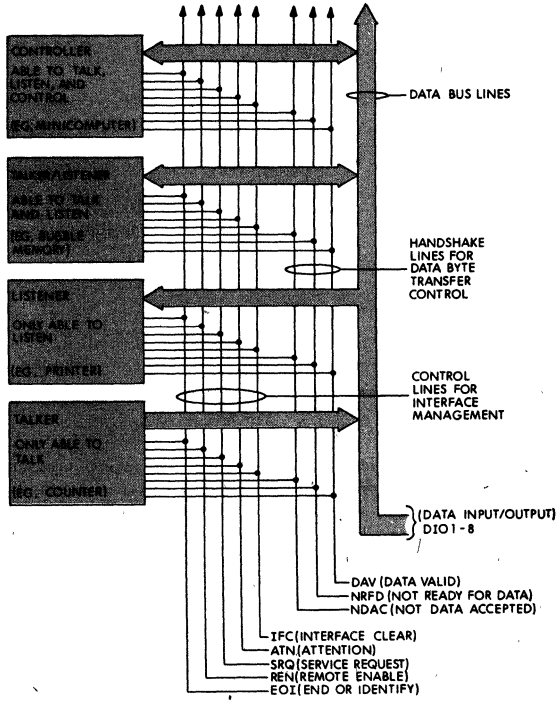


Fig 1 IEEE Std 488 active signal lines for multiple devices. Peripheral devices of different characteristics can be easily connected to standard bus interface. Controller (or processor), such as minicomputer, enables and disables talkers and listeners and manages overall bus activity. Bubble memory functions as both talker and listener. As listener, printer receives characters to be printed. As talker, counter transmits measurements to both controller and listeners

Combinational hardware/software approaches, although faster than direct software implementations, still require enormous design time and cost about $1000 for a typical interface board.

With a recent alternative approach, however, the bus interface is easier and less expensive to incorporate in instrument designs. LSI circuit chips now include as built-in capabilities most of the functional and some of the electrical portions of the Standard's specifications, significantly reducing design time and costing about $50 for bus interfacing. Additionally, Intel's 8291/8292 General Purpose Interface Bus (GPIB) peripheral chip set also incorporates capabilities for bus monitoring, data rate manipulation, and addressing to further simplify bus interface designs.

Bus Signal Definitions

The IEEE Std 488 signals are defined as negative true, where the high state (0 = false, ≥2.0 V) and the low state (1 = true, ≤0.8 V) are based on standard transistor-transistor logic (TTL) levels. Of the 16 active signal lines, 8 are data lines, 5 are interface management lines, and 3 are handshake lines (Fig 1). Data input/output lines (DIO1-DIO8) carry ASCII-coded information, as well as device addresses, universal commands, or program instructions. Interface management lines help to supervise the data lines. The primary management line—Attention (ATN)—determines how data lines are processed. When ATN is true, data lines are interpreted as addresses or universal commands by all bus connected devices. When ATN is false, only those devices addressed can use the data lines; in this case, data transmitted are typically device-dependent. With another management line, Interface Clear (IFC), the bus controller returns the system to a known quiescent state. The Service Request (SRQ) line can be used by any device on the interface bus when it has data to send (talker) or needs to receive data (listener). The Remote Enable (REN) line determines whether the system is under front panel or program control. The End Or Identify (EOI) line can be used as a delimiter by a talker (sending) device to indicate an end of message, or by the controller as a polling line.

Handshake lines control the timing relationship of the interface bus (Fig 2). The Data Valid (DAV) line

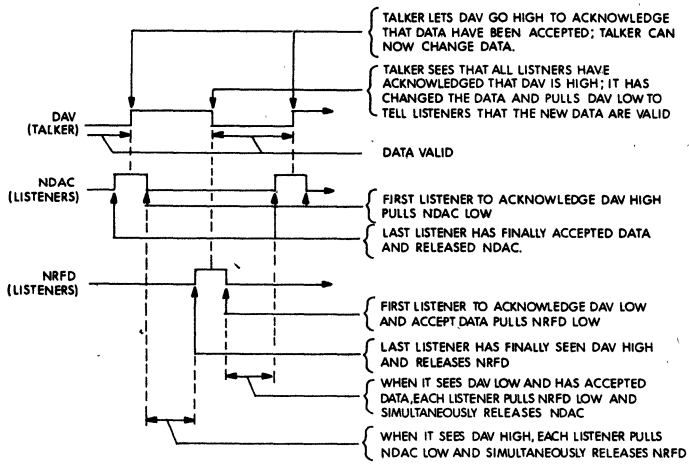


Fig 2 Three-wire handshaking between single talker and several listeners. Before transfer begins, listener indicates it is ready by asserting Ready For Data (RFD) message to true. Talker then drives all eight data input/output lines. Following settling time specified by standard, talker asserts Data Valid (DAV) message to true. While data are being read, RFD message is asserted to false since device is unable to receive additional data. As each listener completes its read, it indicates acceptance by asserting Data Accepted (DAC) message to true; DAC is not sensed true by talker until all listeners have completed read. After each device indicates acceptance, it indicates readiness for data by asserting RFD to true. New cycle begins when all devices have asserted RFD to true

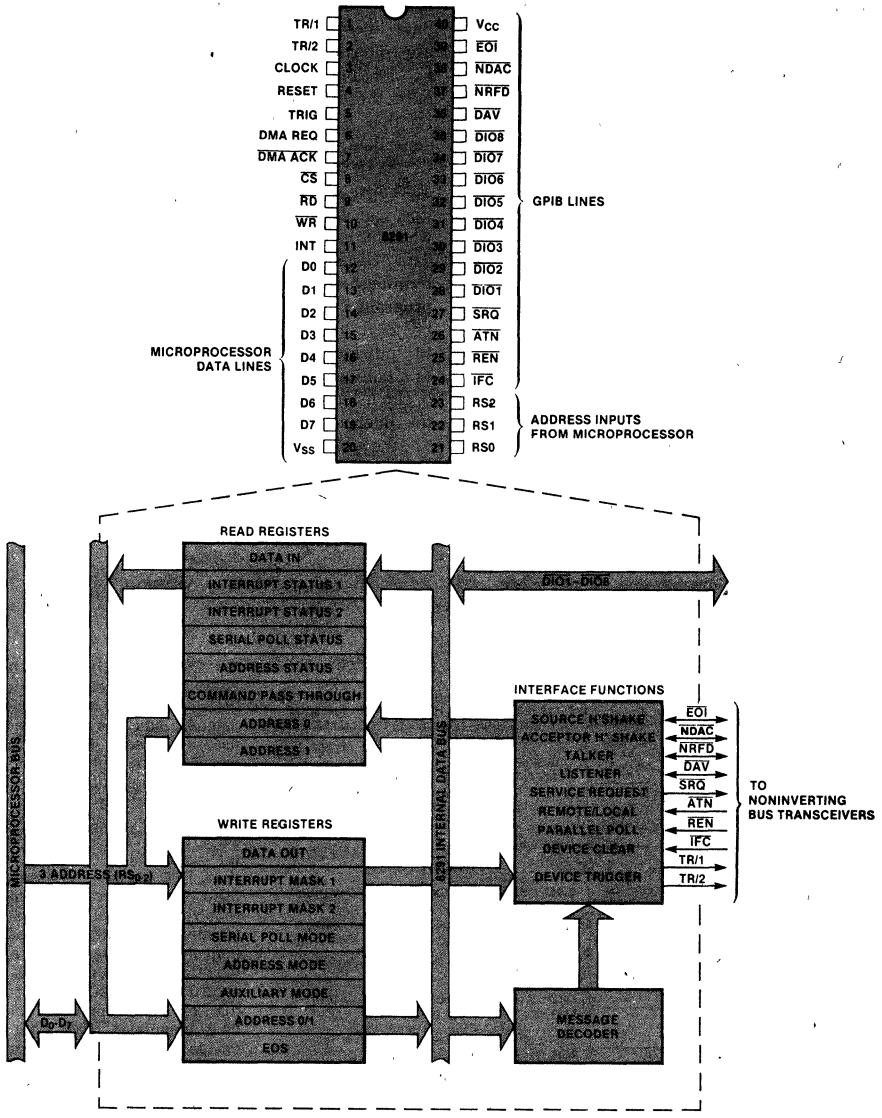
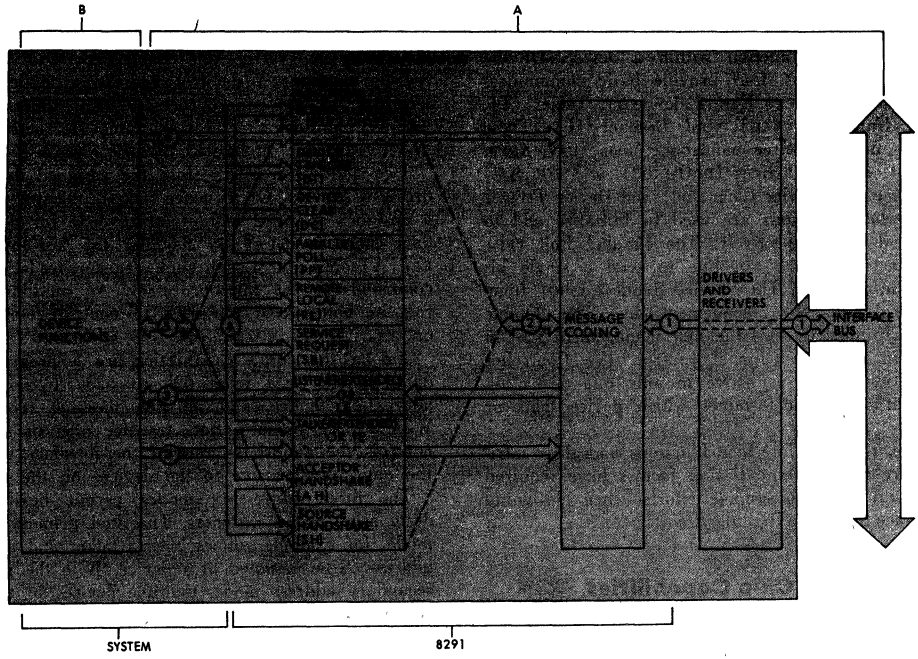


Fig 3 GPIB talker/listener chip. 8291 chip connects 8-bit microprocessor to noninverting bus transceivers, which, in turn, connect to IEEE Std 488 bus. Microprocessor manipulates data bytes after receipt or before transmission, and monitors talker/listener status. Single chip handles all IEEE Std 488 interface functions, except controller functions



A-CAPABILITY DEFINED BY 488-1978 STANDARD
B-CAPABILITY DEFINED BY DESIGNER
1-INTERFACE BUS SIGNAL LINES
2-REMOTE INTERFACE MESSAGES TO AND FROM INTERFACE FUNCTIONS
3-DEVICE DEPENDENT MESSAGES TO AND FROM DEVICE FUNCTIONS
4-STATE LINKAGES BETWEEN INTERFACE FUNCTIONS
5-LOCAL MESSAGES BETWEEN DEVICE FUNCTIONS AND INTERFACE FUNCTIONS (MESSAGES TO INTERFACE FUNCTIONS ARE DEFINED; MESSAGES FROM INTERFACE FUNCTIONS EXIST ACCORDING TO DESIGNER
6-REMOTE INTERFACE MESSAGES SENT BY DEVICE FUNCTIONS WITHIN CONTROLLER (8292)

Fig 4 Bus interface functions. Messages received from interface bus can cause state transitions, just as state transitions can cause messages to be sent on bus (1 and 2). Device dependent data are transferred automatically to microprocessor, without affecting state transitions (3). State changes in one function can cause state changes in another function, resulting in message to be sent (4). Microprocessor can also send local messages to interface functions (5) or remote messages to interface (6)

is used by a talker device to indicate that data are ready to transmit. The Not Ready For Data (NRFD) and Not Data Accepted (NDAC) lines are used by a listener to indicate readiness to receive data and receipt of data, respectively. As a result, a talker knows when all listeners on the bus have received an 8-bit byte of information. Thus, the transmission rate of the bus is only as fast as the slowest listener.

Messages conveyed by all 16 lines are true or false, depending on the states of 10 interface functions. The standard defines each of these interface functions with state diagrams. A function's state can be changed by a controller, another device on the bus, or a state change in another function within a device. Of the 10 interface functions, four provide basic communication capabilities: Source Handshake (SH), Talker (T), Acceptor Handshake (AH), and Listener (L). These functions affect the three handshake lines (DAV, NRFD, and NDAC), eight data lines (DIO1-DIO8), and EOI management line. The Device Clear (DC) and Device Trigger (DT) interface functions are used to initialize and to trigger a device, respectively. The Parallel Poll (PP) function acts with the EOI line to send a single bit of status information. The Service Request (SRQ) function controls the SRQ management line. The Remote Local (RL) interface uses the REN management line in conjunction with front panel control. The Controller (C) function, which is active in only one device on the bus at a time, determines which device talks or listens.

To date, these 10 interface functions and their intricate interrelationship and timing factors have required difficult and time consuming efforts when designing the interface bus into a digital system.

Talker/Listener Chip Capabilities

The 8291 GPIB talker/listener chip, a 40-pin LSI device (Fig 3), performs the inversion necessary to connect an 8-bit microprocessor bus to the negative true IEEE Std 488 bus. In addition, this chip implements most of the Standard's required functions. The microprocessor sets the talker/listener chip to an initial state, manipulates bytes before or after transmission, performs interrupt service routines, causes state changes, monitors other state changes, and enables and disables chip capabilities.

Without microprocessor involvement, the talker/listener chip implements all interface functions, except controller performance, such as handling data transfers, handshake protocols, listener/talker address procedures, device clearing and triggering, service requests, and parallel and serial polling schemes (Fig 4).

Within the chip architecture are eight read (output) and eight write (input) registers. One input register holds the data that are to be moved from the bus to the microprocessor when a device is listening. An output register holds the data byte that is to be transferred to the bus when a device is ready to talk. The other seven write and seven read registers control various chip functions.

Interrupt status registers 1 and 2 store 12 different interrupt flags. For example, one bit in the Interrupt Status 2 register reflects changes in a device's addressed state. The microprocessor can poll both registers to determine which flag caused the interrupt, and can then branch to the appropriate service routine. Two corresponding interrupt mask registers allow designers to mask any interrupt. A serial poll status register holds device status information, and a serial poll mode register is available so that the microprocessor can verify this status. An address mode register contains a device's addressing mode, as determined by the microprocessor. An address status register monitors the address status (ie, active talker or active listener) of a device.

Two address registers store the assigned device addresses. An End-Of-Sequence (EOS) register contains a designer specified end of string code for delimiting data block transfers by flagging the last byte with EOI. A command pass-through register feeds non-GPIB commands to the microprocessor. An auxiliary mode register holds local messages to control reset, power on, etc.

Among the chip's capabilities are a programmable data transfer rate from 62k to 525k bytes/s, three addressing modes, and an EOS message recognition. With a programmable data transfer rate, the designer controls the handshake rate of the interface to match the data transfer rate to the devices on the bus.

The three addressing modes permit flexibility in designating talkers/listeners. The dual primary address mode, for example, allows both a talker and a listener address to be assigned to a device. With the primary/secondary address mode, multiple devices of the same type can have the same primary address, but a different secondary address. In the third addressing mode, devices can have both dual primary and dual secondary addresses.

Data block transfers are made easier with the EOS register. This register holds the character that signals an end-of-block transfer. When a data byte loaded into the data-out register matches the byte in the EOS register, the talker/listener chip asserts the EOI line, signaling an end of transfer.

Controller Chip Capabilities

The 8292 controller chip (Fig 5) implements the controller function of the Standard. In conjunction with the 8291, the controller forms a complete standard interface, including the capability of handling the transfer control protocol. This ability gives the designer an option to accommodate multiple controllers on a single bus.

Additionally, the 8292 performs all the tasks necessary in a complete controller design. It responds to

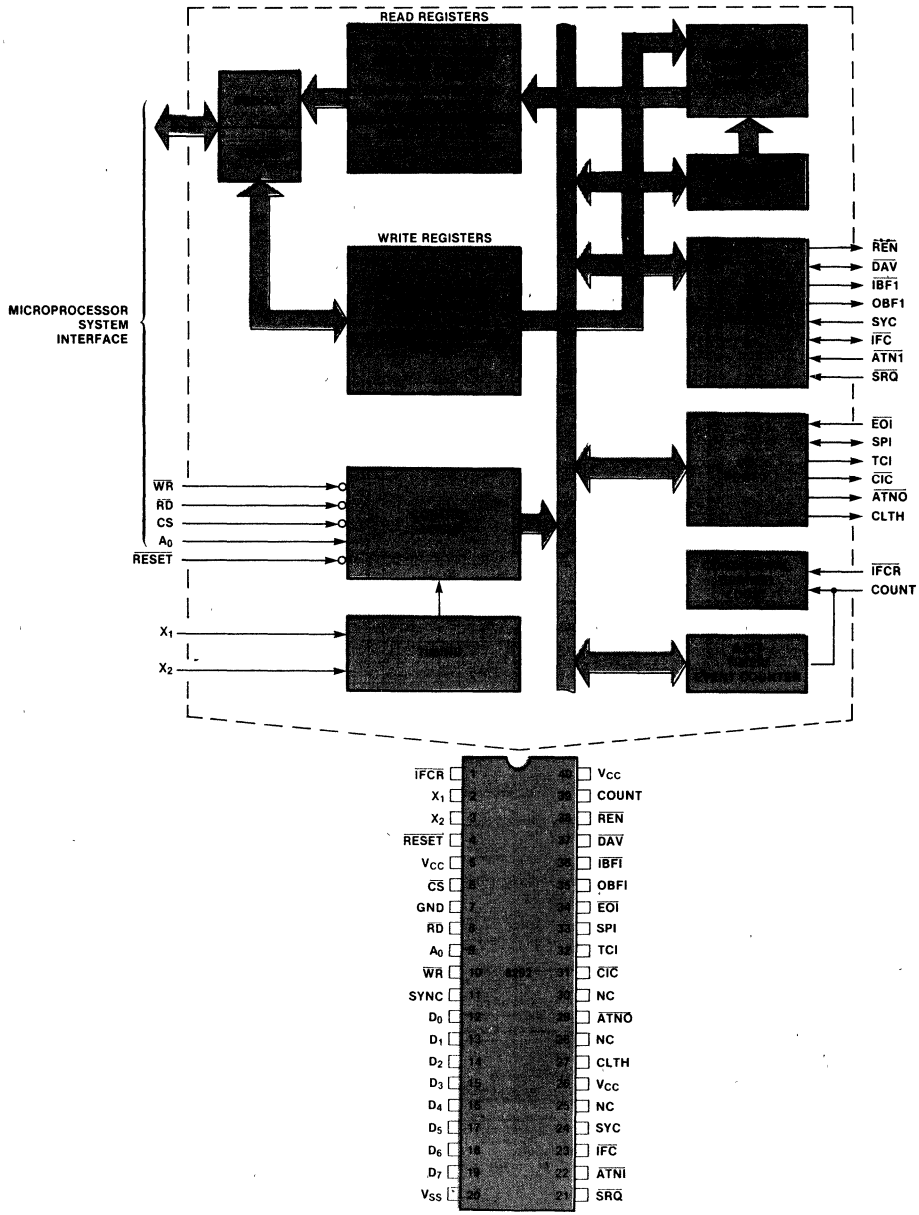


Fig 5 GPIB controller chip. 8292 chip works in conjunction with 8291 to perform GPIB controller interface functions. It implements local control commands from microprocessor according to IEEE Std 488 protocol. Additionally, it processes such inputs from bus as SRQ and EOI. Furthermore, it can send the full repertoire of GPIB control messages, including REN, IFC, ATN, and EOI

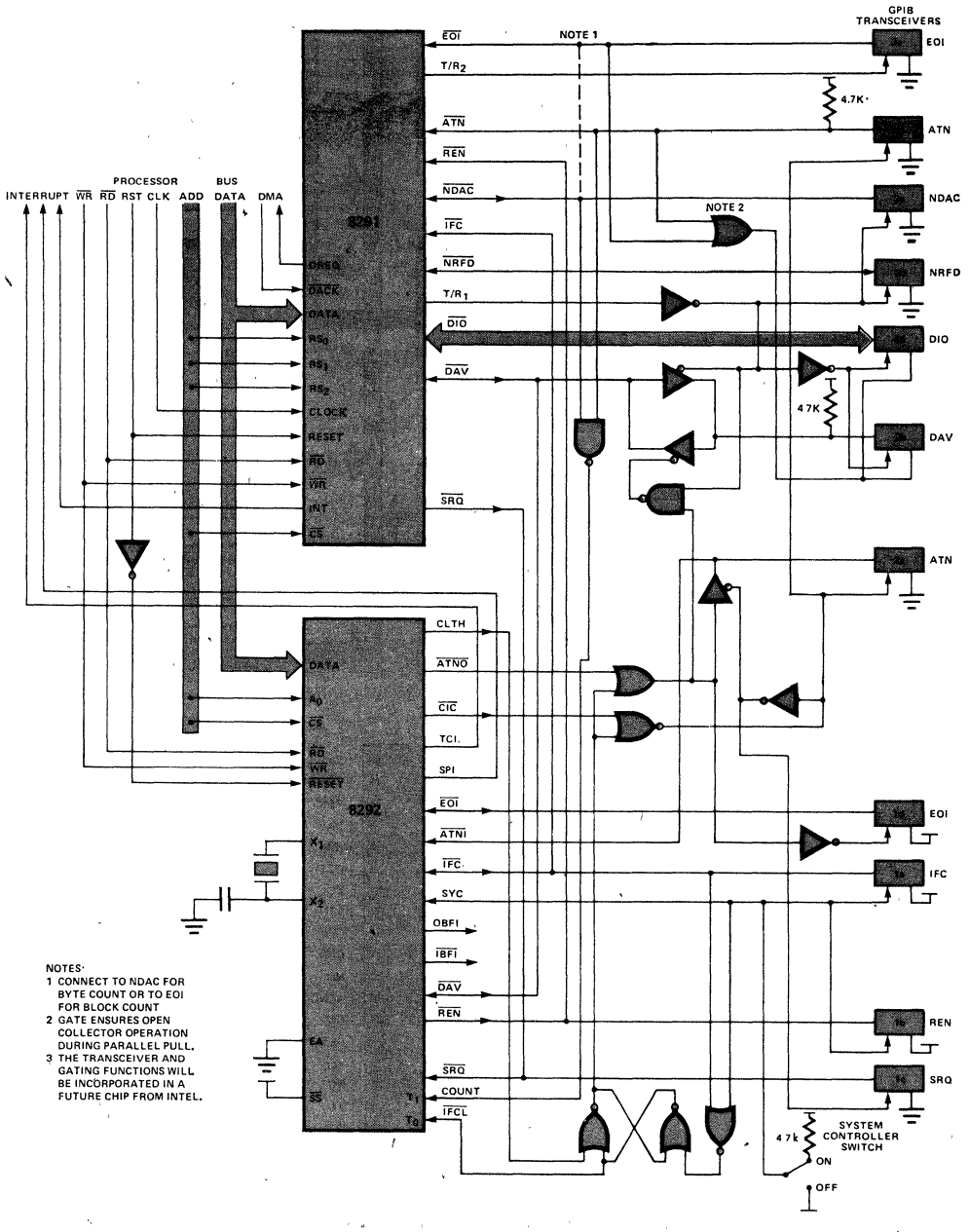


Fig 6 System configuration using chip set. In conjunction with 8291, 8292 performs complete controller function. Together with shared bus transceivers, chip set forms a complete IEEE Std 488 interface. In addition, DMA interface may be implemented through 8291 with 8237 DMA controller

service requests (SRQs), configures other devices on the bus for remote control by sending Remote Enable (REN), and sends Interface Clear (IFC), allowing for control seizure to reinitialize the bus. More importantly, the controller chip can take control of the bus synchronously with the handshake, preventing the destruction of any data transmission in progress.

Internally, the controller chip has 10 dedicated registers for programming and for monitoring status. Through the use of the Interrupt Status and Interrupt Mask registers, the designer can configure the controller to interrupt the microprocessor on selected events. An Event Counter and a corresponding status register are available to monitor and control either byte counts or block counts. A Time-Out register may be set by the designer to program a time-out error function; a corresponding status register contains the current value in the time-out counter. In conjunction with these registers, error control can be programmed with the Error Flags and Error Mask registers. Finally, Controller and GPIB Status registers are available. Each of these registers is read or programmed through a dedicated command buffer.

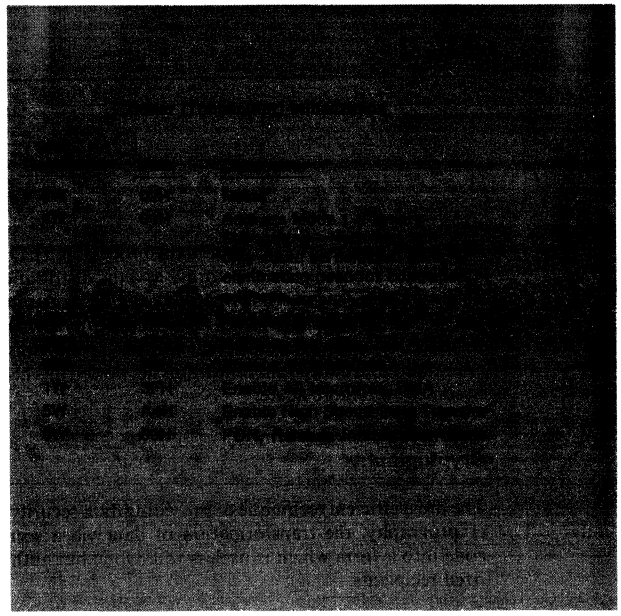
Chip Set Application

The talker/listener and controller chips connect to the standard interface bus through noninverting bus transceivers (Fig 6). These transceivers provide the 48-mA bus drive capability needed to meet the electrical portion of the IEEE Std 488 specification—not directly possible with existing metal oxide semiconductor (MOS) parts. The talker/listener chip can interface directly to microprocessor memory through a direct memory access (DMA) controller, such as an 8237.

The microprocessor drives the talker/listener with a short stored program (see Table), containing initialization conditions, such as data transfer rate, address mode, and other designer requirements. Microprocessor data handling is limited to taking bytes off the bus after they arrive or putting bytes of data on the bus. Interrupt service routines are necessary for each unmasked interrupt. Although 12 interrupts are available, not all have to be used. All other standard bus functions are handled by the 8291.

To send a byte of data, the microprocessor writes the byte into the talker/listener data-out register. The chip then transmits the data byte over the bus lines in conjunction with the handshake lines. Next, the NRFD line is checked to see if it is ready for data. If a ready for data message is detected, the talker/listener sends a DAV signal until it receives a data accepted message from the interface's NDAC line. The 8291 also generates a Byte Out (BO) interrupt, setting the BO flag in the interrupt status register. When its interrupt pin is activated, the microprocessor reads the interrupt status register and responds to the interrupt with an appropriate service routine.

The 8292 handles all hardware aspects of the controller function: SRQ input, ATN, IFC, EOI, and REN outputs. Meanwhile, the designer defined aspects of a given GPIB system are handled by processor software. For example, the processor is responsible for knowing which device on the bus corresponds to which device address. The processor then uses the 8291 to transmit coded Controller commands as the 8292 asserts ATN.



Summary

Bus interface designs that previously required 150 or 200 MSI/SSI chips may now be implemented with a GPIB peripheral chip set. For designers, this hardware set means less design time and cost, resulting in increased reliability and versatility in IEEE Std 488 bus interfaces custom programmed for dedicated applications.

Bibliography

S. C. Baunach, "Design Advantages and Limitations in Connecting Computational and Readout Equipment to the GPIB," Western Electronic Show and Convention, Sept 1976

A. Kaminker and A. Menachem, "LSI Facilitates GPIB Implementation," *IEEE Proceedings on Microcomputer Based Instrumentation*, June 1978

D. C. Loughry and M. S. Allen. "IEEE Standard 488 and Microprocessor Synergism," *Proceedings of the IEEE*, Feb 1978

Ronald M. Williams is a product manager for peripheral controllers in Intel's Microcomputer Components Division. In addition to GPIB devices, he has been involved in introductions of dynamic RAM and CRT controllers. He holds a BS degree from Trinity College, an MS degree from Rensselaer Polytechnic Institute, and an MBA degree from the University of Chicago.

DATA ENCRYPTION TUTORIAL

The proliferation of electronic data processing (EDP) applications that involve the storage and the distribution of potentially sensitive information have demonstrated the need for mechanisms to insure data privacy and security. As society becomes increasingly dependent on computers and data communications networks, this need becomes even more acute.

Cryptography

The most efficient technique of providing data security is cryptography: the transformation of data via a secret code into a form which is useless to anyone but authorized recipients.

A cryptographic algorithm can be presented as a sequence of mathematical transformations. Each transformation has it's unique inverse operation that changes the encrypted data back into the original plain text. In conventional cryptosystems, a set of specific parameters called a key is supplied along with the plain text/cipher text as an input to the enciphering/deciphering algorithm. The key is specified by the user. The transformation of the plain text and the cipher text depends on the key as well as the enciphering and deciphering algorithms. In fact the algorithms themselves can be made public, because the security of the system depends entirely on the secrecy of the key.

The initial interest in encryption for commercial applications came from financial institutions, most notably banks that are heavily involved in Electronic Fund Transfer (EFT). The American banking system alone, moves more than $400 billion between computers every day. The rapid rise of personal computers, workstations and the use of electronic mail and information retrieval services have spread the need for insuring data privacy and security to many other applications.

The DES

In response to the growing commercial need, the National Bureau of Standards has adopted in 1977 a standard algorithm know as the Data Encryption Standard (DES). The DES, originally developed by IBM, is designed for use with sensitive but unclassified information. The National Bureau of Standards requires that the DES be implemented in system hardware. The standardization insures that certified hardware from different suppliers are compatible.

The DES specifies a method for encrypting 64 bit blocks of clear data into corresponding 64 bit blocks of cipher text using a 56 bit key user specified. The 56 bit key (64 bit with parity) gives the user a total of 256 (seventy quadrillion) possible keys. Because the DES algorithm key is so long, a state of the art computer would take years to explore all possible permutations required to break the code. The most critical factor in protecting the data is guaranteeing the secrecy of the key.

Intel Data Encryption Product Line

Intel offers two peripherals supporting the DES algorithm: the 8294A Data Encryption Unit (DEU) and the 82538 Data Ciphering Processor (DCP).

The 8294A -a preprogrammed 8042- can encrypt and decrypt data at a rate up to 400 Byte/Sec. The 8294A is very well suited for data file protection, off line data encryption prior to transmission and phone line applications.

The 82538 is a much faster device: 1.5 Mbyte/Sec. This encryption rate is needed in satellite communications systems, data storage onto hard disks, high performance data communications networks like Ethernet. This rate is high enough to accomodate on the fly encryption in most of the communications systems and eliminate the need for buffers and interfacing circuitry. High encryption and decryption speed is not the only feature of this device. The 82538 supports bi-directional, half-duplex operations at its top speed. It contains three separate write only registers for encryption, decryption and master keys improving system's security and throughput. The DCP can also be configured in any of the three encryption/decryption modes recommended by the NBS (ECB, CBC or CFB).

The Intel Data Encryption product line solves the need for a broad range of applications. Security features can now be economically designed in data entry terminal as well as in satellite communications systems.

PRELIMINARY

8291A
GPIB TALKER/LISTENER

- Designed to Interface Microprocessors (e.g., 8048/49, 8051, 8080/85, 8086/88) to an IEEE Standard 488 Digital Interface Bus
- Programmable Data Transfer Rate
- Complete Source and Acceptor Handshake
- Complete Talker and Listener Functions with Extended Addressing
- Service Request, Parallel Poll, Device Clear, Device Trigger, Remote/Local Functions
- Selectable Interrupts
- On-Chip Primary and Secondary Address Recognition
- Automatic Handling of Addressing and Handshake Protocol
- Provision for Software Implementation of Additional Features
- 1–8 MHz Clock Range
- 16 Registers (8 Read, 8 Write), 2 for Data Transfer, the Rest for Interface Function Control, Status, etc.
- Directly Interfaces to External Non-Inverting Transceivers for Connection to the GPIB
- Provides Three Addressing Modes, Allowing the Chip to be Addressed Either as a Major or a Minor Talker/Listener with Primary or Secondary Addressing
- DMA Handshake Provision Allows for Bus Transfers without CPU Intervention
- Trigger Output Pin
- On-Chip EOS (End of Sequence) Message Recognition Facilitates Handling of Multi-Byte Transfers

The 8291A is an enhanced version of the 8291 GPIB Talker/Listener designed to interface microprocessors to an IEEE Standard 488 Instrumentation Interface Bus. It implements all of the Standard's interface functions except for the controller. The controller function can be added with the 8292 GPIB Controller, and the 8293 GPIB Transceiver performs the electrical interface for Talker/Listener and Talker/Listener/Controller configurations.

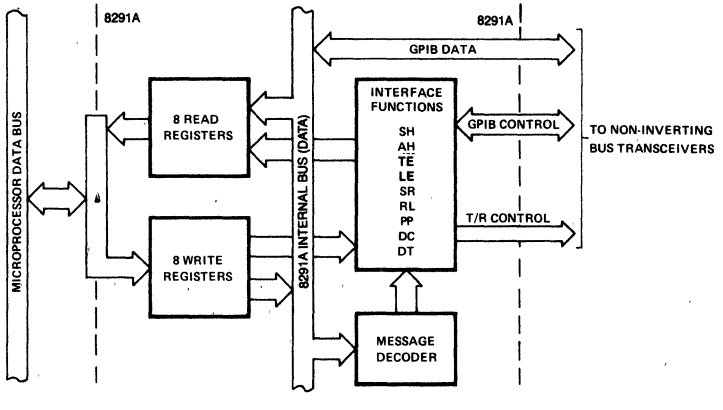


Figure 1. Block Diagram

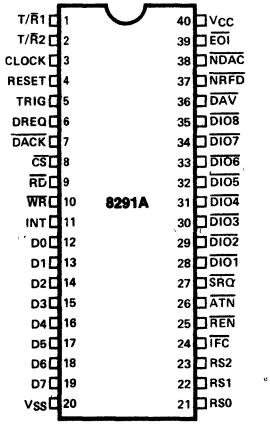


Figure 2. Pin Configuration

8291A FEATURES AND IMPROVEMENTS

The 8291A is an improved design of the 8291 GPIB Talker/Listener. Most of the functions are identical tc the 8291, and the pin configuration is unchanged.

The 8291A offers the following improvements to the 8291:

1. $\overline{EOI}$ is active with the data as a ninth data bit rather than as a control bit. This is to comply with some additions to the 1975 IEEE-488 Standard incorporated in the 1978 Standard.

2. The BO interrupt is not asserted until RFD is true. If the Controller asserts $\overline{ATN}$ synchronously, the data is guaranteed to be transmitted. If the Controller asserts $\overline{ATN}$ asynchronously, the SH (Source Handshake) will return to SIDS (Source Idle State), and the output data will be cleared. The, if $\overline{ATN}$ is released while the 8291A is addressed to talk, a new BO interrupt will be generated. This change fixes 8291 problems which caused data to be lost or repeated and a problem with the RQS bit (sometimes cannot be asserted while talking).

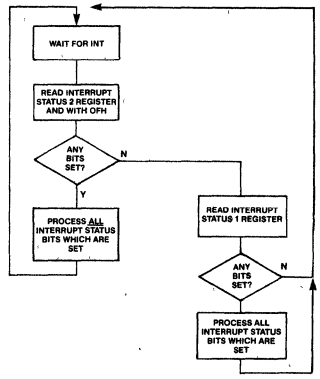
3. LLOC and REMC interrupts are setting flipflops rather than toggling flipflops in the interrupt backup register. This ensures that the CPU knows that these state changes have occurred. The actual state can be determined by checking the LLO and REM status bits in the upper nibble of the Interrupt Status 2 Register.

4. DREQ is cleared by $\overline{DACK}$ ($\overline{RD}$ + $\overline{WR}$). DREQ on the 8291 was cleared only by $\overline{DACK}$ which is not compatible with the 8089 I/O Processor.

5. The INT bit in Interrupt Status 2 Register and bit 7 of Address 0 Register are duplicates. When software polling is used to check interrupts, poll INT in Address 0 Register, instead of Interrupt Status 2 Register. Then, asynchronous status reads and interrupts will not lose interrupts.

   A lockout mechanism prevents all interrupt status bits to be set in both interrupt status registers. A back-up stores any bits, and latches onto the Interrupt Status Registers after the register with the bits set is read.

NOTE:
When an Interrupt Status Register is read, all the interrupt status bits should be checked before disregarding the byte read. A recommended way to handle this 8291A on END interrupt is in the flow chart below.



6. The 8291A's Send $\overline{EOI}$ Auxiliary Command works on any byte including the first byte of a message. The 8291 did not assert $\overline{EOI}$ after this command for a one byte message nor on two consecutive bytes.

7. To avoid confusion between holdoff on DAV versus RFD if a device is readdressed from a talker to a listener role or vice-versa during a holdoff, the "Holdoff on Source Handshake" has been eliminated. Only "Holdoff on Acceptor Handshake" is available.

8. The rsv local message is cleared automatically upon exit from SPAS if (APRS.STRS.SPAS) occurred. The automatic resetting of the bit after the serial poll is complete simplifies the service request software.

9. The SPASC interrupt on the 8291 has been replaced by the SPC (Serial Poll Complete) interrupt on the 8291A. SPC interrupt is set on exit from SPAS if APRS.STRS.SPAS occurred, indicating that the controller has read the bus status byte after the 8291A requested service. The SPASC interrupt was ambiguous because a controller could enter SPAS and exit SPAS generating two SPASC interrupts without reading the serial poll status byte. The SPC interrupt also simplifies the CPU's software by eliminating the interrupt when the serial poll is half way done.

10. The rtl Auxiliary Command in the 8291 has been replaced by Set and Clear rtl Commands in the 8291A. Using the new commands, the CPU has the flexibility to extend the length of local mode or leave it as a short pulse as in the 8291.

11. A holdoff RFD on GET, SDC, and DCL feature has been added to prevent additional bus activity while the CPU is responding to any of these commands. The feature is enabled by a new bit (B_4) in the Auxiliary Register B.

12. On the 8291, BO could cease to occur upon $\overline{IFC}$ going false if $\overline{IFC}$ occurred asynchronously. On the 8291A, BO continues to occur after $\overline{IFC}$ has gone false even if it arrived asynchronously.

    This can be used to set a flag in the user's software which will permit special routines to be executed for each device. It could be included as part of a normal initialization procedure as the first step after a chip reset.

13. User's software can distinguish between the 8291 and the 8291A as follows:
    a) pon (00H to register 5)
    b) RESET (02H to register 5)
    c) Read Interrupt Status 1 Register. If BO interrupt is set, the device is the 8291. If BO is clear, it is the 8291A.

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
D_0–D_7	12–19	I/O	**Data Bus Port:** To be connected to microprocessor data bus.
RS_0–RS_2	21–23	I	**Register Select:** Inputs, to be connected to three non-multiplexed microprocessor address bus lines. Select which of the 8 internal read (write) registers will be read from (written into) with the execution of $\overline{RD}$ ($\overline{WR}$.)
$\overline{CS}$	8	I	**Chip Select:** When low, enables reading from or writing into the register selected by RS_0–RS_2.
$\overline{RD}$	9	I	**Read Strobe:** When low with $\overline{CS}$ or $\overline{DACK}$ low, selected register contents are read.
$\overline{WR}$	10	I	**Write Strobe:** When low with $\overline{CS}$ or $\overline{DACK}$ low, data is written into the selected register.
INT ($\overline{INT}$)	11	O	**Interrupt Request:** To the microprocessor, set high for request and cleared when the appropriate register is accessed by the CPU. May be software configured to be active low.
DREQ	6	O	**DMA Request:** Normally low, set high to indicate byte output or byte input in DMA mode; reset by $\overline{DACK}$.
$\overline{DACK}$	7	I	**DMA Acknowledge:** When low, resets DREQ and selects data in/data out register for DMA data transfer (actual transfer done by $\overline{RD}$/$\overline{WR}$ pulse). Must be high if DMA is not used.
TRIG	5	O	**Trigger Output:** Normally low; generates a triggering pulse with 1 μsec min. width in response to the GET bus command or Trigger auxillary command.
CLOCK	3	I	**External Clock:** Input, used only for T_1 delay generator. May be any speed in 1–8 MHz range.
RESET	4	I	**Reset Input:** When high, forces the device into an "idle" (initialization) mode. The device will remain at "idle" until released by the microprocessor, with the "Immediate Execute pon" local message.
$\overline{DIO_1}$–$\overline{DIO_8}$	28–35	I/O	**8-Bit GPIB Data Port:** Used for bidirectional data byte transfer between 8291A and GPIB via non-inverting external line transceivers.
$\overline{DAV}$	36	I/O	**Data Valid:** GPIB handshake control line. Indicates the availability and validity of information on the $\overline{DIO_1}$–$\overline{DIO_8}$ and $\overline{EOI}$ lines.
$\overline{NRFD}$	37	I/O	**Not Ready for Data:** GPIB handshake control line. Indicates the condition of readiness of device(s) connected to the bus to accept data.
$\overline{NDAC}$	38	I/O	**Not Data Accepted:** GPIB handshake control line. Indicates the condition of acceptance of data by the device(s) connected to the bus.
$\overline{ATN}$	26	I	**Attention:** GPIB command line. Specifies how data on $\overline{DIO}$ lines are to be interpreted.
$\overline{IFC}$	24	I	**Interface Clear:** GPIB command line. Places the interface functions in a known quiescent state.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{SRQ}$	27	O	**Service Request:** GPIB command line. Indicates the need for attention and requests an interruption of the current sequence of events on the GPIB.
$\overline{REN}$	25	I	**Remote Enable:** GPIB command line. Selects (in conjunction with other messages) remote or local control of the device.
$\overline{EOI}$	39	I/O	**End or Identify:** GPIB command line. Indicates the end of a multiple byte transfer sequence or, in conjunction with $\overline{ATN}$, addresses the device during a polling sequence.
$\overline{T/R1}$	1	O	**External Transceivers Control Line:** Set high to indicate output data/signals on the $\overline{DIO_1}$–$\overline{DIO_8}$ and $\overline{DAV}$ lines and input signals on the $\overline{NRFD}$ and $\overline{NDAC}$ lines (active source handshake). Set low to indicate input data/signals on the $\overline{DIO_1}$–$\overline{DIO_8}$ and $\overline{DAV}$ lines and output signals on the $\overline{NRFD}$ and $\overline{NDAC}$ lines (active acceptor handshake).
T/R2	2	O	**External Transceivers Control Line:** Set to indicate output signals on the $\overline{EOI}$ line. Set low to indicate expected input signal on the $\overline{EOI}$ line during parallel poll.
V_{CC}	40	P.S.	**Positive Power Supply:** (5V ±10%).
GND	20	P.S.	**Circuit Ground Potential.**

NOTE:
All signals on the 8291A pins are specified with positive logic. However, IEEE 488 specifies negative logic on its 16 signal lines. Thus, the data is inverted once from D_0–D_7 to $\overline{DIO_0}$–$\overline{DIO_8}$ and non-inverting bus transceivers should be used.

THE GENERAL PURPOSE INTERFACE BUS (GPIB)

The General Purpose Interface Bus (GPIB) is defined in the IEEE Standard 488-1978 "Digital Interface for Programmable Instrumentation." Although a knowledge of this standard is assumed, Figure 4 provides the bus structure for quick reference. Also, Tables 2 and 3 reference the interface state mnemonics and the interface messages respectively. Modified state diagrams for the 8291A are presented in Appendix A.

General Description

The 8291A is a microprocessor-controlled device designed to interface microprocessors, e.g., 8048/49, 8051, 8080/85, 8086/88 to the GPIB. It implements all of the interface functions defined in the IEEE-488 Standard except for the controller function. If an implementation of the Standard's Controller is desired, it can be connected with an Intel® 8292 to form a complete interface.

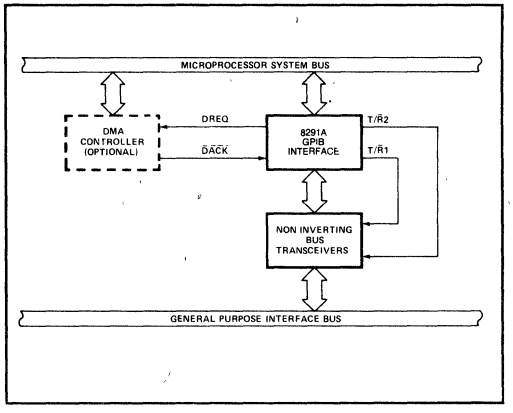


Figure 3. 8291A System Diagram

The 8291A handles communication between a microprocessor-controlled device and the GPIB. Its capabilities include data transfer, handshake protocol, talker/listener addressing procedures, device clearing and triggering, service request, and both serial and parallel polling. In most procedures, it does not disturb the microprocessor unless a byte has arrived (input buffer full) or has to be sent out (output buffer empty).

The 8291A architecture includes 16 registers. Eight of these registers may be written into by the microprocessor. The other eight registers may be read by the microprocessor. One each of these read and

write registers is for direct data transfers. The rest of the write registers control the various features of the chip, while the rest of the read registers provide the microprocessor with a monitor of GPIB states, various bus conditions, and device conditions.

GPIB Addressing

Each device connected to the GPIB must have at least one address whereby the controller device in charge of the bus can configure it to talk, listen, or send status. An 8291A implementation of the GPIB offers the user three alternative addressing modes for which the device can be initialized for each application. The first of these modes allows for the device to have two separate primary addresses. The second mode allows the user to implement a single talker/listener with a two byte address (primary address + secondary address). The third mode again allows for two distinct addresses but in this instance, they can each have a ten-bit address (5 low-order bits of each of two bytes). However, this mode requires that the secondary addresses be passed to the microprocessor for verification. These three addressing schemes are described in more detail in the discussion of the Address Registers.

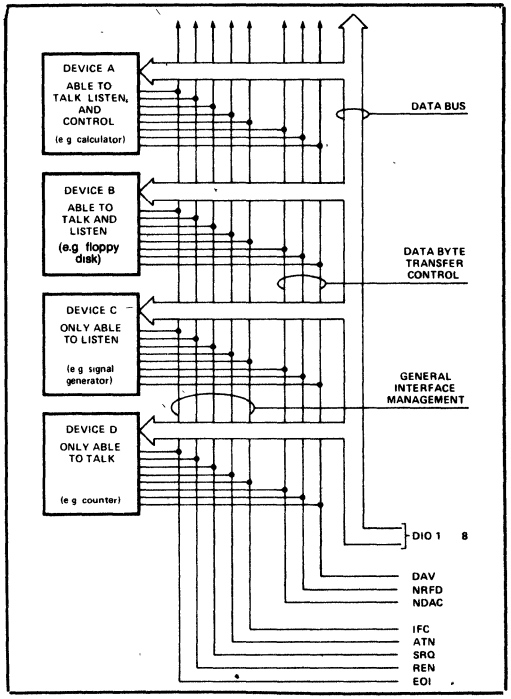


Figure 4. Interface Capabilities and Bus Structure

Table 2. IEEE 488 Interface State Mnemonics

Mnemonic	State Represented
ACDS	Accept Data State
ACRS	Acceptor Ready State
AIDS	Acceptor Idle State
ANRS	Acceptor Not Ready State
APRS	Affirmative Poll Response State
AWNS	Acceptor Wait for New Cycle State
CACS	Controller Active State *
CADS	Controller Addressed State *
CAWS	Controller Active Wait State *
CIDS	Controller Idle State *
CPPS	Controller Parallel Poll State *
CPWS	Controller Parallel Poll Wait State *
CSBS	Controller Standby State *
CSNS	Controller Service Not Requested State *
CSRS	Controller Service Requested State *
CSWS	Controller Synchronous Wait State *
CTRS	Controller Transfer State *
DCAS	Device Clear Active State
DCIS	Device Clear Idle State
DTAS	Device Trigger Active State
DTIS	Device Trigger Idle State
LACS	Listener Active State
LADS	Listener Addressed State
LIDS	Listener Idle State
LOCS	Local State
LPAS	Listener Primary Addressed State
LPIS	Listener Primary Idle State
LWLS	Local With Lockout State
NPRS	Negative Poll Response State

Mnemonic	State Represented
PACS	Parallel Poll Addressed to Configure State
PPAS	Parallel Poll Active State
PPIS	Parallel Poll Idle State
PPSS	Parallel Poll Standby State
PUCS	Parallel Poll Unaddressed to Configure State
REMS	Remote State
RWLS	Remote With Lockout State
SACS	System Control Active State
SDYS	Source Delay State
SGNS	Source Generate State
SIAS	System Control Interface Clear Active State
SIDS	Source Idle State
SIIS	System Control Interface Clear Idle State
SINS	System Control Interface Clear Not Active State
SIWS	Source Idle Wait State
SNAS	System Control Not Active State
SPAS	Serial Poll Active State
SPIS	Serial Poll Idle State
SPMS	Serial Poll Mode State
SRAS	System Control Remote Enable Active State
SRIS	System Control Remote Enable Idle State
SRNS	System Control Remote Enable Not Active State
SRQS	Service Request State
STRS	Source Transfer State
SWNS	Source Wait for New Cycle State
TACS	Talker Active State
TADS	Talker Addressed State
TIDS	Talker Idle State
TPIS	Talker Primary Idle State

*The Controller function is implemented on the Intel® 8292.

Table 3. IEEE 488 Interface Message Reference List

Mnemonic	Message	Interface Function(s)
LOCAL MESSAGES RECEIVED (By Interface Functions)		
[1]gts	go to standby	C
ist	individual status	PP
lon	listen only	L, LE
lpe	local poll enable	PP
nba	new byte available	SH
pon	power on	SH,AH,T,TE,L,LE,SR,RL,PP,C
rdy	ready	AH
[1]rpp	request parallel poll	C
[1]rsc	request system control	C
rsv	request service	SR
rtl	return to local	RL
[1]sic	send interface clear	C
[1]sre	send remote enable	C
[1]tca	take control asynchronously	C
[1]tcs	take control synchronously	AH, C
ton	talk only	T, TE
REMOTE MESSAGES RECEIVED		
ATN	Attention	SH,AH,T,TE,L,LE,PP,C
DAB	Data Byte	(Via L, LE)
DAC	Data Accepted	SH
DAV	Data Valid	AH
DCL	Device Clear	DC
END	End	(via L, LE)
GET	Group Execute Trigger	DT
GTL	Go to Local	RL
IDY	Identify	L,LE,PP
IFC	Interface Clear	T,TE,L,LE,C
LLO	Local Lockout	RL
MLA	My Listen Address	L,LE,RL,T,TE
MSA	My Secondary Address	TE,LE,RL
MTA	My Talk Address	T,TE,L,LE
OSA	Other Secondary Address	TE
OTA	Other Talk Address	T, TE
PCG	Primary Command Group	TE,LE,PP
[2]PPC	Parallel Poll Configure	PP
[2][PPD]	Parallel Poll Disable	PP
[2][PPE]	Parallel Poll Enable	PP
[1]PPR_N	Parallel Poll Response N	(via C)
[2]PPU	Parallel Poll Unconfigure	PP
REN	Remote Enable	RL
RFD	Ready for Data	SH
RQS	Request Service	(via L, LE)
[SDC]	Select Device Clear	DC
SPD	Serial Poll Disable	T, TE
SPE	Serial Poll Enable	T, TE
[1]SQR	Service Request	(via C)
STB	Status Byte	(via L, LE)
[1]TCT or [TCT]	Take Control	C
UNL	Unlisten	L, LE

NOTE:
1. These messages are handled only by Intel's 8292.
2. Undefined commands which may be passed to the microprocessor.

Table 3. (Cont'd)
IEEE 488 Interface Message Reference List

Mnemonic	Message	[3]Interface Function(s)
REMOTE MESSAGES SENT		
ATN	Attention	C
DAB	Data Byte	(via T, TE)
DAC	Data Accepted	AH
DAV	Data Valid	SH
DCL	Device Clear	(via C)
END	End	(via T)
GET	Group Execute Trigger	(via C)
GTL	Go to Local	(via C)
IDY	Identify	C
IFC	Interface Clear	C
LLO	Local Lockout	(via C)
MLA or [MLA]	My Listen Address	(via C)
MSA or [MSA]	My Secondary Address	(via C)
MTA or [MTA]	My Talk Address	(via C)
OSA	Other Secondary Address	(via C)
OTA	Other Talk Address	(via C)
PCG	Primary Command Group	(via C)
PPC	Parallel Poll Configure	(via C)
[PPD]	Parallel Poll Disable	(via C)
[PPE]	Parallel Poll Enable	(via C)
PPR_N	Parallel Poll Response N	PP
PPU	Parallel Poll Unconfigure	(via C)
REN	Remote Enable	C
RFD	Ready for Data	AH
RQS	Request Service	T, TE
[SDC]	Selected Device Clear	(via C)
SPD	Serial Poll Disable	(via C)
SPE	Serial Poll Enable	(via C)
SRQ	Service Request	SR
STB	Status Byte	(via T, TE)
TCT	Take Control	(via C)
UNL	Unlisten	(via C)

NOTE:
3. All Controller messages must be sent via Intel's 8292.

8291A Registers

A bit-by-bit map of the 16 registers on the 8291A is presented in Figure 5. A more detailed explanation of each of these registers and their functions follows. The access of these registers by the microprocessor is accomplished by using the $\overline{CS}$, $\overline{RD}$, $\overline{WR}$, and RS_0-RS_2 pins.

Register	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	RS_0-RS_2
All Read Registers	0	0	1	CCC
All Write Registers	0	1	0	CCC
High Impedance	1	d	d	ddd

Data Registers

DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

DATA-IN REGISTER (0R)

DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

DATA-OUT REGISTER (0W)

The Data-In Register is used to move data from the GPIB to the microprocessor or to memory when the 8291A is addressed to listen. Incoming information is separately latched by this register, and its contents are not destroyed by a write to the data-out

register. The RFD (Ready for Data) message is held false until the byte is removed from the data in register, either by the microprocessor or by DMA. The 8291A then completes the handshake automatically. In RFD holdoff mode (see Auxiliary Register A), the handshake is not finished until a command is sent telling the 8291A to release the holdoff. In this way, the same byte may be read seveal times, or an over anxious talker may be held off until all available data has been processed.

When the 8291A is addressed to talk, it uses the data-out register to move data onto the GPIB. After the BO interrupt is received and a byte is written to this register, the 8291A initiates and completes the handshake while sending the byte out over the bus. In the BO interrupt disable mode, the user should wait until BO is active before writing to the register. (In the DMA mode, this will happen automatically.) A read of the Data-In Register does not destroy the information in the Data-Out Register.

Interrupt Registers

CPT	APT	GET	END	DEC	ERR	BO	BI

INTERRUPT STATUS 1 (1R)

INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC

INTERRUPT STATUS 2 (2R)

CPT	APT	GET	END	DEC	ERR	BO	BI

INTERRUPT ENABLE 1 (1W)

0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC

INTERRUPT ENABLE 2 (2W)

INT	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0

ADDRESS 0 REGISTER

Figure 5. 8291A Registers

READ REGISTERS								RS2	RS1	RS0	WRITE REGISTERS							
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0	0	0	0	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
DATA IN											DATA OUT							
CPT	APT	GET	END	DEC	ERR	BO	BI	0	0	1	CPT	APT	GET	END	DEC	ERR	BO	BI
INTERRUPT STATUS 1											INTERRUPT ENABLE 1							
INT	SPAS	LLO	REM	SPC	LLOC	REMC	ADSC	0	1	0	0	0	DMAO	DMAI	SPC	LLOC	REMC	ADSC
INTERRUPT STATUS 2											INTERRUPT ENABLE 2							
S8	SRQS	S6	S5	S4	S3	S2	S1	0	1	1	S8	rsv	S6	S5	S4	S3	S2	S1
SERIAL POLL STATUS											SERIAL POLL MODE							
ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN	1	0	0	TO	LO	0	0	0	0	ADM1	ADM0
ADDRESS STATUS											ADDRESS MODE							
CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0	1	0	1	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
COMMAND PASS THROUGH											AUX MODE							
INT	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0	1	1	0	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1
ADDRESS 0											ADDRESS 0/1							
X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1	1	1	1	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
ADDRESS 1											EOS							

The 8291A can be configured to generate an interrupt to the microprocessor upon the occurrence of any of 12 conditions or events on the GPIB. Upon receipt of an interrupt, the microprocessor must read the Interrupt Status Registers to determine which event has occurred, and then execute the appropriate service routine (if necessary). Each of the 12 interrupt status bits has a matching enable bit in the interrupt enable registers. These enable bits are used to select the events that will cause the INT pin to be asserted. Writing a logic "1" into any of these bits enables the corresponding interrupt status bits to generate an interrupt. Bits in the Interrupt Status Registers are set regardless of the states of the enable bits. The Interrupt Status Registers are then cleared upon being read or when a local pon (power-on) message is executed. If an event occurs while one of the Interrupt Status Registers is being read, the event is held until after its register is cleared and then placed in the register.

The mnemonics for each of the bits in these registers and a brief description of their respective functions appears in Table 4. This tables also indicates how each of the interrupt bits is set.

NOTE: The INT bit in the Address 0 Register is a duplicate of the INT bit in the Interrupt Status 2 Register. It is only a status bit. It does not generate interrupts and thus does not have a corresponding enable bit.

The BO and BI interrupts enable the user to perform data transfer cycles. BO indicates that a data byte should be written to the Data Out Register. It is set by TACS · (SWNS + SGNS) · RFD. It is reset when the data byte is written, ATN is asserted, or the 8291A exits TACS. Data should never be written to the Data Out Register before BO is set. Similarly, BI is set when an input byte is accepted into the 8291A and reset when the microprocessor reads the Data In Register. BO and BI are also reset by pon (power-on local message) and by a read of the Interrupt

Table 4. Interrupt Bits

Indicates Undefined Commands	CPT	An undefined command has been received.
Set by (TPAS + LPAS)•SCG•ACDS•MODE 3	APT	A secondary address must be passed through to the microprocessor for recognition.
Set by DTAS	GET	A group execute trigger has occurred.
Set by (EOS + EOI)•LACS	END	An EOS or EOI message has been received.
Set by DCAS	DEC	Device Clear Active State has occurred.
Set by TACS•nba•DAC•RFD	ERR	Interface error has occurred; no listeners are active.
TACS•(SWNS + SGNS)	BO	A byte should be output.
Set by LACS•ACDS	BI	A byte has been input.
Shows status of the INT pin	INT	These are status only. They will not generate interrupts, nor do they have corresponding mask bits.
The device has been enabled for a serial poll	SPAS	
The device is in local lock out state. (LWLS+RWLS)	LLO	
The device is in a remote state. (REMS+RWLS)	REM	
SPAS →$\overline{\text{SPAS}}$ if APRS:STRS:SPAS was true	SPC	Serial Poll Complete interrupt.
LLO ⇄ NO LLO	LLOC	Local lock out change interrupt.
Remote ⇄ Local	REMC	Remote/Local change interrupt.
Addressed ⇄ Unaddressed	ADSC	Address status change interrupt.[1]

NOTE: [1]In ton (talk-only) and lon (listen-only) modes, no ADSC interrupt is generated.

Status 1 Register. However, if it is so desired, data transfer cycles may be performed without reading the Interrupt Status 1 Register if all interrupts except for BO or BI are disabled; BO and BI will automatically reset after each byte is transferred.

If the 8291A is used in the interrupt mode, the INT and DREQ pins can be dedicated to data input and output interrupts respectively by enabling BI and DMAO, provided that no other interrupts are enabled. This eliminates the need to read the interrupt status registers if a byte is received or transmitted.

The ERR bit is set to indicate the bus error condition when the 8291A is an active talker and tries to send a byte to the GPIB, but there are no active listeners (e.g., all devices on the GPIB are in AIDS). The logical equivalent of (nba · TACS · DAC · RFD) will set this bit.

The DEC bit is set whenever DCAS has occurred. The user must define a known state to which all device functions will return in DCAS. Typically this state will be a power-on state. However, the state of the device functions at DCAS is at the designer's discretion. It should be noted that DCAS has no effect on the interface functions which are returned to a known state by the IFC (interface clear) message or the pon local message.

The END interrupt bit may be used by the microprocessor to detect that a multi-byte transfer has been completed. The bit will be set when the 8291A is an active listener (LACS) and either EOS (provided the End on EOS Received feature is enabled in the Auxiliary Register A) or EOI is received. EOS will generate an interrupt when the byte in the Data In Register matches the byte in the EOS register. Otherwise the interrupt will be generated when a true input is detected on EOI.

The GET interrupt bit is used by the microprocessor to detect that DTAS has occurred. It is set by the 8291A when the GET message is received while it is addressed to listen. The TRIG output pin of the 8291A fires when the GET message is received. Thus, the basic operation of device trigger may be started without microprocessor software intervention.

The APT interrupt bit indicates to the processor that a secondary address is available in the CPT register for validation. This interrupt will only occur if Mode 3 addressing is in effect. (Refer to the section on addressing.) In Mode 2, secondary addresses will be recognized automatically on the 8291A. They will be ignored in Mode 1.

The CPT interrupt bit flags the occurrence of an undefined command and of all secondary commands following an undefined command. The Command Pass Through feature is enabled by the B0 bit of Auxiliary Register B. Any message not decoded by the 8291A (not included in the state diagrams, in Appendix B) becomes an undefined command. Note that any addressed command is automatically ignored when the 8291A is not addressed.

Undefined commands are read by the CPU from the Command Pass Through register of the 8291A. This register reflects the logic levels present on the data lines at the time it is read. If the CPT feature is enabled, the 8291A will hold off the handshake until this register is read.

An especially useful feature of the 8291A is its ability to generate interrupts from state transitions in the interface functions. In particular, the lower 3 bits of the Interrupt Status 2 Register, if enabled by the corresponding enable bits, will cause an interrupt upon changes in the following states as defined in the IEEE 488 Standard.

Bit 0	ADSC	change in LIDS or TIDS or MJMN
Bit 1	REMC	change in LOCS or REMS
Bit 2	LLOC	change in LWLS or RWLS

The upper 4 bits of the Interrupt Status 2 Register are available to the processor as status bits. Thus, if one of the bits 1 and 2 generates an interrupt indicating a state change has taken place, the corresponding status bit (bits 4 and 5) may be read to determine what the new state is. To determine the nature of a change in addressed status (bit 0) the Address Status Register is available to be read. The SPC interrupt (bit 3 in Interrupt Status 2) is set upon exit from SPAS if APRS:STRS:SPAS occurred which indicates that the GPIB controller has read the bus serial poll status byte after the 8291A requested service (asserted SRQ). The SPC interrupt occurs once after the controller reads the status byte if service was requested.

The controller may read the status byte later, and the byte will contain the last status the 8291A's CPU wrote to the Serial Poll Mode Register, but the SRQS bit will not be set and no interrupt will be generated. Finally, bit 7 monitors the state of the 8291A INT pin. Logically, it is an OR of all enabled interrupt status bits. One should note that bits 4–7 of the Interrupt Status 2 Register do not generate interrupts, but are available only to be read as status bits by the processor. Bit 7 in Interrupt Status 2 is duplicated in Address 0 Register, and the latter should be used when polling for interrupts to avoid losing one of the interrupts in Interrupt Status 2 Register.

Bits 4 and 5 (DMAI, DMAO) of the Interrupt Mask 2 Register are available to enable direct data transfers between memory and the GPIB; DMAI (DMA in) enables the DREQ (DMA request) pin of the 8291A to be asserted upon the occurrence of BI. Similarly, DMAO (DMA out) enables the DREQ pin to be asserted upon the occurrence of BO. One might note that the DREQ pin may be used as a second interrupt output pin, monitoring BI and/or BO and enabled by DMAI and DMAO. One should note that the DREQ pin is not affected by a read of the Interrupt Status 1 Register. It is reset whenever a byte is written to the Data Out Register or read from the Data In Register.

To ensure that an interrupt status bit will not be cleared without being read, and will not remain uncleared after being read, the 8291A implements a special interrupt handling procedure. When an enabled interrupt bit is set in either of the Interrupt Status Registers, the input of the registers are blocked until the set bit is read and reset by the microprocessor. Thus, potential problems arise when interrupt status changes while the register is being blocked. However, the 8291A stores all new interrupts in a temporary register and transfers them to the appropriate Interrupt Status Register after the interrupt has been reset. This transfer takes place only if the corresponding bits were read as zeroes.

Serial Poll Registers

S8	SRQS	S6	S5	S4	S3	S2	S1

SERIAL POLL STATUS (3R)

S8	rsv	S6	S5	S4	S3	S2	S1

SERIAL POLL MODE (3W)

The Serial Poll Mode Register determines the status byte that the 8291A sends out on the GPIB data lines when it receives the SPE (Serial Poll Enable) message. Bit 6 of this register is reserved for the rsv (request service) local message. Setting this bit to 1 causes the 8291A to assert its $\overline{SRQ}$ line, indicating its need for attention from the controller-in-charge of the GPIB. The other bits of this register are available for sending status information over the GPIB. Sometime after the microprocessor initiates a request for service by setting bit 6, the controller of the GPIB sends the SPE message and then addresses the 8291A to talk. At this point, one byte of status is returned by the 8291A via the Serial Poll Mode Register. After the status byte is read by the controller, rsv is automatically cleared by the 8291A and an SPC interrupt is generated. The CPU may request service again by writing another byte to the Serial Poll Mode Register with the rsv bit set. If the controller performs a serial poll when the rsv bit is clear, the last status byte written will be read, but the $\overline{SRQ}$ line will not be driven by the 8291A and the SRQS bit will be clear in the status byte.

The Serial Poll Status Register is available for reading the status byte in the Serial Poll Mode Register. The processor may check the status of a request for service by polling bit 6 of this register, which corresponds to SRQS (Service Request State). When a Serial Poll is conducted and the controller-in-charge reads the status byte, the SRQS bit is cleared. The $\overline{SRQ}$ line and the rsv bit are tied together.

Address Registers

ton	lon	EOI	LPAS	TPAS	LA	TA	MJMN

ADDRESS STATUS (4R)

INT	DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0

ADDRESS 0 (6R)

X	DT1	DL1	AD5-1	AD4-1	AD3-1	AD2-1	AD1-1

ADDRESS 1 (7R)

TO	LO	0	0	0	0	ADM1	ADM0

ADDRESS MODE (4W)

ARS	DT	DL	AD5	AD4	AD3	AD2	AD1

ADDRESS 0/1 (6W)

The Address Mode Register is used to select one of the five modes of addressing available on the 8291A. It determines the way in which the 8291A uses the information in the Address 0 and Address 1 Registers.

—In Mode 1, the contents of the Address 0 Register constitute the "Major" talker/listener address while the Address 1 Register represents the "Minor" talker/listener address. In applications where only one address is needed, the major talker/listener is used, and the minor talker/listener should be disabled. Loading an address via the Address 0/1 Register into Address Registers 0 and 1 enables the major and minor talker/listener functions respectively.

—In Mode 2 the 8291A recognizes two sequential address bytes: a primary followed by a secondary. Both address bytes must be received in order to enable the device to talk or listen. In this manner, Mode 2 addressing implements the extended talker and listener functions as defined in IEEE-488.

To use Mode 2 addressing the primary address must be loaded into the Address 0 Register, and the Secondary Address is placed in the Address 1 Register. With both primary and secondary addresses residing on chip, the 8291A can handle all addressing sequences without processor intervention.

—In Mode 3, the 8291A handles addressing just as it does in Mode 1, except that each Major or Minor primary address must be followed by a secondary address. All secondary addresses must be verified by the microprocessor when Mode 3 is used. When the 8291A is in TPAS or LPAS (talker/listener primary addressed state), and it does not recognize the byte on the DIO lines, an APT interrupt is generated (see section on Interrupt Registers) and the byte is available in the CPT (Command Pass-Through) Register. As part of its interrupt service routine, the microprocessor must read the CPT Register and write one of the following responses to the Auxiliary Mode Register:

1. 07H implies a non-valid secondary address
2. 0FH implies a valid secondary address

Setting the TO bit generates the local ton (talk-only) message and sets the 8291A to a talk-only mode. This mode allows the device to operate as a talker in an interface system without a controller.

Setting the LO bit generates the local lon (listen-only) message and sets the 8291A to a listen-only mode. This mode allows the device to operate as a listener in an interface system without a controller. The above bits may also be used by a controller-in-charge to set itself up for remote command or data communication.

The mode of addressing implemented by the 8291A may be selected by writing one of the following bytes to the Address Mode Register.

Register Contents	Mode
10000000	Enable talk only mode (ton)
01000000	Enable listen only mode (lon)
11000000	The 8291 may talk to itself
00000001	Mode 1, (Primary-Primary)
00000010	Mode 2 (Primary-Secondary)
00000011	Mode 3 (Primary/APT-Primary/APT)

The Address Status Register contains information used by the microprocessor to handle its own addressing. This information includes status bits that monitor the address state of each talker/listener, "ton" and "lon" flags which indicate the talk and listen only states, and an EOI bit which, when set, signifies that the END message came with the last data byte. LPAS and TPAS indicate that the listener or talker primary address has been received. The microprocessor can use these bits when the secondary address is passed through to determine whether the 8291A is addressed to talk or listen. The LA (listener addressed) bit will be set when the 8291A is in LACS (Listener Active State) or in LADS (Listener Addressed State). Similarly, the TA (Talker Addressed bit) will be set to indicate TACS or TADS, but also to indicate SPAS (Serial Poll Active State). The MJMN bit is used to determine whether the information in the other bits applies to the Major or Minor talker/listener. It is set to "1" when the Minor talker/listener is addressed. It should be noted that only one talker/listener may be active at any one time. Thus, the MJMN bit will indicate which, if either, of the talker/listeners is addressed or active.

The Address 0/1 Register is used for specifying the device's addresses according to the format selected in the Address Mode Register. Five bit addresses may be loaded into the Address 0 and Address 1 Registers by writing into the Address 0/1 Register. The ARS bit is used to select which of these registers the other seven bits will be loaded into. The DT and DL bits may be used to disable the talker or listener function at the address signified by the other five

bits. When Mode 1 addressing is used and only one primary address is desired, *both* the talker and the listener should be disabled at the Minor address.

As an example of how the Address 0/1 Register might be used, consider an example where two primary addresses are needed in the device. The Major primary address will be selectable only as a talker and the Minor primary address will be selectable only as a listener. This configuration of the 8291A is formed by the following sequence of writes by the microprocessor.

Operation	$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	Data	RS2-RS0
1. Select addressing Mode 1	0	1	0	00000001	100
2. Load major address into Address 0 Register with listener function disabled.	0	1	0	001AAAAA	110
3. Load minor address into Address 1 Register with talker function disabled.	0	1	0	110BBBBB	110

At this point, the addresses AAAAA and BBBBB are stored in the Address 0 and Address 1 Registers respectively, and are available to be read by the microprocessor. Thus, it is not necessary to store any address information elsewhere. Also, with the information stored in the Address 0 and Address 1 Registers, processor intervention is not required to recognize addressing by the controller. Only in Mode 3, where secondary addresses are passed through, must the processor intervene in the addressing sequence.

The Address 0 Register contains a copy of bit 7 of the Interrupt Status 2 Register (INT). This is to be used when polling for interrupts. Software should poll register 6 checking for INT (bit 7) to be set. When INT is set, the Interrupt Status Register should be read to determine which interrupt was received.

Command Pass Through Register

CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0

COMMAND PASS THROUGH (5R)

The Command Pass Through Register is used to transfer undefined 8-bit remote message codes from the GPIB to the microprocessor. When the CPT feature is enabled (bit B0 in Auxiliary Register B), any message not decoded by the 8291A becomes an undefined command. When Mode 3 addressing is used secondary addresses are also passed through the CPT Register. In either case, the 8291A will hold-off the handshake until the microprocessor reads this register and issues the VSCMD auxiliary command.

The CPT and APT interrupts flag the availablility of undefined commands and secondary addresses in the CPT Register. The details of these interrupts are explained in the section on Interrupt Registers.

An added feature of the 8291A is its ability to handle undefined secondary commands following undefined primaries. Thus, the number of available commands for future IEEE-488 definition is increased; one undefined primary command followed by a sequence of as many as 32 secondary commands can be processed. The IEEE-488 Standard does not permit users to define their own commands, but upgrades of the standard are thus provided for.

The recommended use of the 8291A's undefined command capabilities is for a controller-configured Parallel Poll. The PPC message is an undefined primary command typically followed by PPE, an undefined secondary command. For details on this procedure, refer to the section on Parallel Poll Protocol.

Auxiliary Mode Register

CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0

AUX MODE (5W)

CNT0-2:CONTROL BITS
COM0-4:COMMAND BITS

The Auxiliary Mode Register contains a three-bit control field and a five-bit command field. It is used for several purposes on the 8291A:

1. To load "hidden" auxiliary registers on the 8291A.
2. To issue commands from the microprocessor to the 8291A.
3. To preset an internal counter used to generate T1, delay in the Source Handshake function, as defined in IEEE-488.

Table 5 summarizes how these tasks are performed with the Auxiliary Mode Register. Note that the three control bits determine how the five command bits are interpreted.

Table 5

CODE		COMMAND
CONTROL BITS	COMMAND BITS	
000	0CCCC	Execute auxiliary command CCCC
001	ODDDD	Preset internal counter to match external clock frequency of DDDD MHz (DDDD binary representation of 1 to 8 MHz)
100	DDDDD	Write DDDDD into auxiliary register A
101	DDDDD	Write DDDDD into auxiliary register B
011	$USP_3P_2P_1$	Enable/disable parallel poll either in response to remote messages (PPC followed by PPE or PPD) or as a local lpe message. (Enable if U = 0, disable if U = 1.)

AUXILIARY COMMANDS

Auxiliary commands are executed by the 8291A whenever 0000CCCC is written into the Auxiliary Mode Register, where CCCC is the 4-bit command code.

0000—Immediate Execute pon: This command resets the 8291A to a power up state (local pon message as defined in IEEE-488).
The following conditions constitute the power up state:
1. All talkers and listeners are disabled.
2. No interrupt status bits are set.

The 8291A is designed to power up in certain states as specified in the IEEE-488 state diagrams. Thus, the following states are in effect in the power up state: SIDS, AIDS, TIDS, LIDS, NPRS, LOCS, and PPIS.
The "0000" pon is an immediate execute command (a pon pulse). It is also used to release the "initialize" state generated by either an external reset pulse or the "0010" Chip Reset command.

0010—Chip Reset (Initialize): This command has the same effect as a pulse applied to the Reset pin. (Refer to the section on Reset Procedure.)

0011—Finish Handshake : This command finishes a handshake that was stopped because of a holdoff on RFD. (Refer to Auxiliary Register A.)

0100—Trigger: A "Group Execute Trigger" is forced by this command. It has the same effect as a GET command issued by the controller-in-charge of the GPIB, but does not cause a GET interrupt.

0101, 1101—Clear/Set rtl: These commands correspond to the local rtl message as defined by the IEEE-488. The 8291A will go into local mode when a Set rtl Auxiliary Command is received if local lockout is not in effect. The 8291A will exit local mode after receiving a Clear rtl Auxiliary Command if the 8291A is addressed to listen.

0110—Send EOI: The EOI line of the 8291A may be asserted with this command. The command causes EOI to go true with the next byte transmitted. The EOI line is then cleared upon completion of the handshake for that byte.

0111, 1111—Non Valid/Valid Secondary Address or Command (VSCMD): This command informs the 8291A that the secondary address received by the microprocessor was valid or invalid (0111 = invalid, 1111 = valid). If Mode 3 addressing is used, the processor must field each extended address and respond to it, or the GPIB will hang up. Note that the COM3 bit is the invalid/valid flag.

The valid (1111) command is also used to tell the 8291A to continue from the command-pass-through-state, or from RFD holdoff on GET, SDC or DCL.

1000—pon: This command puts the 8291A into the pon (power on) state and holds it there. It is similar to a Chip Reset except none of the Auxiliary Mode Registers are cleared. In this state, the 8291A does not participate in any bus activity. An Immediate Execute pon releases the 8291A from the pon state and permits the device to participate in the bus activity again.

0001, 1001—Parallel Poll Flag (local "ist" message): This command sets (1001) or clears (0001) the parallel poll flag. A "1" is sent over the assigned data line (PRR = Parallel Poll Response true) only if the parallel poll flag matches the sense bit from the lpe local message (or indirectly from the PPE message). For a more complete description of the Parallel Poll features and procedures refer to the section on Parallel Poll Protocol.

INTERNAL COUNTER

The internal counter determines the delay time allowed for the setting of data on the DIO lines. This delay time is defined as T_1 in IEEE-488 and appears in the Source Handshake state diagram between the

SDYS and STRS. As such, DAV is asserted T_1 *after* the DIO lines are driven. Consequently, T_1 is a major factor in determining the data transfer rate of the 8291A over the GPIB (T_1 = TWRDV2-TWRD15).

When open-collector transceivers are used for connection to the GPIB, T_1 is defined by IEEE-488 to be 2μsec. By writing 0010DDDD into the Auxiliary Mode Register, the counter is preset to match a f_C MHz clock input, where DDDD is the binary representation of N_F [$1 \leq N_F \leq 8$, $N_F=(DDDD)_2$]. When $N_F = f_C$, a 2μsec T_1 delay will be generated before each DAV asserted.

$$T_{1(\mu sec)} = \frac{2N_F}{f_C} + t_{SYNC}\ ,\ 1 \leq N_F \leq 8$$

t_{SYNC} is a synchronization error, greater than zero and smaller than the larger of T clock high and T clock low. (For a 50% duty cycle clock, t_{SYNC} is less than half the clock cycle).

If it is necessary that T_1 be different from 2μsec, N_F may be set to a value other than f_C. In this manner, data transfer rates may be programmed for a given system. In small systems, for example, where transfer rates exceeding GPIB specifications are required, one may set $N_F < f_C$ and decrease T_1.

When tri-state transceivers are used, IEEE-488 allows a higher transfer rate (lower T_1). Use of the 8291A with such transceivers is enabled by setting B_2 in Auxiliary Register B. In this case, setting $N_F=f_C$ causes a T_1 delay of 2μsec to be generated for the first byte transmitted — all subsequent bytes will have a delay of 500 nsec.

$$T_1(\text{High Speed})\ \mu\text{sec} = \frac{N_F}{2f_C} + t_{SYNC}$$

Thus, the shortest T_1 is achieved by setting $N_F=1$ using an 8 MHz clock with a 50% duty cycle clock (t_{SYNC}<63 nsec):

$$T_{1(HS)} = \frac{1}{2x8} + 0.063 = 125 \text{ nsec max.}$$

AUXILIARY REGISTER A

Auxiliary Register A is a "hidden" 5-bit register which is used to enable some of the 8291A features. Whenever a 100 $A_4A_3A_2A_1A_0$ byte is written into the Auxiliary Register, it is loaded with the data $A_4A_3A_2A_1A_0$. Setting the respective bits to "1" enables the following features.

A_0— RFD Holdoff on all Data: If the 8291A is listening, RFD will not be sent true until the "finish handshake" auxiliary command is issued by the microprocessor. The holdoff will be in effect for each data byte.

A_1—RFD Holdoff on End: This feature enables the holdoff on EOI or EOS (if enabled). However, no holdoff will be in effect on any other data bytes.

A_2—End on EOS Received: Whenever the byte in the Data In Register matches the byte in the EOS Register, the END interrupt bit will be set in the Interrupt Status 1 Register.

A_3—Output EOI on EOS Sent: Any occurrence of data in the Data Out Register matching the EOS Register causes the EOI line to be sent true along with the data.

A_4—EOS Binary Compare: Setting this bit causes the EOS Register to function as a full 8-bit word. When it is not set, the EOS Register is a 7-bit word (for ASCII characters).

If $A_0=A_1=1$, a special "continuous Acceptor Handshake cycling" mode is enabled. This mode should be used only in a controller system configuration, where both the 8291A and the 8292 are used. It provides a continuous cycling through the Acceptor Handshake state diagram, requiring no local messages from the microprocessor; the rdy local message is automatically generated when in ANRS. As such, the 8291A Acceptor Handshake serves as the controller Acceptor Handshake. Thus, the controller cycles through the Acceptor Handshake without delaying the data transfer in progress. When the tcs local message is executed, the 8291A should be taken out of the "continuous AH cycling" mode, the GPIB will hang up in ANRS, and a BI interrupt will be generated to indicate that control may be taken. A simpler procedure may be used when a "tcs on end of block" is executed; the 8291A may stay in "continuous AH cycling". Upon the end of a block (EOI or EOS received), a holdoff is generated, the GPIB hangs up in ANRS, and control may be taken.

AUXILIARY REGISTER B

Auxiliary Register B is a "hidden" 4-bit register which is used to enable some of the features of the 8291A. Whenever a 101 $B_4B_3B_2B_1B_0$ is written into the Auxiliary Mode Register, it is loaded with the data $B_4B_3B_2B_1B_0$. Setting the respective bits to "1" enables the following features:

B_0—Enable Undefined Command Pass Through: This feature allows any commands not recognized by the 8291A to be handled in software. If enabled, this feature will cause the 8291A to holdoff the handshake when an undefined command is received. The microprocessor must then read the command from the Command Pass Through Register and send the VSCMD auxiliary command. Until the VSCMD command is sent, the handshake holdoff will be in effect.

B_1—Send EOI in SPAS: This bit enables EOI to be sent with the status byte; EOI is sent true in Serial Poll Active State. Otherwise, EOI is sent false in SPAS.

B_2—Enable High Speed Data Transfer: This feature may be enabled when tri-state external transceivers are used. The data transfer rate is limited by T_1 delay time generated in the Source Handshake function, which is defined according to the type of transceivers used. When the "High Speed" feature is enabled, T_1 = 2 microseconds is generated for the first byte transmitted after each true to false transition of ATN. For all subsequent bytes, T_1 = 500 nanoseconds. Refer to the Internal Counter section for an explanation of T_1 duration as a function of B_2 and of clock frequency.

B_3—Enable Active Low Interrupt: Setting this bit causes the polarity of the INT pin to be reversed, providing an output signal compatible with Intel's MCS-48® Family. Interrupt registers are not affected by this bit.

B_4—Enable RFD Holdoff on GET or DEC: Setting this bit causes RFD to be held false until the "VSCMD" auxiliary command is written after GET, SDC, and DCL commands. This allows the device to hold off the bus until it has completed a clear or trigger similar to an unrecognized command.

PARALLEL POLL PROTOCOL

Writing a 011USP$_3$P$_2$P$_1$ into the Auxiliary Mode Register will enable (U=0) or disable (U=1) the 8291A for a parallel poll. When U=0, this command is the "lpe" (local poll enable) local message as defined in IEEE-488. The "S" bit is the sense in which the 8291A is enabled; only if the Parallel Poll Flag ("ist" local message) matches this bit will the Parallel Poll Response, PPR_N, be sent true (Response= S + ist). The bits $P_3P_2P_1$ specify which of the eight data lines PPR_N will be sent over. Thus, once the 8291A has been configured for Parallel Poll, whenever it senses both EOI and ATN true, it will automatically compare its PP flag with the sense bit and send PPR_N true or false according to the comparison.

If a PP2* implementation is desired, the "lpe" and "ist" local messages are all that are needed. Typically, the user will configure the 8291A for Parallel Poll immediately after initialization. During normal operation the microprocessor will set or clear the Parallel Poll Flag (ist) according to the device's need for service. Consequently the 8291A will be set up to give the proper response to IDY (EOI · ATN) without directly involving the microprocessor.

If a PP1* implementation is desired, the undefined command features of the 8291A must be used. In PP1, the 8291A is indirectly configured for Parallel Poll by the active controller on the GPIB. The sequence at the 8291A being enabled or disabled remotely is as follows:

1. The PPC message is received and is loaded into the Command Pass Through Register as an undefined command. A CPT Interrupt is sent to the microprocessor; the handshake is automatically held off.

2. The microprocessor reads the CPT Register and sends VSCMD to the 8291A, releasing the handshake.

3. Having received an undefined primary command, the 8291A is set up to receive an undefined secondary command (the PPE or PPD message). This message is also received into the CPT Register, the handshake is held off, and the CPT interrupt is generated.

NOTE: *As defined in IEEE Standard 488.

4. The microprocessor reads the PPE or PPD message and writes the command into the Auxiliary Mode Register (bit 7 should be cleared first). Finally, the microprocessor sends VSCMD and the handshake is released.

End of Sequence (EOS) Register

EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0

EOS REGISTER

The EOS Register and its features offer an alternative to the "Send EOI" auxiliary command. A seven or eight bit byte (ASCII or binary) may be placed in the register to flag the end of a block or read. The type of EOS byte to be used is selected in Auxiliary Register bit A_4.

If the 8291A is a listener, and the "End on EOS Received" is enabled with bit A_2, then an END interrupt is generated in the Interrupt Status 1 Register whenever the byte in the Data-In Register matches the byte in the EOS Register.

If the 8291A is a talker, and the "Output EOI on EOS Sent" is enabled with bit A_3, then the EOI line is sent true with the next byte whenever the contents of the Data Out Register match the EOS register.

Reset Procedure

The 8291A is reset to an initialization state either by a pulse applied to its Reset pin, or by a reset auxiliary command (02H written into the Auxiliary Command Register). The following conditions are caused by a reset pulse (or local reset command):

1. A "pon" local message as defined by IEEE-488 is held true until the initialization state is released.
2. The Interrupt Status Registers are cleared (not Interrupt Enable Registers).
3. Auxiliary Registers A and B are cleared.
4. The Serial Poll Mode Register is cleared.
5. The Parallel Poll Flag is cleared.
6. The EOI bit in the Address Status Register is cleared.
7. N_F in the Internal Counter is set to 8 MHz. This setting causes the longest possible T_1 delay to be generated in the Source Handshake (16 μsec for 1 MHz clock).
8. The rdy local message is sent.

The initialization state is released by an "immediate execute pon" command (00H written into the Auxiliary Command Register).

The suggested initialization sequence is:

1. Apply a reset pulse or send the reset auxiliary command.
2. Set the desired intial conditions by writing into the Interrupt Enable, Serial Poll Mode, Address Mode, Address 0/1, and EOS Registers. Auxiliary Registers A and B, and the internal counter should also be initialized.
3. Send the "immediate execute pon" auxiliary command to release the initialization state.
4. If a PP2 Parallel Poll implementation is to be used the "lpe" local message may be sent, enabling the 8291A for a Parallel Poll Response on an assigned line. (Refer to the section on Parallel Poll Protocol.)

Using DMA

The 8291A may be connected to the Intel® 8237 or 8257 DMA Controllers or the 8089 I/O Processor for DMA operation. The 8237 will be used to refer to any DMA controller. The DREQ pin of the 8291A requests a DMA byte transfer from the 8237. It is set by BO or BI flip flops, enabled by the DMAO and DMAI bits in the Interrupt Enable 2 Register. (After reading, the INT1 register BO and BI interrupts will be cleared but not BO and BI in DREQ equation.)

The $\overline{DACK}$ pin is driven by the 8237 in response to the DMA request. When $\overline{DACK}$ is true (active low) it sets $\overline{CS}$= RS0= RS1= RS2=0 such that the $\overline{RD}$ and $\overline{WR}$ signals sent by the 8237 refer to the Data In and Data Out Registers. Also, the DMA request line is reset by $\overline{DACK}$ ($\overline{RD}$ + $\overline{WR}$).

DMA input sequence:

1. A data byte is accepted from the GPIB by the 8291A.
2. A BI interrupt is generated and DREQ is set.
3. $\overline{DACK}$ and $\overline{RD}$ are driven by the 8237, the contents of the Data In Register are transferred to the system bus, and DREQ is reset.
4. The 8291A sends RFD true on the GPIB and proceeds with the Acceptor Handshake protocol.

DMA output sequence:

1. A BO interrupt is generated (indicating that a byte should be output) and DREQ is asserted.

2. $\overline{DACK}$ and $\overline{WR}$ are driven by the 8237, a byte is transferred from the MCS bus into the Data Out Register, and DREQ is reset.
3. The 8291A sends DAV true on the GPIB and proceeds with the Source Handshake protocol.

It should be noted that each time the device is addressed (MTA + MLA + ton + lon), the Address Status Register should be read, and the 8237 should be initialized accordingly. (Refer to the 8237 or 8257 Data Sheets.)

APPLICATION BRIEF

System Configuration

MICROPROCESSOR BUS CONNECTION

The 8291A is 8048/49, 8051, 8080/85, and 8086/88 compatible. The three address pins (RS_0, RS_1, RS_2) should be connected to the non-multiplexed address bus (for example: A_8, A_9, A_{10}). In case of 8080, any address lines may be used. If the three lowest address bits are used (A_0, A_1, A_2), then they must be demultiplexed first.

EXTERNAL TRANSCEIVERS CONNECTION

The 8293 GPIB Transceiver interfaces the 8291A directly to the IEEE-488 bus. The 8291A and two 8293's can be configured as a talker/listener (see Figure 6) or with the 8292 as a talker/listener/controller (see Figure 7). Absolutely no active or passive external components are required to comply with the complete IEEE-488 electrical specification.

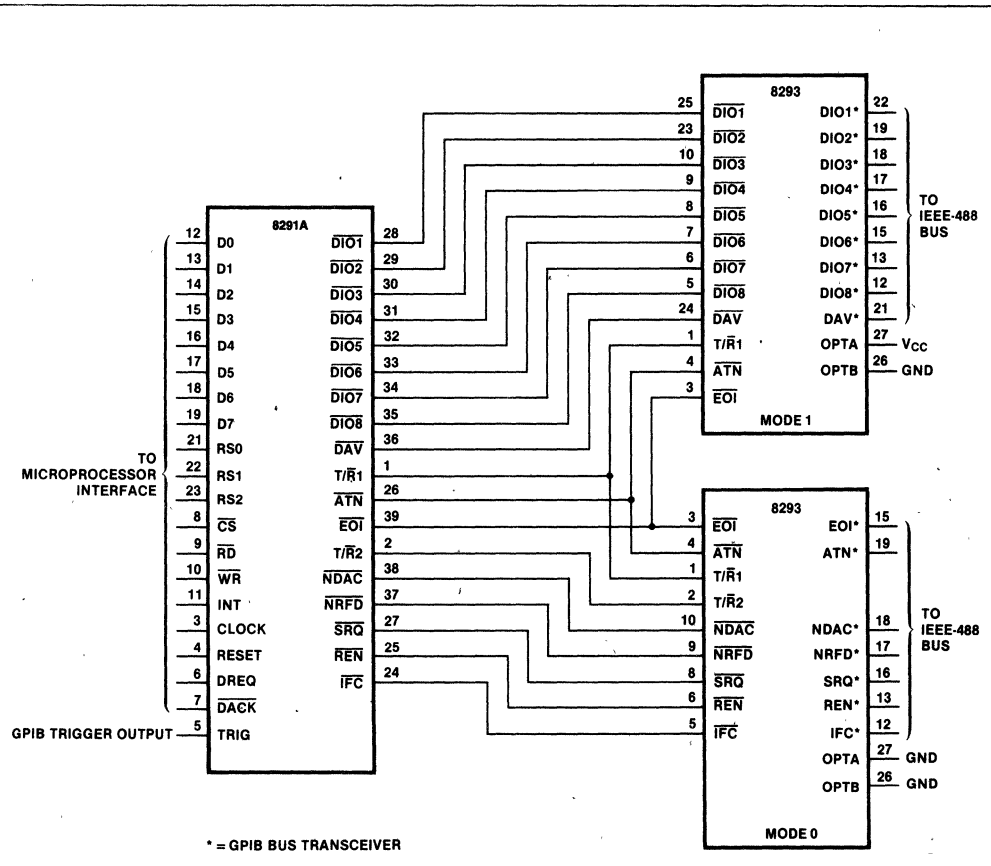


Figure 6. 8291A and 8293 System Configuration

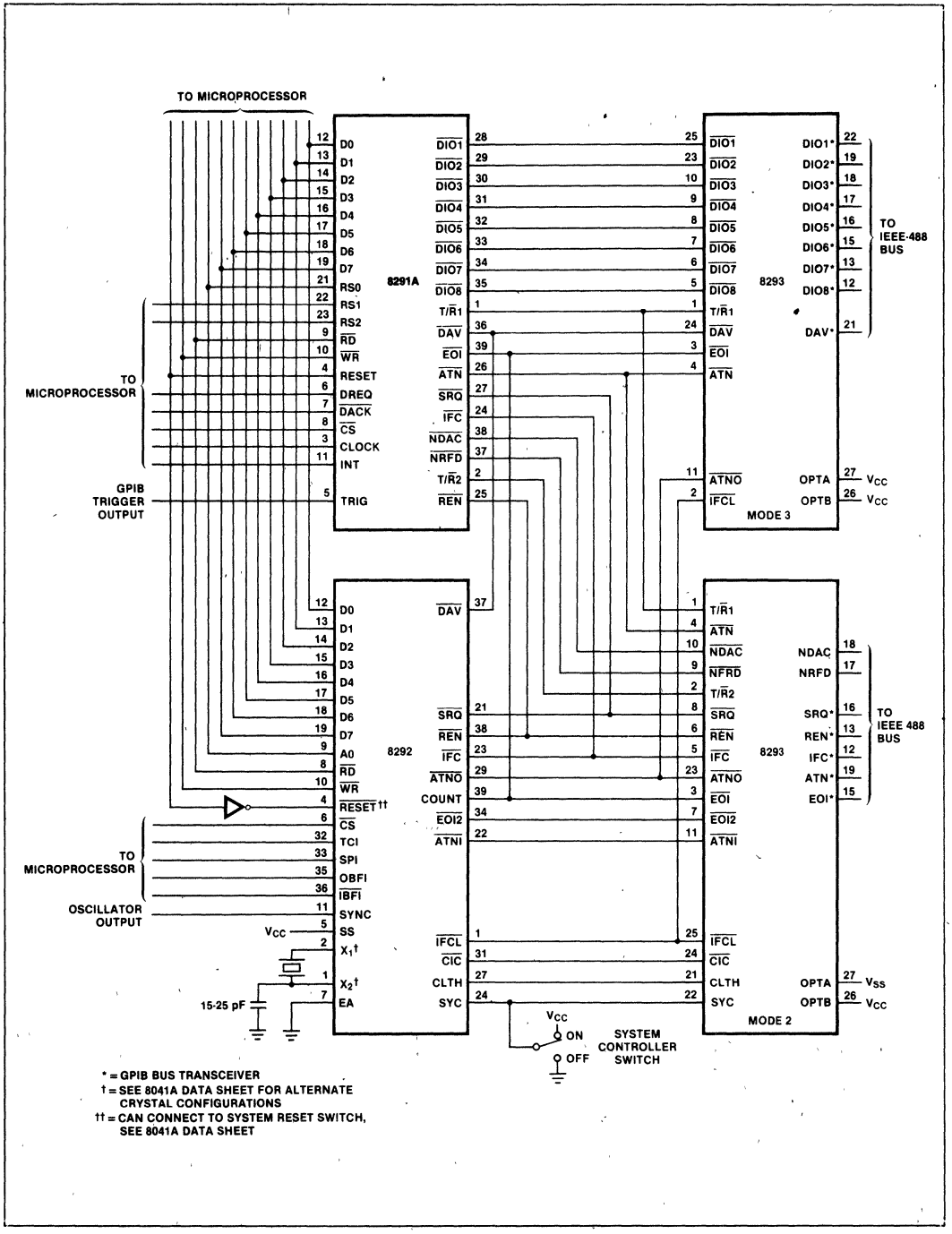


Figure 7. 8291A, 8292, and 8293 System Configuration

Start-Up Procedures

The following section describes the steps needed to initialize a typical 8291A system implementing a talker/listener interface and an 8291A/8292 system implementing a talker/listener/controller interface.

TALKER/LISTENER SYSTEM

Assume a general system configuration with the following features: (i) Polled system interface; (ii) Mode 1 addressing; (iii) same address for talker and listener; (iv) ASCII carriage return as the end-of-sequence (EOS) character; (v) EOI sent true with the last byte; and, (vi) 8 MHz clock.

Initialization. Initialization is accomplished with the following steps:

1. Pulse the RESET input or write 02H to the Auxiliary Mode Register.
2. Write 00H to the Interrupt Enable Registers 1 and 2. This disables interrupt and DMA.
3. Write 01H to the Address Mode Register to select Mode 1 addressing.
4. Write 28H to the Auxiliary Mode Register. This loads 8H to the Auxiliary Register A matching the 8 MHz clock input to the internal T1 delay counter to generate the delay meeting the IEEE spec.
5. Write the talker/listener address to the Address 0/1 register. The three most significant bits are zero.
6. Write an ASCII carriage return (0DH) to the EOS register.
7. Write 88H to the Auxiliary Mode Register to allow $\overline{EOI}$ to be sent true when the EOS character is sent.
8. Write 00H to the Auxiliary Mode Register. This writes the "Immediate Execute pon" message and takes the 8291A from the initialization state into the idle state. The 8291A will remain idle until the controller initiates some activity by driving $\overline{ATN}$ true.

Communication. The local CPU now polls the 8291A to determine which controller command has been received.

The controller addresses the 8291A by driving $\overline{ATN}$, placing MLA (My Listen Address) on the bus and driving $\overline{DAV}$. If the lower five bits of the MLA message match the address programmed into the Address 0/1 register, the 8291A is addressed to listen. It would be addressed to talk if the controller sent the MTA message instead of MLA.

The ADSC bit in the Interrupt Status 2 Register indicates that the 8291A has been addressed or unaddressed. The TA and LA bits in the Address Status Register indicate whether the 8291A is talker (TA=1), listener (LA=1), both (TA=LA=1) or unaddressed (TA=LA=0).

If the 8291A is addressed to listen, the local CPU can read the Data-In Register whenever the BI (Byte In) interrupt occurs in the Interrupt Status 1 Register. If the END bit in the same register is also set, either EOI or a data byte matching the pattern in the EOS register has been received.

In the talker mode, the CPU writes data into the Byte-Out Register on BO (Byte Out) true.

TALKER/LISTENER/CONTROLLER SYSTEM

Combined with the Intel 8292, the 8291A executes a complete IEEE-488-1978 controller function, The 8291A talks and listens via the data and handshake lines ($\overline{NRFD}$, $\overline{NDAC}$ and $\overline{DAV}$). The 8292 controls four of the five bus management lines ($\overline{IFC}$, $\overline{SRQ}$, $\overline{ATN}$ and $\overline{REN}$). $\overline{EOI}$, the fifth line, is shared. The 8291A drives and receives $\overline{EOI}$ when $\overline{EOI}$ is used as an end-of-block indicator. The 8292 drives $\overline{EOI}$ along with $\overline{ATN}$ during a parallel poll command.

Once again, assume a general system configuration with the following features: (i) Polled system interface; (ii) 8292 as the system controller and controller-in-charge; (iii) ASCII carriage return (0DH) as the EOS identifier; (iv) $\overline{EOI}$ sent with the last character; and, (v) an external buffer (8282) used to monitor the TCI line.

Initialization. In order to send a command across the GPIB, the 8292 has to drive $\overline{ATN}$, and the 8291A has to drive the data lines. Both devices therefore need initialization.

To initialize the 8292:

1. Pulse the RESET input. The 8292 will initially drive all outputs high. TCI, SPI, OBFI, IBFI and CLTH will then go low. The Interrupt Status, Interrupt Mask, Error Flag, Error Mask and Timeout registers will be cleared. The interrupt counter will be disabled and loaded with 255. The 8292 will then monitor the status of the SYC pin. If high, the 8292 will pulse IFC true for at least 100μs in compliance with the IEEE-488-1978 standard. It will then take control by asserting $\overline{ATN}$.

To initialize the 8291A, the following is necessary:

1. Write 00H to Interrupt Enable registers 1 and 2. This disables interrupt and DMA.

2. With the 8292 as the controller-in-charge, it is impossible to address the 8292 via the GPIB. Therefore, the ton or lon modes of the 8291A must be used. To send comands, set the 8291A in the ton mode by writing 80H to the Address Mode Register.
3. Write 26H to the Auxiliary Mode Register to match the T1 data settling time to the 6 MHz clock input.
4. Write an ASCII carriage return (0DH) to the EOS Register.
5. Write 84H to the Auxiliary Mode Register in order to enable "Output EOI on EOS sent" and thus send EOI with the last character.
6. Write 00H—Immediate Execute pon—to the Auxiliary Mode Register to put the 8291A in the idle state.

Communication. Since the 8291A is in the ton mode, a BO interrupt is generated as soon as the immediate Execute pon command is written. The CPU writes the command into the Data Out Register, and repeats it on BO becoming true for as many commands as necessary. $\overline{ATN}$ remains continuously true unless the GTSB (Go To Standby) command is sent to the 8292.

$\overline{ATN}$ has to be false in order to send data rather than commands from the controller. To do this, the following steps are needed:

1. Enable the TCI interrupt if not already enabled.
2. Wait for IBF (Input Buffer Full) in the 8292 Interrupt Status Register to be reset.
3. Write the GTSB (F6H) command to the 8292 Command Field Register.
4. Read the 8282 and wait for TCI to be true.
5. Write the ton (80H) and pon (00H) command to the 8291A Address Mode Register and Auxiliary Mode Registers respectively.
6. Wait for the BO interrupt to be set in the 8291A.
7. Write the data to the 8291A Data-Out Register.

Identically, the user could command the controller to listen rather than talk. To do that, write lon (40H) instead of ton into the Address Mode Register. Then wait for BI rather than BO to go true. Read the data Register.

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 0.65 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS [V_{CC} = 5V ±10%, T_A = 0°C to 70°C (Commercial)]

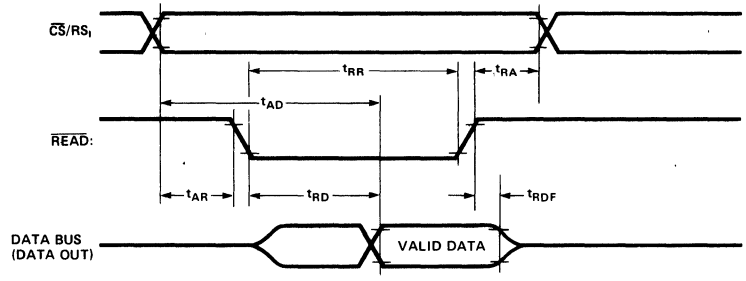
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL}=2mA (4mA for TR1 pin)
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA (-150μA for SRQ pin)
V_{OH-INT}	Interrupt Output High Voltage	2.4		V	I_{OH}=-400μA
		3.5		V	I_{OH}=-50μA
I_{IL}	Input Leakage		10	μA	V_{IN}=0V to V_{CC}
I_{IOFL}	Output Leakage Current		±10	μA	V_{OUT}=0.45V, V_{CC}
I_{CC}	V_{CC} Supply Current		120	mA	T_A=0°C

A.C. CHARACTERISTICS [V_{CC} = 5V ±10%, T_A = 0°C to 70°C (Commercial)]

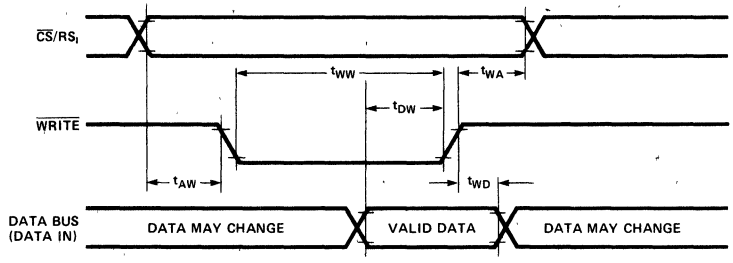
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	Address Stable Before $\overline{READ}$	0		nsec	
t_{RA}	Address Hold After $\overline{READ}$	0		nsec	
t_{RR}	$\overline{READ}$ width	140		nsec	
t_{AD}	Address Stable to Data Valid		250	nsec	
t_{RD}	$\overline{READ}$ to Data Valid		100	nsec	
t_{RDF}	Data Float After $\overline{READ}$	0	60	nsec	
t_{AW}	Address Stable Before $\overline{WRITE}$	0		nsec	
t_{WA}	Address Hold After $\overline{WRITE}$	0			
t_{WW}	$\overline{WRITE}$ Width	170		nsec	
t_{DW}	Data Set Up Time to the Trailing Edge of $\overline{WRITE}$	130		nsec	
t_{WD}	Data Hold Time After $\overline{WRITE}$	0		nsec	
t_{DKDR4}	$\overline{RD}$↓ or $\overline{WR}$↓ to DREQ↓		130	nsec	
t_{DKDA6}	RD↓ to Valid Data (D_0–D_7)		200	nsec	$\overline{DACK}$↓ to $\overline{RD}$↓ 0 ≤t ≤50nsec

WAVEFORMS

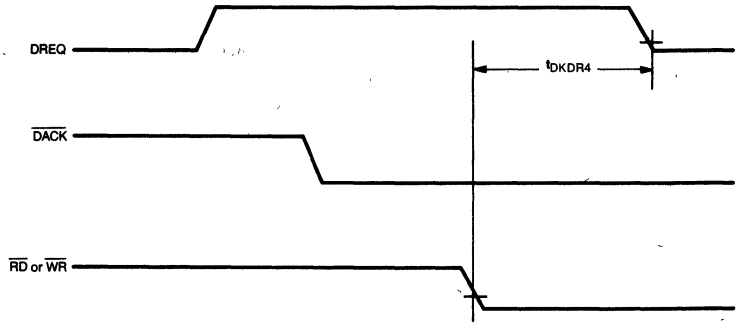
READ



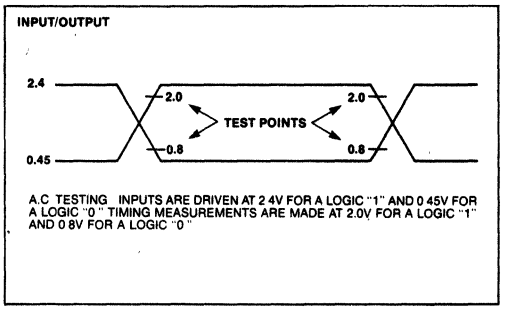
WRITE

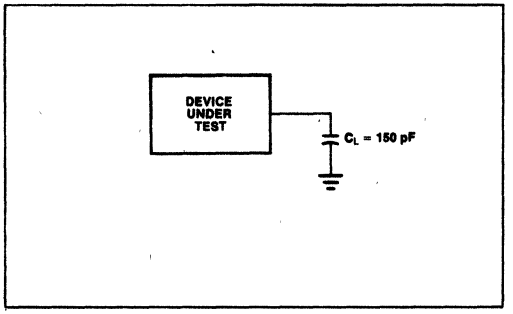


DMA



A.C. TIMING MEASUREMENT POINTS AND LOAD CONDITIONS





GPIB TIMINGS[1]

Symbol	Parameter	Max.	Units	Test Conditions
TEOT13[2]	$\overline{EOI}\downarrow$ to TR1↑	135	nsec	PPSS, ATN=0.45V
TEOD16	$\overline{EOI}\downarrow$ to $\overline{DIO}$ Valid	155	nsec	PPSS, ATN=0.45V
TEOT12	$\overline{EOI}$↑ to TR1↓	155	nsec	PPSS, ATN=0.45V
TATND4	$\overline{ATN}\downarrow$ to $\overline{NDAC}\downarrow$	155	nsec	TACS, AIDS
TATT14	$\overline{ATN}\downarrow$ to TR1↓	155	nsec	TACS, AIDS
TATT24	$\overline{ATN}\downarrow$ to TR2↓	155	nsec	TACS, AIDS
TDVND3-C	$\overline{DAV}\downarrow$ to $\overline{NDAC}$↑	650	nsec	AH, CACS
TNDDV1	$\overline{NDAC}$↑ to $\overline{DAV}$↑	350	nsec	SH, STRS
TNRDR1	$\overline{NRFD}$↑ to DREQ↑	400	nsec	SH
TDVDR3	$\overline{DAV}\downarrow$ to DREQ↑	600	nsec	AH, LACS, ATN=2.4V
TDVND2-C	$\overline{DAV}$↑ to $\overline{NDAC}\downarrow$	350	nsec	AH, LACS
TDVNR1-C	$\overline{DAV}$↑ to $\overline{NRFD}$↑	350	nsec	AH, LACS, rdy=True
TRDNR3	$\overline{RD}\downarrow$ to $\overline{NRFD}$↑	500	nsec	AH, LACS
TWRD15	$\overline{WR}$↑ to $\overline{DIO}$ Valid	280	nsec	SH, TACS, RS=0.4V
TWREO5	$\overline{WR}$↑ to $\overline{EOI}$ Valid	350	nsec	SH, TACS
TWRDV2	$\overline{WR}$↑ to $\overline{DAV}\downarrow$	830 + t_{SYNC}	nsec	High Speed Transfers Enabled, $N_F = f_C$, $t_{SYNC} = 1/2 \cdot f_C$

NOTES:
1. All GPIB timings are at the pins of the 8291A.
2. The last number in the symbol for any GPIB timing parameter is chosen according to the transition directions of the reference signals. The following table describes the numbering scheme.

↑ to ↑	1
↑ to ↓	2
↓ to ↑	3
↓ to ↓	4
↑ to VALID	5
↓ to VALID	6

APPENDIX A

MODIFIED STATE DIAGRAMS

Figure A-1 presents the interface function state diagrams. It is derived from IEEE Std. state diagrams, with the following changes:

A. The 8291A supports the complete set of IEEE-488 interface functions except for the controller. These include: SH1, AH1, T5, TE5, L3, LE3, SR1, RL1, PP1, DC1, DT1, and C0.

B. Addressing modes included in T,L state diagrams.

Note that in Mode 3, MSA, OSA are generated only after secondary address validity check by the microprocessor (APT interrupt).

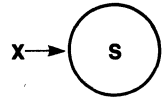
C. In these modified state diagrams, the IEEE-488-1978 convention of negative (low true) logic is followed. This should not be confused with the Intel pin- and signal-naming convention based on positive logic. Thus, while the state diagrams below carry low true logic, the signals described elsewhere in this data sheet are consistent with Intel notation and are based on positive logic.

Level	Logic	Convention	
		IEEE-488	Intel
0	T	DAV	$\overline{DAV}$
1	F	$\overline{DAV}$	DAV
0	T	NDAC	$\overline{NDAC}$
1	F	$\overline{NDAC}$	NDAC
0	T	NRFD	$\overline{NRFD}$
1	F	$\overline{NRFD}$	NRFD

Consider the condition when the Not-Ready-For-Data signal (pin 37) is active. Intel indicates this active low signal with the symbol $\overline{NRFD}$ ($V_{OUT} \leq V_{OL}$ for AH; $V_{IN} \leq V_{IL}$ for SH). The IEEE-488-1978 Standard, in its state diagrams, indicates the active state of this signal (True condition) with NRFD.

D. All remote multiline messages decoded are conditioned by ACDS. The multiplication by ACDS is not drawn to simplify the diagrams.

E. The symbol



indicates:

1. When event X occurs, the function returns to state S.
2. X overrides any other transition condition in the function.

Statement 2 simplifies the diagram, avoiding the explicit use of X to condition all transitions from S to other states.

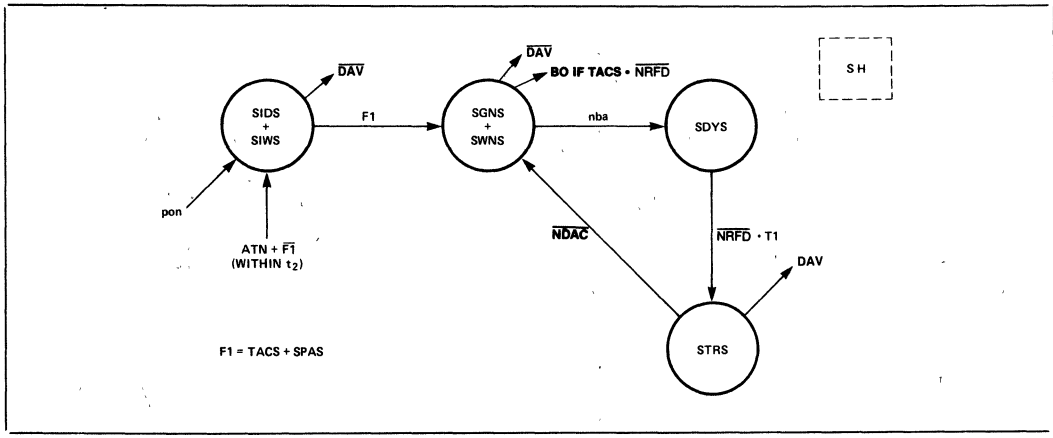


Figure A-1. 8291A State Diagrams (Continued next page)

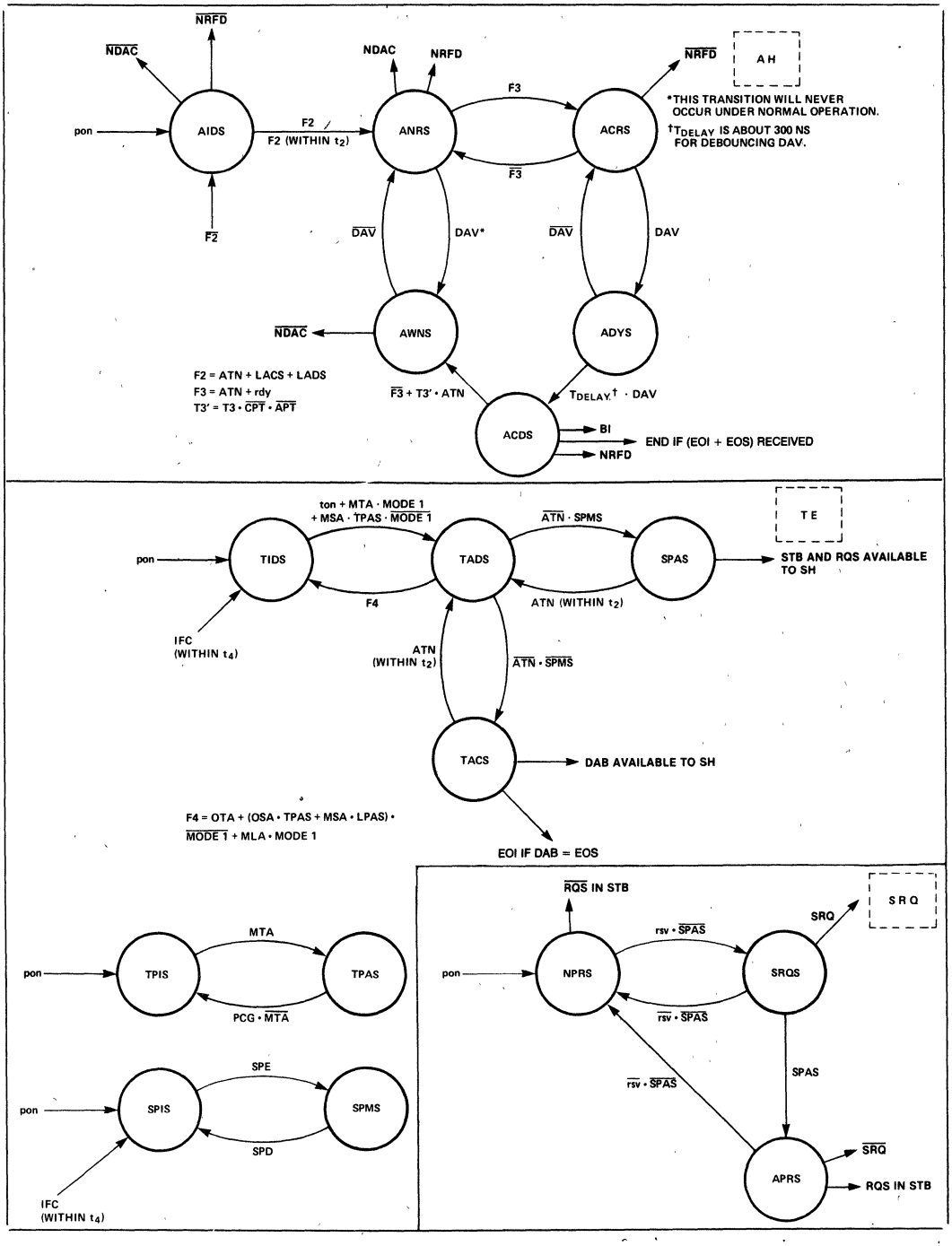


Figure A-1. 8291A State Diagrams (Continued next page)

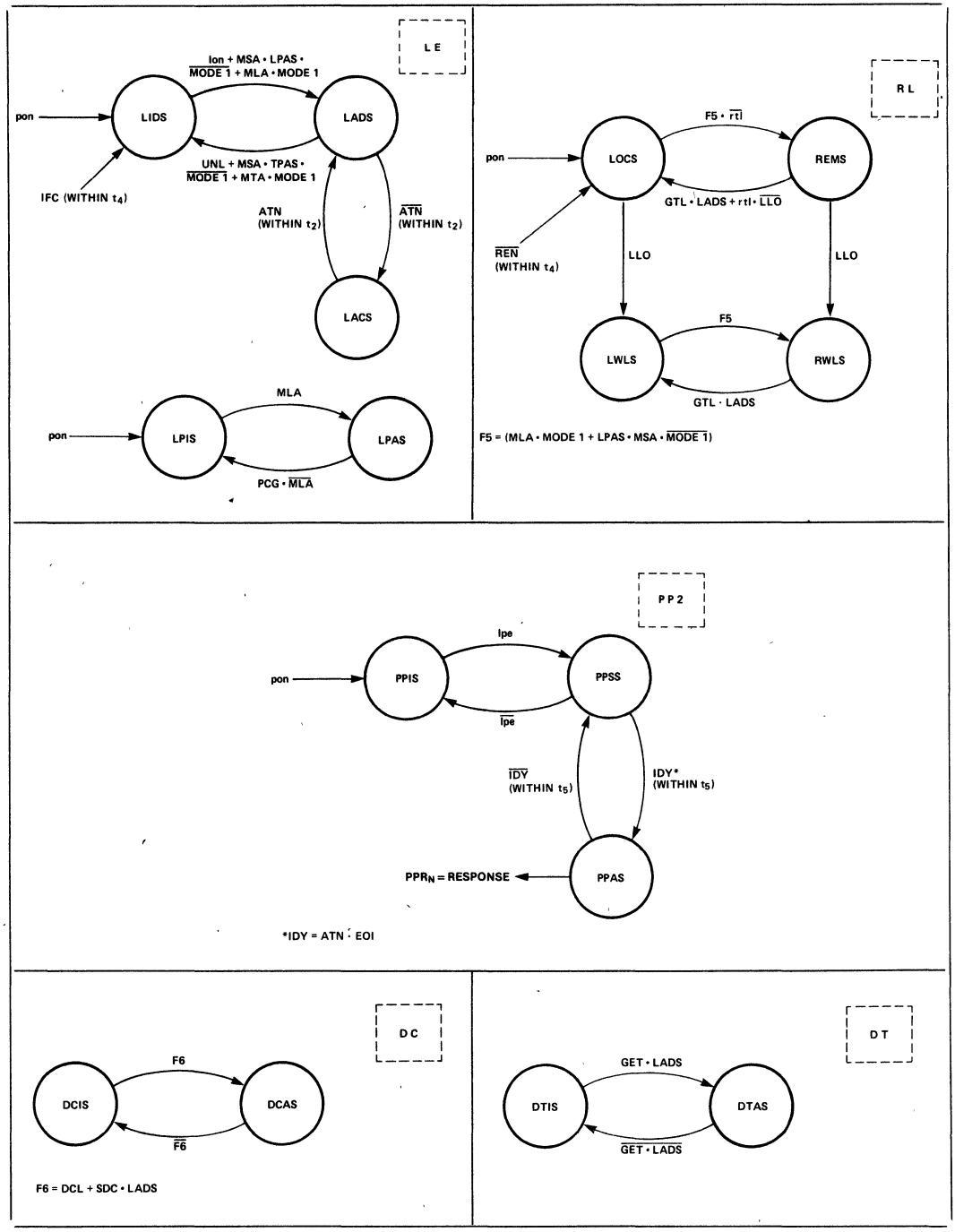


Figure A-1. 8291A State Diagrams

APPENDIX B

Table B-1. IEEE 488 Time Values

Time Value Identifier[1]	Function (Applies to)	Description	Value
T_1	SH	Settling Time for Multiline Messages	$\geq 2\mu s$[2]
t_2	LC,$\overline{IC}$,SH,AH,T,L	Response to ATN	$\leq$ 200ns
T_3	AH	Interface Message Accept Time[3]	> 0[4]
t_4	T,TE,L,LE,C,CE	Response to IFC or REN False	$< 100\mu s$
t_5	PP	Response to ATN+EOI	$\leq$ 200ns
T_6	C	Parallel Poll Execution Time	$\geq 2\mu s$
T_7	C	Controller Delay to Allow Current Talker to see ATN Message	$\geq$ 500 ns
T_8	C	Length of IFC or REN False	$> 100\mu s$
T_9	C	Delay for EOI[5]	$\geq 1.5\mu s$[6]

NOTES:

[1]Time values specified by a lower case t indicate the maximum time allowed to make a state transition. Time values specified by an upper case T indicate the minimum time that a function must remain in a state before exiting.

[2]If three-state drivers are used on the $\overline{DIO}$, $\overline{DAV}$, and $\overline{EOI}$ lines, T_1 may be:

1. $\geq$ 1100 ns.
2. Or $\geq$ 700 ns if it is known that within the controller ATN is driven by a three-state driver.
3. Or $\geq$ 500ns for all subsequent bytes following the first sent after each false transition of ATN (the first byte must be sent in accordance with (1) or (2).
4. Or $\geq$ 350ns for all subsequent bytes following the first sent after each false transition of ATN under conditions specified in Section 5.2.3 and warning note. See IEEE Standard 488.

[3]Time required for interface functions to accept, not necessarily respond to interface messages.

[4]Implementation dependent.

[5]Delay required for $\overline{EOI}$, $\overline{NDAC}$, and $\overline{NRFD}$ signal lines to indicate valid states.

[6]$\geq$ 600 ns for three-state drivers.

APPENDIX C
THE THREE-WIRE HANDSHAKE

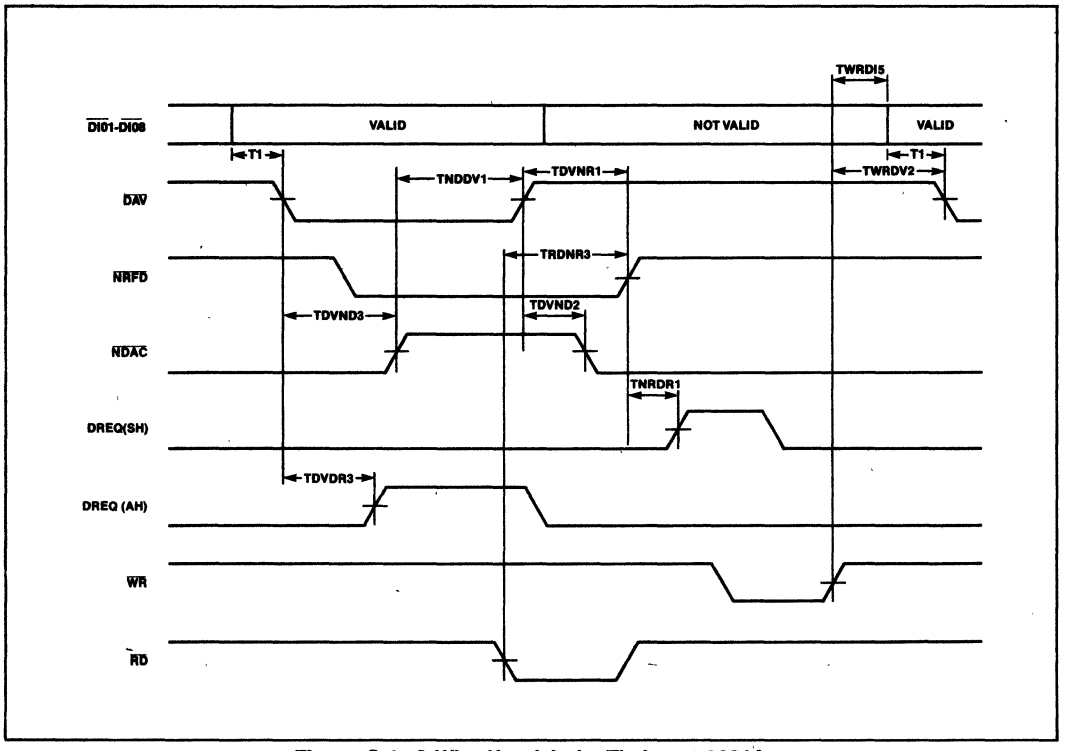


Figure C-1. 3-Wire Handshake Timing at 8291A

8292
GPIB CONTROLLER

- **Complete IEEE Standard 488 Controller Function**
- **Interface Clear (IFC) Sending Capability Allows Seizure of Bus Control and/or Initialization of the Bus**
- **Responds to Service Requests (SRQ)**
- **Sends Remote Enable (REN), Allowing Instruments to Switch to Remote Control**
- **Complete Implementation of Transfer Control Protocol**
- **Synchronous Control Seizure Prevents the Destruction of Any Data Transmission in Progress**
- **Connects with the 8291 to Form a Complete IEEE Standard 488 Interface Talker/Listener/Controller**

The 8292 GPIB Controller is a microprocessor-controlled chip designed to function with the 8291 GPIB Talker/Listener to implement the full IEEE Standard 488 controller function, including transfer control protocol. The 8292 is a pre-programmed Intel® 8041A.

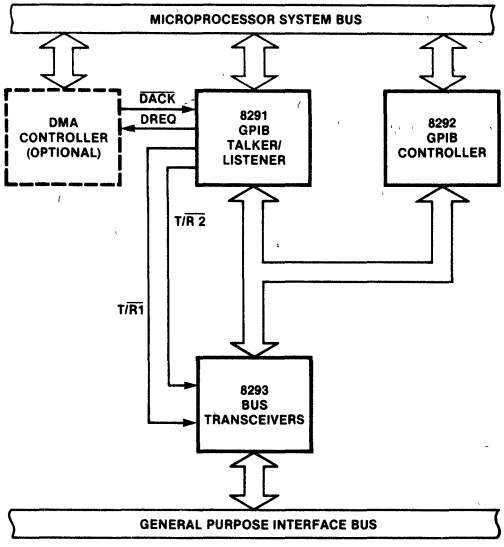


Figure 1. 8291, 8292 Block Diagram

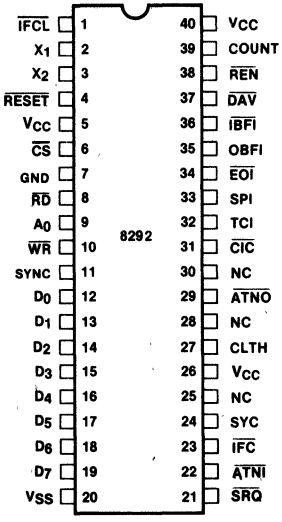


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
$\overline{IFCL}$	1	I	**IFC Received (Latched):** The 8292 monitors the IFC Line (when not system controller) through this pin.
X_1, X_2	2,3	I	**Crystal Inputs:** Inputs for a crystal, LC or an external timing signal to determine the internal oscillator frequency.
$\overline{RESET}$	4	I	**Reset:** Used to initialize the chip to a known state during power on.
$\overline{CS}$	6	I	**Chip Select Input:** Used to select the 8292 from other devices on the common data bus.
$\overline{RD}$	8	I	**Read Enable:** Allows the master CPU to read from the 8292.
A_0	9	I	**Address Line:** Used to select between the data bus and the status register during read operations and to distinguish between data and commands written into the 8292 during write operations.
$\overline{WR}$	10	I	**Write Enable:** Allows the master CPU to write to the 8292.
SYNC	11	O	**Sync:** 8041A instruction cycle synchronization signal; it is an output clock with a frequency of XTAL ÷ 15.
D_0–D_7	12-19	I/O	**Data:** 8 bidirectional lines used for communication between the central processor and the 8292's data bus buffers and status register.
V_{SS}	7, 20	P.S.	**Ground:** Circuit ground potential.
$\overline{SRQ}$	21	I	**Service Request:** One of the IEEE control lines. Sampled by the 8292 when it is controller in charge. If true, SPI interrupt to the master will be generated.
$\overline{ATNI}$	22	I	**Attention In:** Used by the 8292 to monitor the GPIB ATN control line. It is used during the transfer control procedure.
$\overline{IFC}$	23	I/O	**Interface Clear:** One of the GPIB management lines, as defined by IEEE Std. 488-1978, places all devices in a known quiescent state.
SYC	24	I	**System Conroller:** Monitors the system controller switch.
CLTH	27	O	**Clear Latch:** Used to clear the $\overline{IFCR}$ latch after being recognized by the 8292. Usually low (except after hardware Reset), it will be pulsed high when $\overline{IFCR}$ is recognized by the 8292.
$\overline{ATNO}$	29	O	**Attention Out:** Controls the ATN control line of the bus through external logic for tcs and tca procedures. (ATN is a GPIB control line, as defined by IEEE Std. 488-1978.)
V_{CC}	5, 26, 40	P.S.	**Voltage:** +5V supply input ±10%.
COUNT	39	I	**Event Count:** When enabled by the proper command the internal counter will count external events through this pin. High to low transition will increment the internal counter by one. The pin is sampled once per three internal instruction cycles (7.5μsec sample period when using 5 MHz XTAL). It can be used for byte counting when connected to NDAC, or for block counting when connected to the EOI.
$\overline{REN}$	38	O	**Remote Enable:** The Remote Enable bus signal selects remote or local control of the device on the bus. A GPIB bus management line, as defined by IEEE Std. 488-1978.
$\overline{DAV}$	37	I/O	**Data Valid:** Used during parallel poll to force the 8291 to accept the parallel poll status bits. It is also used during the tcs procedure.
$\overline{IBFI}$	36	O	**Input Buffer Not Full:** Used to interrupt the central processor while the input buffer of the 8292 is empty. This feature is enabled and disabled by the interrupt mask register.
OBFI	35	O	**Output Buffer Full:** Used as an interrupt to the central processor while the output buffer of the 8292 is full. The feature can be enabled and disabled by the interrupt mask register.
$\overline{EO12}$	34	I/O	**End Or Identify:** One of the GPIB management lines, as defined by IEEE Std. 488-1978. Used with ATN as Identify Message during parallel poll.
SPI	33	O	**Special Interrupt:** Used as an interrupt on events not initiated by the central processor.
TCI	32	O	**Task Complete Interrupt:** Interrupt to the control processor used to indicate that the task requested was completed by the 8292 and the information requested is ready in the data bus buffer.
CIC	31	O	**Controller In Charge:** Controls the S/R input of the SRQ bus transceiver. It can also be used to indicate that the 8292 is in charge of the GPIB bus.

FUNCTIONAL DESCRIPTION

The 8292 is an Intel 8041A which has been programmed as a GPIB Controller interface element. It is used with the 8291 GPIB Talker/Listener and two 8293 GPIB Transceivers to form a complete IEEE-488 Bus Interface for a microprocessor. The electrical interface is performed by the transceivers, data transfer is done by the talker/listener, and control of the bus is done by the 8292. Figure 3 is a typical controller interface using Intel's GPIB peripherals.

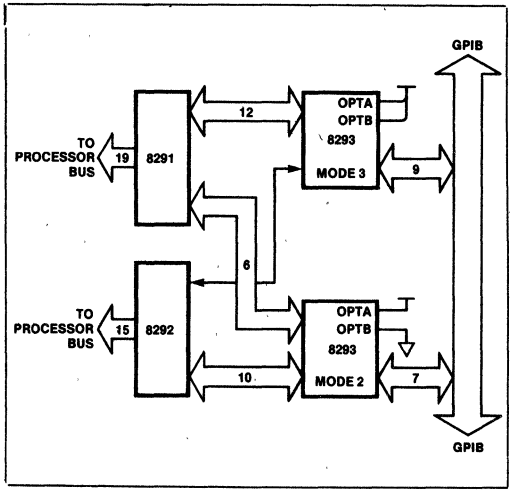


Figure 3. Talker/Listener/Controller Configuration

The internal RAM in the 8041A is used as a special purpose register bank for the 8292. Most of these registers (except for the interrupt flag) can be accessed through commands to the 8292. Table 2 identifies the registers used by the 8292 and how they are accessed.

Interrupt Status Register

D_7							D_0
SYC	ERR	SRQ	EV	X	IFCR	IBF	OBF

The 8292 can be configured to interrupt the microprocessor on one of several conditions. Upon receipt of the interrupt the microprocessor must read the 8292 interrupt status register to determine which event caused the interrupt, and then the appropriate subroutine can be performed. The interrupt status register is read with A_0 high. With the exception of OBF and IBF, these interrupts are enabled or disabled by the SPI interrupt mask. OBF and IBF have their own bits in the interrupt mask (OBFI and $\overline{IBFI}$).

OBF Output Buffer Full. A byte is waiting to be read by the microprocessor. This flag is cleared when the output data bus buffer is read.

IBF Input Buffer Full. The byte previously written by the microprocessor has not been read yet by the 8292. If another byte is written to the 8292 before this flag clears, data will be lost. IBF is cleared when the 8292 reads the data byte.

IFCR Interface Clear Received. The GPIB system controller has set IFC. The 8292 has become idle and is no longer in charge of the bus. The flag is cleared when the IACK command is issued.

EV Event Counter Interrupt. The requested number of blocks or data bytes has been transferred. The EV interrupt flag is cleared by the IACK command.

SRQ Service Request. Notifies the 8292 that a service request (SRQ) message has been received. It is cleared by the IACK command.

ERR Error occurred. The type of error can be determined by reading the error status register. This interrupt flag is cleared by the IACK command.

SYC System Controller Switch Change. Notifies the processor that the state of the system controller switch has changed. The actual state is contained in the GPIB Status Register. This flag is cleared by the IACK command.

Table 2. 8292 Registers

READ FROM 8292

	D_7							D_0	A_0
INTERRUPT STATUS	SYC	ERR	SRQ	EV	X	IFCR	IBF	OBF	1
ERROR FLAG	X	X	USER	X	X	$TOUT_3$	$TOUT_2$	$TOUT_1$	0*
CONTROLLER STATUS	CSBS	CA	X	X	SYCS	IFC	REN	SRQ	0*
GPIB (BUS) STATUS	REN	DAV	EOI	X	SYC	IFC	ANTI	SRQ	0*
EVENT COUNTER STATUS	D	D	D	D	D	D	D	D	0*
TIME OUT STATUS	D	D	D	D	D	D	D	D	0*

WRITE TO 8292

	D_7							D_0	A_0
INTERRUPT MASK	1	SPI	TCI	SYC	OBFI	$\overline{IBFI}$	0	SRQ	0
ERROR MASK	0	0	USER	0	0	$TOUT_3$	$TOUT_2$	$TOUT_1$	0
COMMAND FIELD	1	1	1	OP	C	C	C	C	1
EVENT COUNTER	D	D	D	D	D	D	D	D	0*
TIME OUT	D	D	D	D	D	D	D	D	0*

Note: These registers are accessed by a special utility command, see page 6.

Interrupt Mask Register

1	SPI	TCI	SYC	OBFI	$\overline{IBFI}$	0	SRQ
D_7							D_0

The Interrupt Mask Register is used to enable features and to mask the SPI and TCI interrupts. The flags in the Interrupt Status Register will be active even when masked out. The Interrupt Mask Register is written when A_0 is low and reset by the RINM command. When the register is read, D_1 and D_7 are undefined. An interrupt is enabled by setting the corresponding register bit.

SRQ Enable interrupts on SRQ received.

$\overline{IBFI}$ Enable interrupts on input buffer empty.

OBFI Enable interrupts on output buffer full.

SYC Enable interrupts on a change in the system controller switch.

TCI Enable interrupts on the task completed.

SPI Enable interrupts on special events.

NOTE: The event counter is enabled by the GSEC command, the error interrupt is enabled by the error mask register, and IFC cannot be masked (it will always cause an interrupt).

Controller Status Register

CSBS	CA	X	X	SYCS	IFC	REN	SRQ
D_7							D_0

The Controller Status Register is used to determine the status of the controller function. This register is accessed by the RCST command.

SRQ Service Request line active (CSRS).

REN Sending Remote Enable.

IFC Sending or receiving interface clear.

SYCS System Controller Switch Status (SACS).

CA Controller Active (CACS + CAWS + CSWS).

CSBS Controller Stand-by State (CSBS, CA) = (0,0) — Controller Idle

GPIB Bus Status Register

REN	DAV	EOI	X	SYC	IFC	ATNI	SRQ
D_7							D_0

This register contains GPIB bus status information. It can be used by the microprocessor to monitor and manage the bus. The GPIB Bus Register can be read using the RBST command.

Each of these status bits reflect the current status of the corresponding pin on the 8292.

SRQ Service Request

ATNI Attention In

IFC Interface Clear

SYC System Controller Switch

EOI End or Identify

DAV Data Valid

REN Remote Enable

Event Counter Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

The Event Counter Register contains the initial value for the event counter. The counter can count pulses on pin 39 of the 8292 (COUNT). It can be connected to EOI or NDAC to count blocks or bytes respectively during standby state. A count of zero equals 256. This register cannot be read, and is written using the WEVC command.

Event Counter Status Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

This register contains the current value in the event counter. The event counter counts back from the initial value stored in the Event Counter Register to zero and then generates an Event Counter Interrupt. This register cannot be written and can be read using a REVC command.

Time Out Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

The Time Out Register is used to store the time used for the time out error function. See the individual timeouts (TOUT1, 2, 3) to determine the units of this counter. This Time Out Register cannot be read, and it is written with the WTOUT command.

Time Out Status Register

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

This register contains the current value in the time out counter. The time out counter decrements from the original value stored in the Time Out Register. When zero is reached, the appropriate error interrupt is generated. If the register is read while none of the time out functions are active, the register will contain the last value reached the last time a function was active. The Time Out Status Register cannot be written, and it is read with the RTOUT command.

Error Flag Register

X	X	USER	X	X	$TOUT_3$	$TOUT_2$	$TOUT_1$
D_7							D_0

Four errors are flagged by the 8292 with a bit in the Error Flag Register. Each of these errors can be masked by the Error Mask Register. The Error Flag Register cannot be written, and it is read by the IACK command when the error flag in the Interrupt Status Register is set.

TOUT1 Time Out Error 1 occurs when the current controller has not stopped sending ATN after receiving the TCT message for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 1800 t_{CY}. After flagging the error, the 8292 will remain in a loop trying to take control until the current controller stops sending ATN or a new command is written by the microprocessor. If a new command is written, the 8292 will return to the loop after executing it.

TOUT2 Time Out Error 2 occurs when the transmission between the addressed talker and listener has not started for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 45 t_{CY}. This feature is only enabled when the controller is in the CSBS state.

TOUT3 Time Out Error 3 occurs when the handshake signals are stuck and the 8292 is not succeeding in taking control synchronously for the time period specified by the Time Out Register. Each count in the Time Out Register is at least 1800 t_{CY}. The 8292 will continue checking $\overline{ATNI}$ until it becomes true or a new command is received. After performing the new command, the 8292 will return to the $\overline{ATNI}$ checking loop.

USER User error occurs when request to assert IFC or REN was received and the 8292 was not the system controller.

Error Mask Register

0	0	USER	0	0	$TOUT_3$	$TOUT_2$	$TOUT_1$
D_7							D_0

The Error Mask Register is used to mask the interrupt from a particular type of error. Each type of error interrupt is enabled by setting the corresponding bit in the Error Mask Register. This register can be read with the RERM command and written with A_0 low.

Command Register

1	1	1	OP	C	C	C	C
D_7							D_0

Commands are performed by the 8292 whenever a byte is written with A_0 high. There are two categories of commands distinguished by the OP bit (bit 4). The first category is the operation command (OP = 1). These commands initiate some action on the interface bus. The second category is the utility commands (OP = 0). These commands are used to aid the communication between the processor and the 8292.

OPERATION COMMANDS

Operation commands initiate some action on the GPIB interface bus. It is using these commands that the control functions such as polling, taking and passing control, and system controller functions are performed.

F0 — SPCNI — Stop Counter Interrupts

This command disables the internal counter interrupt so that the 8292 will stop interrupting the master on event counter underflows. However, the counter will continue counting and its contents can still be used.

F1 — GIDL — Go To Idle

This command is used during the transfer of control procedure while transferring control to another controller. The 8292 will respond to this command only if it is in the active state. $\overline{ATNO}$ will go high, and $\overline{CIC}$ will be high so that this 8292 will no longer be driving the ATN line on the GPIB interface bus. TCI will be set upon completion.

F2 — RST — Reset

This command has the same effect as asserting the external reset on the 8292. For details, refer to the reset procedure described later.

F3 — RSTI — Reset Interrupts

This command resets any pending interrupts and clears the error flags. The 8292 will not return to any loop it was in (such as from the time out interrupts).

F4 — GSEC — Go To Standby, Enable Counting

The function causes $\overline{ATNO}$ to go high and the counter will be enabled. If the 8292 was not the active controller, this command will exit immediately. If the 8292 is the active controller, the counter will be loaded with the value stored in the Event Counter Register, and the internal interrupt will be enabled so that when the counter reaches zero, the SPI interrupt will be generated. SPI will be generated every 256 counts thereafter until the controller exits the standby state or the SPCNI command is written. An initial count of 256 (zero in the Event Counter Register) will be used if the WEVC command is not executed. If the data transmission does not start, a TOUT2 error will be generated.

F5 — EXPP — Execute Parallel Poll

This command initiates a parallel poll by asserting EOI when ATN is already active. TCI will be set at the end of the command. The 8291 should be previously configured as a listener. Upon detection of DAV true, the 8291 enters ACDS and latches the parallel poll response (PPR) byte into its data in register. The master will be interrupted by the 8291 BI interrupt when the PPR byte is available. No interrupts except the $\overline{IBFI}$ will be generated by the 8292. The 8292 will respond to this command only when it is the active controller.

F6 — GTSB — Go To Standby

If the 8292 is the active controller, $\overline{ATNO}$ will go high then TCI will be generated. If the data transmission does not start, a TOUT2 error will be generated.

F7 — SLOC — Set Local Mode

If the 8292 is the system controller, then REN will be asserted false and TCI will be set true. If it is not the system controller, the User Error bit will be set in the Error Flag Register.

F8 — SREM — Set Interface To Remote Control

This command will set REN true and TCI true if this 8292 is the system controller. If not, the User Error bit will be set in the Error Flag Register.

F9 — ABORT — Abort All Operation, Clear Interface

This command will cause IFC to be asserted true for at least 100 μsec if this 8292 is the system controller. If it is in CIDS, it will take control over the bus (see the TCNTR command).

FA — TCNTR — Take Control

The transfer of control procedure is coordinated by the master with the 8291 and 8292. When the master receives a TCT message from the 8291, it should issue the TCNTR command to the 8292. The following events occur to take control:

1. The 8292 checks to see if it is in CIDS, and if not, it exits.
2. Then $\overline{\text{ATNI}}$ is checked until it becomes high. If the current controller does not release ATN for the time specified by the Time Out Register, then a TOUT1 error is generated. The 8292 will return to this loop after an error or any command except the RST and RSTI commands.
3. After the current controller releases ATN, the 8292 will assert $\overline{\text{ATNO}}$ and $\overline{\text{CIC}}$ low.
4. Finally, the TCI interrupt is generated to inform the master that it is in control of the bus.

FC — TCASY — Take Control Asynchronously

TCAS transfers the 8292 from CSBS to CACS independent of the handshake lines. If a bus hangup is detected (by an error flag), this command will force the 8292 to take control (asserting ATN) even if the AH function is not in ANRS (Acceptor Not Ready State). This command should be used very carefully since it may cause the loss of a data byte. Normally, control should be taken synchronously. After checking the controller function for being in the CSBS (else it will exit immediately), $\overline{\text{ATNO}}$ will go low, and a TCI interrupt will be generated.

FD — TCSY — Take Control Synchronously

There are two different procedures used to transfer the 8292 from CSBS to CACS depending on the state of the 8291 in the system. If the 8291 is in "continuous AH cycling" mode (Aux. Reg. A0 = A1 = 1), then the following procedure should be followed:

1. The master microprocessor stops the continuous AH cycling mode in the 8291;
2. The master reads the 8291 Interrupt Status 1 Register;
3. If the END bit is set, the master sends the TCSY command to the 8292;
4. If the END bit was not set, the master reads the 8291 Data In Register and then waits for another BI interrupt from the 8291. When it occurs, the master sends the 8292 the TCSY command.

If the 8291 is not in AH cycling mode, then the master just waits for a BI interrupt and then sends the TCSY command. After the TCSY command has been issued, the 8292 checks for $\overline{\text{CSBS}}$. If $\overline{\text{CSBS}}$, then it exits the routine. Otherwise, it then checks the DAV bit in the GPIB status. When DAV becomes false, the 8292 will wait for at least 1.5 μsec. (T10) and then $\overline{\text{ATNO}}$ will go low. If DAV does not go low, a TOUT3 error will be generated. If the 8292 successfully takes control, it sets TCI true.

FE — STCNI — Start Counter Interrupts

This command enables the internal counter interrupt. The counter is enabled by the GSEC command.

UTILITY COMMANDS

All these commands are either Read or Write to registers in the 8292. Note that writing to the Error Mask Register and the Interrupt Mask Register are done directly.

E1 — WTOUT — Write To Time Out Register

The byte written to the data bus buffer (with $A_0 = 0$) following this command will determine the time used for the time out function. Since this function is implemented in software, this will not be an accurate time measurement. This feature is enable or disable by the Error Mask Register. No interrupts except for the $\overline{\text{IBFI}}$ will be generated upon completion.

E2 — WEVC — Write To Event Counter

The byte written to the data bus buffer (with $A_0 = 0$) following this command will be loaded into the Event Counter Register and the Event Counter Status for byte counting or EOI counting. Only $\overline{\text{IBFI}}$ will indicate completion of this command.

E3 — REVC — Read Event Counter Status

This command transfers the contents of the Event Counter into the data bus buffer. A TCI is generated when the data is available in the data bus buffer.

E4 — RERF — Read Error Flag Register

This command transfers the contents of the Error Flag Register into the data bus buffer. A TCI is generated when the data is available.

E5 — RINM — Read Interrupt Mask Register

This command transfers the contents of the Interrupt Mask Register into the data bus buffer. This register is available to the processor so that it does not need to store this information elsewhere. A TCI is generated when the data is available in the data bus buffer.

E6 — RCST — Read Controller Status Register

This command transfers the contents of the Controller Status Register into the data bus buffer and a TCI interrupt is generated.

E7 — RBST — Read GPIB Bus Status Register

This command transfers the contents of the GPIB Bus Status Register into the data bus buffer, and a TCI interrupt is generated when the data is available.

E9 — RTOUT — Read Time Out Status Register

This command transfers the contents of the Time Out Status Register into the data bus buffer, and a TCI interrupt is generated when the data is available.

EA — RERM — Read Error Mask Register

This command transfers the contents of the Error Mask Register to the data bus buffer so that the processor does not need to store this information elsewhere. A TCI interrupt is generated when the data is available.

Interrupt Acknowledge

SYC	ERR	SRQ	EV	1	IFCR	1	1
D_7							D_0

Each named bit in an Interrupt Acknowledge (IACK) corresponds to a flag in the Interrupt Status Register. When the 8292 receives this command, it will clear the SPI and the corresponding bits in the Interrupt Status Register. If not all the bits were cleared, then the SPI will be set true again. If the error flag is not acknowledged by the IACK command, then the Error Flag Register will be transferred to the data bus buffer, and a TCI will be generated.

NOTE: XXXX1X11 is an undefined operation or utility command, so no conflict exists between the IACK operation and utility commands.

SYSTEM OPERATION

8292 To Master Processor Interface

Communication between the 8292 and the Master Processor can be either interrupt based communication or based upon polling the interrupt status register in predetermined intervals.

Interrupt Based Communication

Four different interrupts are available from the 8292:

OBFI Output Buffer Full Interrupt
$\overline{\text{IBFI}}$ Input Buffer Not Full Interrupt
TCI Task Completed Interrupt
SPI Special Interrupt

Each of the interrupts is enabled or disabled by a bit in the interrupt mask register. Since OBFI and $\overline{\text{IBFI}}$ are directly connected to the OBF and IBF flags, the master can write a new command to the input data bus buffer as soon as the previous command has been read.

The TCI interrupt is useful when the master is sending commands to the 8292. The pending TCI will be cleared with each new command written to the 8292. Commands sent to the 8292 can be divided into two major groups:

1. Commands that require response back from the 8292 to the master, e.g., reading register.
2. Commands that initiate some action or enable features but do not require response back from the 8292, e.g., enable data bus buffer interrupts.

With the first group, the TCI interrupt will be used to indicate that the required response is ready in the data bus buffer and the master may continue and read it. With the second group, the interrupt will be used to indicate completion of the required task, so that the master may send new commands.

The SPI should be used when immediate information or special events is required (see the Interrupt Status Register).

"Polling Status" Based Communication

When interrupt based communication is not desired, all interrupts can be masked by the interrupt mask register. The communication with the 8292 is based upon sequential poll of the interrupt status register. By testing the OBF and IBF flags, the data bus buffer status is determined while special events are determined by testing the other bits.

Receiving IFC

The IFC pulse defined by the IEEE-488 standard is at least 100 μsec. In this time, all operation on the bus should be aborted. Most important, the current controller (the one that is in charge at that time) should stop sending ATN or EOI. Thus, IFC must externally gate $\overline{\text{CIC}}$ (controller in charge) and $\overline{\text{ATNO}}$ to ensure that this occurs.

Reset and Power Up Procedure

After the 8292 has been reset either by the external reset pin, the device being powered on, or a RST command, the following sequential events will take place:

1. All outputs to the GPIB interface will go high ($\overline{\text{SRQ}}$, $\overline{\text{ATNI}}$, $\overline{\text{IFC}}$, SYC, CLTH, $\overline{\text{ATNO}}$, $\overline{\text{CIC}}$, TCI, SPI, $\overline{\text{EOI}}$, OBFI, $\overline{\text{IBFI}}$, $\overline{\text{DAV}}$, $\overline{\text{REV}}$).
2. The four interrupt outputs (TCI, SPI, OBFI, $\overline{\text{IBFI}}$) and CLTH output will go low.
3. The following registers will be cleared:
   Interrupt Status
   Interrupt Mask
   Error Flag
   Error Mask
   Time Out
   Event Counter (= 256), Counter is disabled.
4. If the 8292 is the system controller, an ABORT command will be executed, the 8292 will become the controller in charge, and it will enter the CACS state.

   If it is not the system controller, it will remain in CIDS.

System Configuration

The 8291 and 8292 must be interfaced to an IEEE-488 bus meeting a variety of specifications including drive capability and loading characteristics. To interface the 8291 and the 8292 without the 8293's, several external gates are required, using a configuration similar to that used in Figure 5.

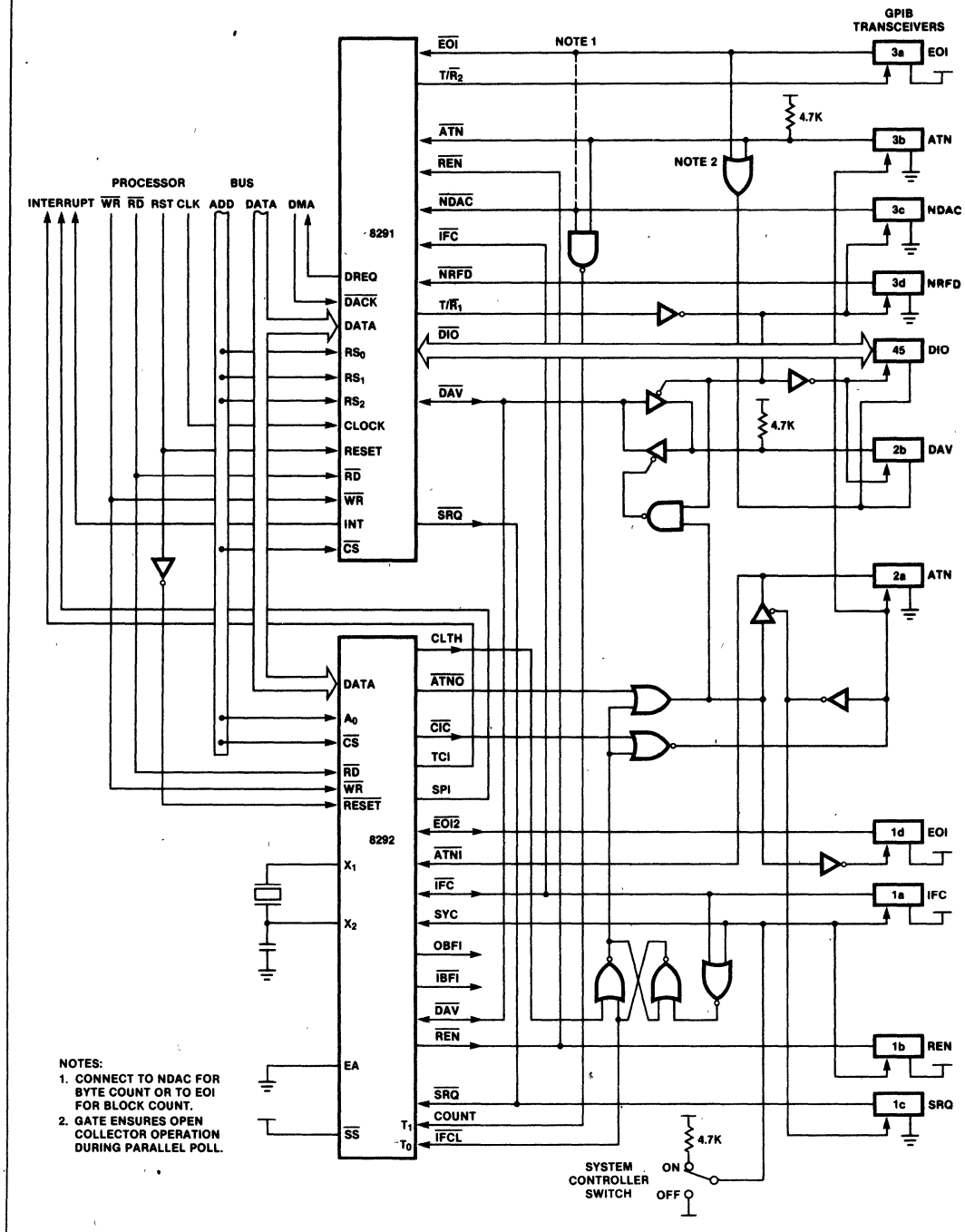


Figure 4. 8291 and 8292 System Configuration

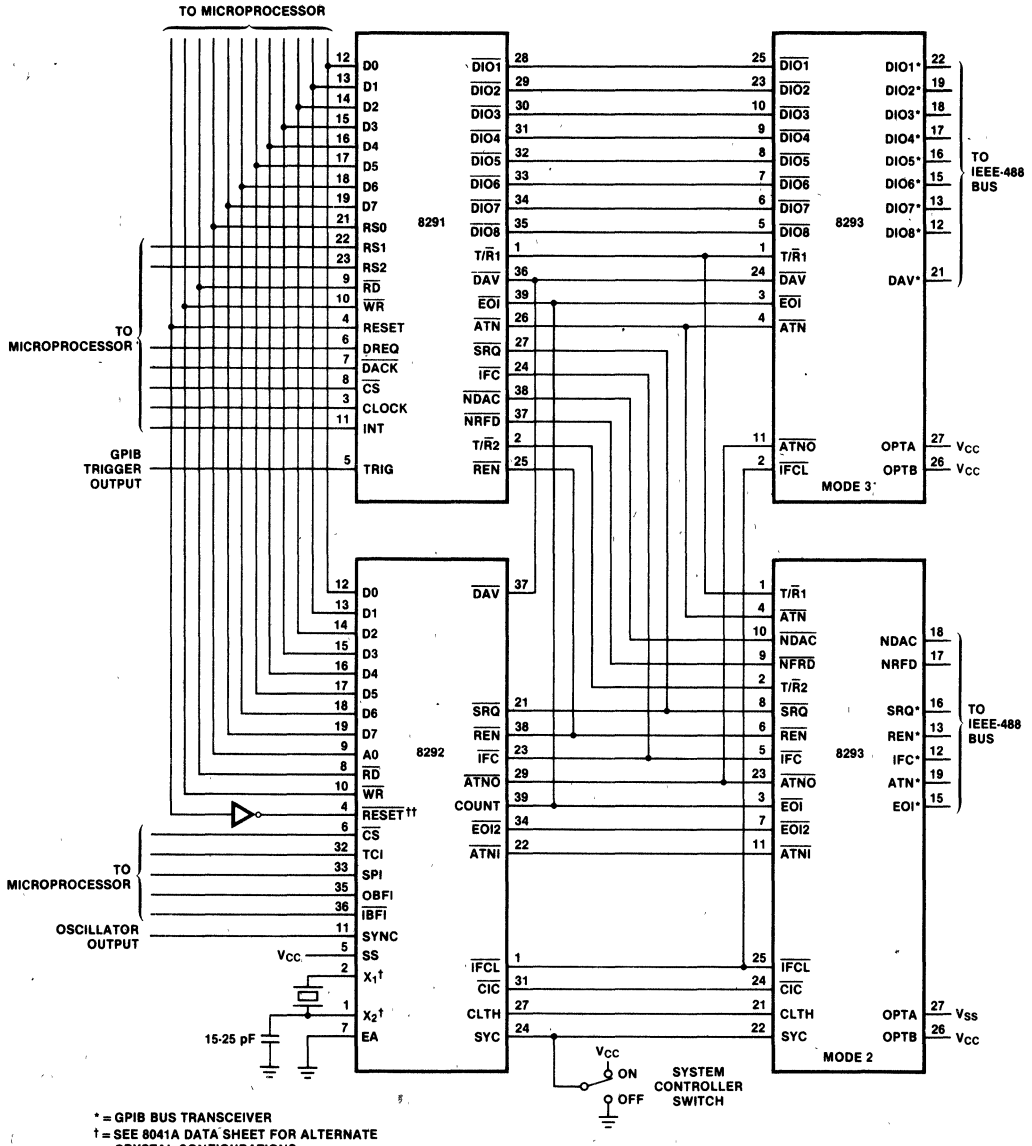


Figure 5. 8291, 8292, and 8293 System Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin With Respect to Ground . . . . . 0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V: 8292, V_{CC} = ±5V ±10%)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL1}	Input Low Voltage (All Except X_1, X_2, $\overline{RESET}$)	−0.5	0.8	V	
V_{IL2}	Input Low Voltage (X_1, X_2, $\overline{RESET}$)	−0.5	0.6	V	
V_{IH1}	Input High Voltage (All Except X_1, X_2, $\overline{RESET}$)	2.2	V_{CC}	V	
V_{IH2}	Input High Voltage (X_1, X_2, $\overline{RESET}$)	3.8	V_{CC}	V	
V_{OL1}	Output Low Voltage (D_0-D_7)		0.45	V	I_{OL} = 2.0 mA
V_{OL2}	Output Low Voltage (All Other Outputs)		0.45	V	I_{OL} = 1.6 mA
V_{OH1}	Output High Voltage (D_0-D_7)	2.4		V	I_{OH} = −400 μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4		V	I_{OH} = −50 μA
I_{IL}	Input Leakage Current (COUNT, $\overline{IFCL}$, $\overline{RD}$, $\overline{WR}$, $\overline{CS}$, A_0)		±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OZ}	Output Leakage Current (D_0-D_7, High Z State)		±10	μA	$V_{SS}+0.45 \leq V_{IN} \leq V_{CC}$
I_{LI1}	Low Input Load Current (Pins 21-24, 27-38)		0.5	mA	V_{IL} = 0.8V
I_{LI2}	Low Input Load Current ($\overline{RESET}$)		0.2	mA	V_{IL} = 0.8V
I_{CC}	Total Supply Current		125	mA	Typical = 65 mA
I_{IH}	Input High Leakage Current (Pins 21-24, 27-38)		100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance		10	pF	
$C_{I/O}$	I/O Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{SS} = 0V: 8292, V_{CC} = +5V ±10%)

DBB READ

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	250		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		225	ns	C_L = 150 pF
t_{RD}	$\overline{RD}$↓ to Data Out Delay		225	ns	C_L = 150 pF
t_{DF}	$\overline{RD}$↑ to Data Float Delay		100	ns	
t_{CY}	Cycle Time	2.5	15	μs	

DBB WRITE

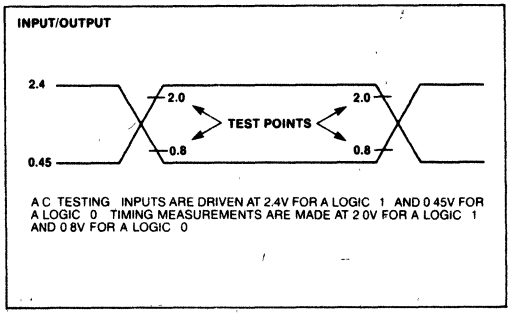
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	CS, A_0 Setup to WR↓	0		ns	
t_{WA}	CS, A_0 Hold After WR↑	0		ns	
t_{WW}	WR Pulse Width	250		ns	
t_{DW}	Data Setup to WR↑	150		ns	
t_{WD}	Data Hold After WR↓	0		ns	

COMMAND TIMINGS[1,3]

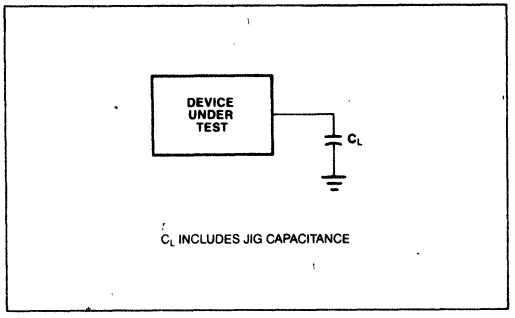
Code	Name	Execution Time	IBFI↑	TCI[2]	SPI	ATNO	CIC	IFC	REN	EOI	DAV	Comments
E1	WTOUT	63	24									
E2	WEVC	63	24									
E3	REVC	71	24	51								
E4	RERF	67	24	47								
E5	RINM	69	24	49								
E6	RCST	97	24	77								
E7	RBST	92	24	72								
E8												
E9	RTOUT	69	24	49								
EA	RERM	69	24	49								
F0	SPCNI	53	24									Count Stops After 39
F1	GIOL	88	24	70		↑61	↑61					
F2	RST	94	24		↓52							Not System Controller
F2	RST	214	24	192	↓52	↓179	↓174	↓101				System Controller
F3	RSTI	61	24									
F4	GSEC	125	24	107		↑98						
F5	EXPP	75	24						↓53 ↑59	↓55 ↑57		
F6	GTSB	118	24	100		↑91						
F7	SLOC	73	24	55				↑46				
F8	SREM	91	24	73				↓64				
F9	ABORT	155	24	133		↓120	↓115	↓42				
FA	TCNTR	108	24	86		↓71	↓68					
FC	TCAS	92	24	67		↓55						
FD	TCSY	115	24	91		↓80						
FE	STCNI	59	24									Starts Count After 43
PIN	RESET	29	—	↓7	↓7							Not System Controller
X	IACK	116	—		↓73 ↑98							If Interrupt Pending

Notes:
1. All times are multiples of t_{CY} from the 8041A command interrupt.
2. TCI clears after 7 t_{CY} on all commands.
3. ↑ indicates a level transition from low to high, ↓ indicates a high to low transition.

A.C. TESTING INPUT, OUTPUT WAVEFORM

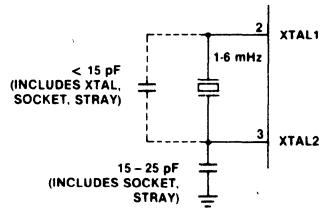


A.C. TESTING LOAD CIRCUIT



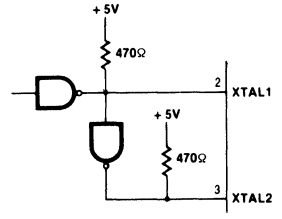
CLOCK DRIVER CIRCUITS

CRYSTAL OSCILLATOR MODE



CRYSTAL SERIES RESISTANCE SHOULD BE <75Ω AT 6 MHz; <180Ω AT 3.6 MHz

DRIVING FROM EXTERNAL SOURCE



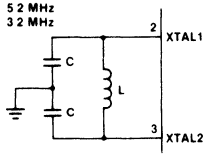
BOTH XTAL1 AND XTAL2 SHOULD BE DRIVEN
RESISTORS TO V_{CC} ARE NEEDED TO ENSURE V_{IH} = 3.8V
IF TTL CIRCUITRY IS USED

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5 2 MHz
120 μH	20 pF	3 2 MHz

$$f \approx \frac{1}{2\pi\sqrt{LC'}}$$

$$C' = \frac{C + 3C_{PP}}{2}$$

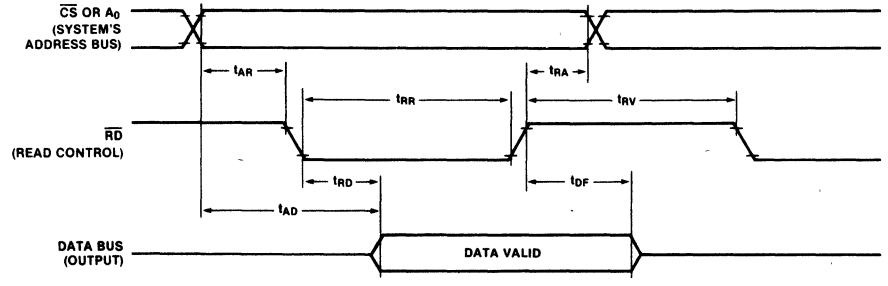


$C_{PP} \approx 5 – 10$ pF PIN TO PIN CAPACITANCE

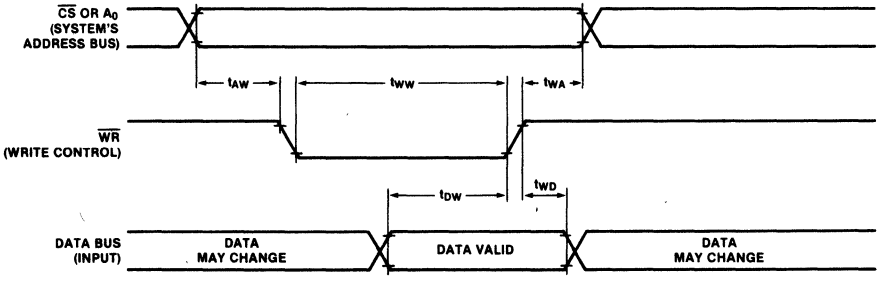
EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE

WAVEFORMS

READ OPERATION—DATA BUS BUFFER REGISTER



WRITE OPERATION — DATA BUS BUFFER REGISTER



APPENDIX

The following tables and state diagrams were taken from the IEEE Standard Digital Interface for Programmable Instrumentation, IEEE Std. 488-1978. This document is the official standard for the GPIB bus and can be purchased from IEEE, 345 East 47th St., New York, NY 10017.

C MNEMONICS

Messages	Interface States
pon = power on	CIDS = controller idle state
rsc = request system control	CADS = controller addressed state
rpp = request parallel poll	CTRS = controller transfer state
gts = go to standby	CACS = controller active state
tca = take control asynchronously	CPWS = controller parallel poll wait state
tcs = take control synchronously	CPPS = controller parallel poll state
sic = send interface clear	CSBS = controller standby state
sre = send remote enable	CSHS = controller standby hold state
IFC = interface clear	CAWS = controller active wait state
ATN = attention	CSWS = controller synchronous wait state
TCT = take control	CSRS = controller service requested state
	CSNS = controller service not requested state
	SNAS = system control not active state
	SACS = system control active state
	SRIS = system control remote enable idle state
	SRNS = system control remote enable not active state
	SRAS = system control remote enable active state
	SIIS = system control interface clear idle state
	SINS = system control interface clear not active state
	SIAS = system control interface clear active state
	(ACDS) = accept data state (AH function)
	(ANRS) = acceptor not ready state (AH function)
	(SDYS) = source delay state (SH function)
	(STRS) = source transfer state (SH function)
	(TADS) = talker addressed state (T function)

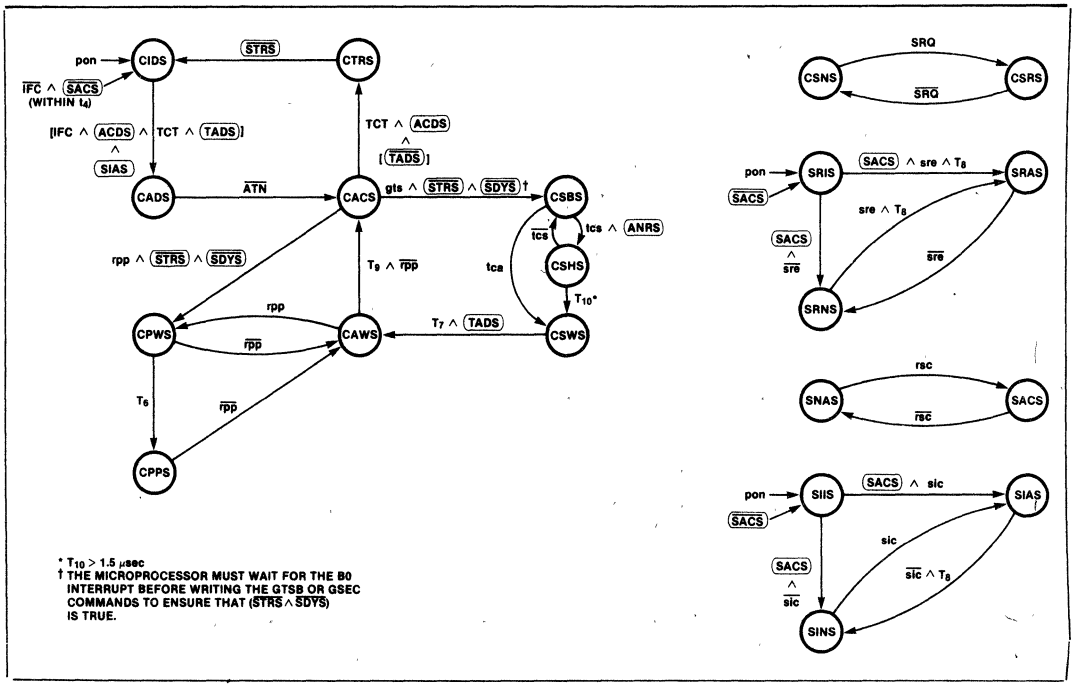


Figure A.1. C State Diagram

REMOTE MESSAGE CODING

Mnemonic	Message Name		TYPE	CLASS	Bus Signal Line(s) and Coding That Asserts the True Value of the Message													
					DIO8	7	6	5	4	3	2	DIO1	NDAVD	NRFDC	ATN	EOI	SRQ	IFC	REN
ACG	Addressed Command Group		M	AC	Y	0	0	0	X	X	X	X	XXX		1	X	X	X	X
ATN	Attention		U	UC	X	X	X	X	X	X	X	X	XXX		1	X	X	X	X
DAB	Data Byte	(Notes 1, 9)	M	DD	D8	D7	D6	D5	D4	D3	D2	D1	XXX		0	X	X	X	X
DAC	Data Accepted		U	HS	X	X	X	X	X	X	X	X	XX0		X	X	X	X	X
DAV	Data Valid		U	HS	X	X	X	X	X	X	X	X	1XX		X	X	X	X	X
DCL	Device Clear		M	UC	Y	0	0	1	0	1	0	0	XXX		1	X	X	X	X
END	End		U	ST	X	X	X	X	X	X	X	X	XXX		0	1	X	X	X
EOS	End of String	(Notes 2, 9)	M	DD	E8	E7	E6	E5	E4	E3	E2	E1	XXX		0	X	X	X	X
GET	Group Execute Trigger		M	AC	Y	0	0	0	1	0	0	0	XXX		1	X	X	X	X
GTL	Go to Local		M	AC	Y	0	0	0	0	0	0	1	XXX		1	X	X	X	X
IDY	Identify		U	UC	X	X	X	X	X	X	X	X	XXX		X	1	X	X	X
IFC	Interface Clear		U	UC	X	X	X	X	X	X	X	X	XXX		X	X	X	1	X
LAG	Listen Address Group		M	AD	Y	0	1	X	X	X	X	X	XXX		1	X	X	X	X
LLO	Local Lock Out		M	UC	Y	0	0	1	0	0	0	1	XXX		1	X	X	X	X
MLA	My Listen Address	(Note 3)	M	AD	Y	0	1	L5	L4	L3	L2	L1	XXX		1	X	X	X	X
MTA	My Talk Address	(Note 4)	M	AD	Y	1	0	T5	T4	T3	T2	T1	XXX		1	X	X	X	X
MSA	My Secondary Address	(Note 5)	M	SE	Y	1	1	S5	S4	S3	S2	S1	XXX		1	X	X	X	X
NUL	Null Byte		M	DD	0	0	0	0	0	0	0	0	XXX		X	X	X	X	X
OSA	Other Secondary Address		M	SE	(OSA = SCG ∧ $\overline{\text{MSA}}$)														
OTA	Other Talk Address		M	AD	(OTA = TAG ∧ $\overline{\text{MTA}}$)														
PCG	Primary Command Group		M	—	(PCG = ACG ∨ UCG ∨ LAG ∨ TAG)														
PPC	Parallel Poll Configure		M	AC	Y	0	0	0	0	1	0	1	XXX		1	X	X	X	X
PPE	Parallel Poll Enable	(Note 6)	M	SE	Y	1	1	0	S	P3	P2	P1	XXX		1	X	X	X	X
PPD	Parallel Poll Disable	(Note 7)	M	SE	Y	1	1	1	D4	D3	D2	D1	XXX		1	X	X	X	X
PPR1	Parallel Poll Response 1	(Note 10)	U	ST	X	X	X	X	X	X	X	1	XXX		1	1	X	X	X
PPR2	Parallel Poll Response 2		U	ST	X	X	X	X	X	X	1	X	XXX		1	1	X	X	X
PPR3	Parallel Poll Response 3		U	ST	X	X	X	X	X	1	X	X	XXX		1	1	X	X	X
PPR4	Parallel Poll Response 4		U	ST	X	X	X	X	1	X	X	X	XXX		1	1	X	X	X
PPR5	Parallel Poll Response 5		U	ST	X	X	X	1	X	X	X	X	XXX		1	1	X	X	X
PPR6	Parallel Poll Response 6		U	ST	X	X	1	X	X	X	X	X	XXX		1	1	X	X	X
PPR7	Parallel Poll Response 7		U	ST	X	1	X	X	X	X	X	X	XXX		1	1	X	X	X
PPR8	Parallel Poll Response 8		U	ST	1	X	X	X	X	X	X	X	XXX		1	1	X	X	X
PPU	Parallel Poll Unconfigure		M	UC	Y	0	0	1	0	1	0	1	XXX		1	X	X	X	X
REN	Remote Enable		U	UC	X	X	X	X	X	X	X	X	XXX		X	X	X	X	1
RFD	Ready for Data		U	HS	X	X	X	X	X	X	X	X	X0X		X	X	X	X	X
RQS	Request Service	(Note 9)	U	ST	X	1	X	X	X	X	X	X	XXX		0	X	X	X	X
SCG	Secondary Command Group		M	SE	Y	1	1	X	X	X	X	X	XXX		1	X	X	X	X
SDC	Selected Device Clear		M	AC	Y	0	0	0	0	1	0	0	XXX		1	X	X	X	X
SPD	Serial Poll Disable		M	UC	Y	0	0	1	1	0	0	1	XXX		1	X	X	X	X
SPE	Serial Poll Enable		M	UC	Y	0	0	1	1	0	0	0	XXX		1	X	X	X	X
SRQ	Service Request		U	ST	X	X	X	X	X	X	X	X	XXX		X	X	1	X	X
STB	Status Byte	(Notes 8, 9)	M	ST	S8	X	S6	S5	S4	S3	S2	S1	XXX		0	X	X	X	X
TCT	Take Control		M	AC	Y	0	0	0	1	0	0	1	XXX		1	X	X	X	X
TAG	Talk Address Group		M	AD	Y	1	0	X	X	X	X	X	XXX		1	X	X	X	X
UCG	Universal Command Group		M	UC	Y	0	0	1	X	X	X	X	XXX		1	X	X	X	X
UNL	Unlisten		M	AD	Y	0	1	1	1	1	1	1	XXX		1	X	X	X	X
UNT	Untalk	(Note 11)	M	AD	Y	1	0	1	1	1	1	1	XXX		1	X	X	X	X

The 1/0 coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

1. D1-D8 specify the device dependent data bits.
2. E1-E8 specify the device dependent code used to indicate the EOS message.
3. L1-L5 specify the device dependent bits of the device's listen address.
4. T1-T5 specify the device dependent bits of the device's talk address.
5. S1-S5 specify the device dependent bits of the device's secondary address.
6. S specifies the sense of the PPR.

   Response = $\overline{S \oplus ist}$

P1-P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
0	0	0	PPR1
.	.	.	.
.	.	.	.
.	.	.	.
1	1	1	PPR8

7. D1-D4 specify don't-care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
8. S1-S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)
9. The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
10. The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
11. This code is provided for system use, see 6.3.

PRELIMINARY

8293
GPIB TRANSCEIVER

- Nine Open-collector or Three-state Line Drivers
- 48 mA Sink Current Capability on Each Line Driver
- Nine Schmitt-type Line Receivers
- High Capacitance Load Drive Capability
- Single 5V Power Supply
- 28-Pin Package
- Low Power HMOS Design
- On-chip Decoder for Mode Configuration
- Power Up/Power Down Protection to Prevent Disrupting the IEEE Bus
- Connects with the 8291A and 8292 to Form an IEEE Standard 488 Interface Talker/Listener/Controller with no Additional Components
- Only Two 8293's Required per GPIB Interface
- On-Chip IEEE-488 Bus Terminations

The Intel® 8293 GPIB Transceiver is a high-current, non-inverting buffer chip designed to interface the 8291A GPIB Talker/Listener, or the 8291A/8292 GPIB Talker/Listener/Controller combination, to the IEEE Standard 488-1978 Instrumentation Interface Bus. Each GPIB interface would contain two 8293 Bus Transceivers. In addition, the 8293 can also be used as a general-purpose bus driver.

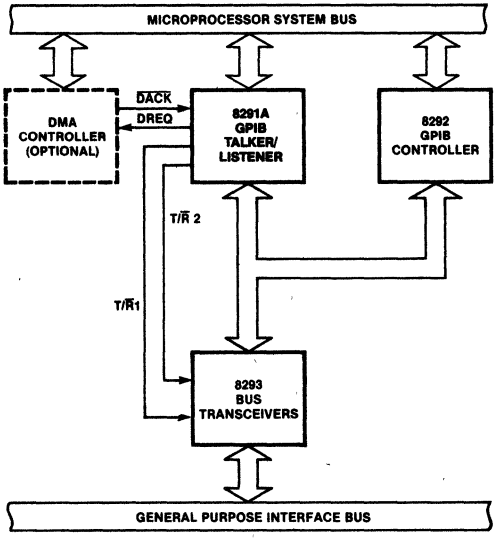


Figure 1. 8291A, 8292, 8293 Block Diagram

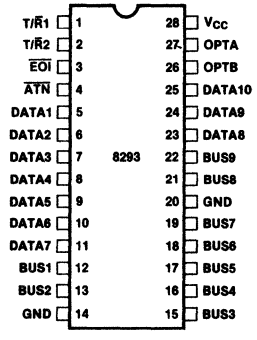


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
BUS1-BUS9	12, 13, 15–19, 21, 22	I/O	**GPIB Lines, GPIB Side:** These are the IEEE-488 bus interface driver/receivers, or TTL-compatible inputs on the 8291A/8292 side, depending on the mode used. Their use is programmed by the two mode select pins, OPTA and OPTB.
DATA1-DATA10	5–11, 23–25	I/O	**GPIB Lines, 8291A/92 Side:** These are the pins to be connected to the 8291A and 8292 to interface with the GPIB. Their use is programmed by the two mode select pins, OPTA and OPTB. All these pins are TTL compatible.
T/$\overline{R}$1	1	I	**Transmit Receive 1:** This pin controls the direction for NDAC, NRFD, DAV, and DIO1-DIO8. Input is TTL compatible.
T/$\overline{R}$2	2	I	**Transmit Receive 2:** This pin controls the direction for EOI. Input is TTL compatible.
$\overline{EOI}$	3	I/O	**End Or Identify:** This pin indicates the end of a multiple byte transfer or, in conjunction with ATN, addresses the device during a polling sequence. It connects to the 8291A and is switched between transmit and receive by T/$\overline{R}$2. This pin is TTL compatible.
$\overline{ATN}$	4	O	**Attention:** This pin is used by the 8291A to monitor the GPIB ATN control line. It specifies how data on the DIO lines is to be interpreted. This output is TTL compatible.
OPTA OPTB	27 26	I I	**Mode Select:** These two pins are to control the function of the 8293. A truth table of how they program the various modes is in Table 2.
V_{CC}	28	P.S.	**Voltage:** Positive power supply (5V ± 10%).
GND	14, 20	P.S.	**Ground:** Circuit ground.

Table 2. 8293 Mode Selection Pin Mapping

Pin Name	Pin No.	IEEE Implementation Name			
		Mode 0	Mode 1	Mode 2	Mode 3
OPTA	27	0	1	0	1
OPTB	26	0	0	1	1
DATA1	5	$\overline{IFC}$	$\overline{DIO8}$	$\overline{IFC}$	$\overline{DIO8}$
BUS1	12	IFC*	DIO8*	IFC*	DIO8*
DATA2	6	$\overline{REN}$	$\overline{DIO7}$	$\overline{REN}$	$\overline{DIO7}$
BUS2	13	REN*	DIO7*	REN*	DIO7*
DATA3	7	NC	$\overline{DIO6}$	$\overline{EOI2}$	$\overline{DIO6}$
BUS3	15	EOI*	DIO6*	EOI*	DIO6*
DATA4	8	$\overline{SRQ}$	$\overline{DIO5}$	$\overline{SRQ}$	$\overline{DIO5}$
BUS4	16	SRQ*	DIO5*	SRQ*	DIO5*
DATA5	9	$\overline{NRFD}$	$\overline{DIO4}$	$\overline{NRFD}$	$\overline{DIO4}$
BUS5	17	NRFD*	DIO4*	NRFD*	DIO4*
DATA6	10	$\overline{NDAC}$	$\overline{DIO3}$	$\overline{NDAC}$	$\overline{DIO3}$
BUS6	18	NDAC*	DIO3*	NDAC*	DIO3*
DATA7	11	T/$\overline{R}$IO1	NC	$\overline{ATNI}$	$\overline{ATNO}$
DATA8	23	T/$\overline{R}$IO2	$\overline{DIO2}$	$\overline{ATNO}$	$\overline{DIO2}$
BUS7	19	ATN*	DIO2*	ATN*	DIO2*
DATA9	24	$\overline{GIO1}$	$\overline{DAV}$	$\overline{CIC}$	$\overline{DAV}$
BUS8	21	GIO1*	DAV*	CLTH	DAV*
DATA10	25	$\overline{GIO2}$	$\overline{DIO1}$	$\overline{IFCL}$	$\overline{DIO1}$
BUS9	22	GIO2*	DIO1*	SYC	DIO1*
T/$\overline{R}$1	1	T/$\overline{R}$1	T/$\overline{R}$1	T/$\overline{R}$1	T/$\overline{R}$1
T/$\overline{R}$2	2	T/$\overline{R}$2	NC	T/$\overline{R}$2	$\overline{IFCL}$
$\overline{EOI}$	3	$\overline{EOI}$	$\overline{EOI}$	$\overline{EOI}$	$\overline{EOI}$
$\overline{ATN}$	4	$\overline{ATN}$	$\overline{ATN}$	$\overline{ATN}$	$\overline{ATN}$

***Note:** These pins are the IEEE-488 bus non-inverting driver/receivers. They include all the bus terminations required by the Standard and may be connected directly to the GPIB bus connector.

GENERAL DESCRIPTION

The 8293 is a bidirectional transceiver. It was designed to interface the Intel 8291A GPIB Talker/Listener and the Intel® 8292 GPIB Controller to the IEEE Standard 488-1978 Instrumentation Bus (also referred to as the GPIB). The Intel GPIB Transceiver meets or exceeds all of the electrical specifications defined in the IEEE Standard 488-1978, Section 3.3–3.5, including the bus termination specifications.

The 8293 can be hardware programmed to one of four modes of operation. These modes allow the 8293 to be configured to support both a Talker/Listener/Controller environment and a Talker/Listener environment. In addition, the 8293 can be used as a general-purpose, three-state (push-pull) or open-collector bus transceiver with nine receiver/drivers. Two modes each are used to support a Talker/Listener (see Figure 3) and a Talker/Listener/Controller environment (see Figure 4). Mode 1 is used in general-purpose environments.

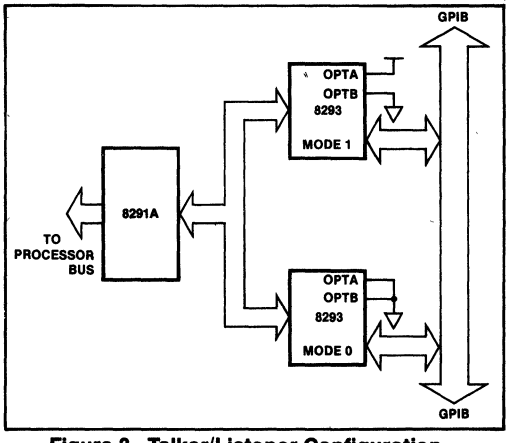


Figure 3. Talker/Listener Configuration

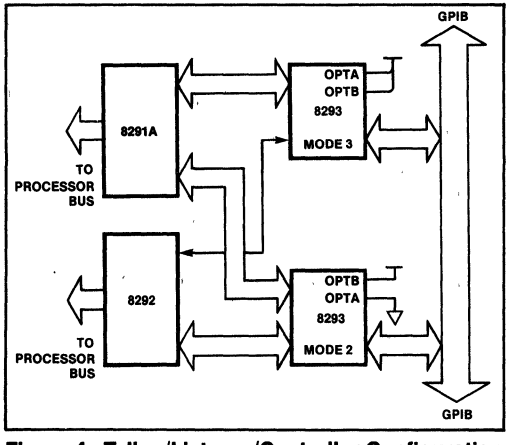


Figure 4. Talker/Listener/Controller Configuration

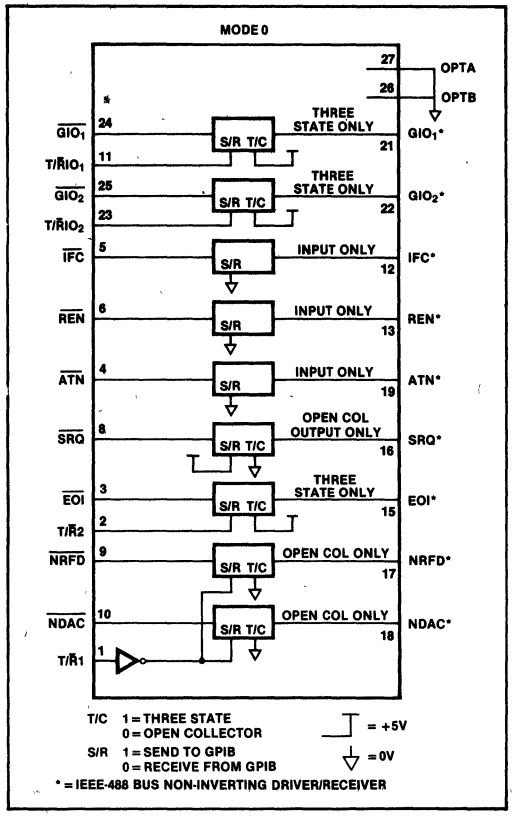


Figure 5. Talker/Listener Control Configuration

Table 3. Mode 0 Pin Description

Symbol	Pin No.	Type	Name and Function
T/R̄1	1	I	**Transmit Receive 1** Direction control for NDAC and NRFD. If T/R1 is high, then NDAC* and NRFD* are receiving. Input is TTL compatible.
NDAC	10	I/O	**Not Data Accepted:** Processor GPIB bus handshake control line; used to indicate the condition of acceptance of data by device(s). It is TTL compatible.
NDAC*	18	I/O	**Not Data Accepted:** IEEE GPIB bus handshake control line. When an input, it is a TTL compatible Schmitt-trigger. When an output, it is an open-collector driver with 48 mA sinking capability.
NRFD	9	I/O	**Not Ready For Data:** Processor GPIB handshake control line; used to indicate the condition of readiness of device(s) to accept data. This pin is TTL compatible.

Table 3. Mode 0 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NRFD*	17	I/O	**Not Ready For Data:** IEEE GPIB bus handshake control line. When an input, it is a TTL compatible Schmitt-trigger. When an output, it is an open-collector driver with a 48 mA current sinking capability.
T/$\overline{R}$2	2	I	**Transmit Receive 2:** Direction control for EOI. If T/R2 is high, EOI* is sending. Input is TTL compatible.
$\overline{EOI}$	3	I/O	**End Or Identify:** Processor GPIB bus control line; is used by a talker to indicate the end of a multiple byte transfer. This pin is TTL compatible.
EOI*	15	I/O	**End Or Identify:** IEEE GPIB bus control line; is used by a talker to indicate the end of a multiple byte transfer. This pin is a three-state (push-pull) driver capable of sinking 48 mA and a TTL compatible receiver with hysteresis.
$\overline{SRQ}$	8	I	**Service Request:** Processor GPIB bus control line; used by a device to indicate the need for service and to request an interruption of the current sequence of events on the GPIB. It is a TTL compatible input.
SRQ*	16	O	**Service Request:** IEEE GPIB bus control line; it is an open collector driver capable of sinking 48 mA.
$\overline{REN}$	6	O	**Remote Enable:** Processor GPIB bus control line; used by a controller (in conjunction with other messages) to select between two alternate sources of device programming data (remote or local control). This output is TTL compatible.
REN*	13	I	**Remote Enable:** IEEE GPIB bus control line. This input is a TTL compatible Schmitt-trigger.
$\overline{ATN}$	4	O	**Attention:** Processor GPIB bus control line; used by the 8291 to determine how data on the DIO signal lines are to be interpreted. This is a TTL compatible output.
ATN*	19	I	**Attention:** IEEE GPIB bus control line; this input is a TTL compatible Schmitt-trigger.
$\overline{IFC}$	5	O	**Interface Clear:** Processor GPIB bus control line; used by a controller to place the interface system into a known quiescent state. It is a TTL compatible output.
IFC*	12	I	**Interface Clear:** IEEE GPIB bus control line. This input is a TTL compatible Schmitt-trigger.
T/$\overline{R}$IO1 T/$\overline{R}$IO2	11 23	I I	**Transmit Receive General IO:** Direction control for the two spare transceivers. These pins are TTL compatible.
$\overline{GIO1}$ $\overline{GIO2}$	24 25	I/O I/O	**General IO:** This is the TTL side of the two spare transceivers. These pins are TTL compatible.
GIO1* GIO2*	21 22	I/O I/O	**General IO:** These are spare three-state (push-pull) drivers/Schmitt-trigger receivers. The drivers can sink 48 mA.

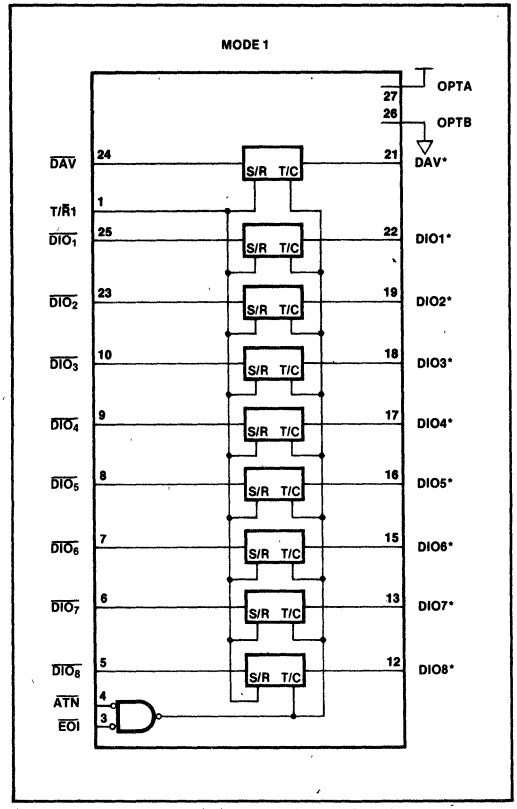


Figure 6. Talker/Listener Data Configuration

Table 4. Mode 1 Pin Description

Symbol	Pin No.	Type	Name and Function
T/$\overline{R1}$	1	I	**Transmit Receive 1:** Controls the direction for DAV and the DIO lines. If T/$\overline{R1}$ is high, then all these lines are sending information to the IEEE GPIB lines. This input is TTL compatible.
$\overline{EOI}$ $\overline{ATN}$	3 4	I I	**End Of Sequence And Attention:** Processor GPIB control lines. These two control signals are ANDed together to determine whether all the transceivers in the 8293 are three-state (push-pull) or open-collector. When both signals are low (true), then the controller is performing a parallel poll and the transceivers are all open-collector. These inputs are TTL compatible.
$\overline{DAV}$	24	I/O	**Data Valid:** Processor GPIB bus handshake control line; used to indicate the condition (availability and validity) of information on the DIO lines. It is TTL compatible.
DAV*	21	I/O	**Data Valid:** IEEE GPIB bus handshake control line. When an input, it is a TTL compatible Schmitt-trigger. When DAV* is an output, it can sink 48 mA.
$\overline{DIO1}$- $\overline{DIO8}$	25, 23, 10, 9, 8, 7, 6, 5	I/O	**Data Input/Output:** Processor GPIB bus data lines; used to carry message and data bytes in a bit-parallel byte-serial form controlled by the three handshake signals. These lines are TTL compatible.
DIO1*- DIO8*	22, 19, 18, 17, 16, 15, 13, 12	I/O	**Data Input/Output:** IEEE GPIB bus data lines. They are TTL compatible Schmitt-triggers when used for input and can sink 48 mA when used for output. See $\overline{ATN}$ and $\overline{EOI}$ description for output mode.

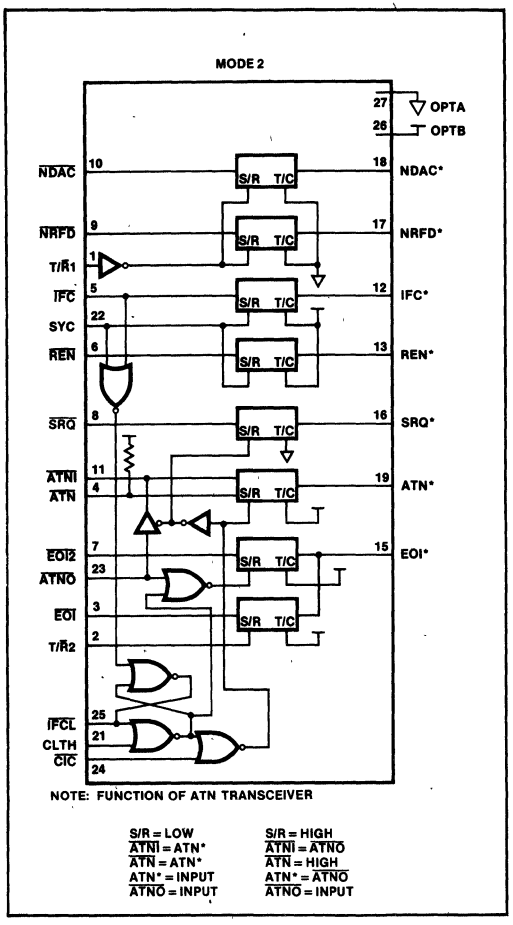


Figure 7. Talker/Listener/Controller Control Configuration

Table 5. Mode 2 Pin Description

Symbol	Pin No.	Type	Name and Function
T/$\overline{R1}$	1	I	**Transmit Receive 1:** Direction control for NDAC and NRFD. If T/$\overline{R1}$ is high, then NDAC and NRFD are receiving. Input is TTL compatible.
$\overline{NDAC}$	10	I/O	**Not Data Accepted:** Processor GPIB bus handshake control line; used to indicate the condition of acceptance of data by device(s). This pin is TTL compatible.
NDAC*	18	I/O	**Not Data Accepted:** IEEE GPIB bus handshake control line. It is a TTL compatible Schmitt-trigger when used for input and an open-collector driver with a 48 mA current sink capability when used for output.
$\overline{NRFD}$	9	I/O	**Not Ready For Data:** Processor GPIB bus handshake control line; used to indicate the condition of readiness of device(s) to accept data. This pin is TTL compatible.
NRFD*	17	I/O	**Not Ready For Data:** IEEE GPIB bus handshake control line. It is a TTL compatible Schmitt-trigger when used for input and an open-collector driver with a 48 mA current sink capability when used for output.
SYC[1]	22	I	**System Controller:** Used to monitor the system controller switch and control the direction for IFC and REN. This pin is a TTL compatible input.
$\overline{REN}$	6	I/O	**Remote Enable:** Processor GPIB control line; used by the active controller (in conjunction with other messages) to select between two alternate sources of device programming data (remote or local control). This pin is TTL compatible.
REN*	13	I/O	**Remote Enable:** IEEE GPIB bus control line. When used as an input, this is a TTL compatible Schmitt-trigger. When an output, it is a three-state driver with a 48 mA current sinking capability.
$\overline{IFC}$	5	I/O	**Interface Clear:** Processor GPIB bus control line; used by the active controller to place the interface system into a known quiescent state. This pin is TTL compatible.
IFC*	12	I/O	**Interface Clear:** IEEE GPIB control line. This is a TTL compatible Schmitt-trigger when used for input and a three-state driver capable of sinking 48 mA current when used for output.
$\overline{CIC}$	24	I	**Controller In Charge:** Used to control the direction of the SRQ and to indicate that the 8292 is in charge of the bus. $\overline{CIC}$ is a TTL compatible input.
CLTH[1]	21	I	**Clear Latch:** Used to clear the IFC Received latch after it has been recognized by the 8292. Normally low (except after a hardware reset). It will be pulsed high when IFC Received is recognized by the 8292. This input is TTL compatible.
$\overline{IFCL}$	25	O	**IFC Received Latch:** The 8292 monitors the IFC line when it is not the active controller through this pin.
$\overline{SRQ}$	8	I/O	**Service Request:** Processor GPIB control line; indicates the need for attention and requests the active controller to interrupt the current sequence of events on the GPIB bus. This pin is TTL compatible.
SRQ*	16	I/O	**Service Request:** IEEE GPIB bus control line. When used as an input, this pin is a TTL compatible Schmitt-trigger. When used as an output, it is an open-collector driver with a 48 mA current sinking capability.
T/$\overline{R2}$	2	I	**Transmit Receive 2:** Controls the direction for EOI. This input is TTL compatible.
$\overline{ATNO}$	23	I	**Attention Out:** Processor GPIB bus control line; used by the 8292 for ATN control of the IEEE bus during "take control synchronously" operations. A low on this input causes ATN to be asserted if $\overline{CIC}$ indicates that this 8292 is in charge. $\overline{ATNO}$ is a TTL compatible input.
$\overline{ATNI}$	11	O	**Attention In:** Processor GPIB bus control line; used by the 8292 to monitor the ATN line. This output is TTL compatible.
$\overline{ATN}$	4	O	**Attention:** Processor GPIB bus control line; used by the 8292 to monitor the ATN line. This output is TTL compatible.
ATN*	19	I/O	**Attention:** IEEE GPIB bus control line; used by a controller to specify how data on the DIO signal lines are to be interpreted and which devices must respond to data. When used as an output, this pin is a three-state driver capable of sinking 48 mA current. As an input, it is a TTL compatible Schmitt-trigger.
$\overline{EOI2}$	7	I/O	**End Or Identify 2:** Processor GPIB bus control line; used in conjunction with ATN by the active controller (the 8292) to execute a polling sequence. This pin is TTL compatible.
$\overline{EOI}$	3	I/O	**End Or Identify:** Processor GPIB bus control line; used by a talker to indicate the end of a multiple byte transfer sequence. This pin is TTL compatible.

NOTES:

1. V_{IL3} is guaranteed at 1.1V on these inputs to accommodate the high current-sourcing capability of these pins during a low input in Mode 2.

Table 5. Mode 2 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
EOI*	15	I/O	**End Or Identify:** IEEE GPIB bus control line; used by a talker to indicate the end of a multiple byte transfer sequence or, by a controller in conjunction with ATN, to execute a polling sequence. When an output, this pin can sink 48 mA current. When an input, it is a TTL compatible Schmitt-trigger.

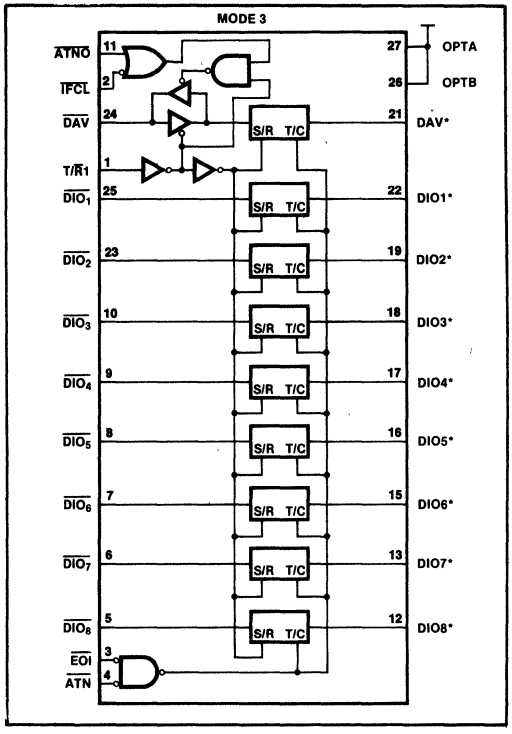


Figure 8. Talker/Listener/Controller Data Configuration

Table 6. Mode 3 Pin Description

Symbol	Pin No.	Type	Name and Function
T/$\overline{R1}$	1	I	**Transmit Receive 1:** Controls the direction for DAV and the DIO lines. If T/$\overline{R1}$ is high, then all these lines are sending information to the IEEE GPIB lines. This input is TTL compatible.
$\overline{EOI}$ $\overline{ATN}$	3 4	I I	**End Of Sequence and Attention:** Processor GPIB control lines. These two control lines are ANDed together to determine whether all the transceivers in the 8293 are push-pull or open-collector. When both signals are low (true), then the controller is performing a parallel poll and the transceivers are all open-collector. These inputs are TTL compatible.
$\overline{ATNO}$	11	I	**Attention Out:** Processor GPIB control line; used by the 8292 during "take control synchronously" operations. This pin is TTL compatible.
$\overline{IFCL}$	2	I	**Interface Clear Latched:** Used to make DAV received after the system controller asserts IFC. This input is TTL compatible
$\overline{DAV}$	24	I/O	**Data Valid:** Processor GPIB handshake control line; used to indicate the condition (availability and validity) of information on the DIO signals. This pin is TTL compatible.
DAV*	21	I/O	**Data Valid:** IEEE GPIB handshake control line. When an input, this pin is a TTL compatible Schmitt-trigger. When DAV* is an output, it can sink 48 mA.
$\overline{DIO1}$- $\overline{DIO8}$	25, 23, 10, 9, 8, 7, 6, 5	I/O	**Data Input/Output:** Processor GPIB bus data lines; used to carry message and data bytes in a bit-parallel byte-serial from controlled by the three handshake signals. These lines are TTL compatible.
DIO1* DIO8*	22, 19, 18, 17, 16, 15, 13, 12	I/O	**Data Input/Output:** IEEE GPIB bus data lines. They are TTL compatible Schmitt-triggers when used for input and can sink 48 mA when used for output.

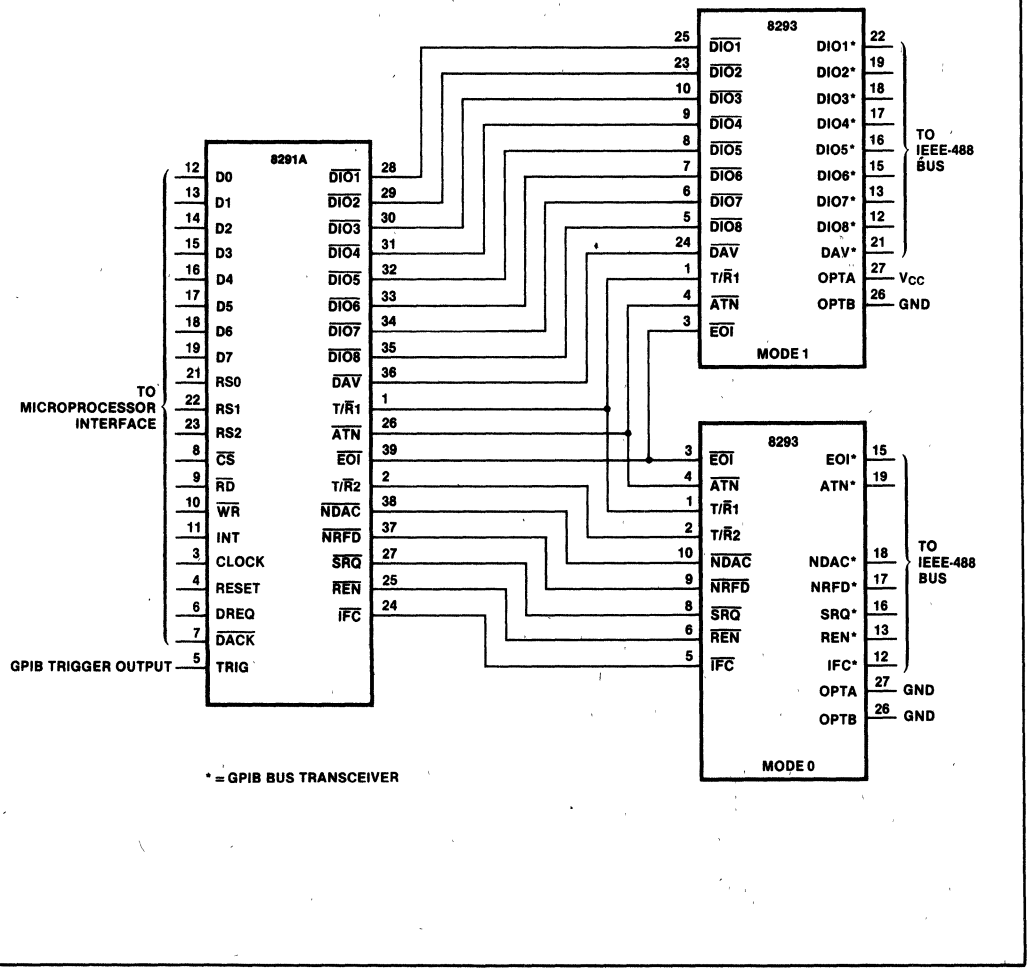


Figure 9. 8291A and 8293 System Configuration

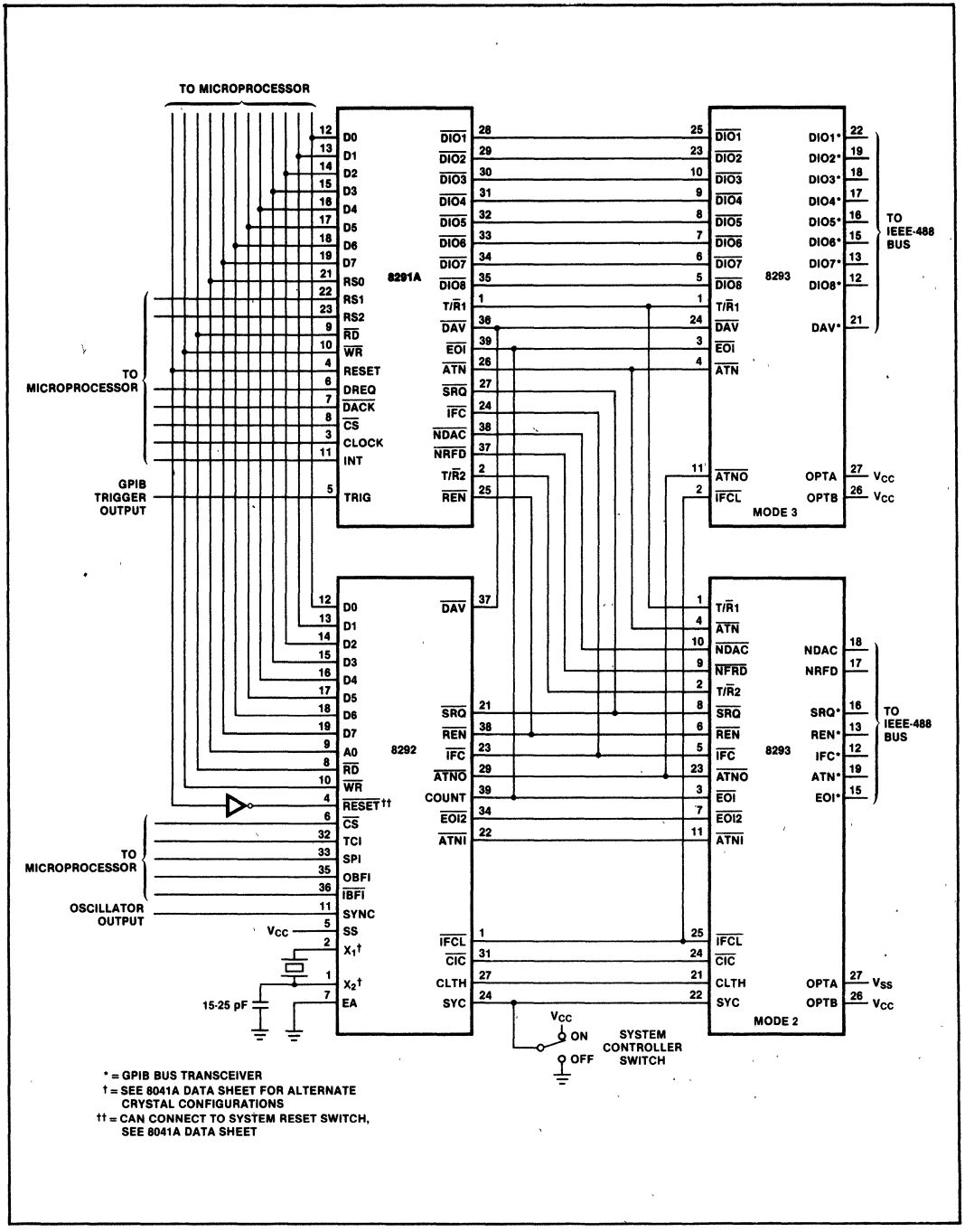


Figure 10. 8291A, 8292, and 8293 System Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0°C to 70°C
Storage Temperature.....−65°C to +150°C
Voltage on any Pin with
Respect to Ground.....−1.0V to +7V
Power Dissipation.....1 Watt

*NOTICE:
1. Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.
2. All devices are guaranteed to operate within the minimum and maximum parameter limits specified below. Typical parameters however are not tested and are not guaranteed. Established statistically, they indicate the performance level expected in a typical device at room temperature (T_A = 25°C) and V_{CC} = 5V.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
V_{IL1}	Input Low Voltage (GPIB Bus Pins)			0.8	V	
V_{IL2}	Input Low Voltage (Option Pins)	−0.1		0.1	V	
V_{IL3}[1]	Input Low Voltage (All Others)			0.8	V	
V_{IH1}	Input High Voltage (GPIB Bus Pins)	2.0		V_{CC}	V	
V_{IH2}	Input High Voltage (Option Pins)	4.5		V_{CC}	V	
V_{IH3}	Input High voltage (All Others)	2.0		V_{CC}	V	
V_{IH4}	Receiver Input Hysteresis	400			mV	
V_{OL1}	Output Low Voltage (GPIB Bus Pins)			0.5	V	I_{OL} = 48 mA
V_{OL2}	Output Low Voltage (All Others)			0.5	V	I_{OL} = 16 mA
V_{OH1}	Output High Voltage (GPIB Bus Pins)	2.4			V	I_{OH} = −5.2 mA
V_{OH2}	Output High Voltage (All Others)	2.4			V	I_{OH} = −800 μA
V_{IT}	Receiver Input Threshold High to Low	0.8			V	
	Receiver Input Threshold Low to High			2.0		
I_{LC}	Input Load Current (GPIB Pins)	See Bus Load Line Diagram				V_{CC} = 5.0V ± 5%
I_{IL}	Input Leakage Current (All Others)			10	μA	0.45 ≤ V_{IN} ≤ V_{CC}
I_{PD}	Bus Power Down Leakage Current			40	μA	0.45V ≤ V_{BUS} ≤ 2.7V
I_{CC}	Power Supply Current		110	175	mA	

NOTES:
1. V_{IL3} = 1.1V max on pins 21 and 22 in Mode 2 for the 8293-10.

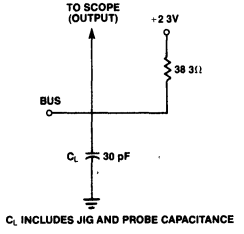
CAPACITANCE

Symbol	Parameter	Min.	Typ.	Max.	Units	Test Conditions
C_{IO1}	I/O Capacitance (GPIB Side)		50	80	pF	V_{IN} = V_{CC}
C_{IO2}	I/O Capacitance (System Side)		35	50	pF	V_{IN} = V_{CC}
C_{ITR}	Input Capacitance (T/R1, T/R2)		7	10	pF	V_{IN} = V_{CC}

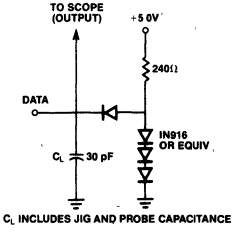
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

Symbol	Parameter	Max.	Units
t_{P1}	Transmitter Propagation Delay (All Lines)	30	ns
t_{P2}	Receiver Propagation Delay (EOI, ATN and Handshake Lines)	50	ns
t_{P3}	Receiver Propagation Delay (All Other Lines)	60	ns
t_{PHZ1}	Transmitter Disable Delay (High to 3-State)	40	ns
t_{PZH1}	Transmitter Enable Delay (3-state to High)	40	ns
t_{PLZ1}	Transmitter Disable Delay (Low to 3-State)	40	ns
t_{PZL1}	Transmitter Enable Delay (3-State to Low)	40	ns
t_{PHZ2}	Receiver Disable Delay (High to 3-State)	40	ns
t_{PZH2}	Receiver Enable Delay (3-State to High)	40	ns
t_{PLZ2}	Receiver Disable Delay (Low to 3-State)	40	ns
t_{PZL2}	Receiver Enable Delay (3-State to Low)	40	ns
t_{MS}	Mode Switch Delay	10	μs

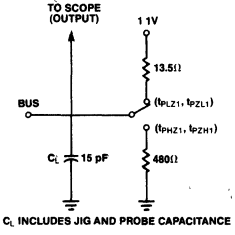
TYPICAL OUTPUT LOADING CIRCUITS



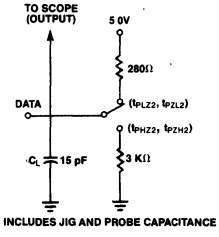
Data Input to Bus Output (Driver)



Bus Input to Data Output (Receiver)

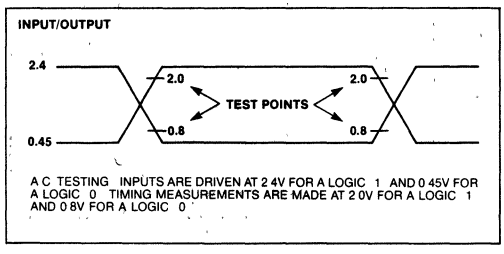


Send/Receive Input to Bus Output (Driver)

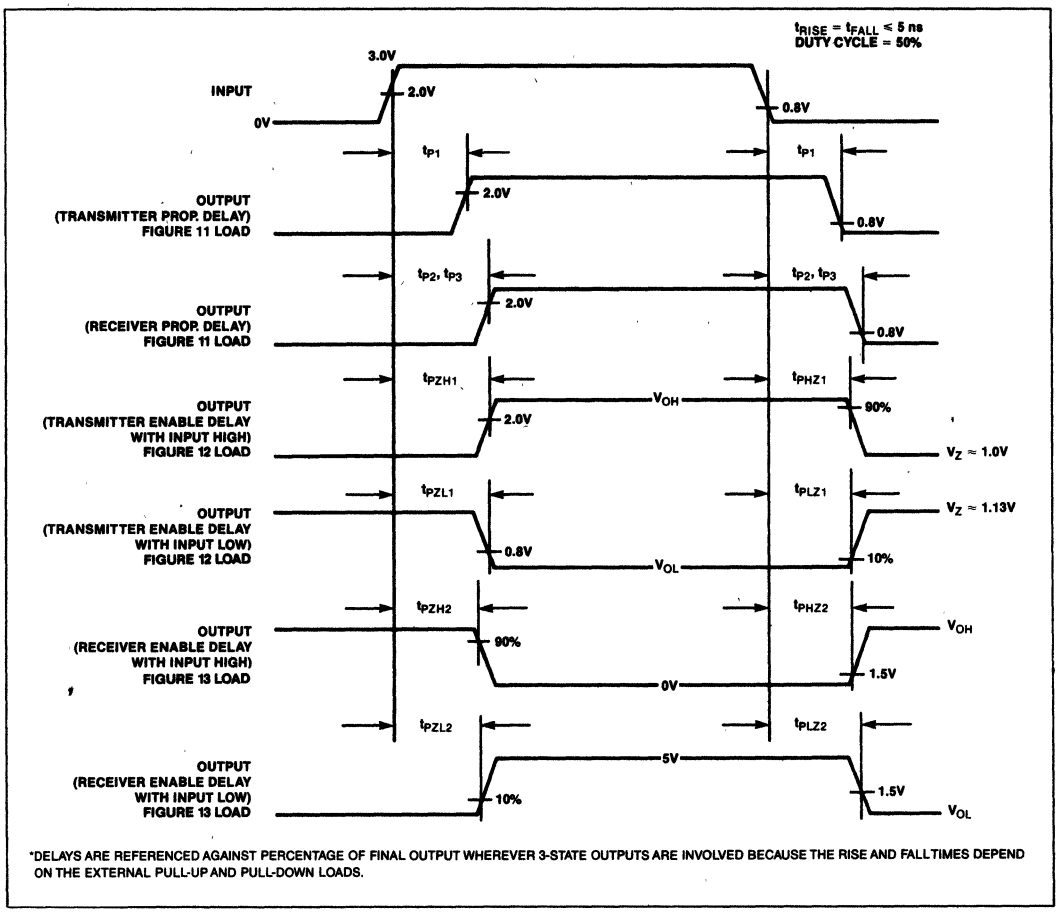


Send/Receive Input to Data Output (Receiver)

A.C. TESTING INPUT, OUTPUT WAVEFORM

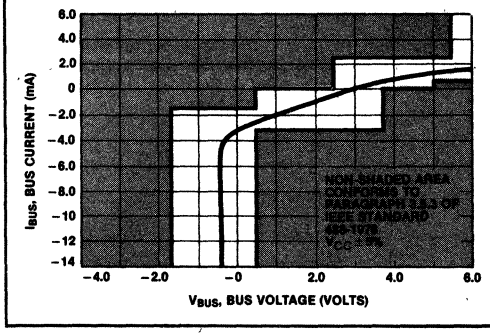


WAVEFORMS

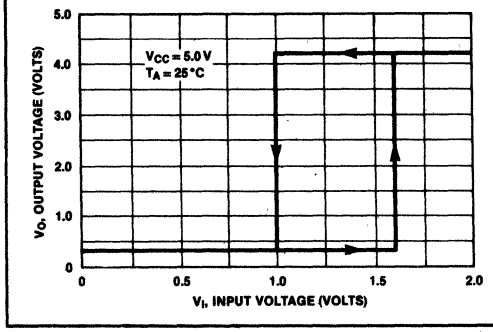


*DELAYS ARE REFERENCED AGAINST PERCENTAGE OF FINAL OUTPUT WHEREVER 3-STATE OUTPUTS ARE INVOLVED BECAUSE THE RISE AND FALL TIMES DEPEND ON THE EXTERNAL PULL-UP AND PULL-DOWN LOADS.

BUS LOAD LINE



TYPICAL RECEIVER HYSTERESIS CHARACTERISTICS



8294A
DATA ENCRYPTION UNIT

- **Certified by National Bureau of Standards**
- **400 Byte/Sec Data Conversion Rate**
- **64-Bit Data Encryption Using 56-Bit Key**
- **DMA Interface**
- **3 Interrupt Outputs to Aid in Loading and Unloading Data**
- **7-Bit User Output Port**
- **Single 5V ± 10% Power Supply**
- **Fully Compatible with iAPX-86,88, MCS-85™, MCS-80™, MCS-51™, and MCS-48™ Processors**
- **Implements Federal Information Processing Data Encryption Standard**
- **Encrypt and Decrypt Modes Available**

The Intel® 8294A Data Encryption Unit (DEU) is a microprocessor peripheral device designed to encrypt and decrypt 64-bit blocks of data using the algorithm specified in the Federal Information Processing Data Encryption Standard. The DEU operates on 64-bit text words using a 56-bit user-specified key to produce 64-bit cipher words. The operation is reversible: if the cipher word is operated upon, the original text word is produced. The algorithm itself is permanently contained in the 8294A; however, the 56-bit key is user-defined and may be changed at any time.

The 56-bit key and 64-bit message data are transferred to and from the 8294A in 8-bit bytes by way of the system data bus. A DMA interface and three interrupt outputs are available to minimize software overhead associated with data transfer. Also, by using the DMA interface two or more DEUs may be operated in parallel to achieve effective system conversion rates which are virtually any multiple of 400 bytes/second. The 8294A also has a 7-bit TTL compatible output port for user-specified functions.

Because the 8294A implements the NBS encryption algorithm it can be used in a variety of Electronic Funds Transfer applications as well as other electronic banking and data handling applications where data must be encrypted.

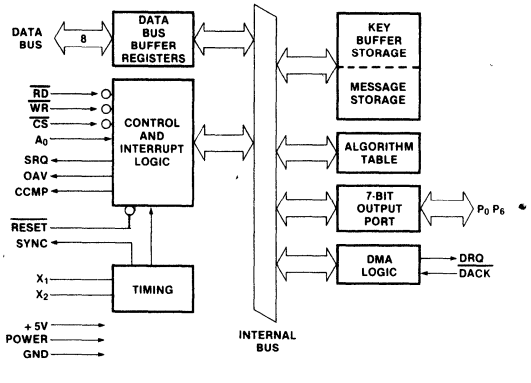


Figure 1. Block Diagram

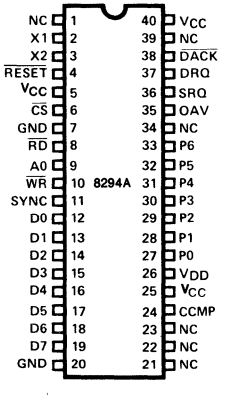


Figure 2. Pin Configuration

SEPT 1983
ORDER NUMBER: 210465-003

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
NC	1		**No Connection.**
X1 X2	2 3	I	**Crystal:** Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency.
$\overline{RESET}$	4	I	**Reset:** A low signal to this pin resets the 8294A.
V_{CC}	5		**Power:** Tied high.
$\overline{CS}$	6	I	**Chip Select:** A low signal to this pin enables reading and writing to the 8294A.
GND	7		**Ground:** This pin must be tied to ground.
$\overline{RD}$	8	I	**Read:** An active low read strobe at this pin enables the CPU to read data and status from the internal DEU registers.
A_0	9	I	**Address:** Address input used by the CPU to select DEU registers during read and write operations.
$\overline{WR}$	10	I	**Write:** An active low write strobe at this pin enables the CPU to send data and commands to the DEU.
SYNC	11	O	**Sync:** High frequency (Clock ÷ 15) output. Can be used as a strobe for external circuitry.
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7	12 13 14 15 16 17 18 19	I/O	**Data Bus:** Three-state, bi-directional data bus lines used to transfer data between the CPU and the 8294A.
GND	20		**Ground:** This pin must be tied to ground.
V_{CC}	40		**Power:** +5 volt power input: +5V ± 10%.

Symbol	Pin No.	Type	Name and Function
NC	39		**No Connection.**
$\overline{DACK}$	38	I	**DMA Acknowledge:** Input signal from the 8257 DMA Controller acknowledging that the requested DMA cycle has been granted.
DRQ	37	O	**DMA Request:** Output signal to the 8257 DMA Controller requesting a DMA cycle.
SRQ	36	O	**Service Request:** Interrupt to the CPU indicating that the 8294A is awaiting data or commands at the input buffer. SRQ=1 implies IBF=0.
OAV	35	O	**Output Available:** Interrupt to the CPU indicating that the 8294A has data or status available in its output buffer. OAV=1 implies OBF=1.
NC	34		**No Connection.**
P6 P5 P4 P3 P2 P1 P0	33 32 31 30 29 28 27	O	**Output Port:** User output port lines Output lines available to the user via a CPU command which can assert selected port lines. These lines have nothing to do with the encryption function. At power-on, each line is in a 1 state.
V_{DD}	26		**Power:** +5V power input. (+5V ±10%) Low power standby pin
V_{CC}	25		**Power:** Tied high.
CCMP	24	O	**Conversion Complete:** Interrupt to the CPU indicating that the encryption/decryption of an 8-byte block is complete.
NC	23		**No Connection.**
NC	22		**No Connection.**
NC	21		**No Connection.**

FUNCTIONAL DESCRIPTION

OPERATION

The data conversion sequence is as follows:

1. A Set Mode command is given, enabling the desired interrupt outputs.
2. An Enter New Key command is issued, followed by 8 data inputs which are retained by the DEU for encryption/decryption. Each byte must have odd parity.
3. An Encrypt Data or Decrypt Data command sets the DEU in the desired mode.

After this, data conversions are made by writing 8 data bytes and then reading back 8 converted data bytes. Any of the above commands may be issued between data conversions to change the basic operation of the DEU; e.g., a Decrypt Data command could be issued to change the DEU from encrypt mode to decrypt mode without changing either the key or the interrupt outputs enabled.

INTERNAL DEU REGISTERS

Four internal registers are addressable by the master processor: 2 for input, and 2 for output. The following table describes how these registers are accessed.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_0	Register
1	0	0	0	Data input buffer
0	1	0	0	Data output buffer
1	0	0	1	Command input buffer
0	1	0	1	Status output buffer
X	X	1	X	Don't care

The functions of each of these registers are described below.

Data Input Buffer — Data written to this register is interpreted in one of three ways, depending on the preceding command sequence.

1. Part of a key.
2. Data to be encrypted or decrypted.
3. A DMA block count.

Data Output Buffer — Data read from this register is the output of the encryption/decryption operation.

Command Input Buffer — Commands to the DEU are written into this register. (See command summary below.)

Status Output Buffer — DEU status is available in this register at all times. It is used by the processor for poll-driven command and data transfer operations.

STATUS BIT:	7	6	5	4	3	2	1	0
FUNCTION:	X	X	X	KPE	CF	DEC	IBF	OBF

OBF Output Buffer Full; OBF = 1 indicates that output from the encryption/decryption function is available in the Data Output Buffer. It is reset when the data is read.

IBF Input Buffer Full; A write to the Data Input Buffer or to the Command Input Buffer sets IBF = 1 The DEU resets this flag when it has accepted the input byte. Nothing should be written when IBF = 1.

DEC Decrypt; indicates whether the DEU is in an encrypt or a decrypt mode. DEC = 1 implies the decrypt mode. DEC = 0 implies the encrypt mode.

After 8294A has accepted a 'Decrypt Data' or 'Encrypt Data' command, 11 cycles are required to update the DEC bit.

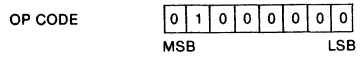
CF Completion Flag; This flag may be used to indicate any or all of three events in the data transfer protocol.

1. It may be used in lieu of a counter in the processor routine to flag the end of an 8-byte transfer.
2. It must be used to indicate the validity of the KPE flag.
3. It may be used in lieu of the CCMP interrupt to indicate the completion of a DMA operation.

KPE Key Parity Error; After a new key has been entered, the DEU uses this flag in conjunction with the CF flag to indicate correct or incorrect parity.

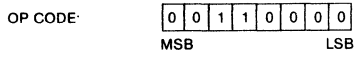
COMMAND SUMMARY

1 — Enter New Key



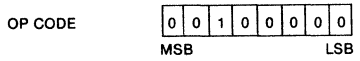
This command is followed by 8 data byte inputs which are retained in the key buffer (RAM) to be used in encrypting and decrypting data. These data bytes must have odd parity represented by the LSB.

2 — Encrypt Data



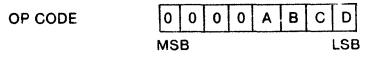
This command puts the 8294A into the encrypt mode.

3 — Decrypt Data



This command puts the 8294A into the decrypt mode.

4 — Set Mode



where:

A is the OAV (Output Available) interrupt enable
B is the SRQ (Service Request) interrupt enable
C is the DMA (Direct Memory Access) transfer enable
D is the CCMP (Conversion Complete) interrupt enable

This command determines which interrupt outputs will be enabled. A "1" in bits A, B, or D will enable the OAV, SRQ, or CCMP interrupts respectively. A "1" in bit C will allow DMA transfers. When bit C is set the OAV and SRQ interrupts should also be enabled (bits A,B = 1). Following the command in which bit C, the DMA bit, is set, the 8294 will expect one data byte to specify the number of 8-byte blocks to be converted using DMA.

5 — Write to Output Port

OP CODE	1	P_6	P_5	P_4	P_3	P_2	P_1	P_0
	MSB							LSB

This command causes the 7 least significant bits of the command byte to be latched as output data on the 8294 output port. The initial output data is 1111111. Use of this port is independent of the encryption/decryption function.

PROCESSOR/DEU INTERFACE PROTOCOL

ENTERING A NEW KEY

The timing sequence for entering a new key is shown in Figure 3. A flowchart showing the CPU software to accommodate this sequence is given in Figure 4.

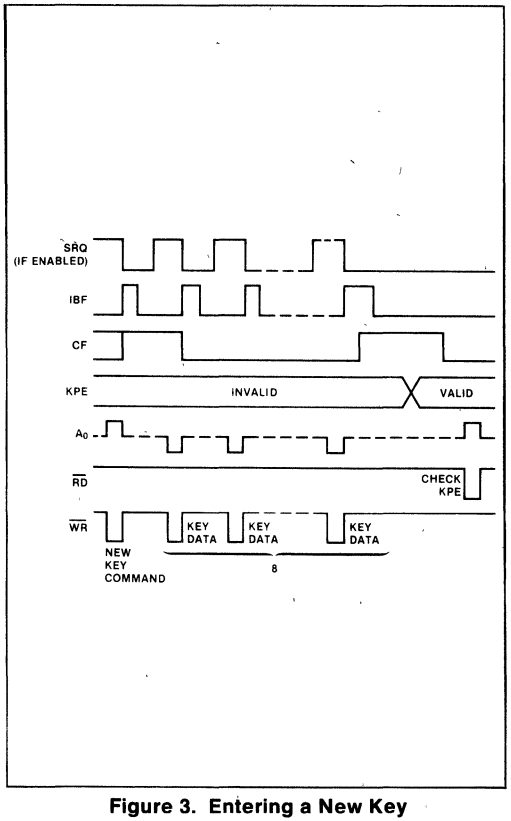


Figure 3. Entering a New Key

After the Enter New Key command is issued, 8 data bytes representing the new key are written to the data input buffer (most significant byte first). After the eighth byte is entered into the DEU, CF goes true (CF=1). The CF bit goes false again when KPE is valid. The CPU can then check the KPE flag. If KPE=1, a parity error has been detected and the DEU has not accepted the key. Each byte is checked for odd parity, where the parity bit is the LSB of each byte.

Since CF=1 only for a short period of time after the last byte is accepted, the CPU which polls the CF flag might miss detecting CF=1 momentarily. Thus, a counter should be used, as in Figure 4, to flag the end of the new key entry. Then CF is used to indicate a valid KPE flag.

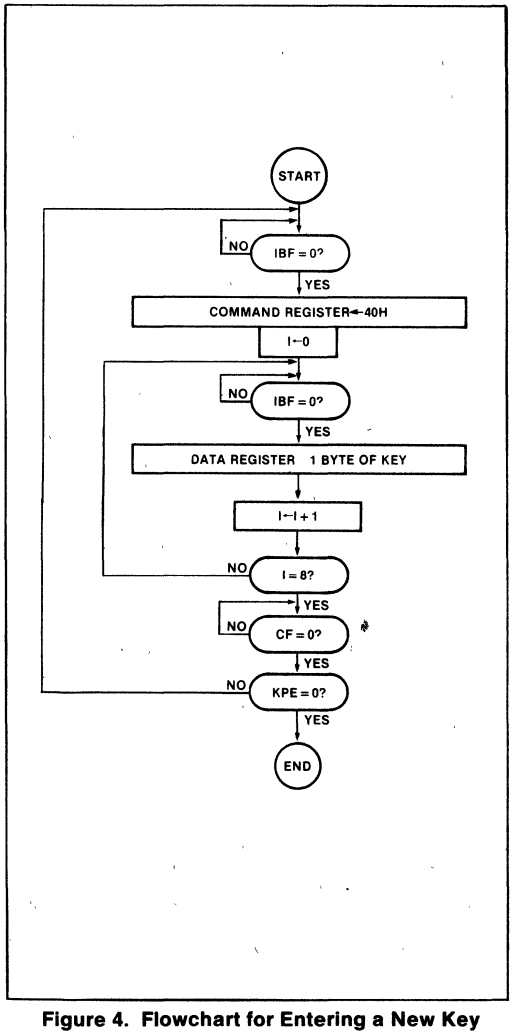


Figure 4. Flowchart for Entering a New Key

ENCRYPTING OR DECRYPTING DATA

Figure 5 shows the timing sequence for encrypting or decrypting data. The CPU writes 8 data bytes to the DEU's data input buffer for encryption/decryption. CF then goes true (CF = 1) to indicate that the DEU has accepted the 8-byte block. Thus, the CPU may test for IBF = 0 and CF = 1 to terminate the input mode, or it may use a software counter. When the encryption/-decryption is complete, the CCMP and OAV interrupts are asserted and the OBF flag is set true (OBF = 1). OAV and OBF are set false again after each of the converted data bytes is read back by the CPU. The CCMP interrupt is set false, and remains false, after the first read. After 8 bytes have been read back by the CPU, CF goes false (CF = 0). Thus, the CPU may test for CF = 0 to terminate the read mode. Also, the CCMP interrupt may be used to initiate a service routine which performs the next series of 8 data reads and 8 data writes.

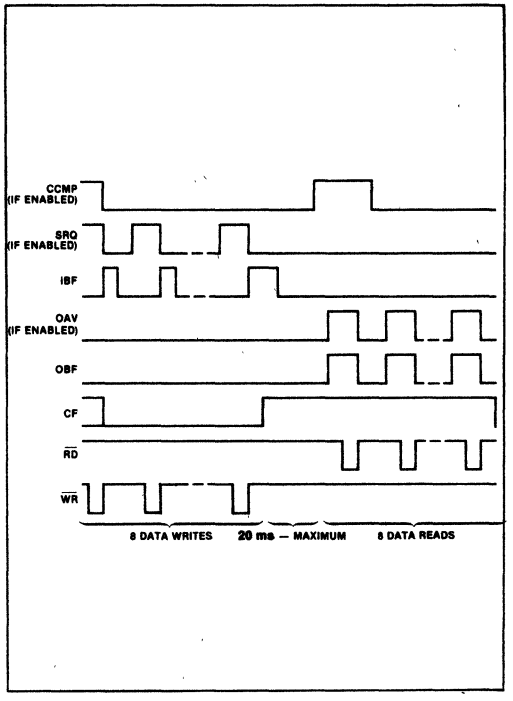


Figure 5. Encrypting/Decrypting Data

Figure 6 offers two flowcharts outlining the alternative means of implementing the data conversion protocol. Either the CF flag or a software counter may be used to end the read and write modes.

SRQ = 1 implies IBF = 0, OAV = 1 implies OBF = 1. This allows interrupt routines to do data transfers without checking status first. However, the OAV service routine must detect and flag the end of a data conversion.

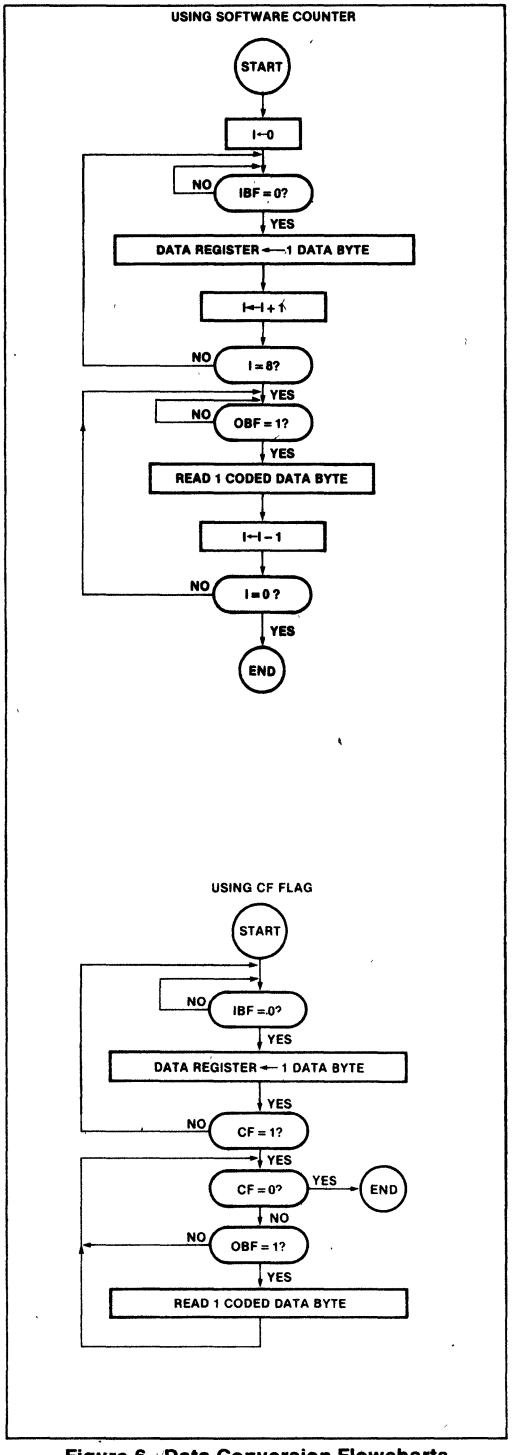


Figure 6. Data Conversion Flowcharts

USING DMA

The timing sequence for data conversions using DMA is shown in Figure 7. This sequence can be better understood when considered in conjunction with the hardware DMA interface in Figure 8. Note that the use of the DMA feature requires 3 external AND gates and 2 DMA channels (one for input, one for output). Since the DEU has only one DMA request pin, the SRQ and OAV outputs are used in conjunction with two of the AND gates to create separate DMA request outputs for the 2 DMA channels. The third AND gate combines the two active-low $\overline{DACK}$ inputs.

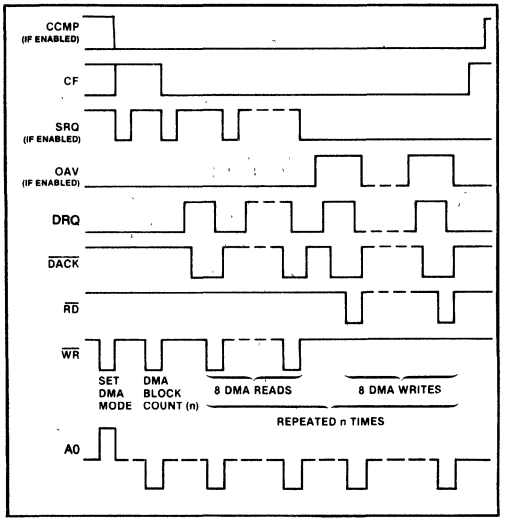


Figure 7. DMA Sequence

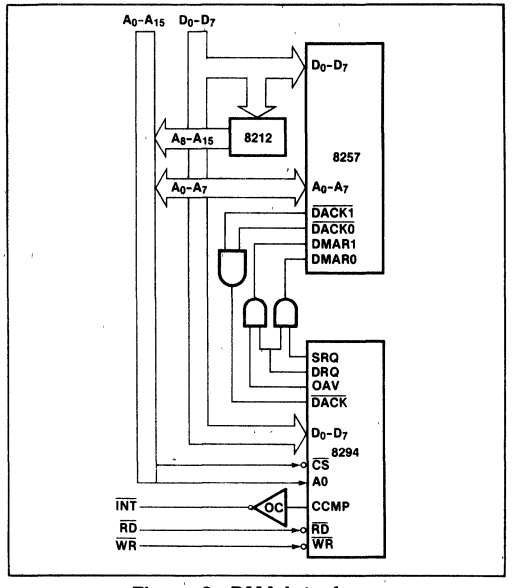


Figure 8. DMA Interface

To initiate a DMA transfer, the CPU must first initialize the two DMA channels as shown in the flowchart in Figure 9. It must then issue a Set Mode command to the DEU enabling the OAV, SRQ, and DMA outputs. The CCMP interrupt may be enabled or disabled, depending on whether that output is desired. Following the Set Mode command, there must be a data byte giving the number of 8-byte blocks of data (n<256) to be converted. The DEU then generates the required number of DMA requests to the 2 DMA channels with no further CPU intervention. When the requested number of blocks has been converted, the DEU will set CF and assert the CCMP interrupt (if enabled). CCMP then goes false again with the next write to the DEU (command or data). Upon completion of the conversion, the DMA mode is disabled and the DEU returns to the encrypt/decrypt mode. The enabled interrupt outputs, however, will remain enabled until another Set Mode command is issued.

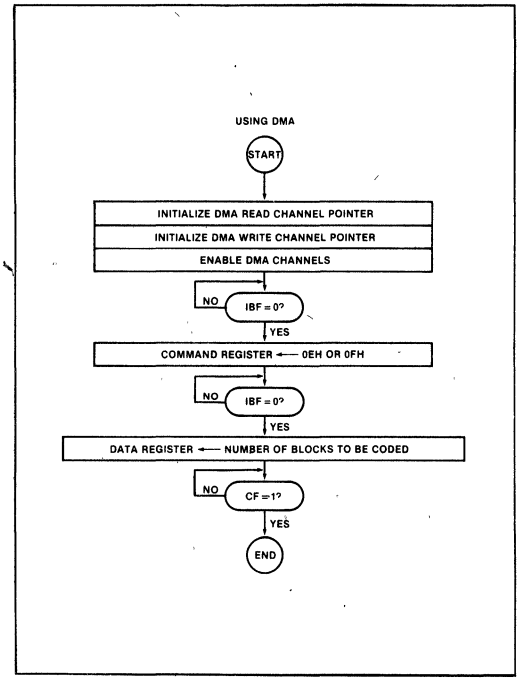


Figure 9. DMA Flowchart

SINGLE BYTE COMMANDS

Figure 10 shows the timing and protocol for single byte commands. Note that any of the commands is effective as a pacify command in that they may be entered at any time, except during a DMA conversion. The DEU is thus set to a known state. However, if a command is issued out of sequence, an additional protocol is required (Figure 11). The CPU must wait until the command is accepted (IBF = 0). A data read must then be issued to clear anything the preceding command sequence may have left in the Data Output Buffer.

CPU/DEU INTERFACES

Figures 12 through 15 illustrate four interface configurations used in the CPU/DEU data transfers. In all cases SRQ will be true (if enabled) and IBF will be false when the DEU is ready to accept data or commands.

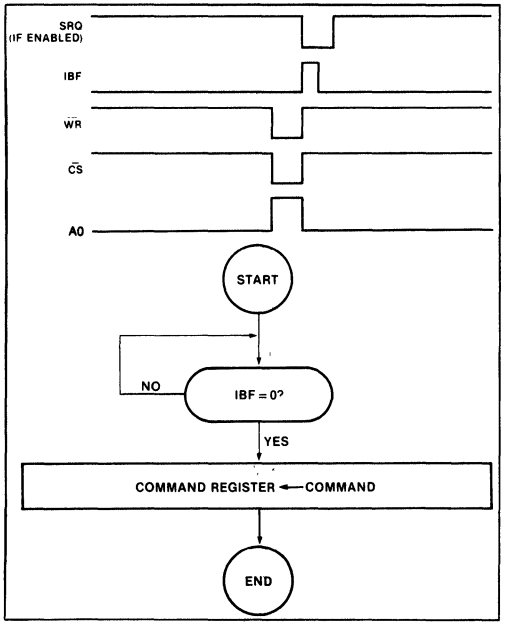


Figure 10. Single Byte Commands

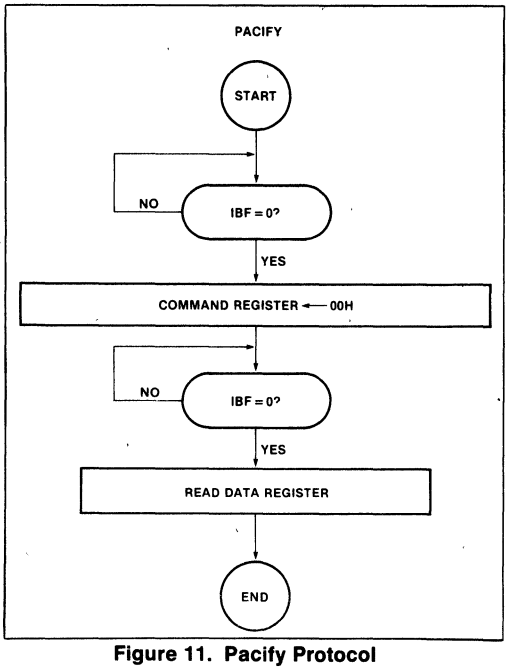


Figure 11. Pacify Protocol

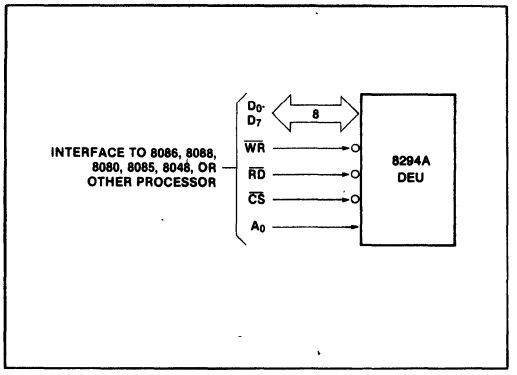


Figure 12. Polling Interface

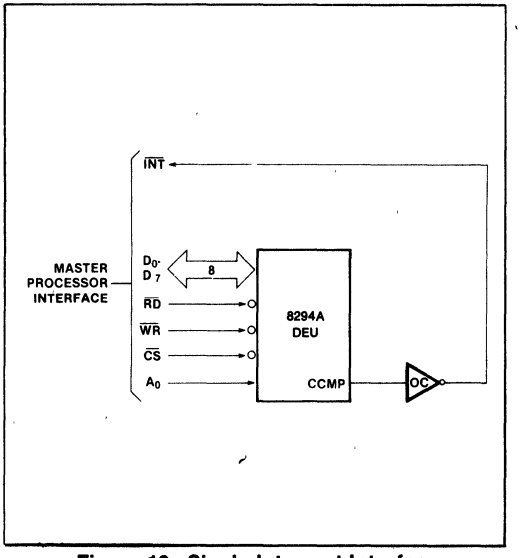


Figure 13. Single Interrupt Interface

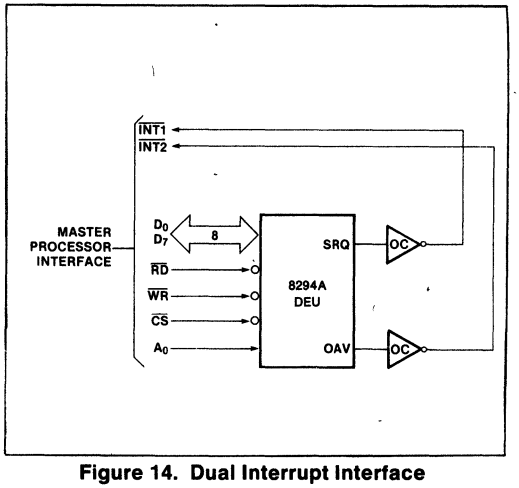


Figure 14. Dual Interrupt Interface

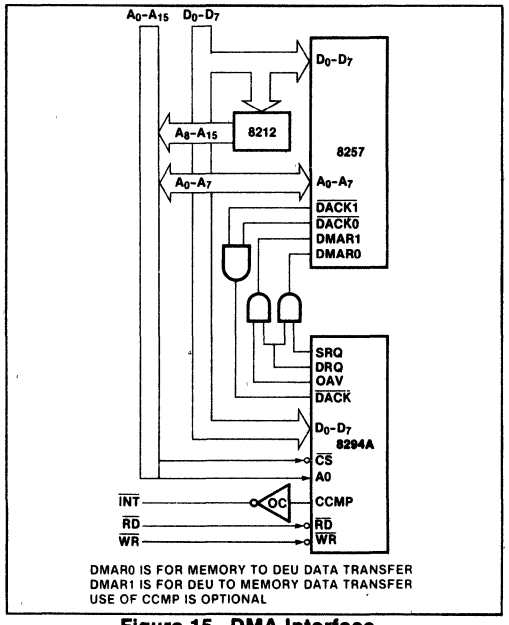


Figure 15. DMA Interface

OSCILLATOR AND TIMING CIRCUITS

The 8294A's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 16.

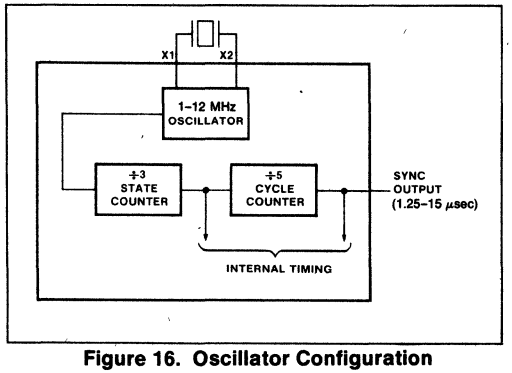


Figure 16. Oscillator Configuration

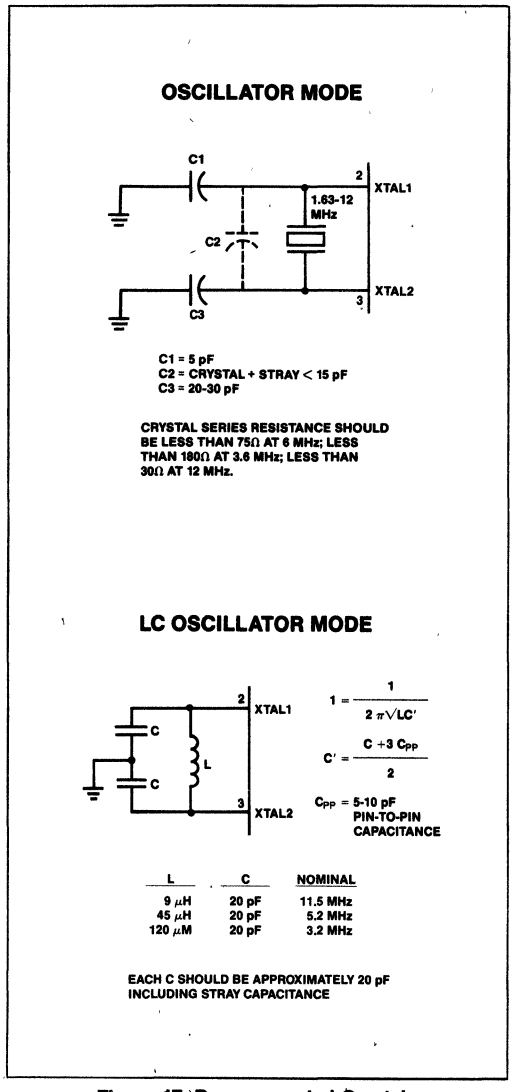
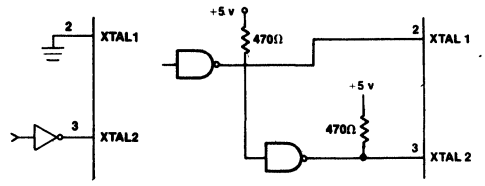


Figure 17. Recommended Crystal

DRIVING FROM EXTERNAL SOURCE—TWO OPTIONS



FOR THE 8294A XTAL2 MUST BE HIGH 35–65% OF THE PERIOD

RISE AND FALL TIMES MUST NOT EXCEED 10 ns

RESISTOR TO V_{CC} IS NEEDED TO ENSURE V_{IH} = 3.0v IF TTL CIRCUITRY IS USED

Figure 18. Recommended Connection for External Clock Signal

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except X_1, X_2, RESET)	−0.5		0.8	V	
V_{IL1}	Input Low Voltage (X_1, X_2, RESET	−0.5		0.6	V	
V_{IH}	Input High Voltage (All Except X_1, X_2, RESET)	2.2		V_{CC}	V	
V_{IH1}	Input High Voltage (X_1, X_2, RESET)	3.0		V_{CC}	V	
V_{IH2}	Input High Voltage (X_2)	2.2		V_{CC}	V	
V_{OL}	Output Low Voltage (D_0-D_7)			0.45	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage (All Other Outputs)			0.45	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage (D_0-D_7)	2.4			V	I_{OH} = −400 μA
V_{OH1}	Output High voltage (All Other Outputs)	2.4			V	I_{OH} = −50 μA
I_{IL}	Input Leakage Current (RD, WR, CS, A_0)			± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{OFL}	Output Leakage Current (D_0-D_7, High Z State)			± 10	μA	V_{SS} + 0.45 ≤ $V_{OUT} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		5	20	mA	
$I_{DD} + I_{CC}$	Total Supply Current		60	135	mA	
I_{LI}	Low Input Load Current (Pins 24, 27-38)			0.3	mA	V_{IL} = 0.8V
I_{LI1}	Low Input Load Current (RESET)			0.2	mA	V_{IL} = 0.8V
I_{IH}	Input High Leakage Current (Pins 24, 27-38)			100	μA	$V_{IN} = V_{CC}$
C_{IN}	Input Capacitance			10	pF	
$C_{I/O}$	I/O Capacitance			20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ± 10%, V_{SS} = 0V)

DBB READ

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	$\overline{CS}$, A_0 Setup to $\overline{RD}$ ↓	0		ns	
t_{RA}	$\overline{CS}$, A_0 Hold After $\overline{RD}$ ↑	0		ns	
t_{RR}	$\overline{RD}$ Pulse Width	160		ns	
t_{AD}	$\overline{CS}$, A_0 to Data Out Delay		130	ns	C_L = 100 pF
t_{RD}	$\overline{RD}$ ↓ to Data Out Delay		130	ns	C_L = 100 pF
t_{DF}	$\overline{RD}$ ↑ to Data Float Delay		85	ns	
t_{CY}	Cycle Time	1.25	15	μs	1-12 MHz Crystal

DBB WRITE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	$\overline{CS}$, A_0 Setup to $\overline{WR}$ ↓	0		ns	
t_{WA}	$\overline{CS}$, A_0 Hold After $\overline{WR}$ ↑	0		ns	
t_{WW}	$\overline{WR}$ Pulse Width	160		ns	
t_{DW}	Data Setup to $\overline{WR}$ ↑	130		ns	
t_{WD}	Data Hold to $\overline{WR}$ ↑	0		ns	

DMA AND INTERRUPT TIMING

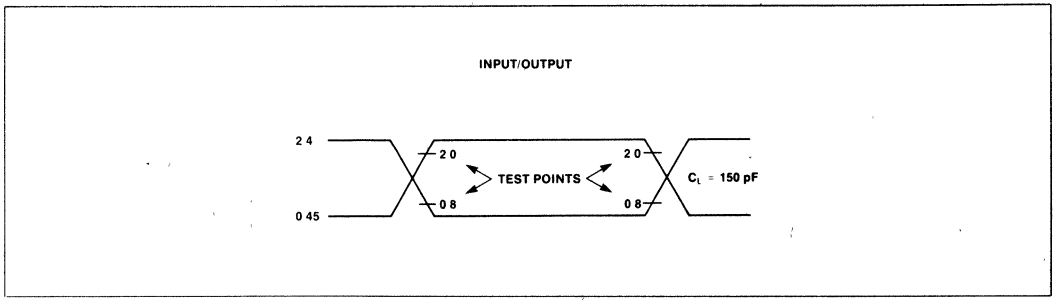
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{ACC}	$\overline{DACK}$ Setup to Control	0		ns	
t_{CAC}	$\overline{DACK}$ Hold After Control	0		ns	
t_{ACD}	$\overline{DACK}$ to Data Valid		130	ns	C_L =100 pF
t_{CRQ}	Control L.E. to DRQ T.E.		100	ns	
t_{CI}	Control T.E. to Interrupt T.E.		400	ns	

CLOCK

Symbol	Parameter	8042		8742		Units
		Min.	Max.	Min.	Max.	
t_{CY}	Cycle Time	1.25	9.20	1.25	9.20	μs[1]
t_{CYC}	Clock Period	83.3	613	83.3	613	ns
t_{PWH}	Clock High Time	33		38		ns
t_{PWL}	Clock Low Time	33		38		ns
t_R	Clock Rise Time		10		10	ns
t_F	Clock Fall Time		10		10	ns

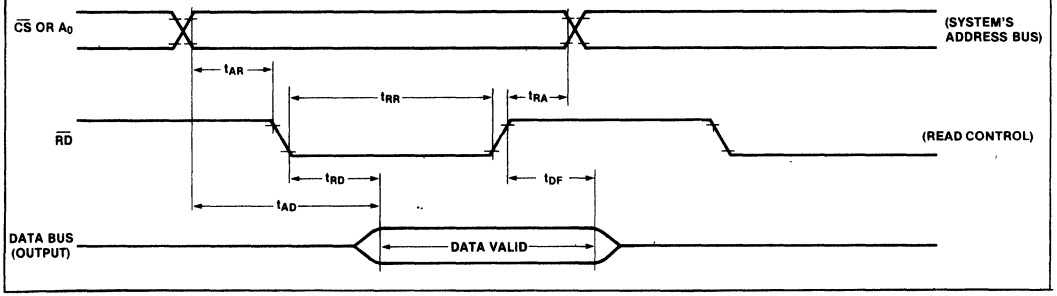
NOTES:
1 t_{CY} = 15/f(XTAL)

A.C. TESTING INPUT, OUTPUT WAVEFORM

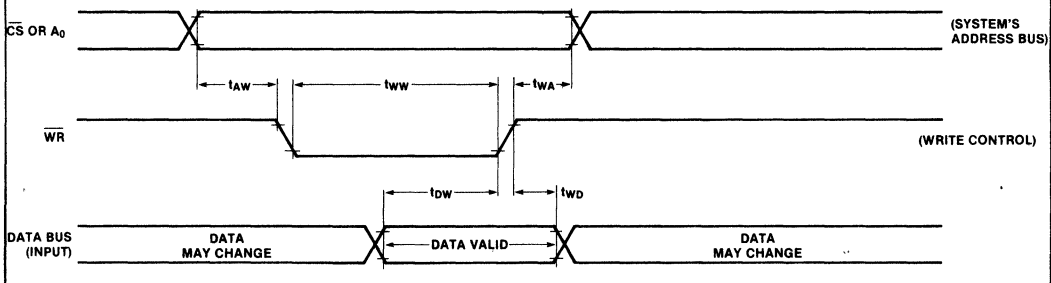


WAVEFORMS

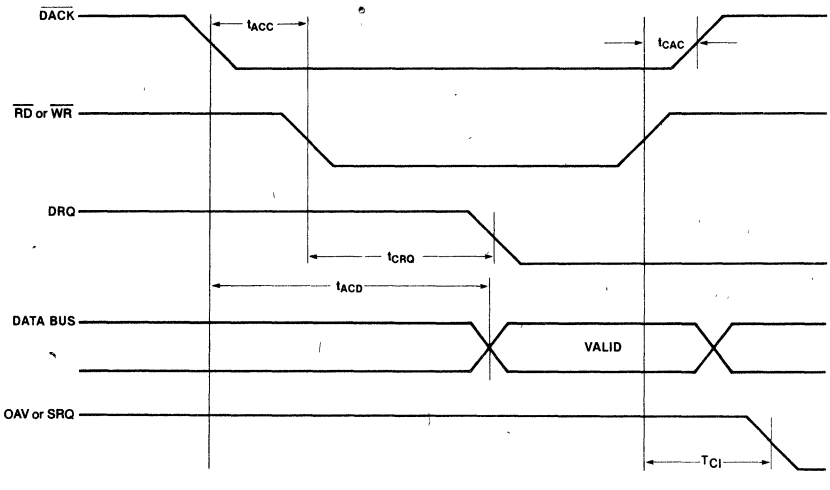
READ OPERATION—OUTPUT BUFFER REGISTER



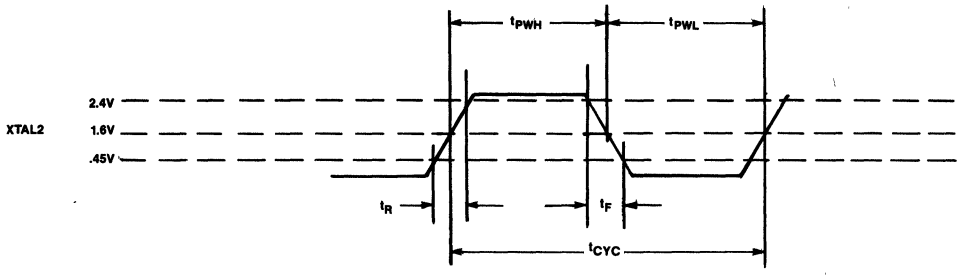
WRITE OPERATION—INPUT BUFFER REGISTER



DMA AND INTERRUPT TIMING



CLOCK TIMING



Video Display 8

Peripherals Section

intel

APPLICATION NOTE

AP-62

November 1979

A Low Cost CRT Terminal Using The 8275

John Katausky
Peripherals Applications

1. INTRODUCTION

The purpose of this application note is to provide the reader with the design concepts and factual tools needed to integrate Intel peripherals and microprocessors into a low cost raster scan CRT terminal. A previously published application note, AP-32, presented one possible solution to the CRT design question. This application note expands upon the theme established in AP-32 and demonstrates how to design a functional CRT terminal while keeping the parts count to a minimum.

For convenience, this application note is divided into seven general sections:

1. Introduction
2. CRT Basics
3. 8275 Description
4. Design Background
5. Circuit Description
6. Software Description
7. Appendix

There is no question that microprocessors and LSI peripherals have had a significant role in the evolution of CRT terminals. Microprocessors have allowed design engineers to incorporate an abundance of sophisticated features into terminals that were previously mere slaves to a larger processor. To complement microprocessors, LSI peripherals have reduced component count in many support areas. A typical LSI peripheral easily replaces between 30 and 70 SSI and MSI packages, and offers features and flexibility that are usually not available in most hardware designs. In addition to replacing a whole circuit board of random logic, LSI circuits also reduce the cost and increase the reliability of design. Fewer interconnects increases mechanical reliability and fewer parts decreases the power consumption and hence, the overall reliability of the design. The reduction of components also yields a circuit that is easier to debug during the actual manufacturing phase of a product.

Until the era of advanced LSI circuitry, a typical CRT terminal consisted of 80 to 200 or more SSI and MSI packages. The first microprocessors and peripherals dropped this component count to between 30 and 50 packages. This application note describes a CRT terminal that uses 20 packages.

2. CRT BASICS

The raster scan display gets its name from the fact that the image displayed on the CRT is built up by generating a series of lines (raster) across the face of the CRT. Usually, the beam starts in the upper left hand corner of the display and simultaneously moves left to right and top to bottom to put a series of zig-zag lines on the screen (Fig. 2.1). Two simultaneously operating independent circuits control the vertical and horizontal movement of the beam.

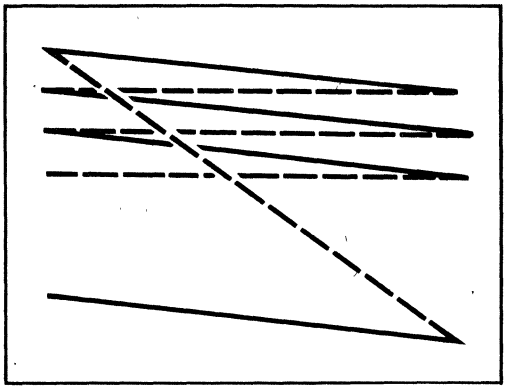


Figure 2-1. Raster Scan

As the electron beam moves across the face of the CRT, a third circuit controls the current flowing in the beam. By varying the current in the electron beam the image on the CRT can be made to be as bright or as dark as the user desires. This allows any desired pattern to be displayed.

When the beam reaches the end of a line, it is brought back to the beginning of the next line at a rate that is much faster than was used to generate the line. This action is referred to as "retrace". During the retrace period the electron beam is usually shut off so that it doesn't appear on the screen.

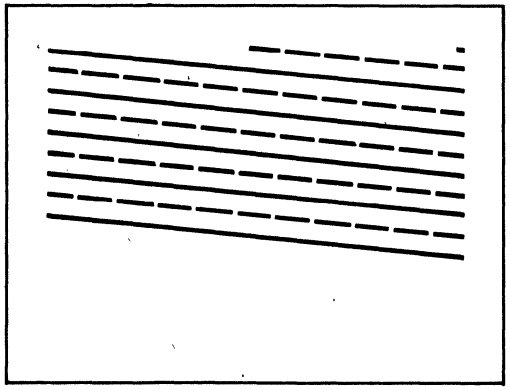
As the electron beam is moving across the screen horizontally, it is also moving downward. Because of this, each successive line starts slightly below the previous line. When the beam finally reaches the bottom right hand corner of the screen, it retraces vertically back to the top left hand corner. The time it takes for the beam to move from the top of the screen to the bottom and back again to the top is usually referred to as a "frame". In the United States, commercial television broadcast use 15,750 Hz as the horizontal sweep frequency (63.5 microseconds per horizontal line) and 60 Hz as the vertical sweep frequency or "frame" (16.67 milliseconds per vertical frame).

Although, the 60 Hz vertical frame and the 15,750 Hz horizontal line are the standards used by commercial broadcasts, they are by no means the only frequency at which CRT's can operate. In fact, many CRT displays use a horizontal scan that is around 18 KHz to 22 KHz and some even exceed 30 KHz. As the

horizontal frequency increases, the number of horizontal lines per frame increases. Hence, the resolution on the vertical axis increases. This increased resolution is needed on high density graphic displays and on special text editing terminals that display many lines of text on the CRT.

Although many CRT's operate at non-standard horizontal frequencies, very few operate at vertical frequencies other than 60 Hz. If a vertical frequency other than 60 Hz is chosen, any external or internal magnetic or electrical variations at 60 Hz will modulate the electron beam and the image on the screen will be unstable. Since, in the United States, the power line frequency happens to be 60 Hz, there is a good chance for 60 Hz interference to exist. Transformers can cause 60 Hz magnetic fields and power supply ripple can cause 60 Hz electrical variations. To overcome this, special shielding and power supply regulation must be employed. In this design, we will assume a standard frame rate of 60 Hz and a standard line rate of 15,750 Hz.

By dividing the 63.5 microsecond horizontal line rate into the 16.67 millisecond vertical rate, it is found that there are 262.5 horizontal lines per vertical frame. At first, the half line may seem a bit odd, but actually it allows the resolution on the CRT to be effectively doubled. This is done by inserting a second set of horizontal lines between the first set (interlacing). In an interlaced system the line sets are not generated simultaneously. In a 60 Hz system, first all of the even-numbered lines are scanned: 0, 2, 4,...524. Then all the odd-numbered lines: 1, 3, 5,... 525. Each set of lines usually contains different data (Fig. 2.2).



EVEN FIELD
ODD FIELD
RETRACE LINES NOT SHOWN

Figure 2-2. Interlaced Scan

Although interlacing provides greater resolution, it also has some distinct disadvantages. First of all, the circuitry needed to generate the extra half horizontal line per frame is quite complex when compared to a noninterlaced design, which requires an integer number of horizontal lines per frame. Next, the overall vertical refresh rate is half that of a noninterlaced display. As a result, flicker may result when the CRT uses high speed phosphors. To keep things as simple as possible, this design uses the noninterlaced approach.

The first thing any CRT controller must do is generate pulses that define the horizontal line timing and the vertical frame timing. This is usually done by dividing a crystal reference source by some appropriate numbers. On most raster scan CRT's the horizontal frequency is very forgiving and can vary by around 500 Hz or so and produce no ill effects. This means that the CRT itself can track a horizontal frequency between 15250 Hz and 16250 Hz, or in other words, there can be 256 to 270 horizontal lines per vertical frame. But, as mentioned earlier, the vertical frequency should be 60 Hz to insure stability.

The characters that are viewed on the screen are formed by a series of dots that are shifted out of the controller while the electron beam moves across the face of the CRT. The circuits that create this timing are referred to as the dot clock and character clock. The character clock is equal to the dot clock divided by the number of dots used to form a character along the horizontal axis and the dot clock is calculated by the following equation:

DOT CLOCK (Hz) = (N + R) * D * L * F

where N is the number of displayed characters per row,

R is the number of retrace character time increments,

D is the number of dots per character,

L is the number of horizontal lines per frame and

F is the frame rate in Hz.

In this design N = 80, R = 20, D = 7, L = 270, and F = 60 Hz. If the numbers are plugged in, the dot clock is found to be 11.34 MHz.

The retrace number, R, may vary from system to system because it is used to establish the margins on the left and right hand sides of the CRT. In this particular design R = 20 was empirically found it be optimum. The number of dots per character may vary depending on the character generator used and the number of dot clocks the designer wants to place between characters. This design uses a 5 X 7 dot matrix and allows 2 dot clock periods between characters (see Fig. 2.3); since 5 + 2 equals 7, we find that D = 7.

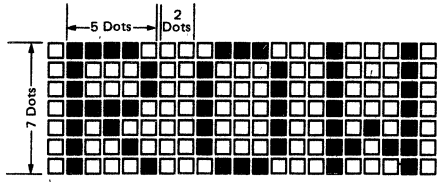


Figure 2-3. 5 X 7 Dot Matrix

The number of lines per frame can be determined by the following equation:

L = (H * Z) + V

where, H is the number of horizontal lines per character,

Z is the number of character lines per frame and

V is the number of horizontal lines during vertical retrace. In this design, a 5 X 7 dot matrix is to be placed on a 7 X 10 field, so H = 10. Also, 25 lines are to be displayed, so Z = 25. As mentioned before, V = 20. When the numbers are plugged into the equation, L is found to be equal to 270 lines per frame.

The designer should be cautioned that these numbers are interrelated and that to guarantee proper operation on a standard raster scan CRT, L should be between 256 and 270. If L does not lie within these bounds the horizontal circuits of the CRT may not be able to lock onto the driving signal and the image will roll horizontally. The chosen L of 270 yields a horizontal frequency of 16,200 KHz on a 60 Hz frame and this number is within the 500 Hz tolerance mentioned earlier.

The V number is chosen to match the CRT in much the same manner as the R number mentioned earlier. When the electron beam reaches the bottom right corner of the screen it must retrace vertically to the top left corner. This retrace action requires time, usually between 900-1200 microseconds. To allow for this, enough horizontal sync times must be inserted during vertical retrace. Twenty horizontal sync times at 61.5 microseconds yield a total of 1234.5 microseconds, which is enough time to allow the beam to return to the top of the screen.

The choices of H and Z largely relate to system design preference. As H increases, the character size along the vertical axis increases. Z is simply the number of lines of characters that are displayed and this, of course, is entirely a system design option.

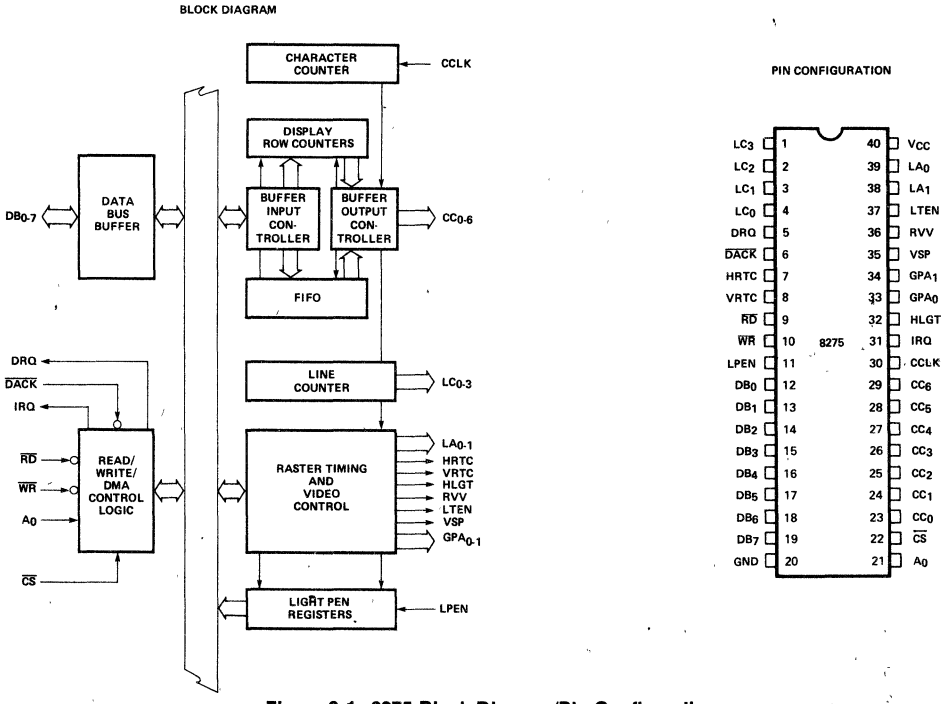


Figure 3-1. 8275 Block Diagram/Pin Configuration

3. 8275 DESCRIPTION

A block diagram and pin configuration of the 8275 are shown in Fig. 3.1. The following is a description of the general capabilities of the 8275.

3.1 CRT DISPLAY REFRESHING

The 8275, having been programmed by the designer to a specific screen format, generates a series of DMA request signals, resulting in the transfer of a row of characters from display memory to the 8275's row buffers. The 8275 presents the character codes to an external character generator ROM by using outputs CCO-CC6. External dot timing logic is then used to transfer the parallel output data from the character generator ROM serially to the video input of the CRT. The character rows are displayed on the CRT one line at a time. Line count outputs LC0-LC3 are applied to the character generator ROM to perform the line selection function. The display process is illustrated in Figure 3.2. The entire process is repeated for each display row. At the beginning of the last displayed row, the 8275 issues an interrupt by setting the IRQ output line. The 8275 interrupt output will normally be connected to the interrupt input of the system central processor.

The interrupt causes the CPU to execute an interrupt service subroutine. The service subroutine typically re-initializes DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and/or executes other appropriate functions. A block diagram of a CRT system implemented with the 8275 CRT Controller is provided in Figure 3.3. Proper CRT refreshing requires that certain 8275 parameters be programmed prior to the beginning of display operation. The 8275 has two types of programming registers, the Command Registers (CREG) and the Parameter Registers (PREG). It also has a Status Register (SREG). The Command Registers may only be written to and the Status Registers may only be read. The 8275 expects to receive a command followed by a sequence of from 0 to 4 parameters, depending on the command. The 8275 instruction set consist of the eight commands shown in Figure 3.4.

To establish the format of the display, the 8275 provides a number of user programmable display format parameters. Display formats having from 1 to 80 characters per row, 1 to 64 rows per screen, and 1 to 16 horizontal lines per row are available.

In addition to transferring characters from memory

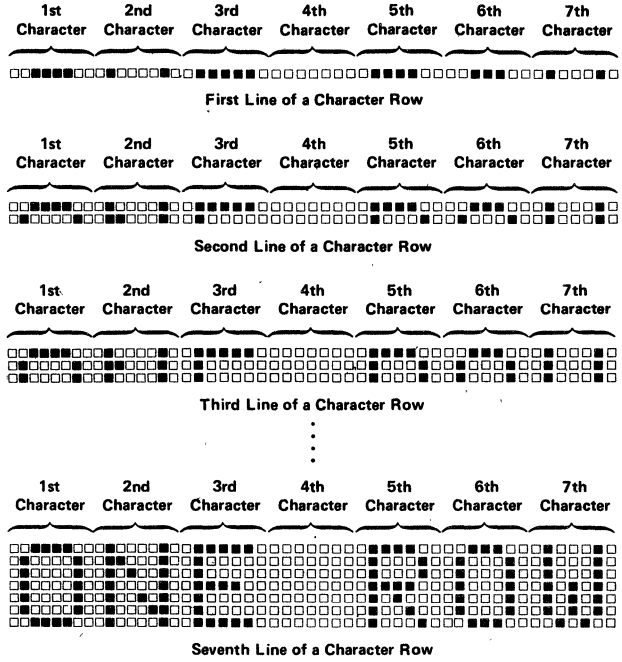


Figure 3-2. 8275 Row Display

AFN-01304A

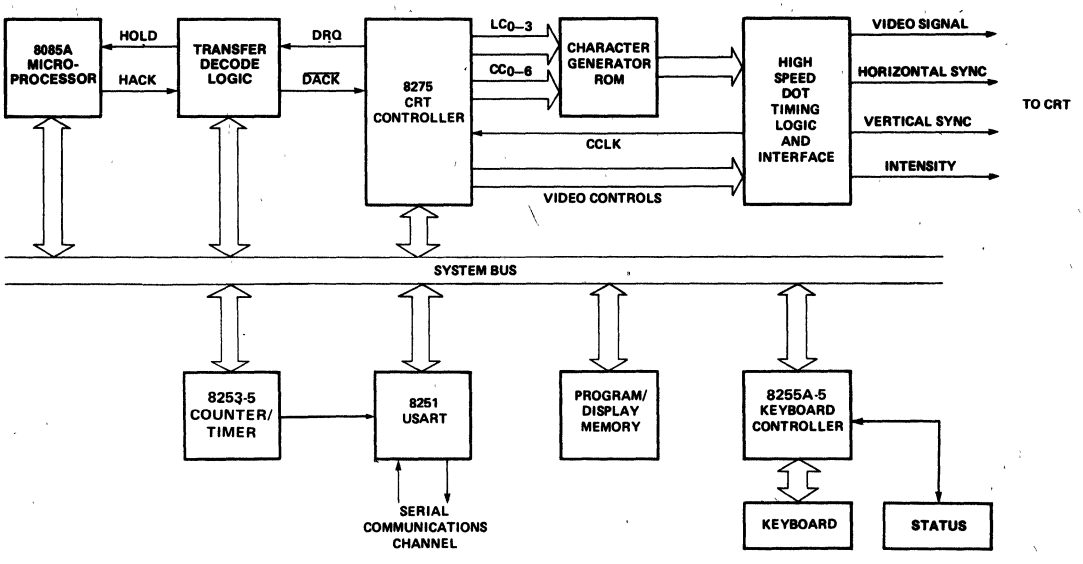


Figure 3-3. CRT System Block Diagram

to the CRT screen, the 8275 features cursor position control. The cursor position may be programmed, via X and Y cursor position registers, to any character position on the display. The user may select from four cursor formats. Blinking or non-blinking underline and reverse video block cursors are available.

3.2 CRT TIMING

The 8275 provides two timing outputs, HRTC and VRTC, which are utilized in synchronizing CRT horizontal and vertical oscillators to the 8275 refresh cycle. In addition, whenever HRTC or VRTC is active, a third timing output, VSP (Video Suppress) is true, providing a blinking signal to the dot timing logic. The dot timing logic will normally inhibit the video output to the CRT during the time when video suppress signal is true. An additional timing output, LTEN (Light Enable) is used to provide the ability to force the video output high regardless of the state of VSP. This feature is used by the 8275 to place a cursor on the screen and to control attribute functions. Attributes will be considered in the next section.

The HLGT (Highlight) output allows an attribute function to increase the CRT beam intensity to a level greater than normal. The fifth timing signal, RVV (Reverse Video) will, when enabled, cause the system video output to be inverted.

COMMAND	NO. OF PARAMETER BYTES	NOTES
RESET	4	Display format parameters required
START DISPLAY	0	DMA operation parameters included in command
STOP DISPLAY	0	—
READ LIGHT PEN	2	—
LOAD CURSOR	2	Cursor X,Y position parameters required
ENABLE INTERRUPT	0	—
DISABLE INTERRUPT	0	—
PRESET COUNTERS	0	Clears all internal counters

Figure 3-4. 8275's Instruction Set

Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS LA_1	LA_0	VSP	LTEN	SYMBOL	DESCRIPTION
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	1	0	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1110	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1111	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						

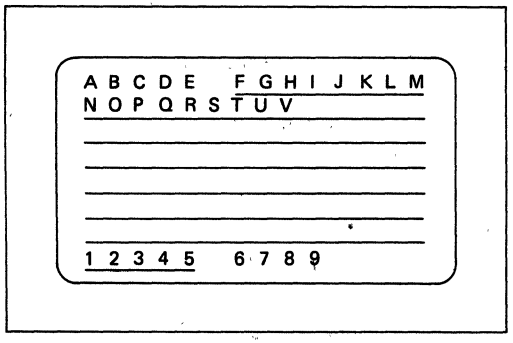
*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

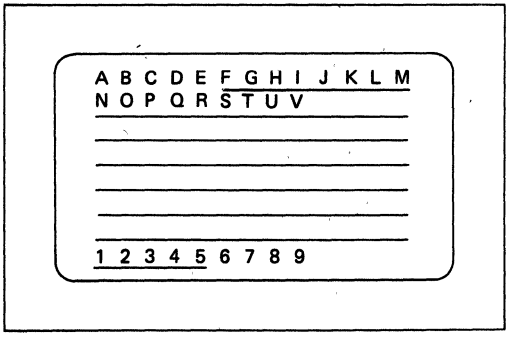
Blinking is active when B = 1.

Highlight is active when H = 1.

Figure 3-5. Character Attributes



EXAMPLE OF THE VISIBLE FIELD ATTRIBUTE MODE (UNDERLINE ATTRIBUTE)



EXAMPLE OF THE INVISIBLE FIELD ATTRIBUTE MODE (UNDERLINE ATTRIBUTE)

Figure 3-6. Field Attribute Examples

3.3 SPECIAL FUNCTIONS

VISUAL ATTRIBUTES—Visual attributes are special codes which, when retrieved from display memory by the 8275, affect the visual characteristics of a character position or field of characters. Two types of visual attributes exist, character attributes and field attributes.

Character Attribute Codes: Character attribute codes can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LAO-LA1), the Video Suppression output (VSP), and the Light Enable output (LTEN). The dot timing logic uses these signals to generate the proper symbols. Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT). Character attributes were designed to produce the graphic symbols shown in Figure 3.5.

Field Attribute Codes: The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the field attribute code up to, and including, the character which precedes the next field attribute code, or up to the end of the frame.

There are six field attributes:

1. *Blink* — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight* — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video* — Characters following the code are caused to appear in reverse video format by activating the Reverse Video output (RVV).
4. *Underline* — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).
5. *General Purpose* — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. These attributes may be used to select colors or perform other desired control functions.

The 8275 can be programmed to provide visible or invisible field attribute characters as shown in Figure 3.6. If the 8275 is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character. If the 8275 is programmed in the invisible field attribute mode, the 8275 row buffer FIFOs are activated. The FIFOs effectively lengthen the row buffers by 16 characters, making room for up to 16 field attribute characters per display row. The FIFOs are 126 characters by 7 bits in size. When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the next character in the proper FIFO. When a field attribute is placed in the buffer output controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CCO-6). The chosen attributes are also activated.

AFN-01304A

LIGHT PEN DETECTION — A light pen consists fundamentally of a switch and light sensor. When the light pen is pressed against the CRT screen, the switch enables the light sensor. When the raster sweep coincides with the light sensor position on the display, the light pen output is input and the row and character position coordinates are stored in two 8275 internal registers. These registers can be read by the microprocessor.

SPECIAL CODES — Four special codes may be used to help reduce memory, software, or DMA overhead. These codes are placed in character positions in display memory.

1. *End Of Row Code* - Activates VSP. VSP remains active until the end of the line is reached. While VSP is active, the screen is blanked.
2. *End Of Row-Stop DMA Code* - Causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the row buffer. It affects the display in the same way as the End of Row Code.
3. *End Of Screen Code* - Activates VSP. VSP remains active until the end of the frame is reached.
4. *End Of Screen-Stop DMA Code* - Causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the row buffer. It affects the display in the same way as the End of Screen Code.

PROGRAMMABLE DMA BURST CONTROL — The 8275 can be programmed to request single-byte DMA transfers of DMA burst transfers of 2, 4, or 8 characters per burst. The interval between bursts is also programmable. This allows the user to tailor the DMA overhead to fit the system needs.

4. DESIGN BACKGROUND

4.1 DESIGN PHILOSOPHY

Since the cost of any CRT system is somewhat proportional to parts count, arriving at a minimum part count solution without sacrificing performance has been the motivating force throughout this design effort. To successfully design a CRT terminal and keep the parts count to a minimum, a few things became immediately apparent.

1. An 8085 should be used.
2. Address and data buffering should be eliminated.
3. Multi-port memory should be eliminated.
4. DMA should be eliminated.

Decision 1 is obvious, the 8085's on-board clock generator, bus controller and vectored interrupts greatly reduce the overall part count considerably. Decision 2 is fairly obvious; if a circuit can be designed so that loading on the data and address lines is kept to a minimum, both the data and address buffers can be eliminated. This easily saves three to eight packages and reduces the power consumption of the design. Both decisions 3 and 4 require a basic understanding of current CRT design concepts.

In any CRT design, extreme time conflicts are created because all essential elements require access to the bus. The CPU needs to access the memory to control the system and to handle the incoming characters, but, at the same time, the CRT controller needs to access the memory to keep the raster scan display refreshed. To resolve this conflict two common techniques are employed, page buffering and line buffering.

In the page buffering approach the entire screen memory is isolated from the rest of the system. This isolation is usually accomplished with three-state buffers or two line to one line multiplexers. Of course, whenever a character needs to be manipulated the CPU must gain access to the buffered memory and, again, possible contention between the CPU and the CRT controller results. This contention is usually resolved in one of two ways, (1) the CPU is always given priority, or; (2) the CPU is allowed to access the buffered memory only during horizontal and vertical retrace times.

Approach 1 is the easiest to implement from a hardware point of view, but if the CPU always has priority the display may temporarily blink or "flicker" while the CPU accesses the display memory. This, of course, occurs because when the CPU accesses the display memory the CRT controller is not able to retrieve a character, so the display must be blanked during this time. Aesethically, this "flickering" is not desirable, so approach 2 is often used.

The second approach eliminates the display flickering encountered in the previously mentioned technique, but additional hardware is required. Usually the vertical and horizontal blank signals are gated with the buffered memory select lines and this line is used to control the CPU's ready line. So, if the CPU wants to use the buffered memory, its ready line is asserted until horizontal or vertical retrace times. This, of course, will impact the CPU's overall through put.

Both page buffered approaches require a significant amount of additional hardware and for the most part are not well suited for a minimum parts count type of terminal. This guides us to the line buffered approach. This approach eliminates the separate buffered memory for the display, but, at the same time, introduces a few new problems that must be solved.

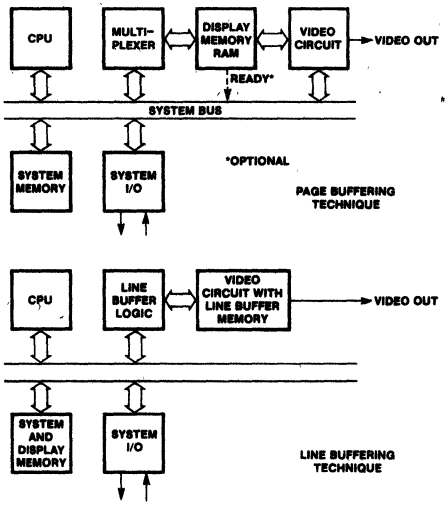


Figure 4-1. Line Buffering Technique

```
CLOCK CYCLES    SEQ     SOURCE STATEMENT

    10          1       PUSH    PSW             ;SAVE A AND FLAGS
    10          2       PUSH    H               ;SAVE H AND L
    10          3       PUSH    D               ;SAVE D AND E
    10          4       LXI     H,0080H         ;ZERO H AND L
    10          5       DAD     SP              ;PUT STACK POINTER IN H AND L
     4          6       XCHG                    ;PUT STACK IN D AND E
    16          7       LHLD    CURAD           ;GET POINTER
     6          8       SPHL                    ;PUT CURRENT LINE INTO SP
     7          9       MVI     A,0C0H          ;SET MASK FOR SIM
     4          10      SIM                     ;SET SPECIAL TRANSFER BIT
   400          11      POP     H               ;DO 40 POPS
     4          12      RRC                     ;SET UP A
     4          13      SIM                     ;GO BACK TO NORMAL MODE
    10          14      LXI     H,0000H         ;ZERO HL
    10          15      DAD     SP              ;ADD STACK
     4          16      XCHG                    ;PUT STACK IN H AND L
     6          17      SPHL                    ;RESTORE STACK
    10          18      LXI     H,LAST          ;PUT BOTTOM DISPLAY IN H AND L
     4          19      XCHG                    ;SWAP REGISTERS
     4          20      MOV     A,D             ;PUT HIGH ORDER IN A
     4          21      CMP     H               ;SEE IF SAME AS H
  7/10          22      JNZ     KPTK            ;IF NOT LEAVE
     4          23      MOV     A,E             ;PUT LOW ORDER IN A
     4          24      CMP     L               ;SEE IF SAME AS L
  7/10          25      JNZ     KPTK            ;IF NOT LEAVE
    10          26      LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF SCREEN MEMORY
    16          27 KPTK: SHLD   CURAD           ;PUT BACK CURRENT ADDRESS
     7          28      MVI     A,18H           ;GET MASK BYTE
     4          29      SIM                     ;SET INTERRUPT MASK
    10          30      POP     D               ;GET D AND E
    10          31      POP     H               ;GET H AND L
    10          32      POP     PSW             ;GET A AND FLAGS
     4          33      EI                      ;ENABLE INTERRUPTS
    10          34      RET                     ;GO BACK

    TOTAL CLOCK CYCLES = 650 (WORST CASE)

    WITH A 6 144 MHZ CRYSTAL TOTAL TIME TO FILL

    ROW BUFFER ON 8275 = 650 * .325 = 211.25 MICROSECONDS
```

Figure 4-2. Routine To Load 8275's Row Buffers

In the line buffered approach both the CPU and the CRT controller share the same memory. Every time the CRT controller needs a new character or line of data, normal processing activity is halted and the CRT controller accesses memory and displays the data. Just how the CRT controller needs to acquire the display data greatly affects the performance of the overall system. Whether the CRT controller needs to gain access to the main memory to acquire a single character or a complete line of data depends on the presence or absence of a separate line or row buffer.

If no row buffer is present the CRT controller must go to the main memory to fetch every character. This of course, is not a very efficient approach because the processor will be forced to relinquish the bus 70% to 80% of the time. So much processor inactivity greatly affects the overall system performance. In fact terminals that use this approach are typically limited to around 1200 to 2400 baud on their serial communication channels. This low baud rate is in general not acceptable, hence this approach was not chosen.

If a separate row buffer is employed the CRT controller only has to access the memory once for each displayed character per line. This forces the processor to relinquish the bus only about 20% to 35% of the time and a full 4800 to 9600 baud can be achieved. Figure 4.1 illustrates these different techniques.

The 8275 CRT controller is ideal for implementing the row buffer approach because the row buffer is contained on the device itself. In fact, the 8275 contains two 80-byte row buffers. The presence of two row buffers allow one buffer to be filled while the other buffer is displaying the data. This dual row buffer approach enhances CPU performance even further.

4.2 USING THE 8275 WITHOUT DMA

Until now the process of filling the row buffer has only been alluded to. In reality, a DMA technique is usually used. This approach was demonstrated in AP-32 where an 8257 DMA controller was mated to an 8275 CRT controller. In order to minimize component count, this design eliminates the DMA controller and its associated circuitry while replacing them with a special interrupt-driven transfer.

The only real concern with using the 8275 in an interrupt-driven transfer mode is speed. Eighty characters must be loaded into the 8275 every 617 microseconds and the processor must also have time to perform all the other tasks that are required. To minimize the overhead associated with loading the characters into the 8275 a special technique was employed. This technique involves setting a special

transfer bit and executing a string of POP instructions. The string of POP instructions is used to rapidly move the data from the memory into the 8275. Figure 4.2 shows the basic software structure.

In this design the 8085's SOD line was used as the special transfer bit. In order to perform the transfer properly this special bit must do two things: (1) turn processor reads into $\overline{DACK}$ plus $\overline{WR}$ for the 8275 and (2) mask processor fetch cycles from the 8275, so that a fetch cycle does not write into the 8275. Conventional logic could have been used to implement this special function, but in this design a small bipolar programmable read only memory was used. Figure 4.3 shows a basic version of the hardware..

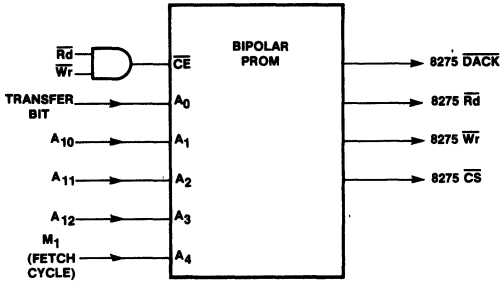


Figure 4-3. Simplified Version of Hardware Decoder

At first, it may seem strange that we are supplying a $\overline{DACK}$ when no DMA controller exist in the system. But the reader should be aware that all Intel peripheral devices that have DMA lines actually use $\overline{DACK}$ as a chip select for the data. So, when you want to write a command or read status you assert $\overline{CS}$ and $\overline{WR}$ or $\overline{RD}$, but when you want to read or write data you assert $\overline{DACK}$ and $\overline{RD}$ or $\overline{WR}$. The peripheral device doesn't "know" if a DMA controller is in the circuit or not. In passing, it should be mentioned that $\overline{DACK}$ and $\overline{CS}$ should not be asserted on the same device at the same time, since this combination yields an undefined result.

This POP technique actually compares quite favorably in terms of time to the DMA technique. One POP instruction transfers two bytes of data to the 8275 and takes 10 CPU clock cycles to execute, for a net transfer rate of one byte every five clock cycles. The DMA controller takes four clock cycles to transfer one byte but, some time is lost in synchronization. So the difference between the two techniques is one clock cycle per byte maximum. If we compare the overall speed of the 8085 to the speed of the 8080 used in AP-32, we find that at 3 MHz we can transfer one byte every 1.67 microseconds using the 8085 and POP technique vs. 2 microseconds per byte for the 2 MHz 8080 using DMA.

5. CIRCUIT DESCRIPTION

5.1 SCOPE OF THE PROJECT

A fully functional, microprocessor-based CRT terminal was designed and constructed using the 8275 CRT controller and the 8085 as the controlling element. The terminal had many of the functions found in existing commercial low-cost terminals and more sophisticated features could easily be added with a modest amount of additional software. In order to minimize component count LSI devices were used whenever possible and software was used to replace hardware.

5.2 SYSTEM TARGET SPECIFICATIONS

The design specifications for the CRT terminal were as follows:

Display Format

- 80 characters per display row
- 25 display rows

Character Format

- 5 X 7 dot matrix character contained within a 7 X 10 matrix
- First and seventh columns blanked
- Ninth line cursor position
- Blinking underline cursor

Special Characters Recognized

- Control characters
- Line feed
- Carriage Return
- Backspace
- Form feed

Escape Sequences Recognized

- ESC, A, Cursor up
- ESC, B, Cursor down
- ESC, C, Cursor right
- ESC, D, Cursor left
- ESC, E, Clear screen
- ESC, H, Home cursor
- ESC, J, Erase to the end of the screen
- ESC, K, Erase the current line

Characters Displayed

- 96 ASCII alphanumeric characters
- Special control characters

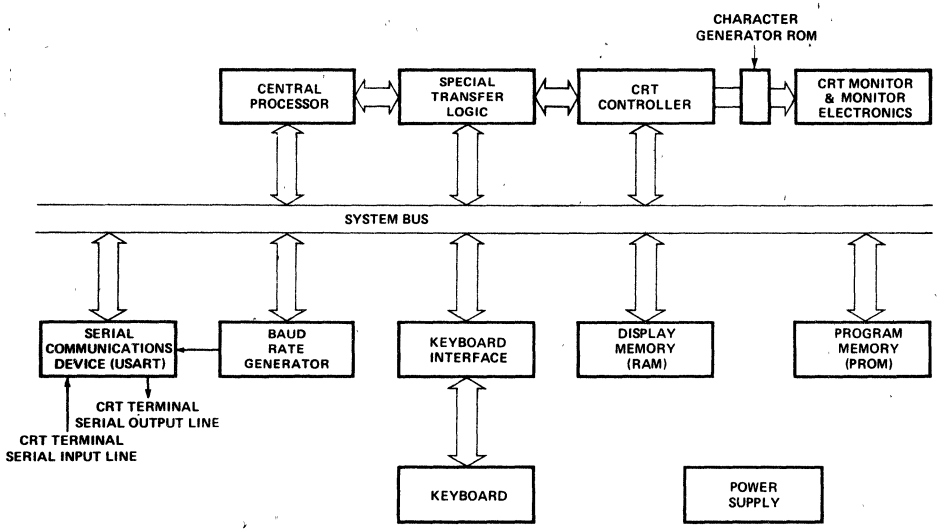


Figure 5-1. CRT Terminal Block Diagram

Characters Transmitted

- 96 ASCII alphanumeric characters
- ASCII control characters

Program Memory

- 2K bytes of 2716 EPROM

Display/Buffer/Stack Memory

- 2K bytes 2114 static memory (4 packages)

Data Rate

- 9600 BAUD using 3MHz 8085

CRT Monitor

- Ball Bros TV-12, 12MHz B.W.

Keyboard

- Any standard un-encoded ASCII keyboard

Screen Refresh Rate

- 60 Hz

5.3 HARDWARE DISCRIPTION

A block diagram of the CRT terminal is shown in Figure 5.1. The diagram shows only the essential system features. A detailed schematic of the CRT is contained in the Appendix. The terminal was constructed on a simple 6" by 6" wire wrap board. Because of the minimum bus loading no buffering of any kind was needed (see Figure 5.2).

The "heart" of the CRT terminal is the 8085 microprocessor. The 8085 initializes all devices in the system, loads the CRT controller, scans the keyboard, assembles the characters to be transmitted, decodes the incoming characters and determines where the character is to be placed on the screen. Clearly, the processor is quite busy.

A standard list of LSI peripheral devices surround the 8085. The 8251A is used as the serial communication link, the 8255A-5 is used to scan the keyboard and read the system variables through a set of

Worst case bus loading:

Data Bus:	8275	20pf
	8255A-5	20pf
	8253-5	20pf
	8253-5	20pf
	8251A	20pf
	2x 2114	10pf
	2716	12pf
	8212	12pf
		114pf max

Only A_8 - A_{15} are important since A_0 - A_7 are latched by the 8212

Address Bus:	4x 2114	20pf
	2716	6pf
		26pf max

This loading assures that all components will be compatible with a 3MHz 8085 and that no wait states will be required

Figure 5-2. Bus Loading

switches, and the 8253 is used as a baud rate generator and as a "horizontal pulse extender" for the 8275.

The 8275 is used as the CRT controller in the system, and a 2716 is used as the character generator. To handle the high speed portion of the terminal the 8275 is surrounded by a small handful of TTL. The program memory is contained in one 2716 EPROM and the data and screen memory use four 2114-type RAMs.

All devices in this system are memory mapped. A bipolar PROM is used to decode all of the addresses for the RAM, ROM, 8275, and 8253. As mentioned earlier, the bipolar prom also turns $\overline{\text{READ}}$s into $\overline{\text{DACK}}$'s and $\overline{\text{WR}}$'s for the 8275. The 8255 and 8253 are decoded by a simple address line chip select method. The total package count for the system is 20, not including the serial line drivers. If this same terminal were designed using the MCS-85 family of integrated circuits, additional part savings could have been realized. The four 2114's could have been replaced by two 8185's and the 8255 and the 2716 program PROM could have been replaced by one 8755. Additionally, since both the 8185 and the 2716 have address latches no 8212 would be needed, so the total parts count could be reduced by three or four packages.

5.4 SYSTEM OPERATION

The 8085 CPU initializes each peripheral to the appropiate mode of operation following system reset. After initialization, the 8085 continually polls the 8251A to see if a character has been sent to the terminal. When a character has been received, the 8085 decodes the character and takes appropriate action. While the 8085 is executing the above "foreground" programs, it is being interrupted once every 617 microseconds by the 8275. This "background" program is used to load the row buffers on the 8275. The 8085 is also interrupted once every frame time, or 16.67 ms, to read the keyboard and the status of the 8275.

As discussed earlier, a special POP technique was used to rapidly move the contents of the display RAM into the 8275's row buffers. The characters are then synchronously transferred to the character code outputs CC0-CC6, connected to the character generator address lines A3-A9 (Figure 5.3). Line count outputs LC0-LC2 from the 8275 are applied to the character generator address lines A0-A2. The 8275 displays character rows one line at a time. The line count outputs are used to determine which line of the character selected by A3-A8 will be displayed. Following the transfer of the first line to the dot timing logic, the line count is incremented and the second line of the character row is selected. This process continues until the last line of the row is transferred to the dot timing logic.

The dot timing logic latches the output of the character generator ROM into a parallel in,serial out synchronous shift register. This shift register is clocked at the dot clock rate (11.34 MHz) and its output constitutes the video input to the CRT.

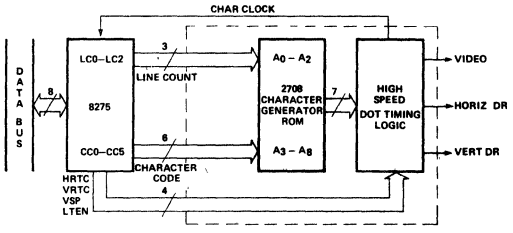


Figure 5-3 Character Generator/Dot Timing Logic Block Diagram

Table 5-1

PARAMETER	RANGE
Vertical Blanking Time (VRTC)	900 μsec nominal
Vertical Drive Pulsewidth	300 μsec ⩽ PW ⩽ 1.4 ms
Horizontal Blanking Time (HRTC)	11 μsec nominal
Horizontal Drive Pulsewidth	25 μsec ⩽ PW ⩽ 30 μsec
Horizontal Repetition Rate	15,750 ±500 pps

5.5 SYSTEM TIMING

Before any specific timing can be calculated it is necessary to determine what constraints the chosen CRT places on the overall timing. The requirements for the Ball Bros. TV-12 monitor are shown in Table 5.1. The data from Table 5.1, the 8275 specifications, and the system target specifications are all that is needed to calculate the system's timing.

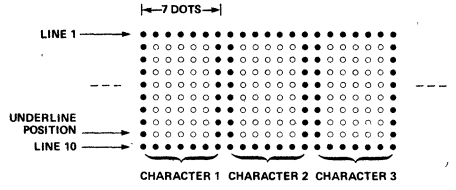


Figure 5-4. Row Format

First, let's select and "match" a few numbers. From our target specifications, we see that each character is displayed on a 7 X 10 field, and is formed by a 5 X 7 dot matrix (Figure 5.4). The 8275 allows the vertical retrace time to be only an integer multiple of the horizontal character line. This means that the total number of horizontal lines in a frame equals 10 times the number of character lines plus the vertical retrace time, which is programmed to be either 1, 2, 3, or 4 character lines. Twenty-five display lines

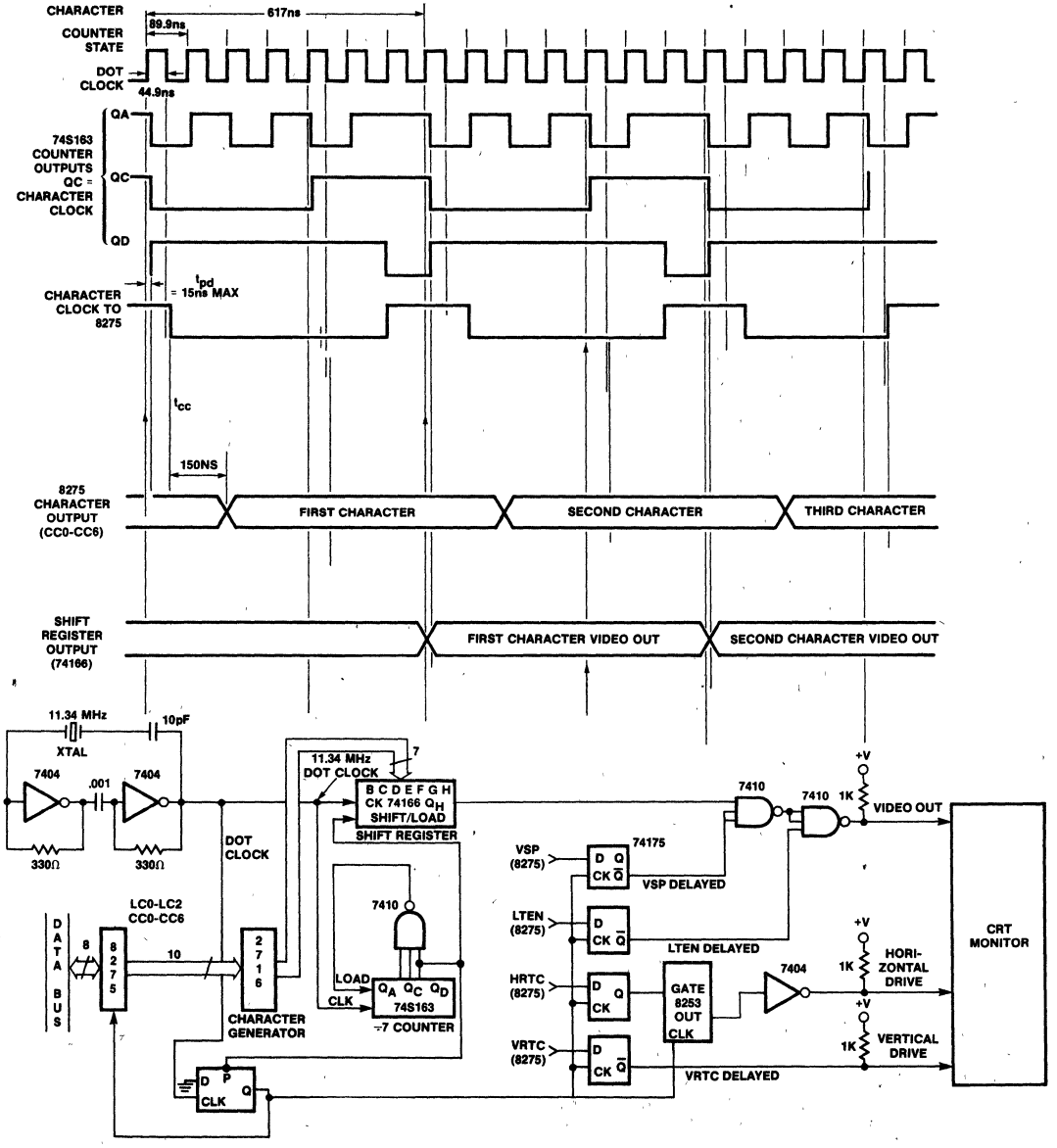


Figure 5-5. Dot Timing Logic

require 250 horizontal lines. So, if we wish to have a horizontal frequency in the neighborhood of 15,750 Hz we must choose either one or two character lines for vertical retrace. To allow for a little more margin at the top and bottom of the screen, two character lines were chosen for vertical retrace. This choice yields a net 250 + 20 = 270 horizontal lines per frame. So, assuming a 60 Hz frame:

60 Hz * 270 = 16,200 Hz (horizontal frequency)

This value falls within our target specification of 15,750 Hz with a 500 Hz variation and also assures timing compatibility with the Ball monitor since, 20 horizontal sync times yield a vertical retract time of:

61.7 microseconds X 20 horizontal sync times = 1.2345 milliseconds

This number meets the nominal VRTC and vertical drive pulse width time for the Ball monitor. A horizontal frequency of 16,200 Hz implies a 1/16,200 = 61.73 microsecond period.

It is now known that the terminal is using 250 horizontal lines to display data and 20 horizontal lines to allow for vertical retrace and that the horizontal frequency is 16,200 Hz. The next thing that needs to be determined is how much time must be allowed for horizontal retrace. Unfortunately, this number depends almost entirely on the monitor used. Usually, this number lies somewhere between 15 and 30 percent of the total horizontal line time, which in this case is 1/16,200 Hz or 61.73 microseconds. Since in most designs a fixed number of characters can be displayed on a horizontal line, it is often useful to express retrace as a given number of character times. In this design, 80 characters can be displayed on a horizontal line and it was empirically found that allowing 20 horizontal character times for retrace gave the best results. So, in reality, there are 100 character times in every given horizontal line, 80 are used to display characters and 20 are used to allow for retrace. It should be noted that if too many character times are used for retrace, less time will be left to display the characters and the display will not "fill out" the screen. Conversely, if not enough character times are allowed for retrace, the display may "run off" the screen.

One hundred character times per complete horizontal line means that each character requires

61.73 microseconds / 100 character times = 617.3 nanoseconds.

If we multiply the 20 horizontal retrace times by the

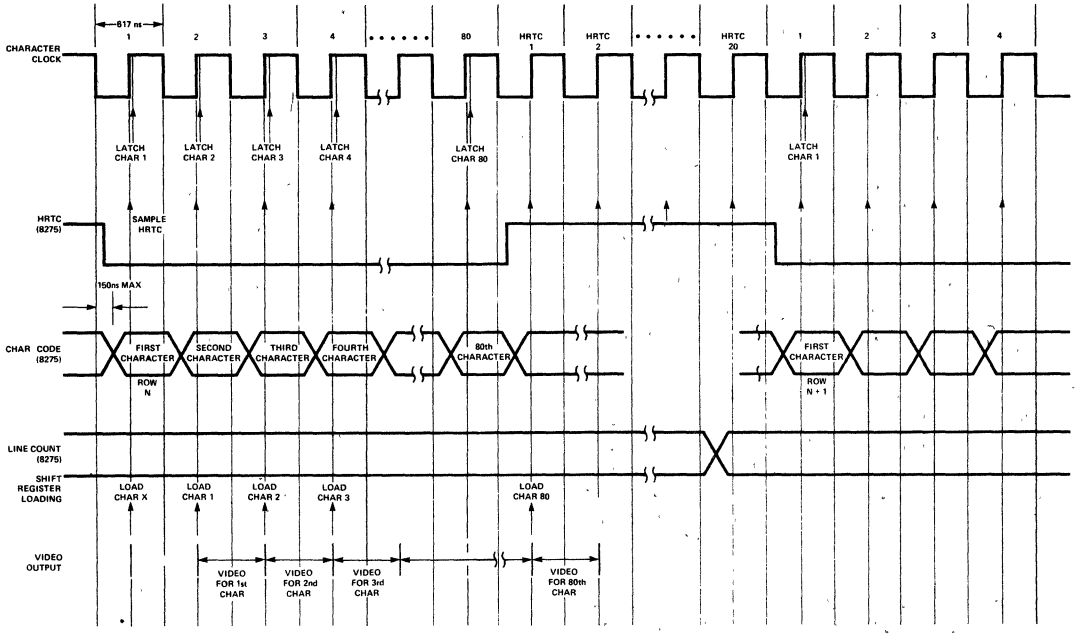


Figure 5-6. CRT System Timing

617.3 nanoseconds needed for each character, we find

617.3 nanoseconds * 20 retrace times = 12.345 microseconds

This value falls short of the 25 to 30 microseconds required by the horizontal drive of the Ball monitor. To correct for this, an 8253 was programmed in the one-shot mode and was used to extend the horizontal drive pulsewidth.

Now that the 617.3 nanosecond character clock period is known, the dot clock is easy to calculate. Since each character is formed by placing 7 dots along the horizontal.

DOT CLOCK PERIOD = 617.3 ns (CHARACTER CLK PERIOD)/ 7 DOTS

DOT CLOCK PERIOD = 88.183 nanoseconds

DOT CLOCK FREQUENCY = 1/PERIOD = 11.34 MHz

Figures 5.5 and 5.6 illustrate the basic dot timing and the CRT system timing, respectively.

6. SYSTEM SOFTWARE

6.1 SOFTWARE OVERVIEW

As mentioned earlier the software is structured on a "foreground-background" basis. Two interrupt-driven routines, FRAME and POPDAT (Fig. 6.1) request service every 16.67 milliseconds and 617 microseconds respectively, frame is used to check the baud rate switches, update the system pointers and decode and assemble the keyboard characters. POPDAT is used to move data from the memory into the 8275's row buffer rapidly.

The foreground routine first examines the line-local switch to see whether to accept data from the USART or the keyboard. If the terminal is in the local mode, action will be taken on any data that is entered through the keyboard and the USART will be ignored on both output and input. If the terminal is in the line mode data entered through the keyboard will be transmitted by the USART and action will be taken on any data read out of the USART.

When data has been entered in the terminal the software first determines if the character received was an escape, line feed, form feed, carriage return, back space, or simply a printable character. If an escape was received the terminal assumes the next received character will be a recognizable escape sequence character. If it isn't no operation is performed.

After the character is decoded, the processor jumps to the routine to perform the required task. Figure 6.2 is a flow chart of the basic software operations; the program is listed in Appendix 6.8.

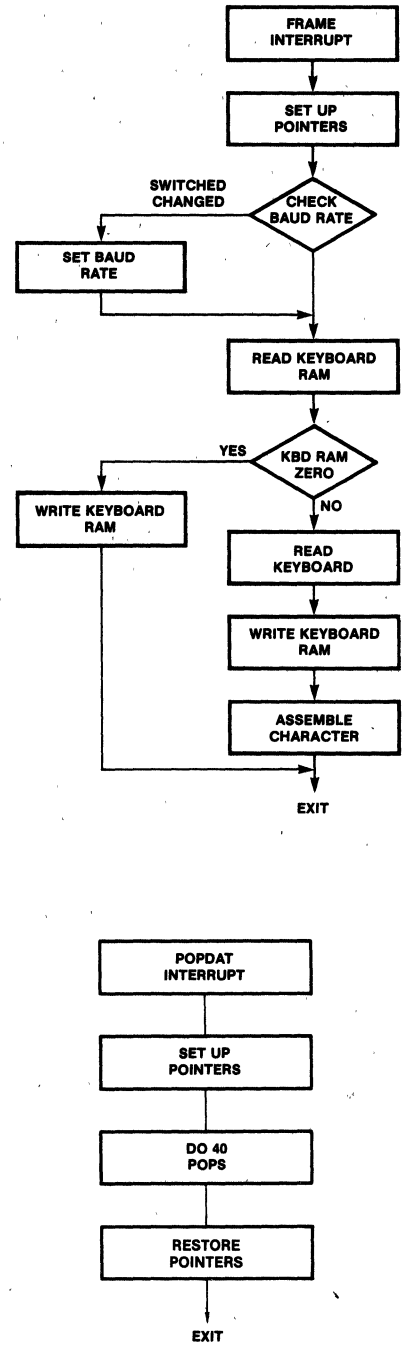


Figure 6-1. Frame and Popdat Interrupt Routines

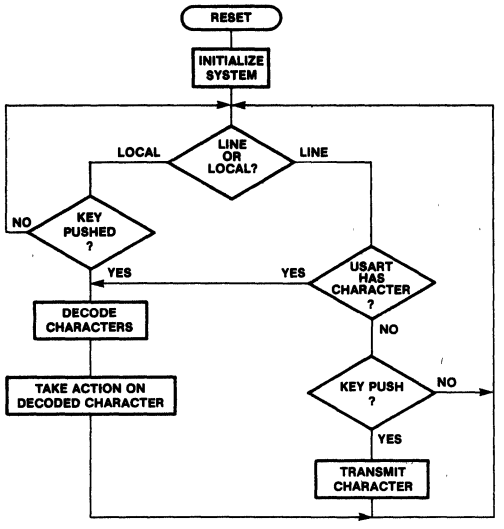


Figure 6-2. Basic Terminal Software

	1st Column	2nd Column ..... 80th Column
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H..... 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H.....0E8FH
ROW 22	0E90H	0E91H.....0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H.....0F7FH
ROW 25	0F80H	0F81H.....0FCFH

Figure 6-3. Screen Display After Initialization

6.2 SYSTEM MEMORY ORGANIZATION

The display memory organization is shown in Figure 6.3. The display begins at location 0800H in memory and ends at location 0FCFH. The 48 bytes of RAM from location 0FD0H to 0FFFH are used as system stack and temporary system storage. 2K bytes of PROM located at 0000H through 07FFH contain the systems program.

6.3 MEMORY POINTERS AND SCROLLING

To calculate the location of a character on the screen, three variables must be defined. Two of these variables are the X and Y position of the cursor (CURSX, CURSY). In addition, the memory address defining the top line of the display must be known, since scrolling on the 8275 is accomplished simply by changing the pointer that loads the 8275's row buffers from memory. So, if it is desired to scroll the display up or down all that must be changed is one 16-bit memory pointer. This pointer is entered into the system by the variable TOPAD (TOP Address) and always defines the top line of the display. Figure 6.4 details screen operation during scrolling.

Subroutines CALCU (Calculate) and ADX (ADd X axis) use these three variables to calculate an absolute memory address. The subroutine CALCU is used whenever a location in the screen memory must be altered.

6.4 SOFTWARE TIMING

One important question that must be asked about the terminal software is, "How fast does it run". This is important because if the terminal is running at 9600 baud, it must be able to handle each received character in 1.04 milliseconds. Figure 6.5 is a flowchart of the subroutine execution times. It should be pointed out that all of the times listed are "worst case" execution times. This means that all routines assume they must do the maximum amount of data manipulation. For instance, the PUT routine assumes that the character is being placed in the last column and that a line feed must follow the placing of the character on the screen.

How fast do the routines need to execute in order to assure operation at 9600 baud? Since POPDAT interrupts occur every 617 microseconds, it is possible to receive two complete interrupt requests in every character time (1042 microseconds) at 9600

ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH

After Initialization

ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH

After 1 Scroll

ROW 3	08A0H	08A1H 08EFH
ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH

After 2 Scrolls

ROW 4	08F0H	08F1H 093FH
ROW 5	0940H	0941H 098FH
ROW 6	0990H	0991H 090FH
ROW 7	09E0H	09E1H 0A2FH
ROW 8	0A30H	0A31H 0A7FH
ROW 9	0A80H	0A81H 0ACFH
ROW 10	0AD0H	0AD1H 0B1FH
ROW 11	0B20H	0B21H 0B6FH
ROW 12	0B70H	0B71H 0BBFH
ROW 13	0BC0H	0BC1H 0C0FH
ROW 14	0C10H	0C11H 0C5FH
ROW 15	0C60H	0C61H 0CAFH
ROW 16	0CB0H	0CB1H 0CFFH
ROW 17	0D00H	0D01H 0D4FH
ROW 18	0D50H	0D51H 0D9FH
ROW 19	0DA0H	0DA1H 0DEFH
ROW 20	0DF0H	0DF1H 0E3FH
ROW 21	0E40H	0E41H 0E8FH
ROW 22	0E90H	0E91H 0EDFH
ROW 23	0EE0H	0EE1H 0F2FH
ROW 24	0F30H	0F31H 0F7FH
ROW 25	0F80H	0F81H 0FCFH
ROW 1	0800H	0801H 084FH
ROW 2	0850H	0851H 089FH
ROW 3	08A0H	08A1H 08EFH

After 3 Scrolls

Figure 6-4. Screen Memory During Scrolling

baud. Each POPDAT interrupt executes in 211 microseconds maximum. This means that each routine must execute in:

1042 - 2 * 211 = 620 microseconds

By adding up the times for any loop, it is clear that all routines meet this speed requirement, with the exception of ESC J. This means that if the terminal is operating at 9600 baud, at least one character time must be inserted after an ESC J sequence.

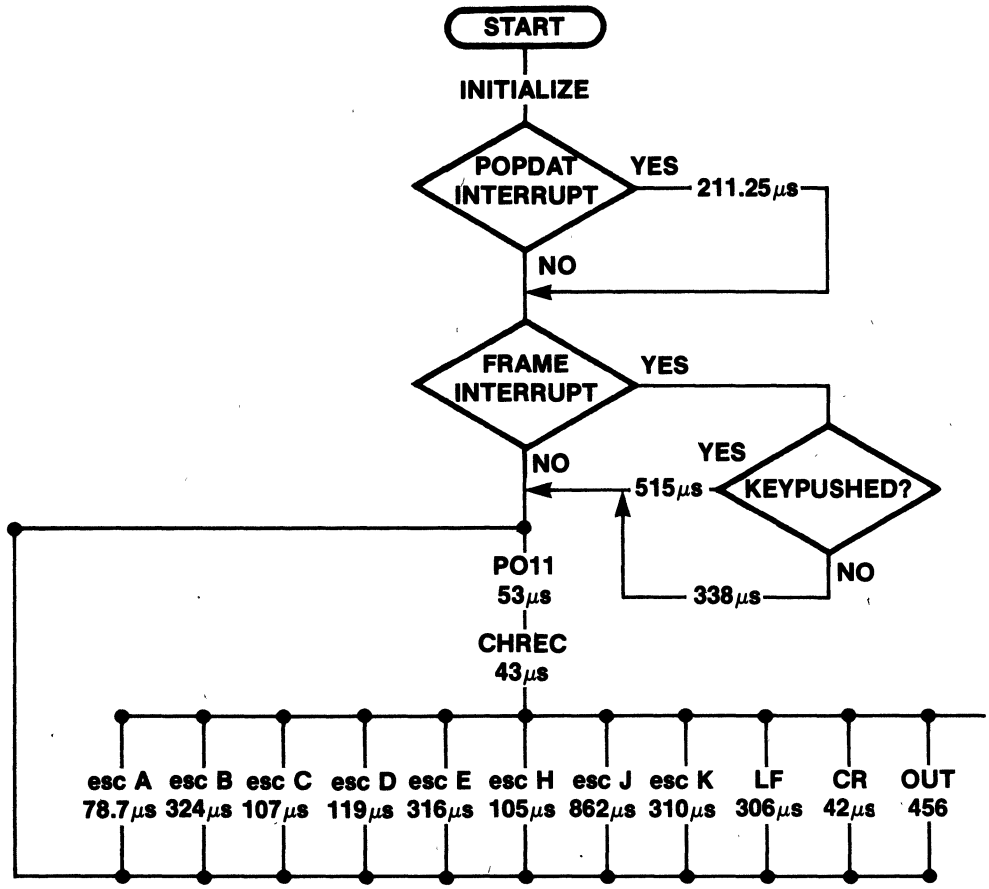
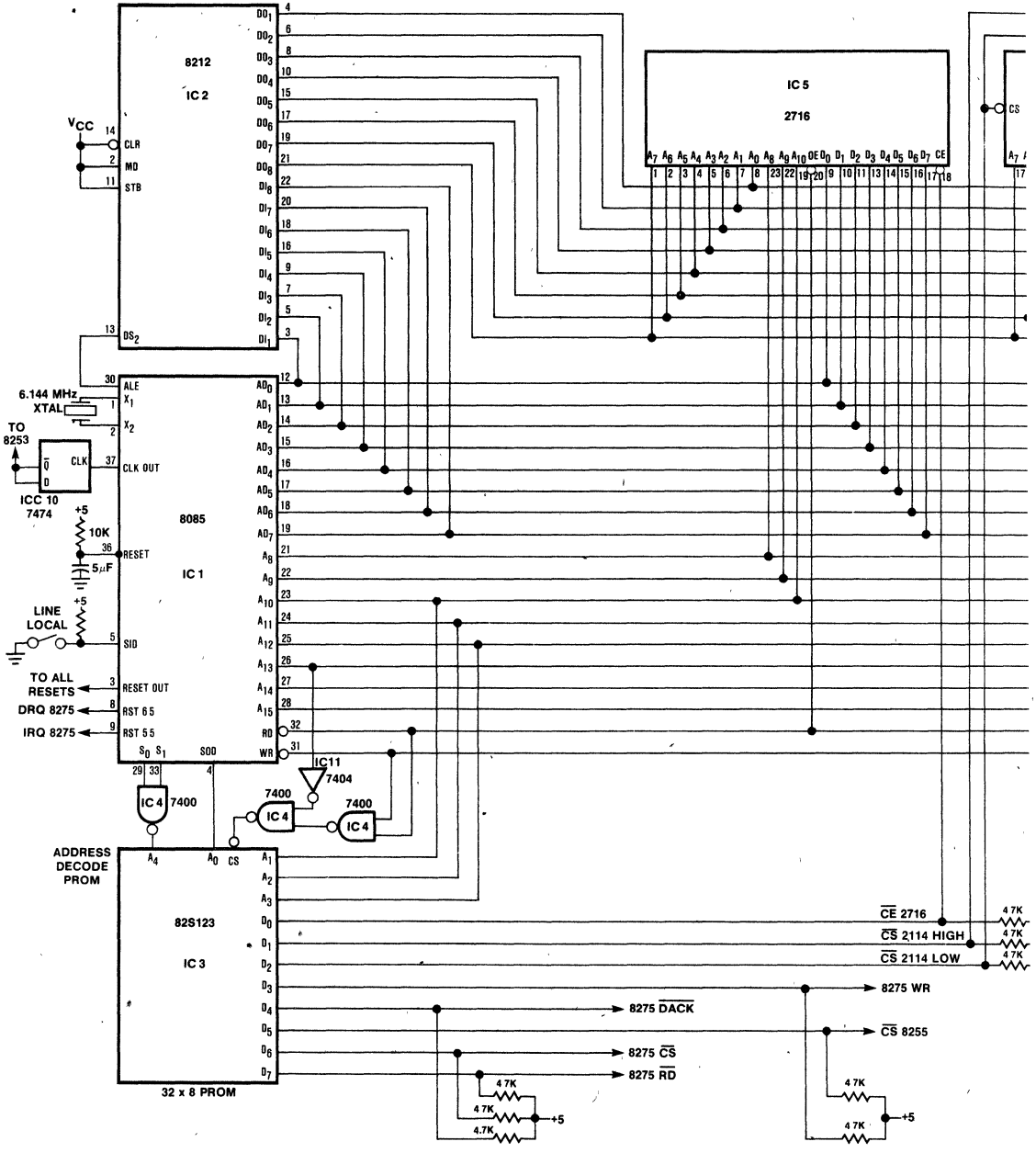
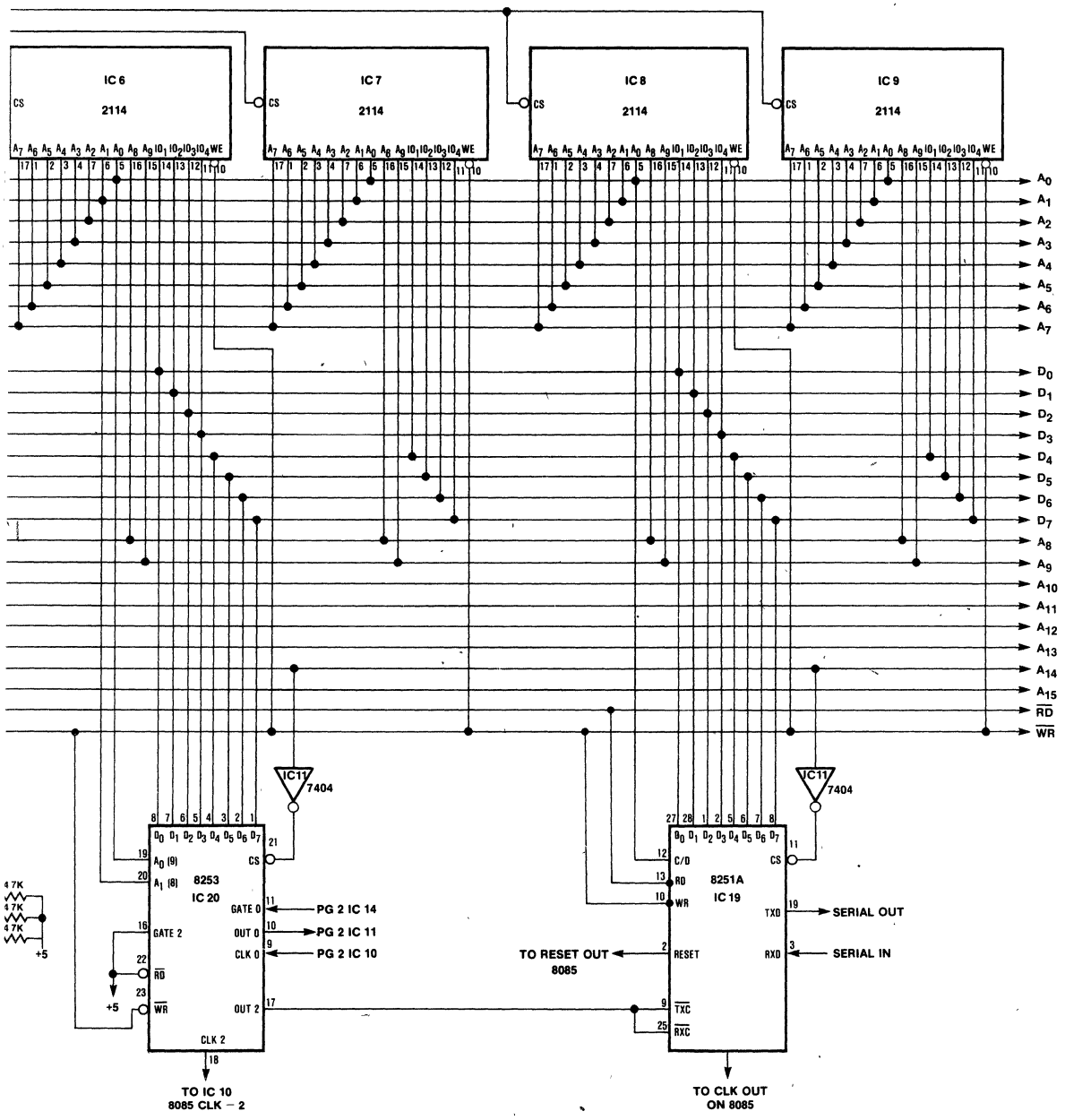


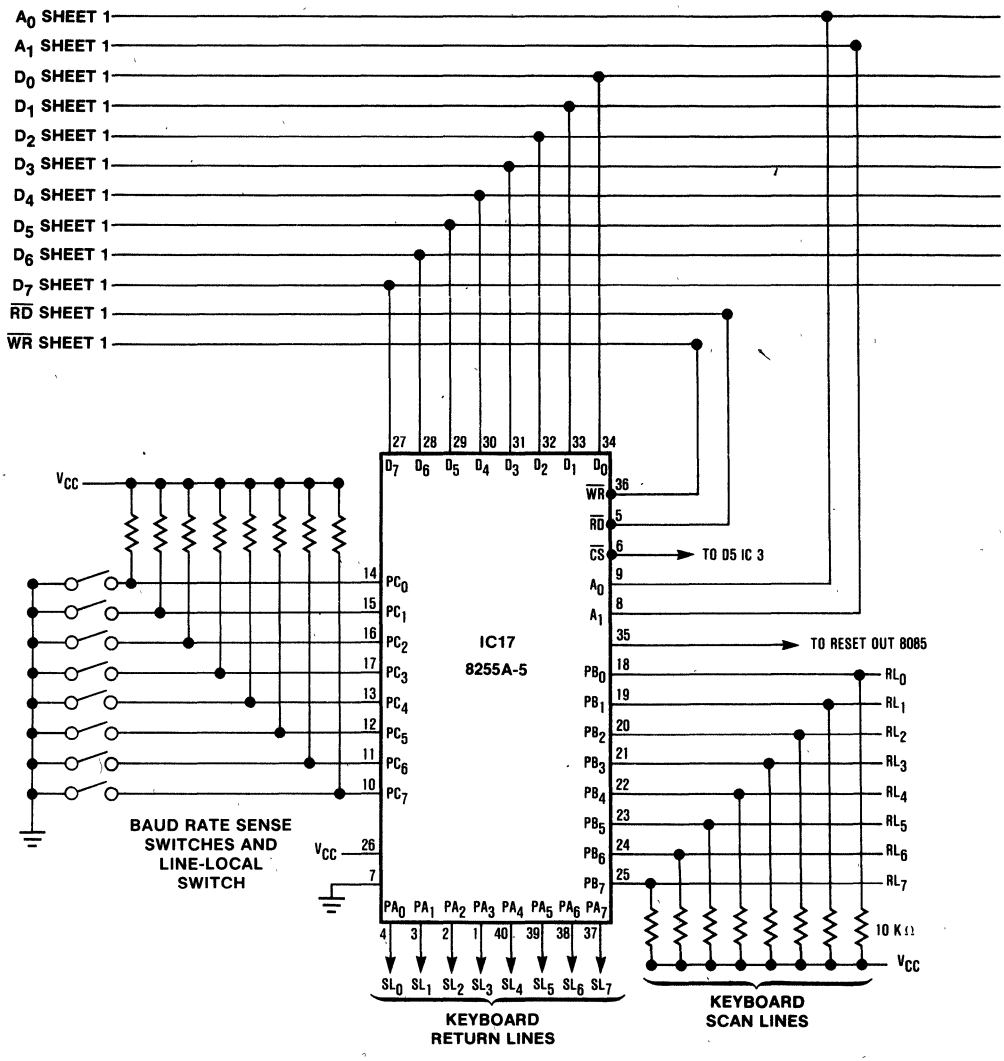
Figure 6-5. Timing Flowchart



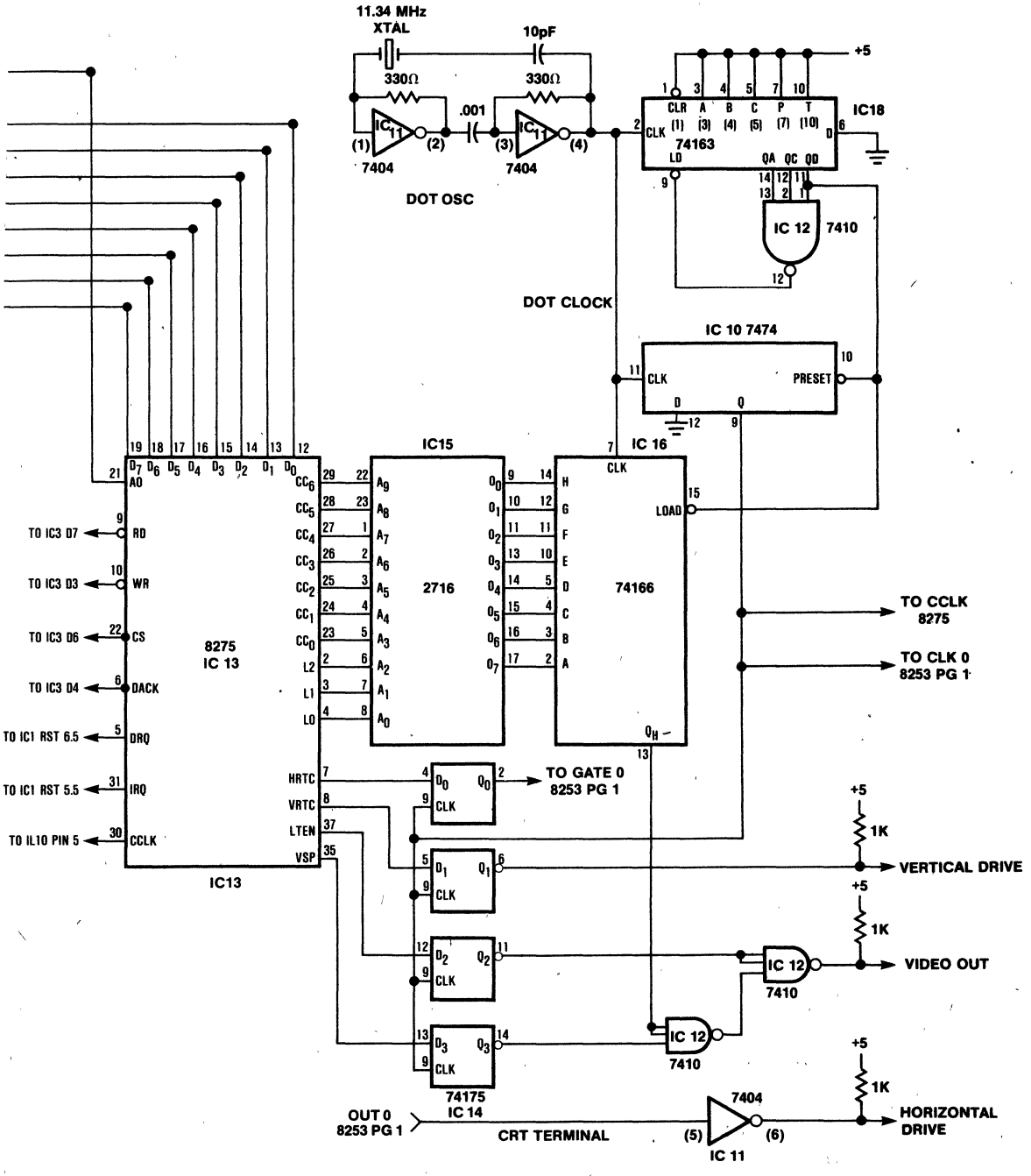
Appendix 7.1
CRT TERMINAL SCHEMATICS

APPLICATIONS





Appendix 7.1
CRT TERMINAL SCHEMATICS


11.34 MHz
XTAL
10pF
330Ω
330Ω
.001
IC 11
(1)
(2)
(3)
(4)
7404
7404
DOT OSC
+5
IC18
CLR A B C P T
(1) (3) (4) (5) (7) (10)
CLK
74163
LD
QA QC QD
IC 12 7410
DOT CLOCK
IC 10 7474
CLK
PRESET
D
Q
IC15
IC 16
2716
74166
CLK
LOAD
8275
IC 13
TO IC3 D7
TO IC3 D3
TO IC3 D6
TO IC3 D4
TO IC1 RST 6.5
TO IC1 RST 5.5
TO IL10 PIN 5
RD
WR
CS
DACK
DRQ
IRQ
CCLK
HRTC
VRTC
LTEN
VSP
IC13
TO CCLK
8275
TO CLK 0
8253 PG 1
TO GATE 0
8253 PG 1
1K
VERTICAL DRIVE
VIDEO OUT
74175
IC 14
OUT 0
8253 PG 1
CRT TERMINAL
HORIZONTAL
DRIVE

Appendix 7.2
KEYBOARD INTERFACE

The keyboard used in this design was a simple unencoded ASCII keyboard. In order to keep the cost to a minimum a simple scan matrix technique was implemented by using two ports of an 8255 parallel I/O device.

When the system is initialized the contents of the eight keyboard RAM locations are set to zero. Once every frame, which is 16.67 milliseconds the contents of the keyboard ram is read and then rewritten with the contents of the current switch matrix. If a non-zero value of one of the keyboard RAM locations is found to be the same as the corresponding current switch matrix, a valid key push is registered and action is taken. By operating the keyboard scan in this manner an automatic debounce time of 16.67 milliseconds is provided.

Figure 7.2A shows the actual physical layout of the keyboard and Figure 7.2B shows how the individual keys were encoded. On Figure 7.2B the scan lines are the numbers on the bottom of each key position and the return lines are the numbers at the top of each key position. The shift, control, and caps lock key were brought in through separate lines of port C of the 8255. Figure 7.3 shows the basic keyboard matrix.

In order to guarantee that two scan lines could not be shorted together if two or more keys are pushed simultaneously, isolation diodes could be added as shown in Figure 7.4.

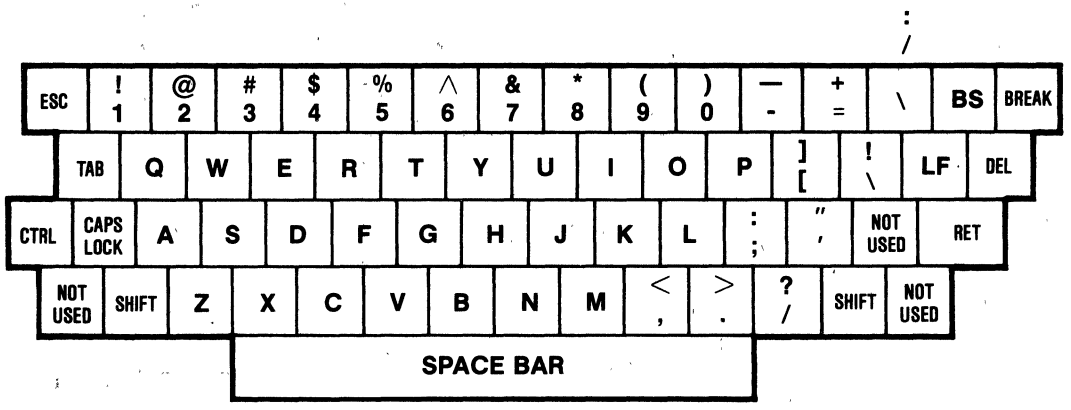
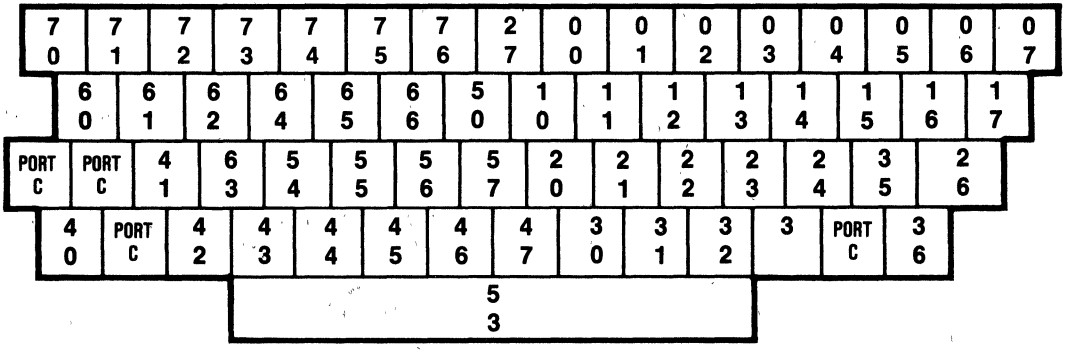


Figure 7-2A. Keyboard Layout



TOP NUMBER = RETURN LINE
BOTTOM NUMBER = SCAN LINE.

Figure 7-2B. Keyboard Encoding

Appendix 7.3
ESCAPE/CONTROL/DISPLAY CHARACTER SUMMARY

	CONTROL CHARACTERS		DISPLAYABLE CHARACTER								ESCAPE SEQUENCE			
BIT	0_{0_0}	0_{0_1}	0_{1_0}	0_{1_1}	1_{0_0}	1_{0_1}	1_{1_0}	1_{1_1}	0_{1_0}	0_{1_1}	1_{0_0}	1_{0_1}	1_{1_0}	1_{1_1}
0000	NUL @	DLE P	SP	ϕ	@	P		P						
0001	SOH A	DC1 Q	!	1	A	Q	A	Q			↑ A			
0010	STX B	DC2 R	"	2	B	R	B	R			↓ B			
0011	ETX C	DC3 S	#	3	C	S	C	S			→ C			
0100	EOT D	DC4 T	$	4	D	T	D	T			← D			
0101	ENQ E	NAK U	%	5	E	U	E	U			CLR E			
0110	ACK F	SYN V	&	6	F	V	F	V						
0111	BEL G	ETB W	'	7	G	W	G	W						
1000	BS H	CAN X	(	8	H	X	H	X			HOME H			
1001	HT I	EM Y	)	9	I	Y	I	Y						
1010	LF J	SUB Z	*	·	J	Z	J	Z			EOS I			
1011	VT K	ESC [	+	;	K	[	K				EL J			
1100	FF L	FS /	,	<	L	\	L							
1101	CR M	GS	-	=	M	]	M							
1110	SO N	RS ^	.	>	N	^	N							
1111	S1 O	US -	/	?	O	-	O							

NOTE Shaded blocks = functions terminal will react to Others can be generated but are ignored up on receipt

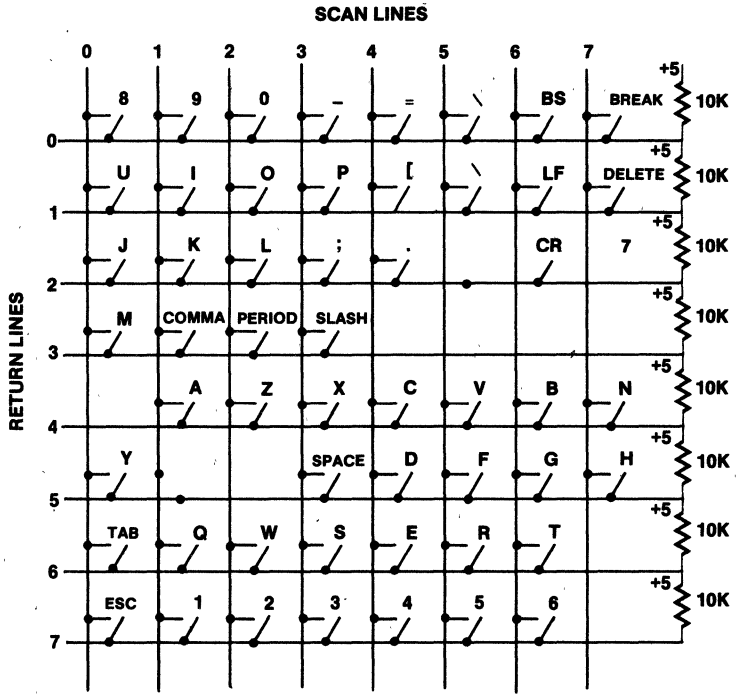


Figure 7-3. Keyboard Matrix

Appendix 7.4
PROM DECODING

As stated earlier, all of the logic necessary to convert the 8275 into a non-DMA type of device was performed by a single small bipolar prom. Besides turning certain processor READS into DACKS and WRITES for the 8275, this 32 by 8 prom decoded addresses for the system ram, rom, as well as for the 8255 parallel I/O port.

Any bipolar prom that has a by eight configuration could function in this application. This particular device was chosen simply because it is the only "by eight" prom available in a 16 pin package. The connection of the prom is shown in detail in Figure 7.5 and its truth table is shown in Figure 7.6. Note that when a fetch cycle (M1) is not being performed, the state of the SOD line is the only thing that determines if memory reads will be written into the 8275's row buffers. This is done by pulling both DACK and WRITE low on the 8275.

Also note that all of the outputs of the bipolar prom MUST BE PULLED HIGH by a resistor. This prevents any unwanted assertions when the prom is disabled.

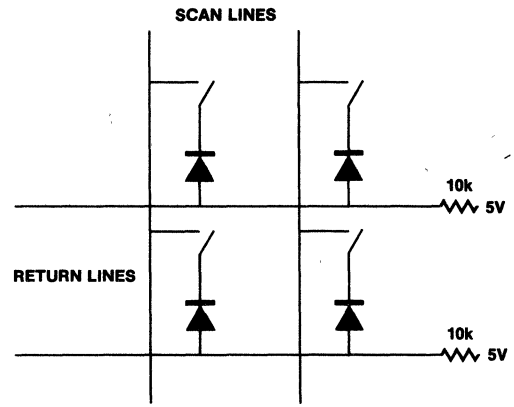


Figure 7-4. Isolating Scan Lines With Diodes

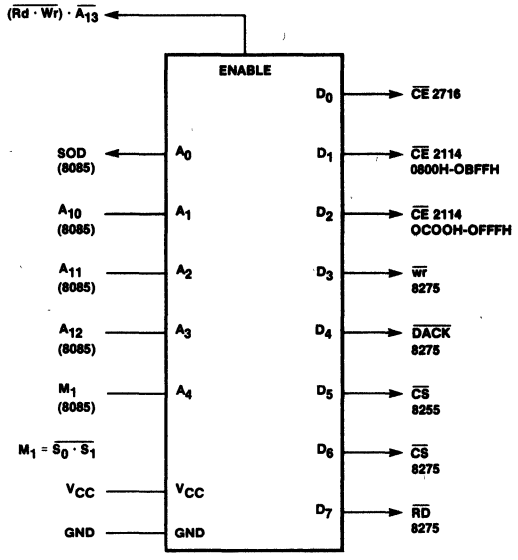


Figure 7-5. Bipolar Prom (825123) Connection

M1	A12	A11	A10	Sod	8275 RD	8275 CS	8255 CS	8275 DACK	8275 WR	2114 H	2114 L	2716
A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0*	0	0	0	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1	1	0
0	0	0	1	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	1	1	0	1
0	0	1	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	1	0	1	1
0	1	0	0	0	1	0	1	1	1	0	1	1
0	1	0	0	1	1	0	1	1	1	0	1	1
0	1	0	1	0	0	0	1	1	1	1	1	1
0	1	0	1	1	0	0	1	1	1	1	1	1
0	1	1	0	0	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	1
0	1	1	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	0	0	1	1	0
1	0	0	1	0	1	1	1	1	1	1	1	0
1	0	0	1	1	1	1	1	0	0	1	1	0
1	0	1	0	0	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	0	0	1	0	1
1	0	1	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	0	0	0	1	1
1	1	0	0	0	1	0	1	1	0	1	1	1
1	1	0	0	1	1	0	1	1	0	1	1	1
1	1	0	1	0	0	0	1	1	1	1	1	1
1	1	0	1	1	0	0	1	1	1	1	1	1
1	1	1	0	0	1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	1	0	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	1	1

Figure 7-6. Truth Table Bipolar Prom

Appendix 7.5
CHARACTER GENERATOR

As previously mentioned, the character generator used in this terminal is a 2716 or 2758 EPROM. A 1K by 8 device is sufficient since a 128 character 5 by 7 dot matrix only requires 8K of memory. Any "standard" or custom character generator could have been used.

The three low-order line count outputs (LC0-LC2) from the 8275 are connected to the three low-order address lines of the character generator and the seven character generator outputs (CC0-CC6) are connected to A3-A9 of the character generator. The output from the character generator is loaded into a shift register and the serial output from the shift register is the video output of the terminal.

Now, let's assume that the letter "E" is to be displayed. The ASCII code for "E" is 45H. So, 45H is presented to address lines A2-A9 of the character generator. The scan lines will now count each line from zero to seven to "form" the character as shown in Fig. 7.7. This same procedure is used to form all 128 possible characters.

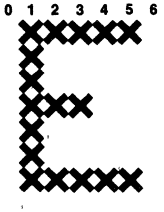
It should be obvious that "custom" character fonts could be made just by changing the bit patterns in the character generator PROM. For reference, Appendix 7.6 contains a HEX dump of the character generator used in this terminal.

45H = 01000101
Address to Prom = 01000101 SL2 SL1 SL0
= 228H - 22FH

Depending on state of Scan lines.

Character generator output

Rom Address	Rom Hex Output	Bit Output*
228H	3E	0 1 2 3 4 5 6 7
229H	02	
22AH	02	
22BH	0E	
22CH	02	
22DH	02	
22EH	3E	
22FH	00	



Bits 0, 6 and 7 are not used.

* note bit output is backward from convention.

Figure 7-7. Character Generation

Appendix 7.6
HEX DUMP OF CHARACTER GENERATOR

```
:100000000100000000000000000000000000000000EF
:10001000000000000000000000000000000000000000E0
:10002000000000000000000000000000000000000000D0
:10003000000000000000000000000000000000000000C0
:10004000000000000000000000000000000000000000B0
:1000500000000000000000000000000000000000000A0
:10006000000000000000000000000000000000000090
:10007000000000000000000000000000000000000080
:10008000000000000000000000000000000000000070
:1000900000000000000082A1C081C2A0830000000BC
:1002A000000000000000000000000000000000000050
:1002B000000000000000000000000000000000000040
:1002C000000000000000000000000000000000000030
:1002D000000000000000000000000000000000000020
:1002E000000000000000000000000000000000000010
:1002F000000000000000000000000000000000000000
:10010000000000000000000000808080808080800BF
:10011000141414000000000143E143E141400C3
:10012000083C0A1C281E08000626100804323000 6D
:10013000040A0A042A122C0708080800000000023
:10014000080402020204080008102020201008000 1
:10015000082A1C081C2A0800080883E0808000 9D
:10016000000000000000808040000003C0000000 3F
:10017000000000000000180000201008040200029
:10018001C22322A26221C00080C0808080 81C0021
:10019001C22201C02023E003E20101820221C00BF
:1001A000101814123E1010003E021E2020221C00C7
:1001B0003804021E22221C003E20100804040400 1
:1001C0001C22221C22221C001C22223C20100E0079
:1001D000000080000080700000000807000080804F3
:1001E0021008040204081000000 03E003E00000059
:1001F00004081020100804001C22201008080000821
:10020001C222A3A1A023C000814222 23E22220012
:10021001E24241C24241E001C22020202221C0074
:10022001E2424242241E003E02020E02023E004C
:10023003E02020E020202003C02023A22223C006E
:10024000222223E222222001C0808080 8081C0044
:10025000702020202020221C0022120A060A122200EE
:10026000202020202023E0022362A2A2222220032
:10027000222262A3222222200 1C22222222221C0092
:10028001E22221E020202001C2222222A122C00FE
:10029001E22221E0A1222003C02021C20201E00E6
:1002A003E0808080808080222222222222 1C00F8
:1002B000222222222214080022222222 2A3522006E
:1002C000222214081422220022222221408080800E4
:1002D0003E2010080402 3E001C0404040 41C0018
:1002E000000204081020000038202020202038 00C0
:1002F000081C2A0808080800000000000000007E12
:10030000890110000000000000 3C203C223C004E
:10031000202 1A2622221E000000380484043800 3B
:10032000202 02C322223C000000 3824BC84B8005B
:10033000382404 0E040404000000BC22223C203CAB
:10034000202 1A2622222200080080808 0890004B
:10035000200020 20A424180202221 20A162200C3
:10036000880808080808900000003 62A2A2222007F
:10037000000 1A2622222200000018242424180 03B
:10038000000 1E22221E020200001C22223C20200D
:10039000000 1A260202020000003804182 01C0087
:1003A00008081C0808089000000022222232 4C0095
:1003B000000022222214080000002222 2A3E1400FB
:1003C000000022140814220000072222 23C20388F
:1003D00000003E1008043E00188889038889190 02F
:1003E000080808080808080C9091219091 0D0051
:1003F00000008C2BB001000000000000000000095
```

Appendix 7.7
COMPOSITE VIDEO

In this design, it was assumed that the monitor required a separate horizontal drive, vertical drive, and video input. However, many monitors require a composite video signal. The schematic shown in Figure 7.8 illustrates how to generate a composite video signal from the output of the 8275.

The dual one-shots are used to provide a small delay and the proper horizontal and vertical pulse to the composite video monitor. The delay introduced in the vertical and horizontal timing is used to "center" the display. VR1 and VR2 control the amount of delay. IC3 is used to mix the vertical and horizontal retrace and Q1 along with the R1, R2, and R3 mix the video and the retrace signal and provide the proper DC levels.

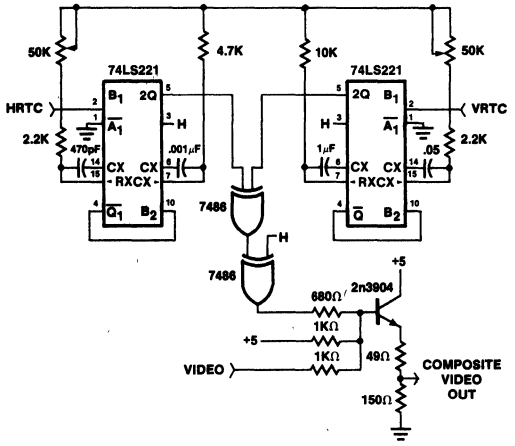


Figure 7-8. Composite Video

Appendix 7.8
SOFTWARE LISTINGS

```
ISIS-II 8080/8085 MACRO ASSEMBLER, X108

LOC  OBJ          SEQ          SOURCE STATEMENT

                    1 $MOD85   MACROFILE
                    2          ;NO DMA 8275 SOFTWARE ALL I/O IS MEMORY MAPPED
                    3          ;SYSTEM ROM 0000H TO 07FFH
                    4          ;SYSTEM RAM 0800H TO 0FFFH
                    5          ;8275 WRITE 1000H TO 13FFH
                    6          ;8275 READ 1400H TO 17FFH
                    7          ;8255 READ/WRITE 1800H TO 1FFF
                    8          ;8253 ENABLED BY A14
                    9          ;8251 ENABLED BY A15
1800                10 PORTA   EQU     1800H          ;8255 PORT A ADDRESS
1801                11 PORTB   EQU     1801H          ;8255 PORT B ADDRESS
1802                12 PORTC   EQU     1802H          ;8255 PORT C ADDRESS
1803                13 CNWD55  EQU     1803H          ;8255 CONTROL PORT ADDRESS
A001                14 USTF    EQU     0A001H         ;8251 FLAGS
A000                15 USTD    EQU     0A000H         ;8251 DATA
6000                16 CNT0    EQU     6000H          ;8253 COUNTER 0
6001                17 CNT1    EQU     6001H          ;8253 COUNTER 1
6002                18 CNT2    EQU     6002H          ;8253 COUNTER 2
6003                19 CNTM    EQU     6003H          ;8253 MODE WORD
1001                20 CRTS    EQU     1001H          ;8275 CONTROL ADDRESS
1000                21 CRTM    EQU     1000H          ;8275 MODE ADDRESS
1401                22 INT75   EQU     1401H          ;8275 INTERRUPT CLEAR
0800                23 TPDIS   EQU     0800H          ;TOP OF DISPLAY RAM
0F80                24 BTDIS   EQU     0F80H          ;BOTTOM OF DISPLAY RAM
0FD0                25 LAST    EQU     0FD0H          ;FIRST BYTE AFTER DISPLAY
0018                26 CURBOT  EQU     18H            ;BOTTOM Y CURSOR
0050                27 LNGTH   EQU     0050H          ;LENGTH OF ONE LINE
0FE0                28 STPTR   EQU     0FE0H          ;LOCATION OF STACK POINTER
                    29         ;
                    30         ;START PROGRAM
                    31         ;ALL VARIABLES ARE INITIALIZED BEFORE ANYTHING ELSE
                    32         ;
0000 F3             33         DI                     ;DISABLE INTERRUPTS
0001 31E00F         34         LXI     SP,STPTR       ;LOAD STACK POINTER
0004 210008         35         LXI     H,TPDIS        ;LOAD H&L WITH TOP OF DISPLAY
0007 22E30F         36         SHLD    TOPAD          ;SET TOP = TOP OF DISPLAY
000A 22E80F         37         SHLD    CURAD          ;STORE THE CURRENT ADDRESS
000D 3E00           38         MVI     A,00H          ;ZERO A
000F 32E10F         39         STA     CURSY          ;ZERO CURSOR Y POINTER
0012 32E20F         40         STA     CURSX          ;ZERO CURSOR X POINTER
0015 32EB0F         41         STA     KBCHR          ;ZERO KBD CHARACTER
0018 32E70F         42         STA     USCHR          ;ZERO USART CHAR BUFFER
001B 32EA0F         43         STA     KEYDWN         ;ZERO KEY DOWN
```

```
001E 32ED0F          44          STA     KEYOK           ;ZERO KEYOK
0021 32EE0F          45          STA     ESCP            ;ZERO ESCAPE
0024 C39800          46          JMP     LPKBD           ;JUMP AND SET EVERYTHING UP
                     47          ;
                     48          ;THIS JUMP VECTOR IS LOCATED AT THE RST 5.5 LOCATION
                     49          ;OF THE 8085. IT IS USED TO READ THE 8275 STATUS AND
                     50          ;READ THE KEYBOARD. THIS ROUTINE IS EXECUTED ONCE EVERY
                     51          ;16.667 MILLISECONDS.
                     52          ;
002C                 53          ORG     002CH
002C C36701          54          JMP     FRAME
                     55          ;
                     56          ;THIS ROUTINE IS LOCATED AT THE RST 6.5 LOCATION OF THE
                     57          ;8085 AND IS USED TO LOAD THE DATA TO BE DISPLAYED INTO
                     58          ;THE 8275. THIS ROUTINE IS EXECUTED ONCE EVERY 617 MICROSECONDS.
                     59          ;
0034                 60          ORG     34H
0034 F5              61 POPDAT:  PUSH    PSW             ;SAVE A AND FLAGS
0035 E5              62          PUSH    H               ;SAVE H AND L
0036 D5              63          PUSH    D               ;SAVE D AND E
0037 210000          64          LXI     H,0000H         ;ZERO H AND L
003A 39              65          DAD     SP              ;PUT STACK POINTER IN H AND L
003B EB              66          XCHG                    ;PUT STACK IN D AND E
003C 2AE80F          67          LHLD    CURAD           ;GET POINTER
003F F9              68          SPHL                    ;PUT CURRENT LINE INTO SP
0040 3EC0            69          MVI     A,0C0H          ;SET MASK FOR SIM
0042 30              70          SIM
                     71          REPT (LNGTH/2)
                     72          POP     H
                     73          ENDM
0043 E1              74+         POP     H
0044 E1              75+         POP     H
0045 E1              76+         POP     H
0046 E1              77+         POP     H
0047 E1              78+         POP     H
0048 E1              79+         POP     H
0049 E1              80+         POP     H
004A E1              81+         POP     H
004B E1              82+         POP     H
004C E1              83+         POP     H
004D E1              84+         POP     H
004E E1              85+         POP     H
004F E1              86+         POP     H
0050 E1              87+         POP     H
0051 E1              88+         POP     H
0052 E1              89+         POP     H
0053 E1              90+         POP     H
0054 E1              91+         POP     H
0055 E1              92+         POP     H
0056 E1              93+         POP     H
0057 E1              94+         POP     H
0058 E1              95+         POP     H
0059 E1              96+         POP     H
005A E1              97+         POP     H
005B E1              98+         POP     H
005C E1              99+         POP     H
005D E1             100+         POP     H
005E E1             101+         POP     H
005F E1             102+         POP     H
0060 E1             103+         POP     H
0061 E1             104+         POP     H
0062 E1             105+         POP     H
0063 E1             106+         POP     H
0064 E1             107+         POP     H
0065 E1             108+         POP     H
0066 E1             109+         POP     H
0067 E1             110+         POP     H
0068 E1             111+         POP     H
0069 E1             112+         POP     H
006A E1             113+         POP     H
006B 0F             114          RRC                     ;SET UP A
006C 30             115          SIM                     ;GO BACK TO NORMAL MODE
006D 210000         116          LXI     H,0000H         ;ZERO HL
0070 39             117          DAD     SP              ;ADD STACK
0071 EB             118          XCHG                    ;PUT STACK IN H AND L
0072 F9             119          SPHL                    ;RESTORE STACK
0073 21D00F         120          LXI     H,LAST          ;PUT BOTTOM DISPLAY IN H AND L
0076 EB             121          XCHG                    ;SWAP REGISTERS
0077 7A             122          MOV     A,D             ;PUT HIGH ORDER IN A
0078 BC             123          CMP     H               ;SEE IF SAME AS H
0079 C28400         124          JNZ     KPTK            ;IF NOT LEAVE
007C 7B             125          MOV     A,E             ;PUT LOW ORDER IN A
007D BD             126          CMP     L               ;SEE IF SAME AS L
007E C28400         127          JNZ     KPTK            ;IF NOT LEAVE
0081 210008         128          LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF SCREEN MEMORY
0084 22E80F         129 KPTK:    SHLD    CURAD           ;PUT BACK CURRENT ADDRESS
0087 3E18           130          MVI     A,18H           ;SET MASK
0089 30             131          SIM                     ;OUTPUT MASK
```

```
008A D1          132          POP     D               ;GET D AND E
008B E1          133          POP     H               ;GET H AND L
008C F1          134          POP     PSW             ;GET A AND FLAGS
008D FB          135          EI                      ;TURN ON INTERRUPTS
008E C9          136          RET                     ;GO BACK
                 137          ;
                 138          ;THIS IS THE EXIT ROUTINE FOR THE FRAME INTERRUPT
                 139          ;
008F 3E18        140 BYPASS:  MVI     A,18H           ;SET MASK
0091 30          141          SIM                     ;OUTPUT THE MASK
0092 C1          142          POP     B               ;GET B AND C
0093 D1          143          POP     D               ;GED D AND E
0094 E1          144          POP     H               ;GET H AND L
0095 F1          145          POP     PSW             ;GET A AND FLAGS
0096 FB          146          EI                      ;ENABLE INTERRUPTS
0097 C9          147          RET                     ;GO BACK
                 148          ;
                 149          ;THIS CLEARS THE AREA OF RAM THAT IS USED
                 150          ;FOR KEYBOARD DEBOUNCE.
                 151          ;
0098 32EF0F      152 LPKBD:   STA     SHCON           ;ZERO SHIFT CONTROL
009B 32F00F      153          STA     RETLIN          ;ZERO RETURN LINE
009E 32F10F      154          STA     SCNLIN          ;ZERO SCAN LINE
                 155          ;
                 156          ;THIS ROUTINE CLEARS THE ENTIRE SCREEN BY PUTTING
                 157          ;SPACE CODES (20H) IN EVERY LOCATION ON THE SCREEN.
                 158          ;
00A1 210008      159          LXI     H,TPDIS         ;PUT TOP OF SCREEN IN HL
00A4 01D00F      160          LXI     B,LAST          ;PUT BOTTOM IN BC
00A7 3620        161 LOOPF:   MVI     M,20H           ;PUT SPACE IN M
00A9 23          162          INX     H               ;INCREMENT POINTER
00AA 7C          163          MOV     A,H             ;GET H
00AB B8          164          CMP     B               ;SEE IF SAME AS B
00AC C2A700      165          JNZ     LOOPF           ;IF NOT LOOP AGAIN
00AF 7D          166          MOV     A,L             ;GET L
00B0 B9          167          CMP     C               ;SEE IF SAME AS C
00B1 C2A700      168          JNZ     LOOPF           ;IF NOT LOOP AGAIN
                 169          ;
                 170          ;8255 INITIALIZATION
                 171          ;
00B4 3E8B        172          MVI     A,8BH           ;MOVE 8255 CONTROL WORD INTO A
00B6 320318      173          STA     CNWD55          ;PUT CONTROL WORD INTO 8255
                 174          ;
                 175          ;8251 INITIALIZATION
                 176          ;
00B9 2101A0      177          LXI     H,USTF          ;GET 8251 FLAG ADDRESS
00BC 3680        178          MVI     M,80H           ;DUMMY STORE TO 8251
00BE 3600        179          MVI     M,00H           ;RESET 8251
00C0 3640        180          MVI     M,40H           ;RESET 8251
00C2 00          181          NOP                     ;WAIT
00C3 36EA        182          MVI     M,0EAH          ;LOAD 8251 MODE WORD
00C5 3605        183          MVI     M,05H           ;LOAD 8251 COMMAND WORD
                 184          ;
                 185          ;8253 INITIALIZATION
                 186          ;
00C7 3E32        187          MVI     A,32H           ;CONTROL WORD FOR 8253
00C9 320360      188          STA     CNTM            ;PUT CONTROL WORD INTO 8253
00CC 3E32        189          MVI     A,32H           ;LSB 8253
00CE 320060      190          STA     CNT0            ;PUT IT IN 8235
00D1 3E00        191          MVI     A,00H           ;MSB 8253
00D3 320060      192          STA     CNT0            ;PUT IT IN 8253
00D6 CDDC00      193          CALL    STBAUD          ;GO DO BAUD RATE
00D9 C3F900      194          JMP     IN75            ;GO DO 8275
                 195          ;
                 196          ;THIS ROUTINE READS THE BAUD RATE SWITCHES FROM PORT C
                 197          ;OF THE 8255 AND LOOKS UP THE NUMBERS NEEDED TO LOAD
                 198          ;THE 8253 TO PROVIDE THE PROPER BAUD RATE.
                 199          ;
00DC 3A0218      200 STBAUD:  LDA     PORTC           ;READ BAUD RATE SWITCHES
00DF E60F        201          ANI     0FH             ;STRIP OFF 4 MSB'S
00E1 32EC0F      202          STA     BAUD            ;SAVE IT
00E4 07          203          RLC                     ;MOVE BITS OVER ONE PLACE
00E5 21C505      204          LXI     H,BDLK          ;GET BAUD RATE LOOK UP TABLE
00E8 1600        205          MVI     D,00H           ;ZERO D
00EA 5F          206          MOV     E,A             ;PUT A IN E
00EB 19          207          DAD     D               ;GET OFFSET
00EC 110360      208          LXI     D,CNTM          ;POINT DE TO 8253
00EF 3EB6        209          MVI     A,0B6H          ;GET CONTROL WORD
00F1 12          210          STAX    D               ;STORE IN 8253
00F2 1B          211          DCX     D               ;POINT AT #2 COUNTER
00F3 7E          212          MOV     A,M             ;GET LSB BAUD RATE
00F4 12          213          STAX    D               ;PUT IT IN 8253
00F5 23          214          INX     H               ;POINT AT MSB BAUD RATE
00F6 7E          215          MOV     A,M             ;GET MSB BAUD RATE
00F7 12          216          STAX    D               ;PUT IT IN 8253
00F8 C9          217          RET                     ;GO BACK
                 218          ;
```

```
                        ;8275 INITIALIZATION
                        ;
00F9 210110     IN75:   LXI     H,CRTS
00FC 3600               MVI     M,00H           ;RESET AND STOP DISPLAY
00FE 2B                 DCX     H               ;HL=1000H
00FF 364F               MVI     M,4FH           ;SCREEN PARAMETER BYTE 1
0101 3658               MVI     M,58H           ;SCREEN PARAMETER BYTE 2
0103 3689               MVI     M,89H           ;SCREEN PARAMETER BYTE 3
0105 36DD               MVI     M,0DDH          ;SCREEN PARAMETER BYTE 4
0107 23                 INX     H               ;HL=1001H
0108 CDB803             CALL    LDCUR           ;LOAD THE CURSOR
010B 36E0               MVI     M,0E0H          ;PRESET COUNTERS
010D 3623               MVI     M,23H           ;START DISPLAY
                        ;
                        ;THIS ROUTINE READS BOTH THE KEYBOARD AND THE USART
                        ;AND TAKES PROPER ACTION DEPENDING ON HOW THE LINE-LOCAL
                        ;SWITCH IS SET
                        ;
010F 3E18       SETUP:  MVI     A,18H           ;SET MASK
0111 30                 SIM                     ;LOAD MASK
0112 FB                 EI                      ;ENABLE INTERRUPTS
                        ;
                        ;READ THE USART
                        ;
0113 20         RXRDY:  RIM                     ;GET LINE LOCAL
0114 E680               ANI     80H             ;IS IT ON OR OFF?
0116 C22101             JNZ     KEYINP          ;LEAVE IF IT IS ON
0119 3A01A0             LDA     USTF            ;READ 8251 FLAGS
011C E602               ANI     02H             ;LOOK AT RXRDY
011E C25C01             JNZ     OK7             ;IF HAVE CHARACTER GO TO WORK
0121 3AEA0F     KEYINP: LDA     KEYDWN          ;GET KEYBOARD CHARACTER
0124 E680               ANI     80H             ;IS IT THERE
0126 C23101             JNZ     KEYS            ;IF KEY IS PUSHED LEAVE
0129 3E00               MVI     A,00H           ;ZERO A
012B 32ED0F             STA     KEYOK           ;CLEAR KEYOK
012E C31301             JMP     RXRDY           ;LOOP AGAIN
0131 3AED0F     KEYS:   LDA     KEYOK           ;WAS KEY DOWN
0134 4F                 MOV     C,A             ;SAVE A IN C
0135 3AEB0F             LDA     KBCHR           ;GET KEYBOARD CHARACTER
0138 B9                 CMP     C               ;IS IT THE SAME AS KEYOK
0139 CA1301             JZ      RXRDY           ;IF SAME LOOP AGAIN
013C 32ED0F             STA     KEYOK           ;IF NOT SAVE IT
013F 32E70F             STA     USCHR           ;SAVE IT
0142 20                 RIM                     ;GET LINE LOCAL
0143 E680               ANI     80H             ;WHICH WAY
0145 CA4B01             JZ      TRANS           ;LEAVE IF LINE
0148 C34E02             JMP     CHREC           ;TIME TO DO SOME WORK
014B 3A01A0     TRANS:  LDA     USTF            ;GET USART FLAGS
014E E601               ANI     01H             ;READY TO TRANSMIT?
0150 CA4B01             JZ      TRANS           ;LOOP IF NOT READY
0153 3AE70F             LDA     USCHR           ;GET CHARACTER
0156 3200A0             STA     USTD            ;PUT IN USART
0159 C30F01             JMP     SETUP           ;LEAVE
015C 3A00A0     OK7:    LDA     USTD            ;READ USART
015F E67F               ANI     07FH            ;STRIP MSB
0161 32E70F             STA     USCHR           ;PUT IT IN MEMORY
0164 C34E02             JMP     CHREC           ;LEAVE
                        ;
                        ;THIS ROUTINE CHECKS THE BAUD RATE SWITCHES, RESETS THE
                        ;SCREEN POINTERS AND READS AND LOOKS UP THE KEYBOARD.
                        ;
0167 F5         FRAME:  PUSH    PSW             ;SAVE A AND FLAGS
0168 E5                 PUSH    H               ;SAVE H AND L
0169 D5                 PUSH    D               ;SAVE D AND E
016A C5                 PUSH    B               ;SAVE B AND C
016B 3A0114             LDA     INT75           ;READ 8275 TO CLEAR INTERRUPT
                        ;
                        ;SET UP THE POINTERS
                        ;
016E 2AE30F             LHLD    TOPAD           ;LOAD TOP IN H AND L
0171 22E80F             SHLD    CURAD           ;STORE TOP IN CURRENT ADDRESS
                        ;
                        ;SET UP BAUD RATE
                        ;
0174 3A0218             LDA     PORTC           ;READ BAUD RATE SWITCHES
0177 E60F               ANI     0FH             ;STRIP OFF 4 MSB'S
0179 47                 MOV     B,A             ;SAVE IN B
017A 3ABC0F             LDA     BAUD            ;GET BAUD RATE
017D B8                 CMP     B               ;SEE IF SAME AS B
017E C4DC00             CNZ     STBAUD          ;IF NOT SAME DO SOMETHING
                        ;
                        ;READ KEYBOARD
                        ;
0181 3AEA0F             LDA     KEYDWN          ;SEE IF A KEY IS DOWN
0184 E640               ANI     40H             ;SET THE FLAGS
0186 C2C201             JNZ     KYDOWN          ;IF KEY IS DOWN JUMP AROUND
0189 CD8F01             CALL    RDKB            ;GO READ THE KEYBOARD
018C C38F00             JMP     BYPASS          ;LEAVE
```

```
018F 21EF0F      307 RDKB:   LXI    H,SHCON      ;POINT HL AT KEYBOARD RAM
0192 3A0218      308         LDA    PORTC        ;GET CONTROL AND SHIFT
0195 77          309         MOV    M,A          ;SAVE IN MEMORY
0196 3EFE        310         MVI    A,0FEH       ;SET UP A
0198 320018      311 LOOPK:  STA    PORTA        ;OUTPUT A
019B 47          312         MOV    B,A          ;SAVE A IN B
019C 3A0118      313         LDA    PORTB        ;READ KEYBOARD
019F 2F          314         CMA                 ;INVERT A
01A0 B7          315         ORA    A            ;SET THE FLAGS
01A1 C2AF01      316         JNZ    SAVKEY       ;LEAVE IF KEY IS DOWN
01A4 78          317         MOV    A,B          ;GET SCAN LINE BACK
01A5 07          318         RLC                 ;ROTATE IT OVER ONE
01A6 DA9801      319         JC     LOOPK        ;DO IT AGAIN
01A9 3E00        320         MVI    A,00H        ;ZERO A
01AB 32EA0F      321         STA    KEYDWN       ;SAVE KEY DOWN
01AE C9          322         RET                 ;LEAVE
01AF 23          323 SAVKEY: INX    H            ;POINT AT RETURN LINE
01B0 2F          324         CMA                 ;PUT A BACK
01B1 77          325         MOV    M,A          ;SAVE RETURN LINE IN MEMORY
01B2 23          326         INX    H            ;POINT H AT SCAN LINE
01B3 70          327         MOV    M,B          ;SAVE SCAN LINE IN MEMORY
01B4 3E40        328         MVI    A,40H        ;SET A
01B6 32EA0F      329         STA    KEYDWN       ;SAVE KEY DOWN
01B9 C9          330         RET                 ;LEAVE
01BA 3E00        331 KYCHNG: MVI    A,00H        ;ZERO 0
01BC 32EA0F      332         STA    KEYDWN       ;RESET KEY DOWN
01BF C38F00      333         JMP    BYPASS       ;LEAVE
01C2 21F10F      334 KYDOWN: LXI    H,SCNLIN     ;GET SCAN LINE
01C5 7E          335         MOV    A,M          ;PUT SCAN LINE IN A
01C6 320018      336         STA    PORTA        ;OUTPUT SCAN LINE TO PORT A
01C9 2B          337         DCX    H            ;POINT AT RETURN LINE
01CA 3A0118      338         LDA    PORTB        ;GET RETURN LINES
01CD B6          339         ORA    M            ;ARE THEY THE SAME?
01CE 2F          340         CMA                 ;INVERT A
01CF B7          341         ORA    A            ;SET FLAGS
01D0 CABA01      342         JZ     KYCHNG       ;IF DIFFERENT KEY HAS CHANGED
01D3 3AEA0F      343         LDA    KEYDWN       ;GET KEY DOWN
01D6 E601        344         ANI    01H          ;HAS THIS BEEN DONE BEFORE?
01D8 C28F00      345         JNZ    BYPASS       ;LEAVE IF IT HAS
01DB 3A0118      346         LDA    PORTB        ;GET RETURN LINE
01DE 06FF        347         MVI    B,0FFH       ;GET READY TO ZERO B
01E0 04          348 UP:     INR    B            ;ZERO B
01E1 0F          349         RRC                 ;ROTATE A
01E2 DAE001      350         JC     UP           ;DO IT AGAIN
01E5 23          351         INX    H            ;POINT H AT SCAN LINES
01E6 7E          352         MOV    A,M          ;GET SCAN LINES
01E7 0EFF        353         MVI    C,0FFH       ;GET READY TO LOOP
01E9 0C          354 UP1:    INR    C            ;START C COUNTING
01EA 0F          355         RRC                 ;ROTATE A
01EB DAE901      356         JC     UP1          ;JUMP TO LOOP
01EE 78          357         MOV    A,B          ;GET RETURN LINES
01EF 07          358         RLC                 ;MOVE OVER ONCE
01F0 07          359         RLC                 ;MOVE OVER TWICE
01F1 07          360         RLC                 ;MOVE OVER THREE TIMES
01F2 B1          361         ORA    C            ;OR SCAN AND RETURN LINES
01F3 47          362         MOV    B,A          ;SAVE A IN B
01F4 3A0218      363         LDA    PORTC        ;GET SHIFT CONTROL
01F7 E640        364         ANI    40H          ;IS CONTROL SET
01F9 4F          365         MOV    C,A          ;SAVE A IN C
01FA 3AEF0F      366         LDA    SHCON        ;GET SHIFT CONTROL
01FD 57          367         MOV    D,A          ;SAVE A IN D
01FE E640        368         ANI    40H          ;STRIP CONTROL
0200 B1          369         ORA    C            ;SET BIT
0201 CA3E02      370         JZ     CNTDWN       ;IF SET LEAVE
0204 3A0218      371         LDA    PORTC        ;READ IT AGAIN,
0207 E620        372         ANI    20H          ;STRIP SHIFT
0209 4F          373         MOV    C,A          ;SAVE A
020A 7A          374         MOV    A,D          ;GET SHIFT CONTROL
020B E620        375         ANI    20H          ;STRIP CONTROL
020D B1          376         ORA    C            ;ARE THEY THE SAME?
020E CA4702      377         JZ     SHDWN        ;IF SET LEAVE
0211 58          378 SCR:    MOV    E,B          ;PUT  TARGET IN E
0212 1600        379         MVI    D,00H        ;ZERO D
0214 210705      380         LXI    H,KYLKUP     ;GET LOOKUP TABLE
0217 19          381         DAD    D            ;GET OFFSET
0218 7E          382         MOV    A,M          ;GET CHARACTER
0219 47          383         MOV    B,A          ;PUT CHARACTER IN B
021A 3A0218      384         LDA    PORTC        ;GET PORTC
021D E610        385         ANI    10H          ;STRIP BIT
021F CA2E02      386         JZ     CAPLOC       ;CAPS LOCK
0222 78          387         MOV    A,B          ;GET A BACK
0223 32EB0F      388 STKEY:  STA    KBCHR        ;SAVE CHARACTER
0226 3EC1        389         MVI    A,0C1H       ;SET A
0228 32EA0F      390         STA    KEYDWN       ;SAVE KEY DOWN
022B C38F00      391         JMP    BYPASS       ;LEAVE
                 392         ;
                 393         ;IF THE CAP LOCK BUTTON IS PUSHED THIS ROUTINE SEES IF
                 394         ;THE CHARACTER IS BETWEEN 61H AND 7AH AND IF IT IS THIS
```

```
                        395          ;ROUTINE ASSUMES THAT THE CHARACTER IS LOWER CASE ASCII
                        396          ;AND SUBTRACTS 20H, WHICH CONVERTS THE CHARACTER TO
                        397          ;UPPER CASE ASCII
                        398          ;
022E 78                 399 CAPLOC:  MOV     A,B             ;GET A BACK
022F FE60               400          CPI     60H             ;HOW BIG IS IT?
0231 DA2302             401          JC      STKEY           ;LEAVE IF IT'S TOO SMALL
0234 FE7B               402          CPI     7BH             ;IS IT TOO BIG
0236 D22302             403          JNC     STKEY           ;LEAVE IF TOO BIG
0239 D620               404          SUI     20H             ;ADJUST A
023B C32302             405          JMP     STKEY           ;STORE THE KEY
                        406          ;
                        407          ;THE ROUTINES SHDWN AND CNTDWN SET BIT 6 AND 7 RESPECTIVLY
                        408          ;IN THE ACC.
                        409          ;
023E 3E80               410 CNTDWN:  MVI     A,80H           ;SET BIT 7 IN A
0240 B0                 411          ORA     B               ;OR WITH CHARACTER
0241 E6BF               412          ANI     0BFH            ;MAKE SURE SHIFT IS NOT SET
0243 47                 413          MOV     B,A             ;PUT IT BACK IN B
0244 C31102             414          JMP     SCR             ;GO BACK
0247 3E40               415 SHDWN:   MVI     A,40H           ;SET BIT 6 IN A
0249 B0                 416          ORA     B               ;OR WITH CHARACTER
024A 47                 417          MOV     B,A             ;PUT IT BACK IN B
024B C31102             418          JMP     SCR             ;GO BACK
                        419          ;
                        420          ;THIS ROUTINE CHECKS FOR ESCAPE CHARACTERS, LF, CR,
                        421          ;FF, AND BACK SPACE
                        422          ;
024E 3AEE0F             423 CHREC:   LDA     ESCP            ;ESCAPE SET?
0251 FE80               424          CPI     80H             ;SEE IF IT IS
0253 CA7B02             425          JZ      ESSQ            ;LEAVE IF IT IS
0256 3AE70F             426          LDA     USCHR           ;GET CHARACTER
0259 FE0A               427          CPI     0AH             ;LINE FEED
025B CAF603             428          JZ      LNFD            ;GO TO LINE FEED
025E FE0C               429          CPI     0CH             ;FORM FEED
0260 CACA03             430          JZ      FMFD            ;GO TO FORM FEED
0263 FE0D               431          CPI     0DH             ;CR
0265 CAAD03             432          JZ      CGRT            ;DO A CR
0268 FE08               433          CPI     08H             ;BACK SPACE
026A CA6E03             434          JZ      LEFT            ;DO A BACK SPACE
026D FE1B               435          CPI     1BH             ;ESCAPE
026F CAA503             436          JZ      ESKAP           ;DO AN ESCAPE
0272 B7                 437          ORA     A               ;CLEAR CARRY
0273 C6E0               438          ADI     0E0H            ;SEE IF CHARACTER IS PRINTABLE
0275 DA7704             439          JC      CHRPUT          ;IF PRINTABLE DO IT
0278 C30F01             440          JMP     SETUP           ;GO BACK AND READ USART AGAIN
                        441          ;
                        442          ;THIS ROUTINE RESETS THE ESCAPE LOCATION AND DECODES
                        443          ;THE CHARACTERS FOLLOWING AN ESCAPE. THE COMMANDS ARE
                        444          ;COMPATABLE WITH INTELS CREDIT TEXT EDITOR
                        445          ;
0278 3E00               446 ESSQ:    MVI     A,00H           ;ZERO A
027D 32EE0F             447          STA     ESCP            ;RESET  ESCP
0280 3AE70F             448          LDA     USCHR           ;GET CHARACTER
0283 FE42               449          CPI     42H             ;DOWN
0285 CAAE02             450          JZ      DOWN            ;MOVE CURSOR DOWN
0288 FE45               451          CPI     45H             ;CLEAR SCREEN CHARACTER
028A CACF02             452          JZ      CLEAR           ;CLEAR THE SCREEN
028D FE4A               453          CPI     4AH             ;CLEAR REST OF SCREEN
028F CAD502             454          JZ      CLRST           ;GO CLEAR THE REST OF THE SCREEN
0292 FE4B               455          CPI     4BH             ;CLEAR LINE CHARACTER
0294 CA2703             456          JZ      CLRLIN          ;GO CLEAR A LINE
0297 FE41               457          CPI     41H             ;CURSOR UP CHARACTER
0299 CA3303             458          JZ      UPCUR           ;MOVE CURSOR UP
029C FE43               459          CPI     43H             ;CURSOR RIGHT CHARACTER
029E CA4503             460          JZ      RIGHT           ;MOVE CURSOR TO THE RIGHT
02A1 FE44               461          CPI     44H             ;CURSOR LEFT CHARACTER
02A3 CA6E03             462          JZ      LEFT            ;MOVE CURSOR TO THE LEFT
02A6 FE48               463          CPI     48H             ;HOME CURSOR CHARACTER
02A8 CA9703             464          JZ      HOME            ;HOME THE CURSOR
02AB C30F01             465          JMP     SETUP           ;LEAVE
                        466          ;
                        467          ;THIS ROUTINE MOVES THE CURSOR DOWN ONE CHARACTER LINE
                        468          ;
02AE 3AE10F             469 DOWN:    LDA     CURSY           ;PUT CURSOR Y IN A
02B1 FE18               470          CPI     CURBOT          ;SEE IF ON BOTTOM OF SCREEN
02B3 CA0F01             471          JZ      SETUP           ;LEAVE IF ON BOTTOM
02B6 3C                 472          INR     A               ;INCREMENT Y CURSOR
02B7 32E10F             473          STA     CURSY           ;SAVE NEW CURSOR
02BA CDB803             474          CALL    LDCUR           ;LOAD THE CURSOR
02BD CDA504             475          CALL    CALCU           ;CALCULATE ADDRESS
02C0 7E                 476          MOV     A,M             ;GET FIRST LOCATION OF THE LINE
02C1 FEF0               477          CPI     0F0H            ;SEE IF CLEAR SCREEN CHARACTER
02C3 C20F01             478          JNZ     SETUP           ;LEAVE IF IT IS NOT
02C6 22E50F             479          SHLD    LOC80           ;SAVE BEGINNING OF THE LINE
02C9 CD1504             480          CALL    CLLINE          ;CLEAR THE LINE
02CC C30F01             481          JMP     SETUP           ;LEAVE
                        482          ;
```

```
                        483          ;THIS ROUTINE CLEARS THE SCREEN.
                        484          ;
02CF CDE403             485 CLEAR:   CALL    CLSCR          ;GO CLEAR THE SCREEN
02D2 C30F01             486          JMP     SETUP          ;GO BACK
                        487          ;
                        488          ;THIS ROUTINE CLEARS ALL LINES BENEATH THE LOCATION
                        489          ;OF THE CURSOR.
                        490          ;
02D5 CDA504             491 CLRST:   CALL    CALCU          ;CALCULATE ADDRESS
02D8 CDCD04             492          CALL    ADX            ;ADD X POSITION
02DB 01204F             493          LXI     B,4F20H        ;PUT SPACE AND LAST X IN B AND C
02DE 3AE20F             494          LDA     CURSX          ;GET X CURSOR
02E1 B8                 495          CMP     B              ;SEE IF AT END OF LINE
02E2 CAEC02             496          JZ      OVR1           ;LEAVE IF X IS AT END OF LINE
02E5 3C                 497 LLP:     INR     A              ;MOVE A OVER ONE X POSITION
02E6 23                 498          INX     H              ;INCREMENT MEMORY POINTER
02E7 71                 499          MOV     M,C            ;PUT A SPACE IN MEMORY
02E8 B8                 500          CMP     B              ;SEE IF A = 4FH
02E9 C2E502             501          JNZ     LLP            ;IF NOT LOOP AGAIN
02EC 01D00F             502 OVR1:    LXI     B,LAST         ;PUT LAST LINE IN BC
02EF 23                 503          INX     H              ;POINT HL TO LAST LINE
02F0 78                 504          MOV     A,B            ;GET B
02F1 BC                 505          CMP     H              ;SAME AS H?
02F2 C2FD02             506          JNZ     CONCL          ;LEAVE IF NOT
02F5 79                 507          MOV     A,C            ;GET C
02F6 BD                 508          CMP     L              ;SAME AS L?
02F7 C2FD02             509          JNZ     CONCL          ;LEAVE IF NOT
02FA 210008             510          LXI     H,TPDIS        ;GET TOP OF DISPLAY
02FD 3AE10F             511 CONCL:   LDA     CURSY          ;GET Y CURSOR
0300 FE18               512          CPI     CURBOT         ;IS IT ON THE BOTTOM
0302 CA0F01             513          JZ      SETUP          ;LEAVE IF IT IS
0305 3C                 514          INR     A              ;MOVE IT DOWN ONE LINE
0306 47                 515          MOV     B,A            ;SAVE CURSOR IN B FOR LATER
0307 115000             516          LXI     D,LNGTH        ;PUT LENGTH OF ONE LINE IN D
030A 36F0               517 CLOOP:   MVI     M,0F0H         ;PUT EOR IN MEMORY
030C 78                 518          MOV     A,B            ;GET CURSOR Y
030D FE18               519          CPI     CURBOT         ;ARE WE ON THE BOTTOM
030F CA0F01             520          JZ      SETUP          ;LEAVE IF WE ARE
0312 3C                 521          INR     A              ;MOVE CURSOR DOWN ONE
0313 19                 522          DAD     D              ;GET NEXT LINE
0314 47                 523          MOV     B,A            ;SAVE A
0315 7C                 524          MOV     A,H            ;PUT H IN A
0316 FE0F               525          CPI     0FH            ;COMPARE TO HIGH LAST
0318 C20A03             526          JNZ     CLOOP          ;LEAVE IF IT IS NOT
031B 7D                 527          MOV     A,L            ;PUT L IN A
031C FED0               528          CPI     0D0H           ;COMPARE TO LOW LAST
031E C20A03             529          JNZ     CLOOP          ;LEAVE IF IT IS NOT
0321 210008             530          LXI     H,TPDIS        ;PUT TOP DISPLAY IN H AND L
0324 C30A03             531          JMP     CLOOP          ;LOOP AGAIN
                        532          ;
                        533          ;THIS ROUTINE CLEARS THE LINE THE CURSOR IS ON.
                        534          ;
0327 CDA504             535 CLRLIN:  CALL    CALCU          ;CALCULATE ADDRESS
032A 22E50F             536          SHLD    LOC80          ;STORE H AND L TO CLEAR LINE
032D CD1504             537          CALL    CLLINE         ;CLEAR THE LINE
0330 C30F01             538          JMP     SETUP          ;GO BACK
                        539          ;
                        540          ;THIS ROUTINE MOVES THE CURSOR UP ONE LINE.
                        541          ;
0333 3AE10F             542 UPCUR:   LDA     CURSY          ;GET Y CURSOR
0336 FE00               543          CPI     00H            ;IS IT ZERO
0338 CA0F01             544          JZ      SETUP          ;IF IT IS LEAVE
033B 3D                 545          DCR     A              ;MOVE CURSOR UP
033C 32E10F             546          STA     CURSY          ;SAVE NEW CURSOR
033F CD8803             547          CALL    LDCUR          ;LOAD THE CURSOR
0342 C30F01             548          JMP     SETUP          ;LEAVE
                        549          ;
                        550          ;THIS ROUTINE MOVES THE CURSOR ONE LOCATION TO THE RIGHT
                        551          ;
0345 3AE20F             552 RIGHT:   LDA     CURSX          ;GET X CURSOR
0348 FE4F               553          CPI     4FH            ;IS IT ALL THE WAY OVER?
034A C25403             554          JNZ     NTOVER         ;IF NOT JUMP AROUND
034D 3AE10F             555          LDA     CURSY          ;GET Y CURSOR
0350 FE18               556          CPI     CURBOT         ;SEE IF ON BOTTOM
0352 CA5903             557          JZ      GD18           ;IF WE ARE JUMP
0355 3C                 558          INR     A              ;INCREMENT Y CURSOR
0356 32E10F             559          STA     CURSY          ;SAVE IT
0359 3E00               560 GD18:    MVI     A,00H          ;ZERO A
035B 32E20F             561          STA     CURSX          ;ZERO X CURSOR
035E CD8803             562          CALL    LDCUR          ;LOAD THE CURSOR
0361 C30F01             563          JMP     SETUP          ;LEAVE
0364 3C                 564 NTOVER:  INR     A              ;INCREMENT X CURSOR
0365 32E20F             565          STA     CURSX          ;SAVE IT
0368 CD8803             566          CALL    LDCUR          ;LOAD THE CURSOR
036B C30F01             567          JMP     SETUP          ;LEAVE
                        568          ;
                        569          ;THIS ROUTINE MOVES THE CURSOR LEFT ONE CHARACTER POSITION
```

```
                        570
                        571 ;
036E 3AE20F             572 LEFT:   LDA     CURSX           ;GET X CURSOR
0371 FE00               573         CPI     00H             ;IS IT ALL THE WAY OVER
0373 C28D03             574         JNZ     NOVER           ;IF NOT JUMP AROUND
0376 3AE10F             575         LDA     CURSY           ;GET CURSOR Y
0379 FE00               576         CPI     00H             ;IS IT ZERO?
037B CA0F01             577         JZ      SETUP           ;IF IT IS JUMP
037E 3D                 578         DCR     A               ;MOVE CURSOR Y UP
037F 32E10F             579         STA     CURSY           ;SAVE IT
0382 3E4F               580         MVI     A,4FH           ;GET LAST X LOCATION
0384 32E20F             581         STA     CURSX           ;SAVE IT
0387 CDB803             582         CALL    LDCUR           ;LOAD THE CURSOR
038A C30F01             583         JMP     SETUP
038D 3D                 584 NOVER:  DCR     A               ;ADJUST X CURSOR
038E 32E20F             585         STA     CURSX           ;SAVE CURSOR X
0391 CDB803             586         CALL    LDCUR           ;LOAD THE CURSOR
0394 C30F01             587         JMP     SETUP           ;LEAVE
                        588         ;
                        589         ;THIS ROUTINE HOMES THE CURSOR.
                        590         ;
0397 3E00               591 HOME:   MVI     A,00H           ;ZERO A
0399 32E20F             592         STA     CURSX           ;ZERO X CURSOR
039C 32E10F             593         STA     CURSY           ;ZERO Y CURSOR
039F CDB803             594         CALL    LDCUR           ;LOAD THE CURSOR
03A2 C30F01             595         JMP     SETUP           ;LEAVE
                        596         ;
                        597         ;THIS ROUTINE SETS THE ESCAPE BIT
                        598         ;
03A5 3E80               599 ESKAP:  MVI     A,80H           ;LOAD A WITH ESCAPE BIT
03A7 32EE0F             600         STA     ESCP            ;SET ESCAPE LOCATION
03AA C30F01             601         JMP     SETUP           ;GO BACK AND READ USART
                        602         ;
                        603         ;THIS ROUTINE DOES A CR
                        604         ;
03AD 3E00               605 CGRT:   MVI     A,00H           ;ZERO A
03AF 32E20F             606         STA     CURSX           ;ZERO  CURSOR X
03B2 CDB803             607         CALL    LDCUR           ;LOAD CURSOR INTO 8275
03B5 C30F01             608         JMP     SETUP           ;POLL USART AGAIN
                        609         ;
                        610         ;THIS ROUTINE LOADS THE CURSOR
                        611         ;
03B8 3E80               612 LDCUR:  MVI     A,80H           ;PUT 80H INTO A
03BA 320110             613         STA     CRTS            ;LOAD CURSOR INTO 8275
03BD 3AE20F             614         LDA     CURSX           ;GET CURSOR X
03C0 320010             615         STA     CRTM            ;PUT IT IN 8275
03C3 3AE10F             616         LDA     CURSY           ;GET CURSOR Y
03C6 320010             617         STA     CRTM            ;PUT IT IN 8275
03C9 C9                 618         RET
                        619         ;
                        620         ;THIS ROUTINE DOES A FORM FEED
                        621         ;
03CA CDE403             622 FMFD:   CALL    CLSCR           ;CALL CLEAR SCREEN
03CD 210008             623         LXI     H,TPDIS         ;PUT TOP DISPLAY IN HL
03D0 22E50F             624         SHLD    LOC80           ;PUT IT IN LOC80
03D3 CD1504             625         CALL    CLLINE          ;CLEAR TOP LINE
03D6 3E00               626         MVI     A,00H           ;ZERO A
03D8 32E20F             627         STA     CURSX           ;ZERO CURSOR X
03DB 32E10F             628         STA     CURSY           ;ZERO CURSOR Y
03DE CDB803             629         CALL    LDCUR           ;LOAD THE CURSOR
03E1 C30F01             630         JMP     SETUP           ;BACK TO USART
                        631         ;
                        632         ;THIS ROUTINE CLEARS THE SCREEN BY WRITING END OF ROW
                        633         ;CHARACTERS INTO THE FIRST LOCATION OF ALL LINES ON
                        634         ;THE SCREEN.
                        635         ;
03E4 3EF0               636 CLSCR:  MVI     A,0F0H          ;PUT EOR CHARACTER IN A
03E6 0618               637         MVI     B,CURBOT        ;LOAD B WITH MAX Y
03E8 04                 638         INR     B               ;GO TO MAX PLUS ONE
03E9 210008             639         LXI     H,TPDIS         ;LOAD H AND L WITH TOP OF RAM
03EC 115000             640         LXI     D,LNGTH         ;MOVE 50H = 80D INTO D AND E
03EF 77                 641 LOADX:  MOV     M,A             ;MOVE EOR INTO MEMORY
03F0 19                 642         DAD     D               ;CHANGE POINTER BY 80D
03F1 05                 643         DCR     B               ;COUNT THE LOOPS
03F2 C2EF03             644         JNZ     LOADX           ;CONTINUE IF NOT ZERO
03F5 C9                 645         RET                     ;GO BACK
                        646         ;
                        647         ;THIS ROUTINE DOES A LINE FEED
                        648         ;
03F6 CDFC03             649 LNFD:   CALL    LNFD1           ;CALL ROUTINE
03F9 C30F01             650         JMP     SETUP           ;POLL FLAGS
                        651         ;
                        652         ;LINE FEED
                        653         ;
03FC 3AE10F             654 LNFD1:  LDA     CURSY           ;GET Y LOCATION OF CURSOR
03FF FE18               655         CPI     CURBOT          ;SEE IF AT BOTTOM OF SCREEN
0401 CA5304             656         JZ      ONBOT           ;IF WE ARE, LEAVE
0404 3C                 657         INR     A               ;INCREMENT A
0405 32E10F             658         STA     CURSY           ;SAVE NEW CURSOR
```

```
0408 CDA504          658          CALL    CALCU           ;CALCULATE ADDRESS
040B 22E50F          659          SHLD    LOC80           ;SAVE TO CLEAR LINE
040E CD1504          660          CALL    CLLINE          ;CLEAR THE LINE
0411 CDB803          661          CALL    LDCUR           ;LOAD THE CURSOR
0414 C9              662          RET                     ;LEAVE
                     663          ;
                     664          ;THIS ROUTINE CLEARS THE LINE WHOSE FIRST ADDRESS
                     665          ;IS IN LOC80. PUSH INSTRUCTIONS ARE USED TO RAPIDLY
                     666          ;CLEAR THE LINE
                     667          ;
0415 F3              668 CLLINE:  DI                      ;NO INTERRUPTS HERE
0416 2AE50F          669          LHLD    LOC80           ;GET LOC80
0419 115000          670          LXI     D,LNGTH         ;GET OFFSET
041C 19              671          DAD     D               ;ADD OFFSET
041D EB              672          XCHG                    ;PUT START IN DE
041E 210000          673          LXI     H,0000H         ;ZERO HL
0421 39              674          DAD     SP              ;GET STACK
0422 EB              675          XCHG                    ;PUT STACK IN DE
0423 F9              676          SPHL                    ;PUT START IN SP
0424 212020          677          LXI     H,2020H         ;PUT SPACES IN HL
                     678          ;
                     679          ;NOW DO 40 PUSH INSTRUCTIONS TO CLEAR THE LINE
                     680          ;
                     681          REPT    (LNGTH/2)
                     682          PUSH    H
                     683          ENDM
0427 E5              684+         PUSH    H
0428 E5              685+         PUSH    H
0429 E5              686+         PUSH    H
042A E5              687+         PUSH    H
042B E5              688+         PUSH    H
042C E5              689+         PUSH    H
042D E5              690+         PUSH    H
042E E5              691+         PUSH    H
042F E5              692+         PUSH    H
0430 E5              693+         PUSH    H
0431 E5              694+         PUSH    H
0432 E5              695+         PUSH    H
0433 E5              696+         PUSH    H
0434 E5              697+         PUSH    H
0435 E5              698+         PUSH    H
0436 E5              699+         PUSH    H
0437 E5              700+         PUSH    H
0438 E5              701+         PUSH    H
0439 E5              702+         PUSH    H
043A E5              703+         PUSH    H
043B E5              704+         PUSH    H
043C E5              705+         PUSH    H
043D E5              706+         PUSH    H
043E E5              707+         PUSH    H
043F E5              708+         PUSH    H
0440 E5              709+         PUSH    H
0441 E5              710+         PUSH    H
0442 E5              711+         PUSH    H
0443 E5              712+         PUSH    H
0444 E5              713+         PUSH    H
0445 E5              714+         PUSH    H
0446 E5              715+         PUSH    H
0447 E5              716+         PUSH    H
0448 E5              717+         PUSH    H
0449 E5              718+         PUSH    H
044A E5              719+         PUSH    H
044B E5              720+         PUSH    H
044C E5              721+         PUSH    H
044D E5              722+         PUSH    H
044E E5              723+         PUSH    H
044F EB              724          XCHG                    ;PUT STACK IN HL
0450 F9              725          SPHL                    ;PUT IT BACK IN SP
0451 FB              726          EI                      ;ENABLE INTERRUPTS
0452 C9              727          RET                     ;GO BACK
                     728          ;
                     729          ;IF CURSOR IS ON THE BOTTOM OF THE SCREEN THIS ROUTINE
                     730          ;IS USED TO IMPLEMENT THE LINE FEED
                     731          ;
0453 2AE30F          732 ONBOT:   LHLD    TOPAD           ;GET TOP ADDRESS
0456 22E50F          733          SHLD    LOC80           ;SAVE IT IN LOC80
0459 115000          734          LXI     D,LNGTH         ;LINE LENGTH
045C 19              735          DAD     D               ;ADD HL + DE
045D 01D00F          736          LXI     B,LAST          ;GET BOTTOM LINE
0460 7C              737          MOV     A,H             ;GET H
0461 B8              738          CMP     B               ;SAME AS B
0462 C26D04          739          JNZ     ARND            ;LEAVE IF NOT SAME
0465 7D              740          MOV     A,L             ;GET L
0466 B9              741          CMP     C               ;SAME AS C
0467 C26D04          742          JNZ     ARND            ;LEAVE IF NOT SAME
046A 210008          743          LXI     H,TPDIS         ;LOAD HL WITH TOP OF DISPLAY
046D 22E30F          744 ARND:    SHLD    TOPAD           ;SAVE NEW TOP ADDRESS
```

```
0470 CD1504          745           CALL    CLLINE          ;CLEAR LINE
0473 CDB803          746           CALL    LDCUR           ;LOAD THE CURSOR
0476 C9              747           RET
                     748           ;
                     749           ;THIS ROUTINE PUTS A CHARACTER ON THE SCREEN AND
                     750           ;INCREMENTS THE X CURSOR POSITION. A LINE FEED IS
                     751           ;INSERTED IF THE INCREMENTED CURSOR EQUALS 81D
                     752           ;
0477 CDA504          753 CHRPUT:   CALL    CALCU           ;CALCULATE SCREEN POSITION
047A 7E              754           MOV     A,M             ;GET FIRST CHARACTER
047B FEF0            755           CPI     0F0H            ;IS IT A CLEAR LINE
047D 22E50F          756           SHLD    LOC80           ;SAVE LINE TO CLEAR
0480 CC1504          757           CZ      CLLINE          ;CLEAR LINE
0483 2AE50F          758           LHLD    LOC80           ;GET LINE
0486 CDCD04          759           CALL    ADX             ;ADD CURSOR X
0489 3AE70F          760           LDA     USC'R           ;GET CHARACTER
048C 77              761           MOV     M,A             ;PUT IT ON SCREEN
048D 3AE20F          762           LDA     CURSX           ;GET CURSOR X
0490 3C              763           INR     A               ;INCREMENT CURSOR X
0491 FE50            764           CPI     LNGTH           ;HAS IT GONE TOO FAR?
0493 C29C04          765           JNZ     OK1             ;IF NOT GOOD
0496 CDFC03          766           CALL    LNFD1           ;DO A LINE FEED
0499 C3AD03          767           JMP     CGRT            ;DO A CR
049C 32E20F          768 OK1:      STA     CURSX           ;SAVE CURSOR
049F CDB803          769           CALL    LDCUR           ;LOAD THE CURSOR
04A2 C30F01          770           JMP     SETUP           ;LEAVE
                     771           ;
                     772           ;THIS ROUTINE TAKES THE TOP ADDRESS AND THE Y CURSOR
                     773           ;LOCATION AND CALCULATES THE ADDRESS OF THE LINE
                     774           ;THAT THE CURSOR IS ON. THE RESULT IS RETURNED IN H
                     775           ;AND L AND ALL REGISTERS ARE USED.
                     776           ;
04A5 21D504          777 CALCU:    LXI     H,LINTAB        ;GET LINE TABLE INTO H AND L
04A8 3AE10F          778           LDA     CURSY           ;GET CURSOR INTO A
04AB 07              779           RLC                     ;SET UP A FOR LOOKUP TABLE
04AC 0600            780           MVI     B,00H           ;ZERO B
04AE 4F              781           MOV     C,A             ;PUT CURSOR INTO A
04AF 09              782           DAD     B               ;ADD LINE TABLE TO Y CURSOR
04B0 7E              783           MOV     A,M             ;PUT LOW LINE TABLE INTO A
04B1 4F              784           MOV     C,A             ;PUT LOW LINE TABLE INTO C
04B2 23              785           INX     H               ;CHANGE MEMORY POINTER
04B3 7E              786           MOV     A,M             ;PUT HIGH LINE TABLE INTO A
04B4 47              787           MOV     B,A             ;PUT HIGH LINE TABLE INTO B
04B5 2100F8          788           LXI     H,0F800H        ;TWOS COMPLEMENT SCREEN LOCATION
04B8 09              789           DAD     B               ;SUBTRACT OFFSET
04B9 EB              790           XCHG                    ;SAVE HL IN DE
04BA 2AE30F          791           LHLD    TOPAD           ;GET TOP ADDRESS IN H AND L
04BD 19              792           DAD     D               ;GET DISPLACED ADDRESS
04BE EB              793           XCHG                    ;SAVE IT IN D
04BF 2130F0          794           LXI     H,0F030H        ;TWOS COMPLEMENT SCREEN LOCATION
04C2 19              795           DAD     D               ;SEE IF WE ARE OFF THE SCREEN
04C3 DAC804          796           JC      FIX             ;IF WE ARE FIX IT
04C6 EB              797           XCHG                    ;GET DISPLACED ADDRESS BACK
04C7 C9              798           RET                     ;GO BACK
04C8 2130F8          799 FIX:      LXI     H,0F830H        ;SCREEN BOUNDRY
04CB 19              800           DAD     D               ;ADJUST SCREEN
04CC C9              801           RET                     ;GO BACK
                     802           ;
                     803           ;THIS ROUTINE ADDS THE X CURSOR LOCATION TO THE ADDRESS
                     804           ;THAT IS IN THE H AND L REGISTERS AND RETURNS THE RESULT
                     805           ;IN H AND L
                     806           ;
04CD 3AE20F          807 ADX:      LDA     CURSX           ;GET CURSOR
04D0 0600            808           MVI     B,00H           ;ZERO B
04D2 4F              809           MOV     C,A             ;PUT CURSOR X IN C
04D3 09              810           DAD     B               ;ADD CURSOR X TO H AND L
04D4 C9              811           RET                     ;LEAVE
                     812           ;
                     813           ;THIS TABLE CONTAINS THE OFFSET ADDRESSES FOR EACH
                     814           ;OF THE 25 DISPLAYED LINES.
                     815           ;
0000                 816 LINTAB:   LINNUM SET 0
                     817           REPT (CURBOT+1)
                     818           DW      TPDIS+(LNGTH*LINNUM)
                     819           LINNUM SET (LINNUM+1)
                     820           ENDM
04D5 0008            821+          DW      TPDIS+(LNGTH*LINNUM)
0001                 822+          LINNUM SET (LINNUM+1)
04D7 5008            823+          DW      TPDIS+(LNGTH*LINNUM)
0002                 824+          LINNUM SET (LINNUM+1)
04D9 A008            825+          DW      TPDIS+(LNGTH*LINNUM)
0003                 826+          LINNUM SET (LINNUM+1)
04DB F008            827+          DW      TPDIS+(LNGTH*LINNUM)
0004                 828+          LINNUM SET (LINNUM+1)
04DD 4009            829+          DW      TPDIS+(LNGTH*LINNUM)
0005                 830+          LINNUM SET (LINNUM+1)
04DF 9009            831+          DW      TPDIS+(LNGTH*LINNUM)
```

```
0006                    832+        LINNUM SET (LINNUM+1)
04E1 E009               833+        DW      TPDIS+(LNGTH*LINNUM)
0007                    834+        LINNUM SET (LINNUM+1)
04E3 300A               835+        DW      TPDIS+(LNGTH*LINNUM)
0008                    836+        LINNUM SET (LINNUM+1)
04E5 800A               837+        DW      TPDIS+(LNGTH*LINNUM)
0009                    838+        LINNUM SET (LINNUM+1)
04E7 D00A               839+        DW      TPDIS+(LNGTH*LINNUM)
000A                    840+        LINNUM SET (LINNUM+1)
04E9 200B               841+        DW      TPDIS+(LNGTH*LINNUM)
000B                    842+        LINNUM SET (LINNUM+1)
04EB 700B               843+        DW      TPDIS+(LNGTH*LINNUM)
000C                    844+        LINNUM SET (LINNUM+1)
04ED C00B               845+        DW      TPDIS+(LNGTH*LINNUM)
000D                    846+        LINNUM SET (LINNUM+1)
04EF 100C               847+        DW      TPDIS+(LNGTH*LINNUM)
000E                    848+        LINNUM SET (LINNUM+1)
04F1 600C               849+        DW      TPDIS+(LNGTH*LINNUM)
000F                    850+        LINNUM SET (LINNUM+1)
04F3 B00C               851+        DW      TPDIS+(LNGTH*LINNUM)
0010                    852+        LINNUM SET (LINNUM+1)
04F5 000D               853+        DW      TPDIS+(LNGTH*LINNUM)
0011                    854+        LINNUM SET (LINNUM+1)
04F7 500D               855+        DW      TPDIS+(LNGTH*LINNUM)
0012                    856+        LINNUM SET (LINNUM+1)
04F9 A00D               857+        DW      TPDIS+(LNGTH*LINNUM)
0013                    858+        LINNUM SET (LINNUM+1)
04FB F00D               859+        DW      TPDIS+(LNGTH*LINNUM)
0014                    860+        LINNUM SET (LINNUM+1)
04FD 400E               861+        DW      TPDIS+(LNGTH*LINNUM)
0015                    862+        LINNUM SET (LINNUM+1)
04FF 900E               863+        DW      TPDIS+(LNGTH*LINNUM)
0016                    864+        LINNUM SET (LINNUM+1)
0501 E00E               865+        DW      TPDIS+(LNGTH*LINNUM)
0017                    866+        LINNUM SET (LINNUM+1)
0503 300F               867+        DW      TPDIS+(LNGTH*LINNUM)
0018                    868+        LINNUM SET (LINNUM+1)
0505 800F               869+        DW      TPDIS+(LNGTH*LINNUM)
0019                    870+        LINNUM SET (LINNUM+1)
                        871         ;
                        872         ;KEYBOARD LOOKUP TABLE
                        873         ;THIS TABLE CONTAINS ALL THE ASCII CHARACTERS
                        874         ;THAT ARE TRANSMITTED BY THE TERMINAL
                        875         ;THE CHARACTERS ARE ORGANIZED SO THAT BITS 0,1 AND 2
                        876         ;ARE THE SCAN LINES, BITS 3,4 AND 5 ARE THE RETURN LINES
                        877         ;BIT 6 IS SHIFT AND BIT 7 IS CONTROL
                        878         ;
0507 38                 879 KYLKUP: DB      38H,39H         ;8 AND 9
0508 39
0509 30                 880         DB      30H,2DH         ;0 AND -
050A 2D
050B 3D                 881         DB      3DH,5CH         ;= AND \
050C 5C
050D 08                 882         DB      08H,00H         ;BS AND BREAK
050E 00
050F 75                 883         DB      75H,69H         ;LOWER CASE U AND I
0510 69
0511 6F                 884         DB      6FH,70H         ;LOWER CASE O AND P
0512 70
0513 5B                 885         DB      5BH,5CH         ;[ AND \
0514 5C
0515 0A                 886         DB      0AH,7FH         ;LF AND DELETE
0516 7F
0517 6A                 887         DB      6AH,6BH         ;LOWER CASE J AND K
0518 6B
0519 6C                 888         DB      6CH,3BH         ;LOWER CASE L AND ;
051A 3B
051B 27                 889         DB      27H,00H         ;' AND NOTHING
051C 00
051D 0D                 890         DB      0DH,37H         ;CR AND 7
051E 37
051F 6D                 891         DB      6DH,2CH         ;LOWER CASE M AND COMMA
0520 2C
0521 2E                 892         DB      2EH,2FH         ;PERIOD AND SLASH
0522 2F
0523 00                 893         DB      00H,00H         ;BLANK AND NOTHING
0524 00
0525 00                 894         DB      00H,00H         ;NOTHING AND NOTHING
0526 00
0527 00                 895         DB      00H,61H         ;NOTHING AND LOWER CASE A
0528 61
0529 7A                 896         DB      7AH,78H         ;LOWER CASE Z AND X
052A 78
052B 63                 897         DB      63H,76H         ;LOWER CASE C AND V
052C 76
052D 62                 898         DB      62H,6EH         ;LOWER CASE B AND N
052E 6E
```

```
052F 79          899          DB          79H,00H          ;LOWER CASE Y AND NOTHING
0530 00
0531 00          900          DB          00H,20H          ;NOTHING AND SPACE
0532 20
0533 64          901          DB          64H,66H          ;LOWER CASE D AND F
0534 66
0535 67          902          DB          67H,68H          ;LOWER CASE G AND H
0536 68
0537 00          903          DB          00H,71H          ;TAB AND LOWER CASE Q
0538 71
0539 77          904          DB          77H,73H          ;LOWER CASE W AND S
053A 73
053B 65          905          DB          65H,72H          ;LOWER CASE E AND R
053C 72
053D 74          906          DB          74H,00H          ;LOWER CASE T AND NOTHING
053E 00
053F 1B          907          DB          1BH,31H          ;ESCAPE AND I
0540 31
0541 32          908          DB          32H,33H          ; 2 AND 3
0542 33
0543 34          909          DB          34H,35H          ; 4 AND 5
0544 35
0545 36          910          DB          36H,00H          ; 6 AND NOTHING
0546 00
0547 2A          911          DB          2AH,28H          ;* AND )
0548 28
0549 29          912          DB          29H,5FH          ;( AND -
054A 5F
054B 2B          913          DB          2BH,00H          ;+ AND NOTHING
054C 00
054D 08          914          DB          08H,00H          ;BS AND BREAK
054E 00
054F 55          915          DB          55H,49H          ;U AND I
0550 49
0551 4F          916          DB          4FH,50H          ;O AND P
0552 50
0553 5D          917          DB          5DH,00H          ;] AND NO CHARACTER
0554 00
0555 0A          918          DB          0AH,7FH          ;LF AND DELETE
0556 7F
0557 4A          919          DB          4AH,4BH          ;J AND K
0558 4B
0559 4C          920          DB          4CH,3AH          ;L AND :
055A 3A
055B 22          921          DB          22H,00H          ;" AND NO CHARACTER
055C 00
055D 0D          922          DB          0DH,26H          ;CR AND &
055E 26
055F 4D          923          DB          4DH,3CH          ;M AND <
0560 3C
0561 3E          924          DB          3EH,3FH          ;> AND ?
0562 3F
0563 00          925          DB          00H,00H          ;BLANK AND NOTHING
0564 00
0565 00          926          DB          00H,00H          ;NOTHING AND NOTHING
0566 00
0567 00          927          DB          00H,41H          ;NOTHING AND A
0568 41
0569 5A          928          DB          5AH,58H          ;Z AND X
056A 58
056B 43          929          DB          43H,56H          ;C AND V
056C 56
056D 42          930          DB          42H,4EH          ;B AND N
056E 4E
056F 59          931          DB          59H,00H          ;Y AND NOTHING
0570 00
0571 00          932          DB          00H,20H          ;NO CHARACTER AND SPACE
0572 20
0573 44          933          DB          44H,46H          ;D AND F
0574 46
0575 47          934          DB          47H,48H          ;G AND H
0576 48
0577 00          935          DB          00H,51H          ;TAB AND Q
0578 51
0579 57          936          DB          57H,53H          ;W AND S
057A 53
057B 45          937          DB          45H,52H          ;E AND R
057C 52
057D 54          938          DB          54H,00H          ;T AND NO CONNECTION
057E 00
057F 1B          939          DB          1BH,21H          ;ESCAPE AND !
0580 21
0581 40          940          DB          40H,23H          ;@ AND #
0582 23
0583 24          941          DB          24H,25H          ;$ AND %
0584 25
0585 5E          942          DB          5EH,00H          ;^ AND NO CONNECTION
```

```
0586 00
                        943     ;
                        944     ;THIS IS WHERE THE CONTROL CHARACTERS ARE LOOKED UP
                        945     ;
0587 00                 946     DB      00H,00H         ;NOTHING
0588 00
0589 00                 947     DB      00H,00H         ;NOTHING
058A 00
058B 00                 948     DB      00H,00H         ;NOTHING
058C 00
058D 00                 949     DB      00H,00H         ;NOTHING
058E 00
058F 15                 950     DB      15H,09H         ;CONTROL U AND I
0590 09
0591 0F                 951     DB      0FH,10H         ;CONTROL O AND P
0592 10
0593 0B                 952     DB      0BH,0CH         ;CONTROL [ AND \
0594 0C
0595 0A                 953     DB      0AH,7FH         ;LF AND DELETE
0596 7F
0597 0A                 954     DB      0AH,0BH         ;CONTROL J AND K
0598 0B
0599 0C                 955     DB      0CH,00H         ;CONTROL L AND NOTHING
059A 00
059B 00                 956     DB      00H,00H         ;NOTHING
059C 00
059D 0D                 957     DB      0DH,00H         ;CR AND NOTHING
059E 00
059F 0D                 958     DB      0DH,00H         ;CONTROL M AND COMMA
05A0 00
05A1 00                 959     DB      00H,00H         ;NOTHING
05A2 00
05A3 00                 960     DB      00H,00H         ;NOTHING
05A4 00
05A5 00                 961     DB      00H,00H         ;NOTHING AND NOTHING
05A6 00
05A7 1A                 962     DB      1AH,18H         ;CONTROL Z AND X
05A8 18
05A9 03                 963     DB      03H,16H         ;CONTROL C AND V
05AA 16
05AB 02                 964     DB      02H,0EH         ;CONTROL B AND N
05AC 0E
05AD 19                 965     DB      19H,00H         ;CONTROL Y AND NOTHING
05AE 00
05AF 00                 966     DB      00H,20H         ;NOTHING AND SPACE
05B0 20
05B1 04                 967     DB      04H,06H         ;CONTROL D AND F
05B2 06
05B3 07                 968     DB      07H,08H         ;CONTROL G AND H
05B4 08
05B5 00                 969     DB      00H,11H         ;NOTHING AND CONTROL Q
05B6 11
05B7 17                 970     DB      17H,13H         ;CONTROL W AND S
05B8 13
05B9 06                 971     DB      06H,12H         ;CONTROL E AND R
05BA 12
05BB 14                 972     DB      14H,00H         ;CONTROL W AND NOTHING
05BC 00
05BD 1B                 973     DB      1BH,1DH         ;ESCAPE AND HOME(CREDIT)
05BE 1D
05BF 1E                 974     DB      1EH,1CH         ;CURSOR UP AND DOWN(CREDIT)
05C0 1C
05C1 14                 975     DB      14H,1FH         ;CURSOR RIGHT AND LEFT(CREDIT)
05C2 1F
05C3 00                 976     DB      00H,00H         ;NOTHING
05C4 00
                        977     ;
                        978     ;LOOK UP TABLE FOR 8253 BAUD RATE GENERATOR
                        979     ;
05C5 00                 980 BDLK:   DB      00H,05H,69H,03H ;75 AND 110 BAUD
05C6 05
05C7 69
05C8 03
05C9 80                 981     DB      80H,02H,40H,01H ;150 AND 300 BAUD
05CA 02
05CB 40
05CC 01
05CD A0                 982     DB      0A0H,00H        ;600 BAUD
05CE 00
05CF 50                 983     DB      50H,00H         ;1200 BAUD
05D0 00
05D1 28                 984     DB      28H,00H         ;2400 BAUD
05D2 00
05D3 14                 985     DB      14H,00H         ;4800 BAUD
05D4 00
05D5 0A                 986     DB      0AH,00H         ;9600 BAUD
05D6 00
```

```
                        987             ;
                        988             ;DATA AREA
                        989             ;
0FE1                    990             ORG     0FE1H
0001                    991 CURSY:      DS      1
0001                    992 CURSX:      DS      1
0002                    993 TOPAD:      DS      2
0002                    994 LOC80:      DS      2
0001                    995 USCHR:      DS      1
0002                    996 CURAD:      DS      2
0001                    997 KEYDWN:     DS      1
0001                    998 KBCHR:      DS      1
0001                    999 BAUD:       DS      1
0001                   1000 KEYOK:      DS      1
0001                   1001 ESCP:       DS      1
0001                   1002 SHCON:      DS      1
0001                   1003 RETLIN:     DS      1
0001                   1004 SCNLIN:     DS      1
                       1005             END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
ADX     A 04CD  ARND    A 046D  BAUD    A 0FEC  BDLK    A 05C5  BTDIS   A 0F80  BYPASS  A 008F
CAPLOC  A 022E  CGRT    A 03AD  CHREC   A 024E  CHRPUT  A 0477  CLEAR   A 02CF  CLLINE  A 0415
CLRLIN  A 0327  CLRST   A 02D5  CLSCR   A 03E4  CNT0    A 6000  CNT1    A 6001  CNT2    A 6002
CNTM    A 6003  CNWD55  A 1803  CONCL   A 02FD  CRTM    A 1000  CRTS    A 1001  CURAD   A 0FE8
CURSX   A 0FE2  CURSY   A 0FE1  DOWN    A 02AE  ESCP    A 0FEE  ESKAP   A 03A5  ESSQ    A 027B
FMFD    A 03CA  FRAME   A 0167  GD18    A 0359  HOME    A 0397  IN75    A 00F9  INT75   A 1401
KEYDWN  A 0FEA  KEYINP  A 0121  KEYOK   A 0FED  KEYS    A 0131  KPTK    A 0084  KYCHNG  A 01BA
KYLKUP  A 0507  LAST    A 0FD0  LDCUR   A 03B8  LEFT    A 036E  LINNUM  A 0019  LINTAB  A 04D5
LNFD    A 03F6  LNFD1   A 03FC  LNGTH   A 0050  LOADX   A 03EF  LOC80   A 0FE5  LOOPF   A 00A7
LPKBD   A 0098  NOVER   A 038D  NTOVER  A 0364  OK1     A 049C  OK7     A 015C  ONBOT   A 0453
POPDAT  A 0034  PORTA   A 1800  PORTB   A 1801  PORTC   A 1802  RDKB    A 018F  RETLIN  A 0FF0
RXRDY   A 0113  SAVKEY  A 01AF  SCNLIN  A 0FF1  SCR     A 0211  SETUP   A 010F  SHCON   A 0FEF
STBAUD  A 00DC  STKEY   A 0223  STPTR   A 0FE0  TOPAD   A 0FE3  TPDIS   A 0800  TRANS   A 014B
UP1     A 01E9  UPCUR   A 0333  USCHR   A 0FE7  USTD    A A000  USTF    A A001

ASSEMBLY COMPLETE,  NO ERRORS
```

September 1983

Low-Cost CRT Control Does More with Less

Thomas Rossi, Applications Manager
Peripheral Components Division

Reprinted with permission from Electronic Design', Vol.29, No.9; copyright Hayden Publishing Co., Inc., 1981

ORDER NUMBER: 210507-001

Fewer parts make a microprocessor-based CRT controller cost-effective, and interrupt-driven software cuts overhead on the system's CPU.

Low-cost CRT control does more with less

The multitude of components and the CPU overhead long associated with cathode-ray-tube controllers are rapidly becoming conspicuous by their absence. In particular, an intelligent terminal based on Intel's iAPX 88/10 (8088) microprocessor and 8276 small-system CRT controller eliminates all but 22 of the nearly 40 chips required by other CRT controllers (even those with microprocessors and integrated peripherals). It also cuts overhead on the processor to less than 25%, so that the 88/10 is free to implement such intelligent terminal functions as local data processing.

The iAPX 88/10 implementation supplies characters directly to the 8276 by means of interrupt-driven software, eliminating the need for a direct-memory-access (DMA) controller. The design interfaces directly with standard CRT monitors, contact-closure keyboards, and RS-232C serial-communication links (asynchronous or bisynchronous), to provide a complete stand-alone operator interface.

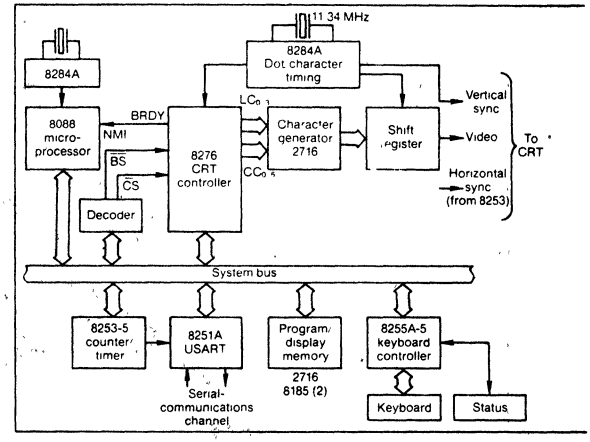
Although the primary design goal—implementing a low-cost CRT terminal—has excluded some useful CRT features, these are easily made available through additional external hardware. For example, composite video is added with two TTL packages, a transistor, and some resistors and capacitors. Another simple option involves the two general-purpose attribute outputs on the 8276 and lets users select any one of four colors on a color monitor.

Basic system configuration and architecture

Central to the 22-chip CRT controller design is an iAPX 88/10 8-bit microprocessor operating at 5 MHz and supported by two 8185 1-kbit × 8 static RAMs and a 2716 control software PROM (Fig. 1). An 8251A programmable communication interface provides synchronous or asynchronous serial communications. Three manual switches on the PC board select the baud rate, and one of the 8253's three independent programmable interval timers generates the 8251A's baud-rate clock under software control.

The three PC-board switches are monitored by the iAPX 88/10 to determine the desired baud rate. When the CPU detects a change in the switch positions, the 8253 is loaded with the appropriate count for the new baud rate.

An 8255A provides three 8-bit parallel I/O ports. Two I/O ports contribute keyboard scanning, and the



1. Intelligent terminals, built with Intel's iAPX 88/10 (8088) microprocessor and new 8276 small-system CRT controller, take this basic configuration to reduce parts count and minimize overhead on the system CPU.

Thomas Rossi, Applications Mgr. Peripheral Components
Intel Corp.
3065 Bowers Ave., Santa Clara, CA 95051

third port senses option-switch settings and the vertical-retrace signal from the 8276 (for CRT synchronization upon reset).

The CRT dot and character timing is generated by an 8284A clock generator. Another 8253 timer provides the appropriate horizontal-retrace timing for the CRT monitor. In its programmable one-shot mode, this timer generates a 32-μs horizontal-retrace pulse for the CRT monitor (Ball Brothers TV-12). A simple user-initiated change in the software will modify this delay time to suit different CRT monitors. The third and last timer in the 8253 is available for any user-defined need.

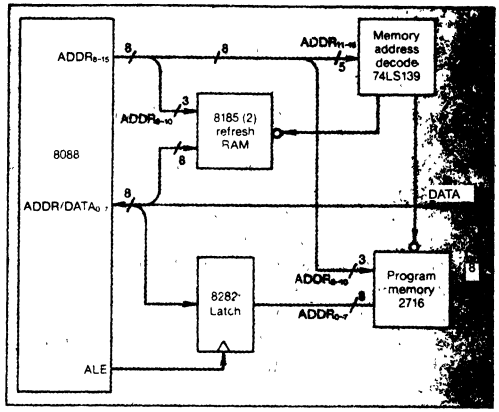
A 2716 EPROM on the controller board serves as a user-programmable character generator. A shift register transforms the data from the character EPROM into a serial-bit stream to illuminate dots on the CRT screen. The 2716 character generator helps to create special symbols and characters for word processing, industrial-control applications, or foreign-language displays.

The controller hardware is divided into processor and support, serial and parallel I/O, and CRT-control sections. The processor and support section consists of an iAPX 88/10 microprocessor, which is supported by two 8185 1-kbit × 8 static-RAM devices, and another 2716 EPROM (containing 2 kbytes of control firmware). The iAPX 88/10 uses a 15-MHz crystal (with an 8284A) to operate at a 5-MHz clock rate. The 8185 memories attach directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A_0-A_7) from the multiplexed bus for 2716-program memory access (Fig. 2).

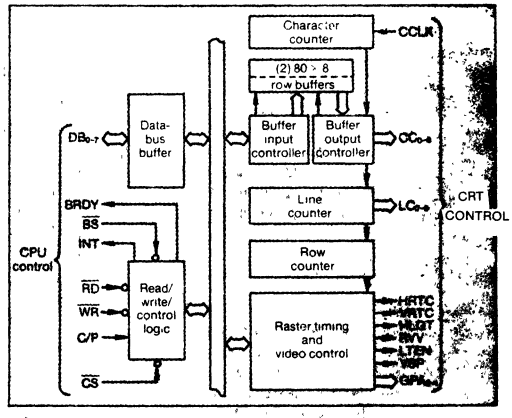
The serial and parallel I/O section of the terminal includes the 8255A programmable peripheral interface, and the CRT section contains the 8276 CRT controller and support circuits. All of the controller's I/O operations are memory mapped (see table).

Memory map of controller I/O operations

Address range	Selected device	Comments
00000 - 00003	RAM	Interrupt vector
00004 - 00029	RAM	Stack, local variables
00030 - 007FF	RAM	Display buffer
01000 - 01001	8276 $\overline{CS}$	8276 command/status
01900	8276 $\overline{BS}$	8276 row buffers
12000 - 12001	8251A	Serial channel
14000 - 14003	8253	Baud-rate timer
18000 - 18003	8255A	Keyboard, switches
FF800 - FFFFF	2716	Program storage



2. The processor and support section of the intelligent terminal's hardware contains two 8185 RAMs attached directly to the iAPX 88/10 multiplexed bus. An 8282 latches eight address lines (A_0-A_7) from the multiplexed bus for 2716-program memory access.



3. Here are the major functional blocks of the 8276 programmable CRT controller. This device permits software specification of most CRT-screen format characteristics (cursor position, characters/row, rows/frame).

How the controller board communicates

The CRT-controller board communicates to computer systems and other CRT units through a serial interface. Both RS-232C and TTL-compatible interfaces are available at the J_1 connector. The unit's standard software supports eight data-transmission rates: 9600, 4800, 2400, 1200, 600, 300, 150, and 110 baud. These rates are switch-selectable on the PC board. Since the baud-rate clock is generated by an 8253, baud rates may be easily modified in software.

Keyboard scanning is supported through the A and B ports of a 8255A programmable peripheral interface. Therefore, low-cost unencoded keyboards can be used. The eight scan lines (port B) and eight return lines (port A) support a 64-contact closure-key matrix. The three switches attached to port C permit baud-rate selection. Four general-purpose

inputs on port C permit the software to sense depression of the caps-lock key, the control key, and the shift key, as well as the position of the line/local switch. The last input on port C senses the status of the vertical retrace (VRTC) output of the 8276, so that the controller can synchronize with the CRT display on power up or after a hardware reset.

All keyboard I/O connects to the terminal board by means of a 40-pin header on its edge. All seven option-switch inputs are also brought to the connector, so that option switches may be installed on the keyboard if desired.

Software specifies the screen format

The CRT display is controlled by the 8276 programmable CRT controller (Fig. 3). With this device, most CRT screen-format characteristics—such as the cursor position, the number of characters per row, and the number of rows per frame—can be specified through software. The 8276 handles all display timing including retrace time delays.

In the current design, 2000 characters are displayed on the CRT screen (25 rows of 80 characters). Each character is formed as a 5 × 7-dot matrix within a larger 7 × 10 matrix (Fig. 4). Other screen formats (e.g., 16 rows of 64 characters) can be easily implemented with a few software changes and no hardware changes.

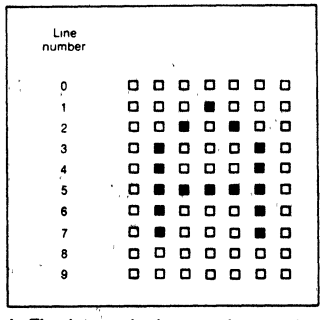
The 8276 contains two 80-character row buffers (see "Row Buffers Reduce System Overhead"). While one buffer displays the current character line on the screen, the 8276 fills the other row buffer from memory. This data transfer begins when the 8276 issues a data request (by means of the BRDY pin), causing an interrupt to the CPU. In response to this interrupt, the CPU activates the RAM's CS and RD inputs, while simultaneously activating the 8276 BS and WR inputs (Fig. 5). Through this technique, a single bus cycle suffices to transfer each byte from the RAM into the CRT row buffer. After the row buffer is filled, the CPU exits the interrupt-service routine.

But the 8276 can do more than simply paint characters on a CRT screen. Its end-of-row-stop buffer-loading code allows the control software to blank individual display lines. Also, the end-of-the-screen-stop buffer-loading code initiates an erase to the end of the screen.

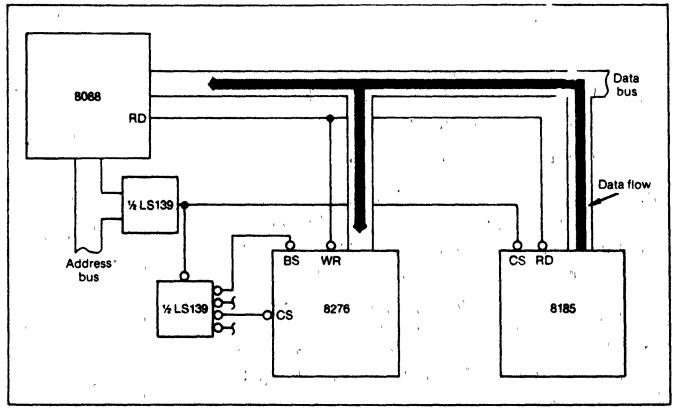
The 8276 supports software selection of visible-field "attributes" that can blink, underline, or highlight (intensify) characters on the screen and can reverse the video-character fields (black letters on a white background). Two general-purpose attribute outputs are provided to control the user-defined display capabilities.

Hardware provides three support functions

The 8276 is supported by three hardware functions: a dot/character-clock oscillator, an EPROM character generator, and a character-shift register (Fig. 6). The dot/character-clock oscillator consists of an 8284A operating at 11.34 MHz and providing an 88.2-ns dot clock. A 74LS163 divides this clock by 7 to generate a 1.62-MHz (617-ns) character clock.



4. The dot-matrix character font used in the low-cost CRT controller creates a 5 × 7 character in a 7 × 10 matrix (example shown is an upper-case A). Top and bottom lines are blanked for character separation, and the remaining line is reserved for cursor/underline display.



5. Row-buffer loading for the 8276 begins when a single 8088 string instruction moves data bytes from the 8185 RAM to the 8276 row buffer. The 8088 CPU "thinks" it is loading the AX register.

The 8276 is programmed to display one raster line every 61.7 μs—a complete character line every 617 μs (ten raster lines). The 8276 is also programmed to refresh the screen every 16.7 ms (60 Hz).

Each character row consists of ten raster lines. Seven lines display the 5 × 7-character matrix, two lines are blanked for row spacing, and one line displays the cursor and underline.

The 8276 uses the line count (LC_0-LC_3) outputs to indicate the current raster line during the display of each character. These outputs, combined with the character-code outputs (CC_0-CC_6), are sent to the 2716, which generates the dot pattern for display. This dot pattern is loaded into the shift register and is serially clocked for display by the 11.34-MHz dot clock.

During the vertical-retrace interval, the row buffer for the first line of the next frame is loaded by the iAPX 88/10. When the frame starts, the 8276 outputs the first character on its CC_0-CC_6 pins; the LC outputs are all zero. Exactly 617 ns later, the next character code is emitted by the 8276. This process continues every 617 ns until all 80 characters have been output. Then the 8276 generates a horizontal-retrace pulse, which is converted to the appropriate pulse width for the CRT monitor by the 8253.

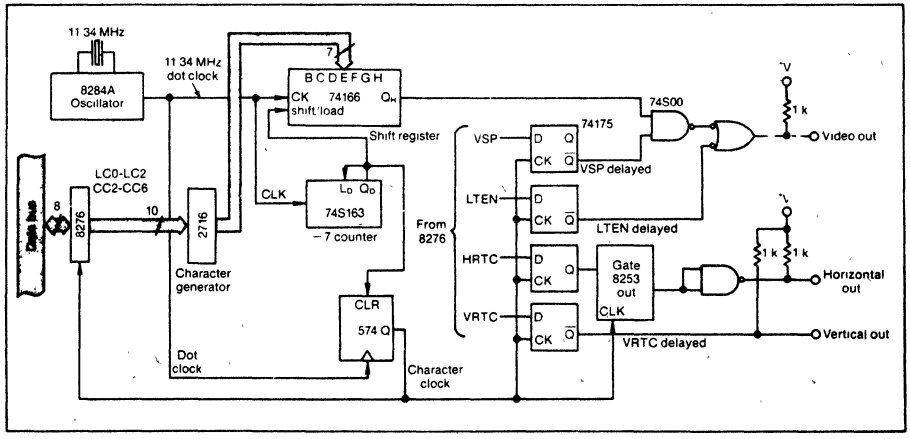
At the end of the first raster line, the 8276 increments the LC outputs. The next nine raster lines are similar to the first—the 8276 outputs the same 80 character codes on the CC_0-CC_6 pins for each of the raster lines, and the LC outputs are incremented after each raster line.

While the ten raster lines are being displayed, the 8276 is also filling the next row buffer. After the tenth raster line is completed, the 8276 resets the LC count and outputs character codes for the second row on the CC_0-CC_6 pins. As this row is displayed, the first row buffer is filled with information for the third row. The 8276 alternates row buffers until all 25 rows are displayed. At this time, the vertical-retrace signal is activated, and the scanning process is repeated for the next frame.

During display, the 8276 automatically activates the video-suppress pin (VSP) and/or light-enable outputs (LTEN), as appropriate, to control retrace blanking, generate the cursor, or underline characters.

Software is split between two priorities

The software for the CRT controller is divided into high and low-priority sections. The high-priority "foreground" software is activated each time the 8276 requests (through the iAPX 88/10 NMI interrupt) that an 80-character row buffer be filled. The 8276 row buffer is filled by performing 80 sequential memory reads. As each read is performed, the



6. CRT control logic supports the 8276. Three hardware functions are involved: a dot/character clock oscillator, an EPROM character generator, and a character-shift register.

Row buffers reduce system overhead

If no row buffer is present, the CRT controller must go to main memory to fetch every character during every dot scan line. Thus, the central processing unit is forced to relinquish the system bus 90 to 95% of the time. That CPU inactivity (overhead) greatly degrades total system performance and efficiency. CRT terminals using this approach are typically limited to between 1200 and 2400 baud on their serial-communications channels.

However, with the 8276's row-buffered architecture, the CRT controller need only access the main memory once for each displayed character row. This approach reduces system bus overhead for CRT refreshing to 25% (maximum). The CPU is then free to perform other local-processing functions, for instance, processing data at 9600 baud on a serial-communications channel.

```
        PUSHF           ; save registers
        PUSH   SI       ; used by
        PUSH   CX       ; subroutine

        MOV    SI,CURAD ; point to current line
        ADD    SI, OFFSET ;
        CLD             ; auto increment
        MOv    CX,40
REP     LODS   WDPTR    ; move 40 words
        CMP    SI, LAST ; check for end of screen
        JNZ    KTPK     ; jump if not at end
        MOv    SI,TOPDIS ; end-set to top
KTPK    MOv    CURAD,SI

        POP    CX       ; restore
        POP    SI       ;
        POPF            ;
```

7. A screen-refresh routine illustrates how the iAPX 88/10 load-string (LODS) instruction fills an 8276 row buffer. The 15 lines take 167 μs and are run every ten CRT lines (every 617 μs).

```
XOR     AX, AX          ; clear AX
MOV     BX, ESCTBL      ; load table, pointer
MOV     AL, USCHR       ; read character
CMP     AL, 41H         ; check for 41H
JL      SETUP           ; not valid
CMP     AL, 48H         ; check for 48H
JG      SETUP           ; not valid
XLAT                    ; translate to routine
                          address
JMP     (AX)
```

8. This routine checks the keyboard character to see if it is a valid escape-sequence command (41H through 48H). If the character is valid, a translate table jumps to a service routine. With the powerful iAPX 88/10 translate instruction, the service routine takes just 7 μs.

hardware automatically sends a write (over buffer-select and write pins) to the 8276.

The simultaneous memory-read and 8276-write commands transfer characters from the 8185 RAM to the 8276 in a single memory cycle—without a direct-memory-access (DMA) controller. The 80 reads are under the control of the CPU load string (LODS) instruction, which handles 40 word loads with iAPX 88/10 code (Fig. 7). The complete refresh sequence for one line requires approximately 167 μs. As a result, processor overhead for refresh operations is approximately 27%.

Foreground software also involves keyboard scanning that is performed only at the end of each display frame (after 25 rows or 16.7 ms). If a key depression is noted during one of these scans, the information is stored for further background processing. An iAPX 88/10 routine checks the character to determine whether it is a valid escape-character command (Fig. 8). In this procedure, the iAPX 88/10's translate instruction (XLAT) takes care of table lookup.

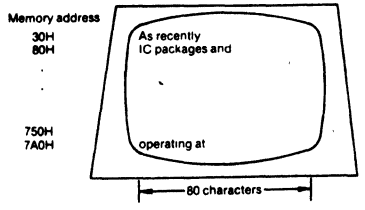
The low-priority software section handles "background" processing. It monitors the 8251A serial I/O port and provides processing for characters entered via the keyboard or with the serial interface. Background software executes continuously except when interrupted for the higher-priority foreground processing.

Cumbersome scrolling technique avoided

A refresh-buffer memory stores all 2000 characters that can be displayed on the CRT screen. The foreground software transfers one row (of 80 characters) at a time to the 8276. Two pointers are employed during normal operation. Under the control of foreground processing software, the current-row pointer contains the address of the next row to be displayed. This pointer must always be correct, so that a row can be transferred to the 8276 when requested. The buffer pointer contains the address of the next CRT buffer location to be written into (from either the keyboard or the serial port). Controlled by the background software, the buffer pointer indicates the cursor's actual location.

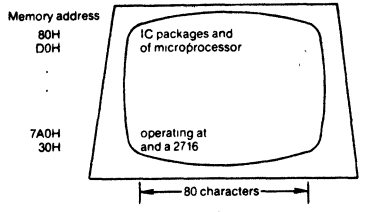
The simplest refresh-buffer organization associates the first memory address with the upper left position on the CRT screen. All other characters are stored sequentially (Fig. 9). But this method makes CRT screen scrolling difficult. Scrolling requires that each display line be moved up one row. The top line of the CRT is lost, the bottom line is blanked, and the cursor is placed at the beginning of the bottom line.

With this fixed sequential organization, all characters in the refresh buffer must be moved forward



Memory address	ASCII	Hex	Notes
30H	A	41	First character, first row
31H	s	73	
32H	space	20	
80H	I	49	First character, second row
81H	C	43	
7FFH	space	20	Last screen character

9. This memory/screen-character relationship exists when all characters are stored sequentially, making scrolling difficult.



Memory address	ASCII	Hex	Notes
30H	a	61	First character, 25th row
31H	n	6E	
32H	d	64	
80H	I	49	First character, first row
81H	C	43	
7FFH	space	20	Last character first row

10. If sequential memory orientation is retained but characters do not have to be moved in memory, scrolling can be much more efficient. Here, scrolling is accomplished simply by changing the display-start pointer. The memory/screen-character relationship is shown after a scroll of one line from the positions illustrated in Fig. 9.

by 80 characters (memory locations) to scroll the screen. (Each line moves up one row on the CRT and the last 80 characters in the buffer are blanked.) Moving 1920 characters each time the screen scrolls a single line is very slow and cumbersome.

The low-cost CRT controller avoids this problem with a slight modification of the fixed-sequential scrolling technique. Here, sequential memory orientation is retained while the need to move characters in memory is eliminated. This approach requires an additional display-start pointer that points to the memory location of the first character to be displayed.

At system initialization, the display-start pointer is set to 30H, the buffer-start address. During each vertical-retrace interval, the current-row pointer is initialized from the display-start pointer. Scrolling is performed by merely changing the display-start pointer.

For a single row scroll, the display-start pointer moves ahead 80 characters to location 80H, and the first 80 characters in the buffer are blanked. During the next vertical retrace, the foreground software sets the current-row pointer to the display-start location (80H), and begins transferring characters to the 8276 from this address.

The character in memory-location 80H (previously the first character in the second row) now occupies the first display position on the CRT screen (first character of the first row). When the foreground software reaches the end of the display buffer, the next row is read from the beginning of the buffer (location 30H). Thus, the first 80 characters in the buffer appear on the last display row (Fig. 10).

Each subsequent scroll moves the display start pointer forward by 80 characters. Buffer operations automatically "roll over" to the physical beginning of the buffer after passing the last buffer location.

Since the row-by-row character display is controlled by iAPX 88/10 software, other display techniques may be used. In particular, a linked list structure is extremely adaptable to word-processing and text-editing functions. This method allows each row within a file to be changed independently of other rows.

Because the rows are linked or "chained together" by pointers, rows may be easily inserted or deleted by simply changing pointers. To display a CRT frame, the processor simply follows the pointer chain from one row to the next.□

How useful?	Circle
Immediate design application	547
Within the next year	548
Not applicable	549

8275
PROGRAMMABLE CRT CONTROLLER

- **Programmable Screen and Character Format**
- **6 Independent Visual Field Attributes**
- **11 Visual Character Attributes (Graphic Capability)**
- **Cursor Control (4 Types)**
- **Light Pen Detection and Registers**
- **MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible**
- **Dual Row Buffers**
- **Programmable DMA Burst Mode**
- **Single +5V Supply**

The Intel® 8275 Programmable CRT Controller is a single chip device to interface CRT raster scan displays with Intel® microcomputer systems. It is manufactured on Intel's advanced NMOS process. Its primary function is to refresh the display by buffering the information from main memory and keeping track of the display position of the screen. The flexibility designed in the 8275 will allow simple interface to almost any raster scan CRT display with a minimum of external hardware and software overhead.

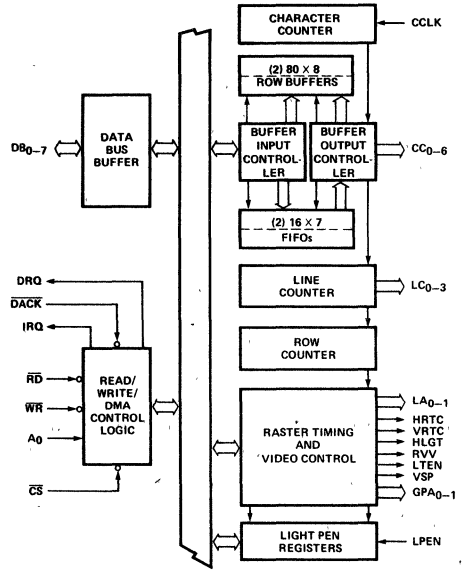


Figure 1. Block Diagram

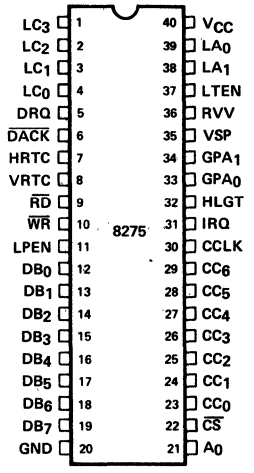


Figure 2. Pin Configuration

APRIL 1982
ORDER NUMBER: 210464-001

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**Line Count:** Output from the line counter which is used to address the character generator for the line positions on the screen.
DRQ	5	O	**DMA Request:** Output signal to the 8257 DMA controller requesting a DMA cycle.
$\overline{DACK}$	6	I	**DMA Acknowledge:** Input signal from the 8257 DMA controller acknowledging that the requested DMA cycle has been granted.
HRTC	7	O	**Horizontal Retrace:** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**Vertical Retrace:** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**Read Input:** A control signal to read registers.
$\overline{WR}$	10	I	**Write Input:** A control signal to write commands into the control registers or write data into the row buffers during a DMA cycle.
LPEN	11	I	**Light Pen:** Input signal from the CRT system signifying that a light pen signal has been detected.
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**Bi-Directional Three-State Data Bus Lines:** The outputs are enabled during a read of the C or P ports.
Ground	20		**Ground.**

Symbol	Pin No.	Type	Name and Function
V_{CC}	40		**+5V Power Supply.**
LA_0 LA_1	39 38	O	**Line Attribute Codes:** These attribute codes have to be decoded externally by the dot/timing logic to generate the horizontal and vertical line combinations for the graphic displays specified by the character attribute codes.
LTEN	37	O	**Light Enable:** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**Reverse Video:** Output signal used to indicate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**Video Suppression:** Output signal used to blank the video signal to the CRT. This output is active: —during the horizontal and vertical retrace intervals. —at the top and bottom lines of rows if underline is programmed to be number 8 or greater. —when an end of row or end of screen code is detected. —when a DMA underrun occurs. —at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for character and field attributes)—to create blinking displays as specified by cursor, character attribute, or field attribute programming.
GPA_1 GPA_0	34 33	O	**General Purpose Attribute Codes:** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**Highlight:** Output signal used to intensify the display at particular positions on the screen as specified by the character attribute codes or field attribute codes.
IRQ	31	O	**Interrupt Request.**
CCLK	30	I	**Character Clock (from dot/timing logic).**
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**Character Codes:** Output from the row buffers used for character selection in the character generator.
CS	22	I	**Chip Select:** The read and write are enabled by CS.
A_0	21	I	**Port Address:** A high input on A_0 selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8275 to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

A_0	OPERATION	REGISTER
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	1	0	0	Write 8275 Parameter
0	0	1	0	Read 8275 Parameter
1	1	0	0	Write 8275 Command
1	0	1	0	Read 8275 Status
X	1	1	0	Three-State
X	X	X	1	Three-state

$\overline{RD}$ (Read)

A "low" on this input informs the 8275 that the CPU is reading data or status information from the 8275.

$\overline{WR}$ (Write)

A "low" on this input informs the 8275 that the CPU is writing data or control words to the 8275.

$\overline{CS}$ (Chip Select)

A "low" on this input selects the 8275. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus in the float state and $\overline{RD}$ and $\overline{WR}$ will have no effect on the chip.

DRQ (DMA Request)

A "high" on this output informs the DMA Controller that the 8275 desires a DMA transfer.

$\overline{DACK}$ (DMA Acknowledge)

A "low" on this input informs the 8275 that a DMA cycle is in progress.

IRQ (Interrupt Request)

A "high" on this output informs the CPU that the 8275 desires interrupt service.

FUNCTIONAL DESCRIPTION

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be a derivative of the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Sweeps) per character row. Its outputs are used to address the external character generator ROM.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Light Pen Registers

The Light Pen Registers are two registers that store the contents of the character counter and the row counter whenever there is a rising edge on the LPEN (Light Pen) input.

Note: Software correction is required.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of LA_{0-1} (Line Attribute), HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80 character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

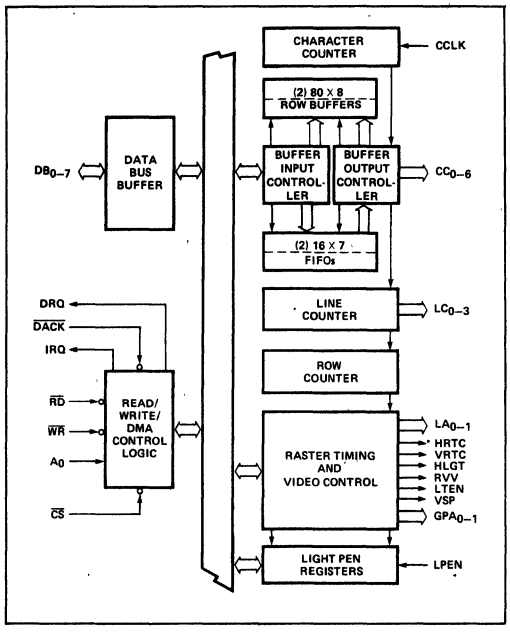


Figure 3. 8275 Block Diagram Showing Counter and Register Functions

FIFOs

There are two 16 character FIFOs in the 8275. They are used to provide extra row buffer length in the Transparent Attribute Mode (see Detailed Operation section).

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a character attribute, field attribute or special code, these controllers control the appropriate action. (Examples: An "End of Screen–Stop DMA" special code will cause the Buffer Input Controller to stop further DMA requests. A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)

SYSTEM OPERATION

The 8275 is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding, cursor timing, and light pen detection.

It is designed to interface with the 8257 DMA Controller and standard character generator ROMs for dot matrix decoding. Dot level timing must be provided by external circuitry.

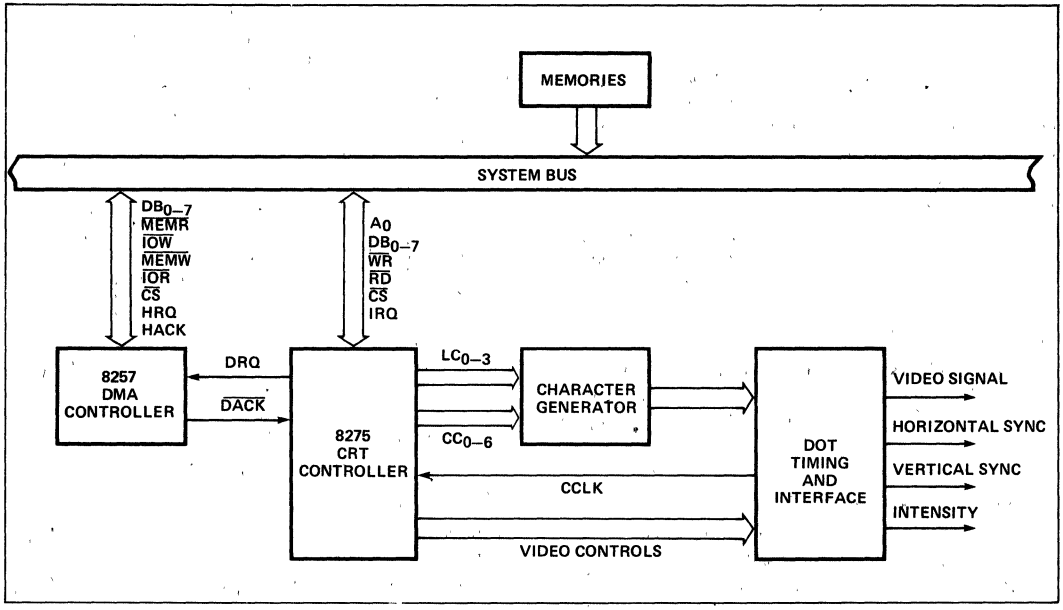


Figure 4. 8275 Systems Block Diagram Showing Systems Operation

General Systems Operational Description

The 8275 provides a "window" into the microcomputer system memory.

Display characters are retrieved from memory and displayed on a row by row basis. The 8275 has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8275 requests DMA to fill the row buffer that is not being used for display. DMA burst length and spacing is programmable. (See Programming Section.)

The 8275 displays character rows one line at a time. The number of lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8275 provides special Control Codes which can be used to minimize DMA or software overhead. It also provides Visual Attribute Codes to cause special action or symbols on the screen without the use of the character generator (see Visual Attributes Section).

The 8275 also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is programmable.

The 8275 can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

The 8275 has a light pen input and registers. The light pen input is used to load the registers. Light pen registers can be read on command. (See Programming Section.)

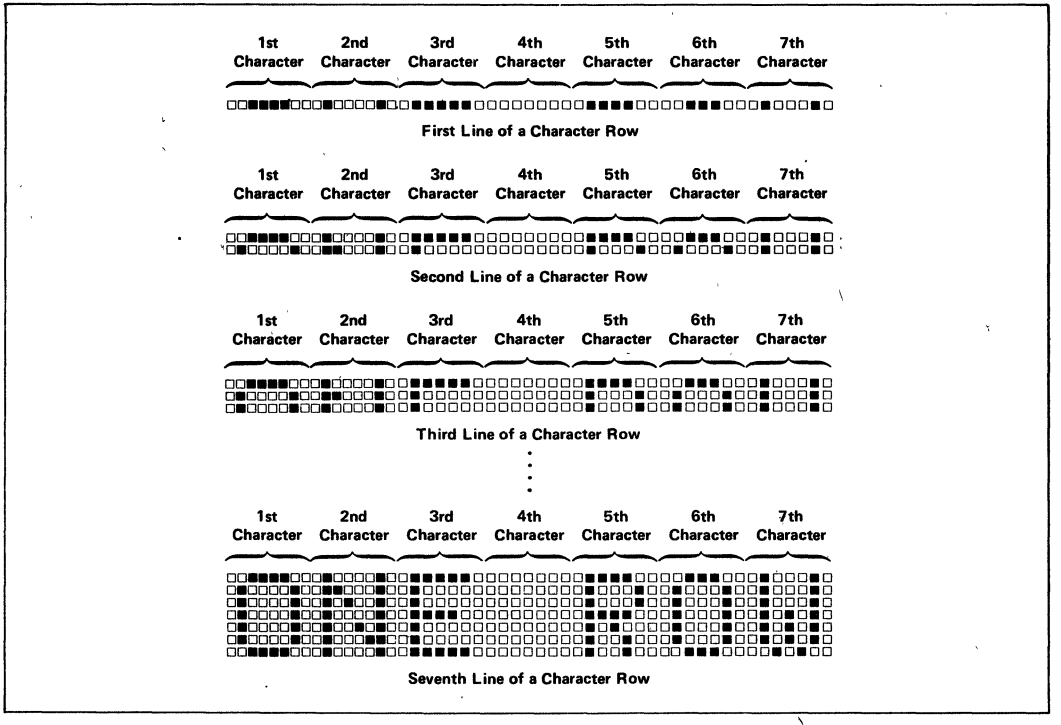


Figure 5. Display of a Character Row

Display Row Buffering

Before the start of a frame, the 8275 requests DMA and one row buffer is filled with characters.

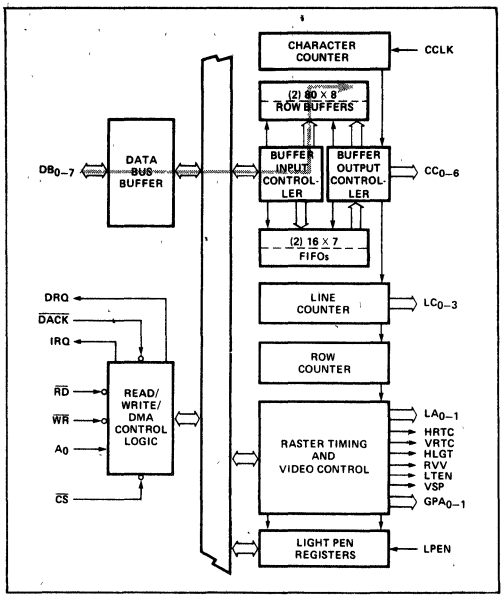


Figure 6. First Row Buffer Filled

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, DMA begins filling the other row buffer with the next row of characters.

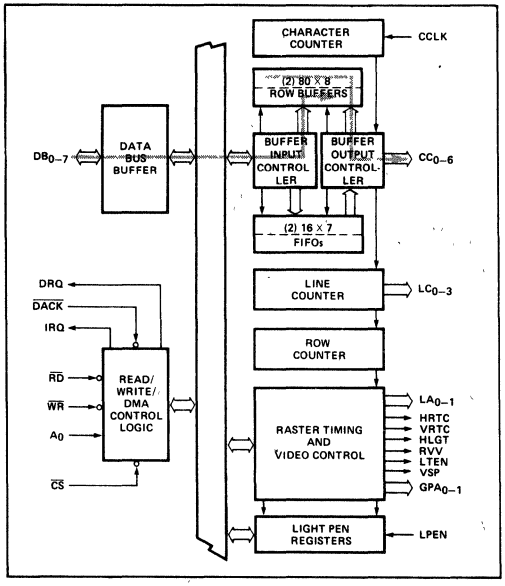


Figure 7. Second Buffer Filled, First Row Displayed

After all the lines of the character row are scanned, the roles of the two row buffers are reversed and the same procedure is followed for the next row.

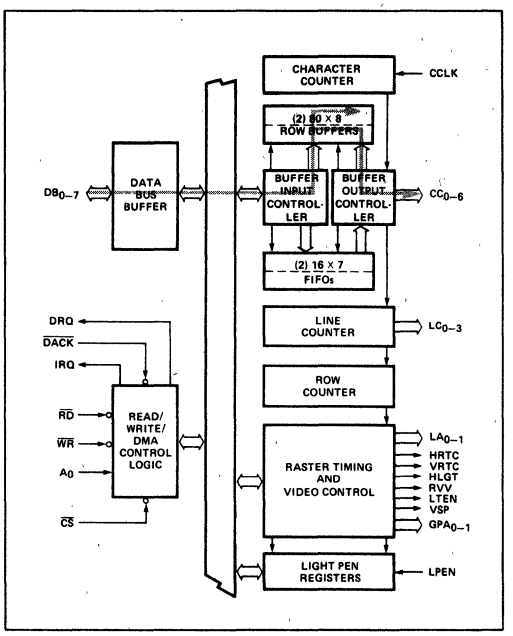


Figure 8. First Buffer Filled with Third Row, Second Row Displayed

This is repeated until all of the character rows are displayed.

Display Format

Screen Format

The 8275 can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

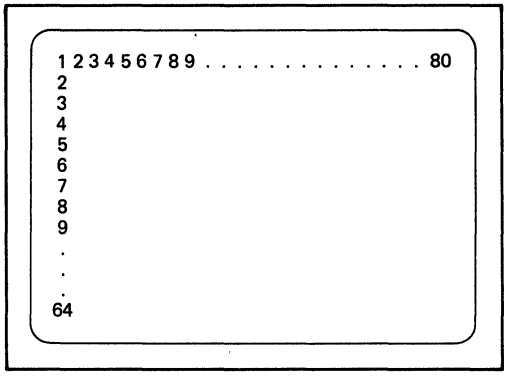


Figure 9. Screen Format

The 8275 can also be programmed to blank alternate rows. In this mode, the first row is displayed, the second blanked, the third displayed, etc. DMA is not requested for the blanked rows.

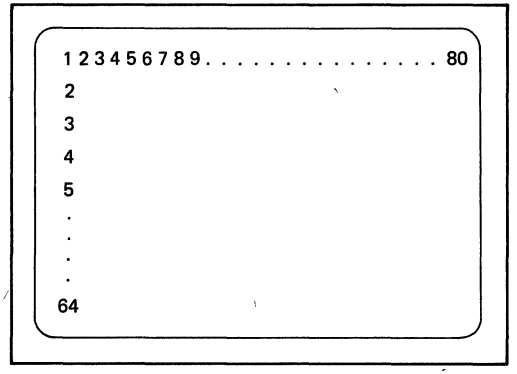


Figure 10. Blank Alternate Rows Mode

Row Format

The 8275 is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the whole character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number*.

In mode 1, the line *counter* is offset by one from the line *number*.

Note: In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

Line Number		Line Counter Mode 0	Line Counter Mode 1
0	□ □ □ □ □ □ □ □ □	0000	1111
1	□ □ □ □ ■ □ □ □ □	0001	0000
2	□ □ □ ■ □ ■ □ □ □	0010	0001
3	□ □ ■ □ □ □ ■ □ □	0011	0010
4	□ ■ □ □ □ □ □ ■ □	0100	0011
5	□ ■ □ □ □ □ □ ■ □	0101	0100
6	□ ■ ■ ■ ■ ■ ■ ■ □	0110	0101
7	□ ■ □ □ □ □ □ ■ □	0111	0110
8	□ ■ □ □ □ □ □ ■ □	1000	0111
9	□ ■ □ □ □ □ □ ■ □	1001	1000
10	□ □ □ □ □ □ □ □ □	1010	1001
11	□ □ □ □ □ □ □ □ □	1011	1010
12	□ □ □ □ □ □ □ □ □	1100	1011
13	□ □ □ □ □ □ □ □ □	1101	1100
14	□ □ □ □ □ □ □ □ □	1110	1101
15	□ □ □ □ □ □ □ □ □	1111	1110

Figure 11. Example of a 16-Line Format

Line Number		Line Counter Mode 0	Line Counter Mode 1
0	□ □ □ □ □ □ □	0000	1001
1	□ □ □ ■ □ □ □	0001	0000
2	□ □ ■ □ ■ □ □	0010	0001
3	□ ■ □ □ □ ■ □	0011	0010
4	□ ■ □ □ □ ■ □	0100	0011
5	□ ■ ■ ■ ■ ■ □	0101	0100
6	□ ■ □ □ □ ■ □	0110	0101
7	□ ■ □ □ □ ■ □	0111	0110
8	□ □ □ □ □ □ □	1000	0111
9	□ □ □ □ □ □ □	1001	1000

Figure 12. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

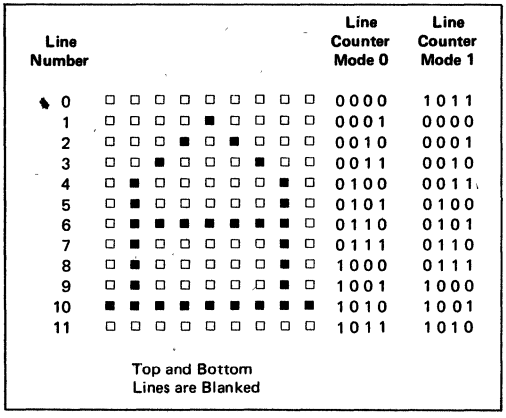


Figure 13. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

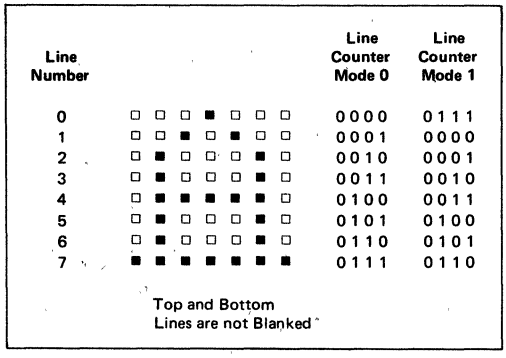


Figure 14. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

Dot Format

Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

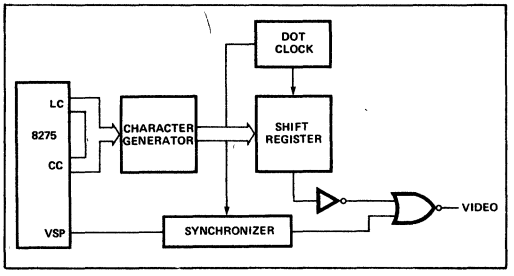


Figure 15. Typical Dot Level Block Diagram

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

Note: Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This is constantly repeated.

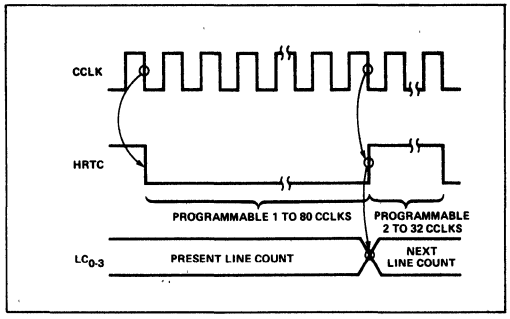


Figure 16. Line Timing

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

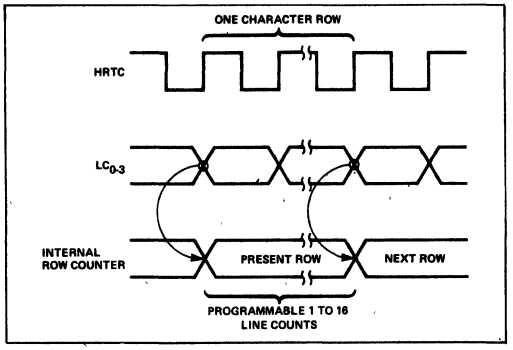


Figure 17. Row Timing

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

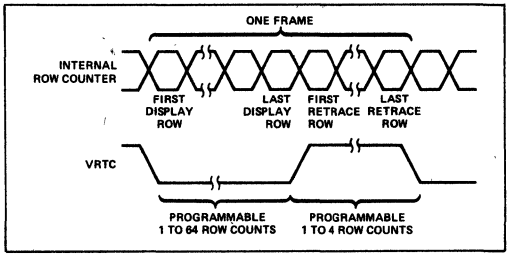


Figure 18. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.

DMA Timing

The 8275 can be programmed to request burst DMA transfers of 1 to 8 characters. The interval between bursts is also programmable (from 0 to 55 character clock periods ±1). This allows the user to tailor his DMA overhead to fit his system needs.

The first DMA request of the frame occurs one *row time* before the end of vertical retrace. DMA requests continue as programmed, until the row buffer is filled. If the row buffer is filled in the middle of a burst, the 8275 terminates the burst and resets the burst counter. No more DMA requests will occur until the *beginning* of the *next* row. At that time, DMA requests are activated as programmed until the other buffer is filled.

The first DMA request for a row will start at the first character clock of the preceding row. If the burst mode is used, the first DMA request may occur a number of character clocks later. This number is equal to the programmed burst space.

If, for any reason, there is a DMA underrun, a flag in the status word will be set.

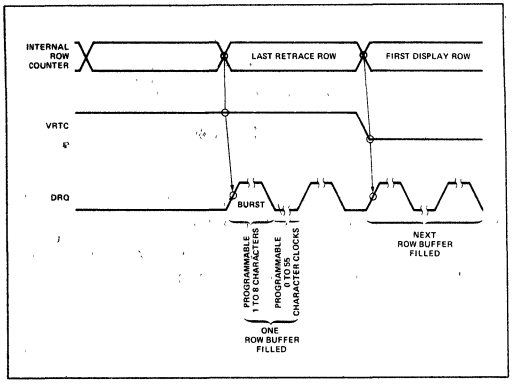


Figure 19. DMA Timing

The DMA controller is typically initialized for the next frame at the end of the current frame.

Interrupt Timing

The 8275 can be programmed to generate an interrupt request at the end of each frame. This can be used to reinitialize the DMA controller. If the 8275 interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row*.

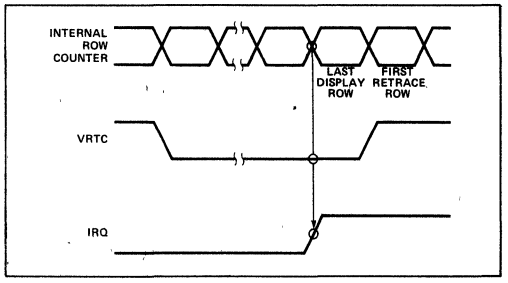


Figure 20. Beginning of Interrupt Request

IRQ will go inactive after the status register is read.

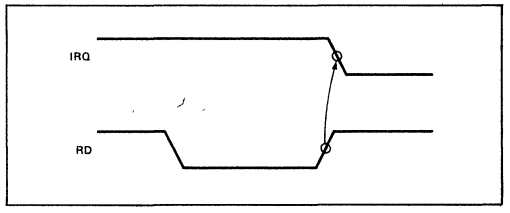


Figure 21. End of Interrupt Request

A reset command will also cause IRQ to go inactive, but this is not recommended during normal service.

Another method of reinitializing the DMA controller is to have the DMA controller itself interrupt on terminal count. With this method, the 8275 interrupt enable flag should not be set.

Note: Upon power-up, the 8275 Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8275 before system interrupts are enabled.

VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8275 are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Visual Attribute or Special Code (MSB = 1).

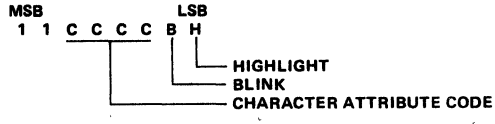
There are two types of Visual Attribute Codes. They are Character Attributes and Field Attributes.

Character Attribute Codes

Character attribute codes are codes that can be used to generate graphics symbols without the use of a character generator. This is accomplished by selectively activating the Line Attribute outputs (LA_{0-1}), the Video Suppression output (VSP), and the Light Enable output. The dot level timing circuitry can use these signals to generate the proper symbols.

Character attributes can be programmed to blink or be highlighted individually. Blinking is accomplished with the Video Suppression output (VSP). Blink frequency is equal to the screen refresh frequency divided by 32. Highlighting is accomplished by activating the Highlight output (HGLT).

Character Attributes



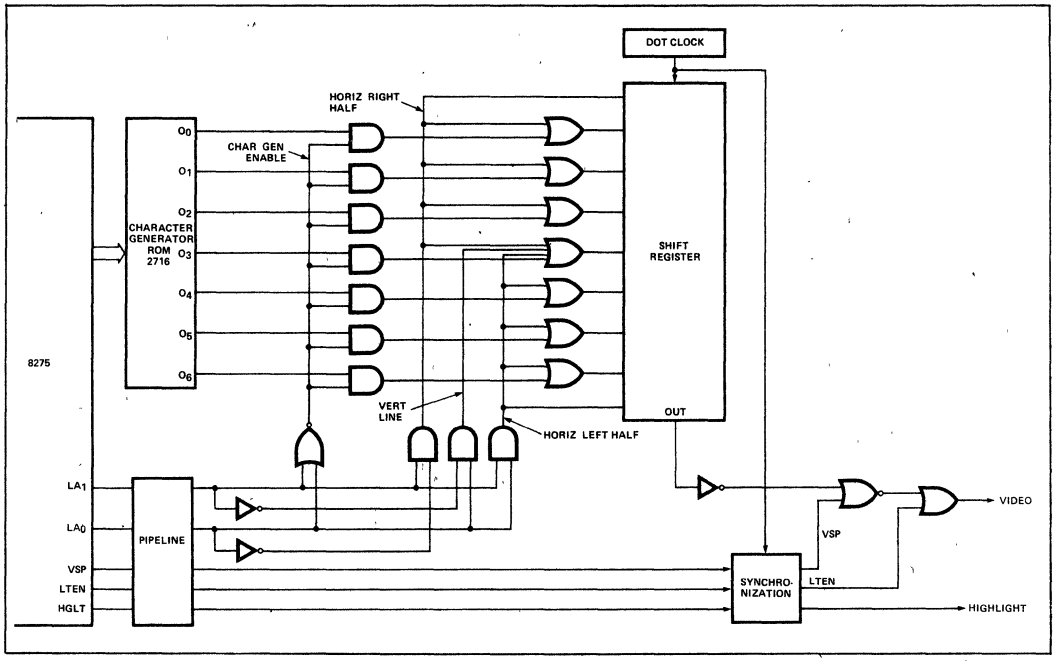
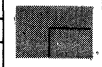
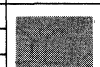
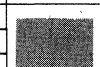


Figure 22. Typical Character Attribute Logic

Table 2. Character Attributes

Character attributes were designed to produce the following graphics:

CHARACTER ATTRIBUTE CODE "CCCC"		OUTPUTS LA_1	OUTPUTS LA_0	OUTPUTS VSP	OUTPUTS LTEN	SYMBOL	DESCRIPTION
0000	Above Underline	0	0	1	0		Top Left Corner
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0001	Above Underline	0	0	1	0		Top Right Corner
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0010	Above Underline	0	1	0	0		Bottom Left Corner
	Underline	1	0	0	0		
	Below Underline	0	0	1	0		
0011	Above Underline	0	1	0	0		Bottom Right Corner
	Underline	1	1	0	0		
	Below Underline	0	0	1	0		
0100	Above Underline	0	0	1	0		Top Intersect
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
0101	Above Underline	0	1	0	0		Right Intersect
	Underline	1	1	0	0		
	Below Underline	0	1	0	0		
0110	Above Underline	0	1	0	0		Left Intersect
	Underline	1	0	0	0		
	Below Underline	0	1	0	0		
0111	Above Underline	0	1	0	0		Bottom Intersect
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1000	Above Underline	0	0	1	0		Horizontal Line
	Underline	0	0	0	1		
	Below Underline	0	0	1	0		
1001	Above Underline	0	1	0	0		Vertical Line
	Underline	0	1	0	0		
	Below Underline	0	1	0	0		
1010	Above Underline	0	1	0	0		Crossed Lines
	Underline	0	0	0	1		
	Below Underline	0	1	0	0		
1011	Above Underline	0	0	0	0		Not Recommended *
	Underline	0	0	0	0		
	Below Underline	0	0	0	0		
1100	Above Underline	0	0	1	0		Special Codes
	Underline	0	0	1	0		
	Below Underline	0	0	1	0		
1101	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1110	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						
1111	Above Underline						Illegal
	Underline		Undefined				
	Below Underline						

*Character Attribute Code 1011 is not recommended for normal operation. Since none of the attribute outputs are active, the character Generator will not be disabled, and an indeterminate character will be generated.

Character Attribute Codes 1101, 1110, and 1111 are illegal.

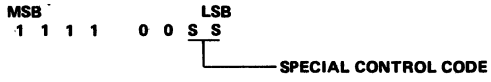
Blinking is active when B = 1.

Highlight is active when H = 1.

Special Codes

Four special codes are available to help reduce memory, software, or DMA overhead.

Special Control Character



S S	FUNCTION
0 0	End of Row
0 1	End of Row-Stop DMA
1 0	End of Screen
1 1	End of Screen-Stop DMA

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop DMA Code (01) causes the DMA Control Logic to stop DMA for the rest of the row when it is written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop DMA Code (11) causes the DMA Control Logic to stop DMA for the rest of the frame when it is written into the Row Buffer. It affects the display in the same way as the End of Screen Code (10).

If the Stop DMA feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

Note: If a Stop DMA character is not the last character in a burst or row, DMA is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop DMA character.

Field Attributes

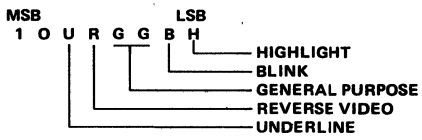
The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

There are six field attributes:

1. *Blink* — Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight* — Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video* — Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).
4. *Underline* — Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5,6. *General Purpose* — There are two additional 8275 outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

Field Attribute Code



H = 1 FOR HIGHLIGHTING
B = 1 FOR BLINKING
R = 1 FOR REVERSE VIDEO
U = 1 FOR UNDERLINE
GG = GPA_1, GPA_0

*More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

The 8275 can be programmed to provide visible or invisible field attribute characters.

If the 8275 is programmed in the visible field attribute mode, all field attributes will occupy a position on the screen. They will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

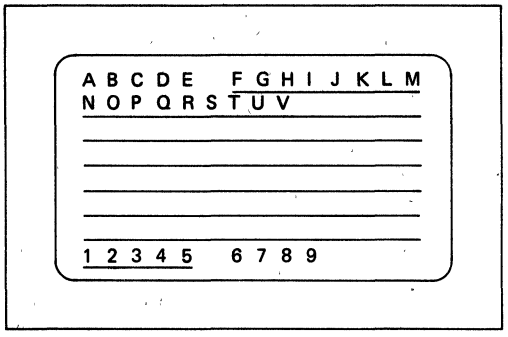


Figure 23. Example of the Visible Field Attribute Mode (Underline Attribute)

If the 8275 is programmed in the invisible field attribute mode, the 8275 FIFO is activated.

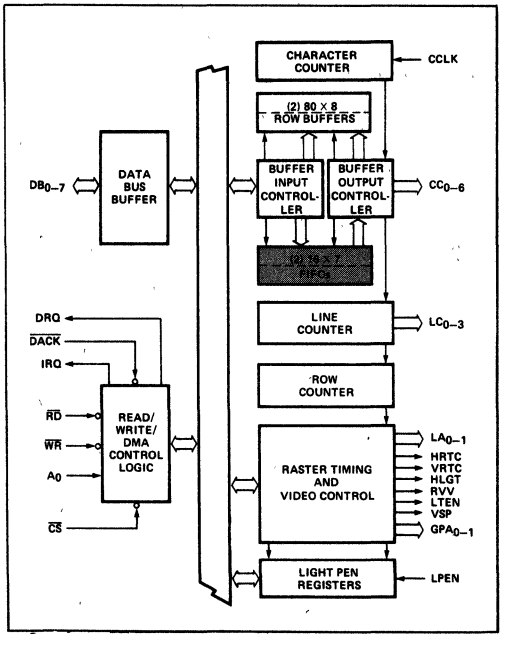


Figure 24. Block Diagram Showing FIFO Activation

Each row buffer has a corresponding FIFO. These FIFOs are 16 characters by 7 bits in size.

When a field attribute is placed in the row buffer during DMA, the buffer input controller recognizes it and places the *next* character in the proper FIFO.

When a field attribute is placed in the Buffer Output Controller during display, it causes the controller to immediately put a character from the FIFO on the Character Code outputs (CC_{0-6}). The chosen Visual Attributes are also activated.

Since the FIFO is 16 characters long, no more than 16 field attribute characters may be used per line in this mode. If more are used, a bit in the status word is set and the first characters in the FIFO are written over and lost.

Note: Since the FIFO is 7 bits wide, the MSB of any characters put in it are stripped off. Therefore, a Visual Attribute or Special Code must *not* immediately follow a field attribute code. If this situation does occur, the Visual Attribute or Special Code will be treated as a normal display character.

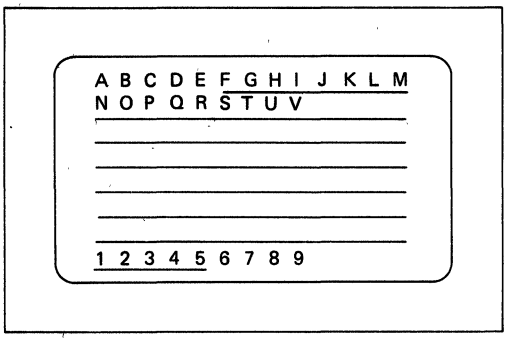


Figure 25. Example of the Invisible Field Attribute Mode (Underline Attribute)

Field and Character Attribute Interaction

Character Attribute Symbols are affected by the Reverse Video (RVV) and General Purpose (GPA_{0-1}) field attributes. They are not affected by Underline, Blink or Highlight field attributes; however, these characteristics can be programmed *individually* for Character Attribute Symbols.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field*, the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field*, the cursor will not be visible.

Light Pen Detection

A light pen consists of a micro switch and a tiny light sensor. When the light pen is pressed against the CRT screen, the micro switch enables the light sensor. When the raster sweep reaches the light sensor, it triggers the light pen output.

If the output of the light pen is presented to the 8275 LPEN input, the row and character position coordinates are stored in a pair of registers. These registers can be read on command. A bit in the status word is set, indicating that the light pen signal was detected. The LPEN input must be a 0 to 1 transition for proper operation.

Note: Due to internal and external delays, the character position coordinate will be off by at least three character positions. This has to be corrected in software.

Device Programming

The 8275 has two programming registers, the Command Register (CREG) and the Parameter Register (PREG). It also has a Status Register (SREG). The Command Register can only be written into and the Status Registers can only be read from. They are addressed as follows:

A_0	OPERATION	REGISTER
0	Read	PREG
0	Write	PREG
1	Read	SREG
1	Write	CREG

The 8275 expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

INSTRUCTION SET

The 8275 instruction set consists of 8 commands.

COMMAND	NO. OF PARAMETER BYTES
Reset	4
Start Display	0
Stop Display	0
Read Light Pen	2
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8275 (SREG) can be read by the CPU at any time.

1. Reset Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Reset Command	0	0	0	0	0	0	0	0
Parameters	Write	0	Screen Comp Byte 1	S	H	H	H	H	H	H	H
	Write	0	Screen Comp Byte 2	V	V	R	R	R	R	R	R
	Write	0	Screen Comp Byte 3	U	U	U	U	L	L	L	L
	Write	0	Screen Comp Byte 4	M	F	C	C	Z	Z	Z	Z

Action – After the reset command is written, DMA requests stop, 8275 interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up.

As parameters are written, the screen composition is defined.

Parameter – S Spaced Rows

S	FUNCTIONS
0	Normal Rows
1	Spaced Rows

Parameter – HHHHHHH Horizontal Characters/Row

H	H	H	H	H	H	H	NO. OF CHARACTERS PER ROW
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	2
0	0	0	0	0	1	0	3
			.				.
			.				.
			.				.
1	0	0	1	1	1	1	80
1	0	1	0	0	0	0	Undefined
			.				.
			.				.
			.				.
1	1	1	1	1	1	1	Undefined

Parameter – VV Vertical Retrace Row Count

V	V	NO. OF ROW COUNTS PER VRTC
0	0	1
0	1	2
1	0	3
1	1	4

Parameter – RRRRRR Vertical Rows/Frame

R	R	R	R	R	R	NO. OF ROWS/FRAME
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
			.			.
			.			.
			.			.
1	1	1	1	1	1	64

Parameter – UUUU Underline Placement

U	U	U	U	LINE NUMBER OF UNDERLINE
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		.		.
		.		.
		.		.
1	1	1	1	16

Parameter – LLLL Number of Lines per Character Row

L	L	L	L	NO. OF LINES/ROW
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		.		.
		.		.
		.		.
1	1	1	1	16

Parameter – M Line Counter Mode

M	LINE COUNTER MODE
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter – F Field Attribute Mode

F	FIELD ATTRIBUTE MODE
0	Transparent
1	Non-Transparent

Parameter – CC Cursor Format

C	C	CURSOR FORMAT
0	0	Blinking reverse video block
0	1	Blinking underline
1	0	Nonblinking reverse video block
1	1	Nonblinking underling

Parameter – ZZZZ Horizontal Retrace Count

Z	Z	Z	Z	NO. OF CHARACTER COUNTS PER HRTC
0	0	0	0	2
0	0	0	1	4
0	0	1	0	6
		.		.
		.		.
		.		.
1	1	1	1	32

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. Start Display Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Start Display	0	0	1	S	S	S	B	B
No parameters											

S S S BURST SPACE CODE

S S S	NO. OF CHARACTER CLOCKS BETWEEN DMA REQUESTS
0 0 0	0
0 0 1	7
0 1 0	15
0 1 1	23
1 0 0	31
1 0 1	39
1 1 0	47
1 1 1	55

B B BURST COUNT CODE

B B	NO. OF DMA CYCLES PER BURST
0 0	1
0 1	2
1 0	4
1 1	8

Action – 8275 interrupts are enabled, DMA requests begin, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. Stop Display Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Stop Display	0	1	0	0	0	0	0	0
No parameters											

Action – Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to re-enable the display.

4. Read Light Pen Command

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Read Light Pen	0	1	1	0	0	0	0	0
Parameters	Read	0	Char. Number	(Char. Position in Row)							
	Read	0	Row Number	(Row Number)							

Action – The 8275 is conditioned to supply the contents of the light pen position registers in the next two read cycles of the parameter register. Status flags are not affected.

Note: Software correction of light pen position is required.

5. Load Cursor Position:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Load Cursor	1	0	0	0	0	0	0	0
Parameters	Write	0	Char Number	(Char. Position in Row)							
	Write	0	Row Number	(Row Number)							

Action – The 8275 is conditioned to place the next two parameter bytes into the cursor position registers. Status flags not affected.

6. Enable Interrupt Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Enable Interrupt	1	0	1	0	0	0	0	0
No parameters											

Action – The interrupt enable status flag is set and interrupts are enabled.

7. Disable Interrupt Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Disable Interrupt	1	1	0	0	0	0	0	0
No parameters											

Action – Interrupts are disabled and the interrupt enable status flag is reset.

8. Preset Counters Command:

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB							LSB
Command	Write	1	Preset Counters	1	1	1	0	0	0	0	0
No parameters											

Action – The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

	OPERATION	A_0	DESCRIPTION	DATA BUS MSB LSB
Command	Read	1	Status Word	0 IE IR LP IC VE DU F0

IE – (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR – (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

LP – This flag is set when the light pen input (LPEN) is activated and the light pen registers have been loaded. This flag is automatically reset after a status read.

IC – (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE – (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

DU – (DMA Underrun) This flag is set whenever a data underrun occurs during DMA transfers. Upon detection of DU, the DMA operation is stopped and the screen is blanked until after the vertical retrace interval. This flag is reset after a status read.

FO – (FIFO Overrun) This flag is set whenever the FIFO is overrun. It is reset on a status read.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to +150°C
Voltage On Any Pin
With Respect to Ground . . . . . -0.5V to +7V
Power Dissipation . . . . . 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance		10	pF	f_c = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}.

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Bus Parameters

READ CYCLE

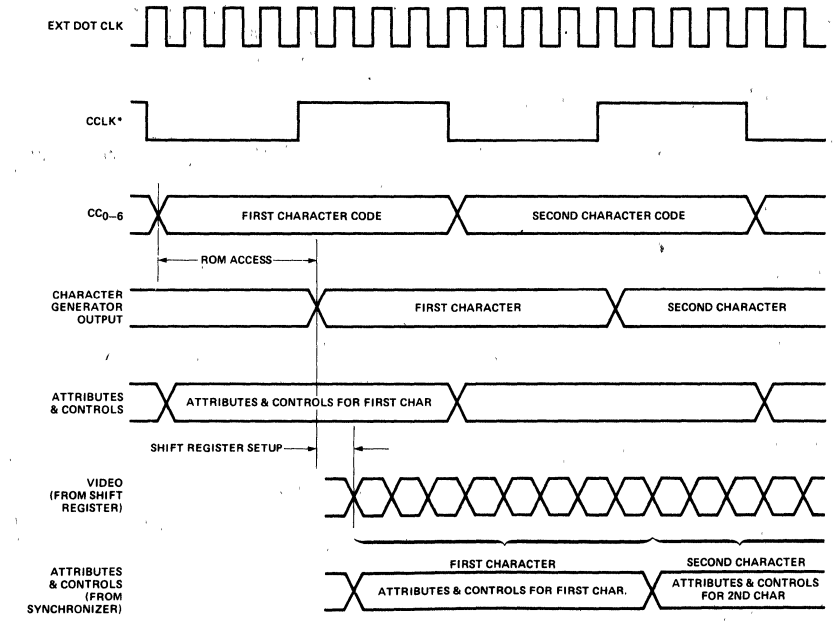
Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AR}	Address Stable Before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	C_L = 150 pF
t_{DF}	READ to Data Floating		100	ns	C_L = 150 pF

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AW}	Address Stable Before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

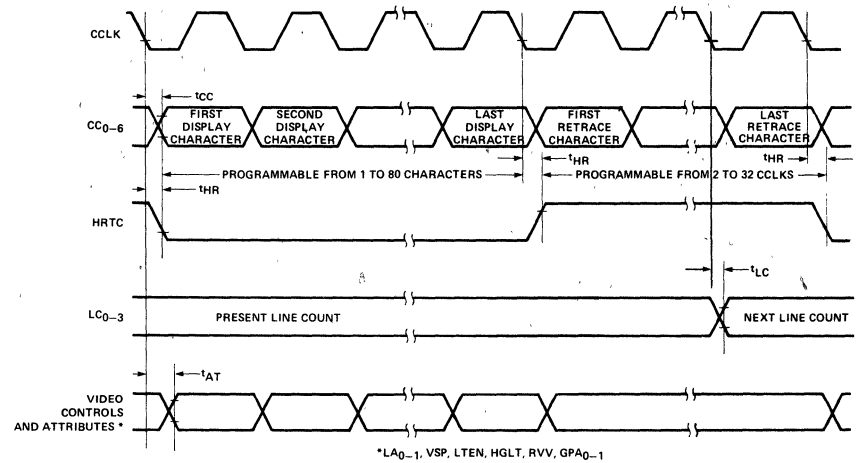
WAVEFORMS

TYPICAL DOT LEVEL TIMING



*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8275

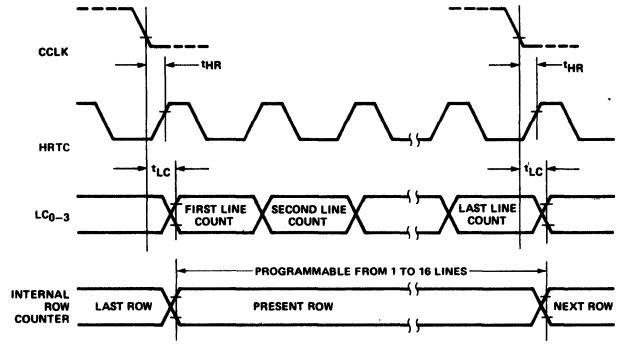
LINE TIMING



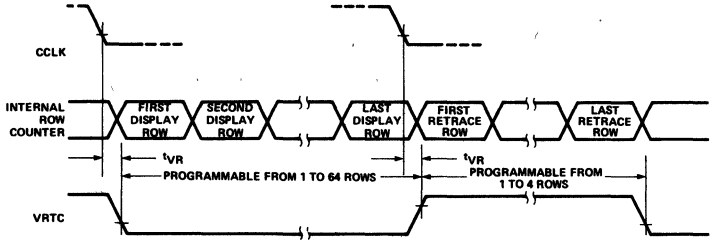
*LA0–1, VSP, LTEN, HGLT, RVV, GPA0–1

WAVEFORMS (Continued)

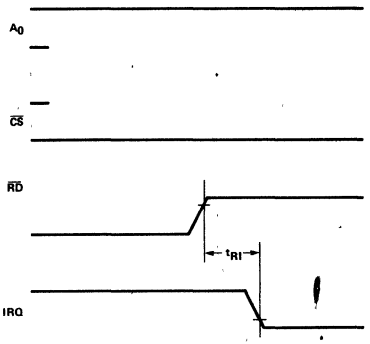
ROW TIMING



FRAME TIMING

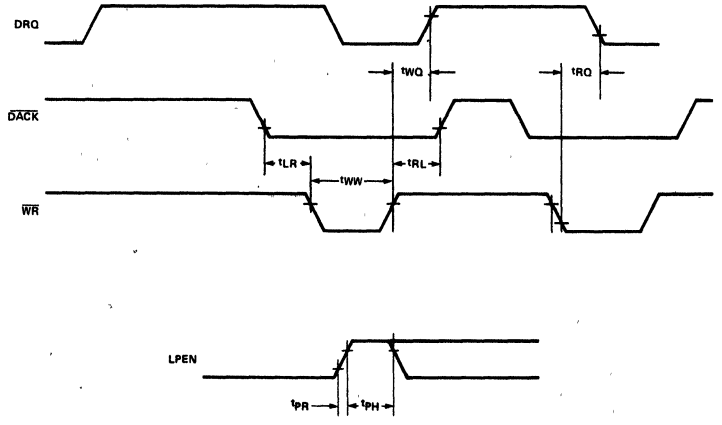


INTERRUPT TIMING

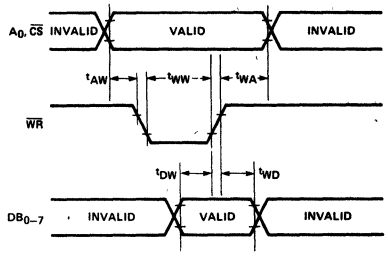


WAVEFORMS (Continued)

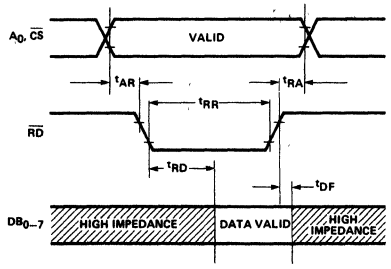
DMA TIMING



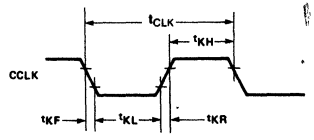
WRITE TIMING



READ TIMING



CLOCK TIMING



A.C. CHARACTERISTICS (Continued)

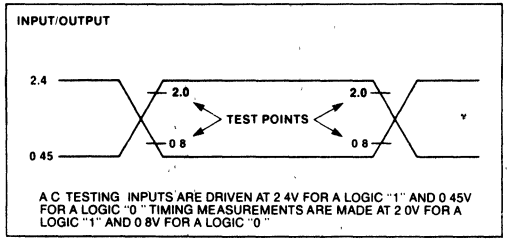
CLOCK TIMING

Symbol	Parameter	8275 Min.	8275 Max.	8275-2 Min.	8275-2 Max.	Units	Test Conditions
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

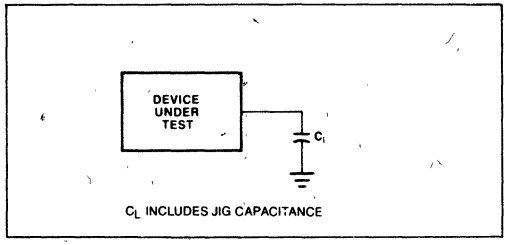
Symbol	Parameter	8275 Min.	8275 Max.	8275-2 Min.	8275-2 Max.	Units	Test Conditions
t_{CC}	Character Code Output Delay		150		150	ns	C_L = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	C_L = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	C_L = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	C_L = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	C_L = 50 pF
t_{RI}	IRQ↓ from RD↑		250		250	ns	C_L = 50 pF
t_{WQ}	DRQ↑ from WR↑		250		250	ns	C_L = 50 pF
t_{RQ}	DRQ↓ from WR↓		200		200	ns	C_L = 50 pF
t_{LR}	DACK↓ to WR↓	0		0		ns	
t_{RL}	WR↑ to DACK↑	0		0		ns	
t_{PR}	LPEN Rise		50		50	ns	
t_{PH}	LPEN Hold	100		100		ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0"

A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

8276
SMALL SYSTEM CRT CONTROLLER

- **Programmable Screen and Character Format**
- **6 Independent Visual Field Attributes**
- **Cursor Control (4 Types)**
- **MCS-51®, MCS-85®, iAPX 86, and iAPX 88 Compatible**
- **Dual Row Buffers**
- **Single +5V Supply**
- **40-Pin Package**
- **3 MHz Clock with 8276-2**

The Intel 8276 Small System CRT Controller is a single chip device intended to interface CRT raster scan displays with Intel microcomputers in minimum device-count systems. Its primary function is to refresh the display by buffering character information from main memory and keeping track of the display position of the screen. The flexibility designed into the 8276 will allow simple interface to almost any raster scan CRT display. It can be used with the 8051 Single Chip Microcomputer for a minimum IC count design.

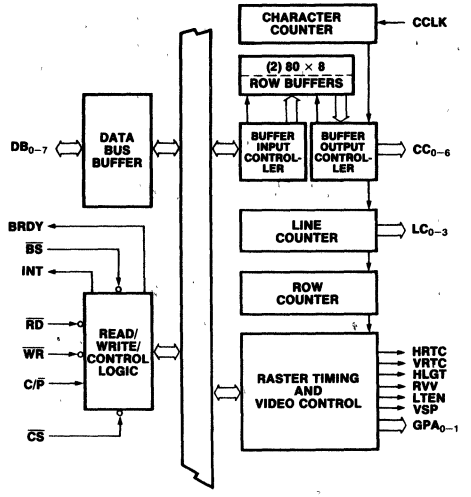


Figure 1. Block Diagram

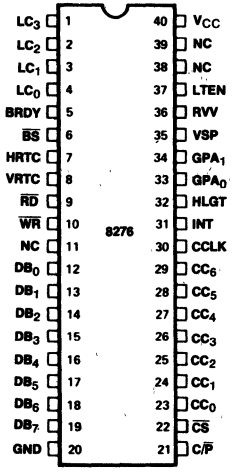


Figure 2. Pin Configuration

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
LC_3 LC_2 LC_1 LC_0	1 2 3 4	O	**Line count.** Output from the line counter which is used to address the character generator for the line positions on the screen.
BRDY	5	O	**Buffer ready.** Output signal indicating that a Row Buffer is ready for loading of character data.
$\overline{BS}$	6	I	**Buffer select.** Input signal enabling $\overline{WR}$ for character data into the Row Buffers.
HRTC	7	O	**Horizontal retrace.** Output signal which is active during the programmed horizontal retrace interval. During this period the VSP output is high and the LTEN output is low.
VRTC	8	O	**Vertical retrace.** Output signal which is active during the programmed vertical retrace interval. During this period the VSP output is high and the LTEN output is low.
$\overline{RD}$	9	I	**Read input.** A control signal to read registers.
$\overline{WR}$	10	I	**Write input.** A control signal to write commands into the control registers or write data into the row buffers.
NC	11		**No connection.**
DB_0 DB_1 DB_2 DB_3 DB_4 DB_5 DB_6 DB_7	12 13 14 15 16 17 18 19	I/O	**Bidirectional data bus.** Three-state lines. The outputs are enabled during a read of the C or P ports.
Ground	20		**Ground.**

Symbol	Pin No.	Type	Name and Function
V_{CC}	40		**+5V power supply.**
NC	39		**No connection.**
NC	38		**No connection.**
LTEN	37	O	**Light enable.** Output signal used to enable the video signal to the CRT. This output is active at the programmed underline cursor position, and at positions specified by attribute codes.
RVV	36	O	**Reverse video.** Output signal used to activate the CRT circuitry to reverse the video signal. This output is active at the cursor position if a reverse video block cursor is programmed or at the positions specified by the field attribute codes.
VSP	35	O	**Video suppression.** Output signal used to blank the video signal to the CRT. This output is active: — during the horizontal and vertical retrace intervals. — at the top and bottom lines of rows if underline is programmed to be number 8 or greater. — when an end of row or end of screen code is detected. — when a Row Buffer underrun occurs. — at regular intervals (1/16 frame frequency for cursor, 1/32 frame frequency for attributes)—to create blinking displays as specified by cursor or field attribute programming.
GPA_1 GPA_0	34 33	O	**General purpose attribute codes.** Outputs which are enabled by the general purpose field attribute codes.
HLGT	32	O	**Highlight.** Output signal used to intensify the display at particular positions on the screen as specified by the field attribute codes.
INT	31	O	**Interrupt output.**
CCLK	30	I	**Character clock** (from dot/timing logic).
CC_6 CC_5 CC_4 CC_3 CC_2 CC_1 CC_0	29 28 27 26 25 24 23	O	**Character codes.** Output from the row buffers used for character selection in the character generator.
$\overline{CS}$	22	I	**Chip select.** Enables $\overline{RD}$ of status or $\overline{WR}$ of command or parameters.
C/ $\overline{P}$	21	I	**Port address.** A high input on this pin selects the "C" port or command registers and a low input selects the "P" port or parameter registers.

FUNCTIONAL DESCRIPTION

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8276 to the system Data Bus.

This functional block accepts inputs from the System Control Bus and generates control signals for overall device operation. It contains the Command, Parameter, and Status Registers that store the various control formats for the device functional definition.

C/$\overline{P}$	OPERATION	REGISTER
0	Read	RESERVED
0	Write	PARAMETER
1	Read	STATUS
1	Write	COMMAND

$\overline{RD}$ (READ)
A "low" on this input informs the 8276 that the CPU is reading status information from the 8276.

$\overline{WR}$ (WRITE)
A "low" on this input informs the 8276 that the CPU is writing data or control words to the 8276.

$\overline{CS}$ (CHIP SELECT)
A "low" on this input selects the 8276 for $\overline{RD}$ or $\overline{WR}$ of Commands, Status, and Parameters.

BRDY (BUFFER READY)
A "high" on this output indicates that the 8276 is ready to receive character data.

$\overline{BS}$ (BUFFER SELECT)
A "low" on this input enables $\overline{WR}$ of character data to the 8276 row buffers.

INT (INTERRUPT)
A "high" on this output informs the CPU that the 8276 needs interrupt service.

C/$\overline{P}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	$\overline{BS}$	
0	0	1	0	1	Reserved
0	1	0	0	1	Write 8276 Parameter
1	0	1	0	1	Read 8276 Status
1	1	0	0	1	Write 8276 Command
X	1	0	1	0	Write 8276 Row Buffer
X	1	1	X	X	High Impedance
X	X	X	1	1	High Impedance

Character Counter

The Character Counter is a programmable counter that is used to determine the number of characters to be displayed per row and the length of the horizontal retrace interval. It is driven by the CCLK (Character Clock) input, which should be derived from the external dot clock.

Line Counter

The Line Counter is a programmable counter that is used to determine the number of horizontal lines (Raster Scans) per character row. Its outputs are used to address the external character generator.

Row Counter

The Row Counter is a programmable counter that is used to determine the number of character rows to be displayed per frame and length of the vertical retrace interval.

Raster Timing and Video Controls

The Raster Timing circuitry controls the timing of the HRTC (Horizontal Retrace) and VRTC (Vertical Retrace) outputs. The Video Control circuitry controls the generation of HGLT (Highlight), RVV (Reverse Video), LTEN (Light Enable), VSP (Video Suppress), and GPA_{0-1} (General Purpose Attribute) outputs.

Row Buffers

The Row Buffers are two 80-character buffers. They are filled from the microcomputer system memory with the character codes to be displayed. While one row buffer is displaying a row of characters, the other is being filled with the next row of characters.

Buffer Input/Output Controllers

The Buffer Input/Output Controllers decode the characters being placed in the row buffers. If the character is a field attribute or special code, they control the appropriate action. (Example: A "Highlight" field attribute will cause the Buffer Output Controller to activate the HGLT output.)

SYSTEM OPERATION

The 8276 is programmable to a large number of different display formats. It provides raster timing, display row buffering, visual attribute decoding and cursor timing.

It is designed to interface with standard character generators for dot matrix decoding. Dot level timing must be provided by external circuitry.

General Systems Operational Description

Display characters are retrieved from memory and displayed on a row-by-row basis. The 8276 has two row buffers. While one row buffer is being used for display, the other is being filled with the next row of characters to be displayed. The number of display characters per row and the number of character rows per frame are software programmable, providing easy interface to most CRT displays. (See Programming Section.)

The 8276 uses BRDY to request character data to fill the row buffer that is not being used for display.

The 8276 displays character rows one scan line at a time. The number of scan lines per character row, the underline position, and blanking of top and bottom lines are programmable. (See Programming Section.)

The 8276 provides special Control Codes which can be used to minimize overhead. It also provides Visual Attribute Codes to cause special action on the screen without the use of the character generator. (See Visual Attributes Section.)

The 8276 also controls raster timing. This is done by generating Horizontal Retrace (HRTC) and Vertical Retrace (VRTC) signals. The timing of these signals is also programmable.

The 8276 can generate a cursor. Cursor location and format are programmable. (See Programming Section.)

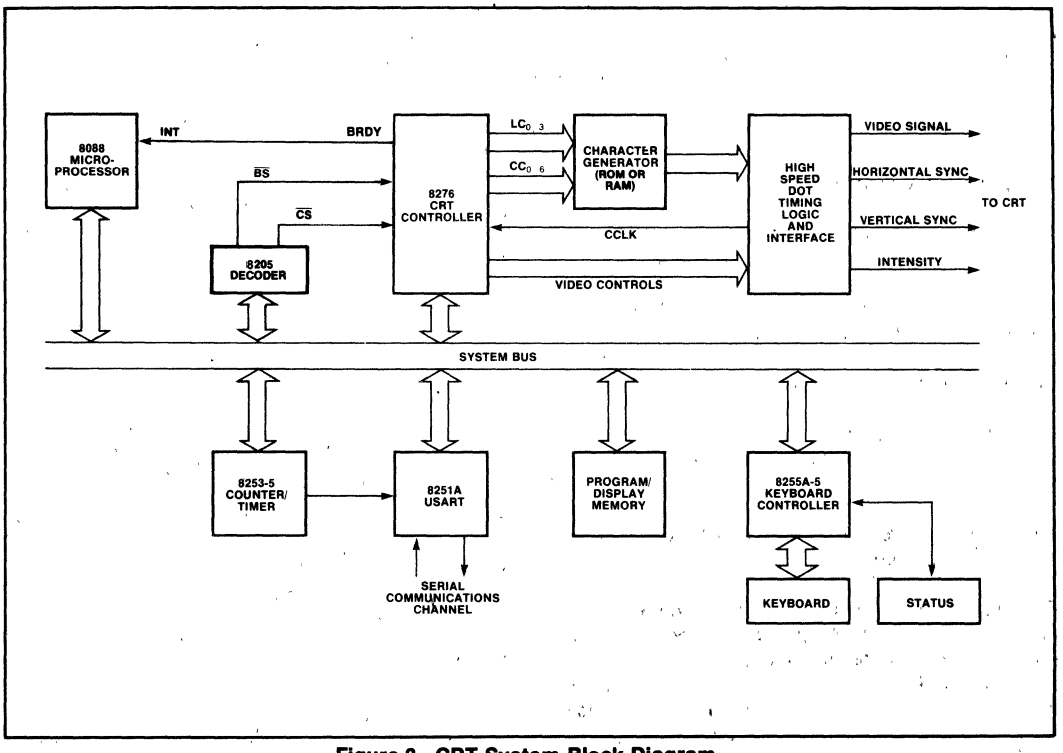


Figure 3. CRT System Block Diagram

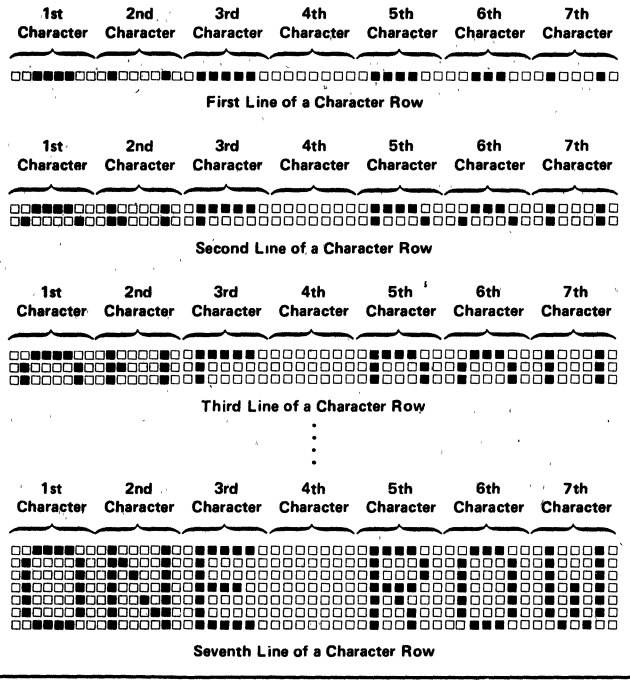


Figure 4. Display Of A Character Row

Display Row Buffering

Before the start of a frame, the 8276 uses BRDY and $\overline{BS}$ to fill one row buffer with characters.

When the first horizontal sweep is started, character codes are output to the character generator from the row buffer just filled. Simultaneously, the other row buffer is filled with the next row of characters.

After all the lines of the character row are scanned, the buffers are swapped and the same procedure is followed for the next row.

This process is repeated until all of the character rows are displayed.

Row Buffering allows the CPU access to the display memory at all times except during Buffer Loading (about 25%). This compares favorably to alternative approaches which restrict CPU access to the display memory to occur only during horizontal and vertical retrace intervals (80% of the bus time is used to refresh the display.)

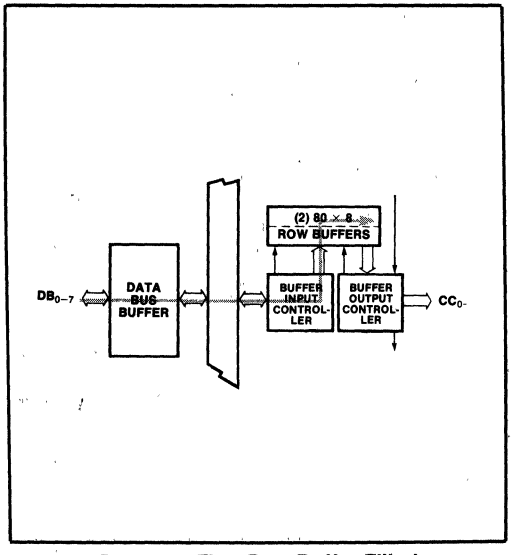


Figure 5. First Row Buffer Filled

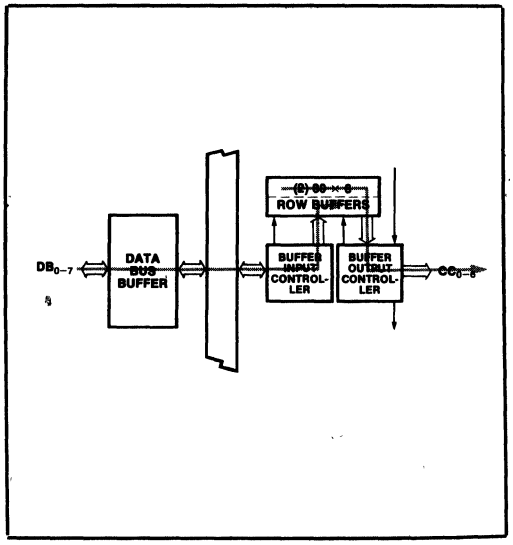


Figure 6. Second Row Buffer Filled, First Row Displayed

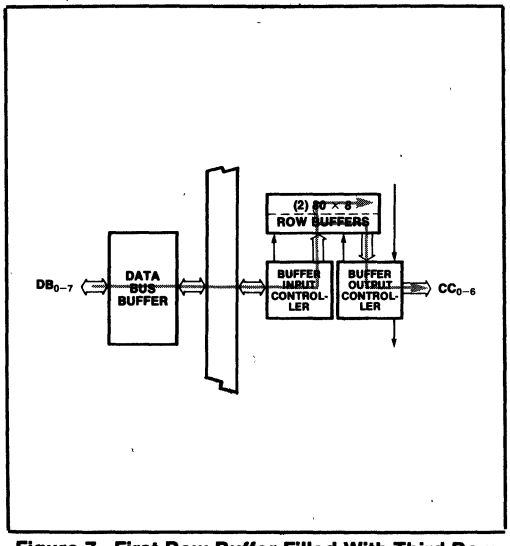


Figure 7. First Row Buffer Filled With Third Row, Second Row Displayed

Display Format

SCREEN FORMAT

The 8276 can be programmed to generate from 1 to 80 characters per row, and from 1 to 64 rows per frame.

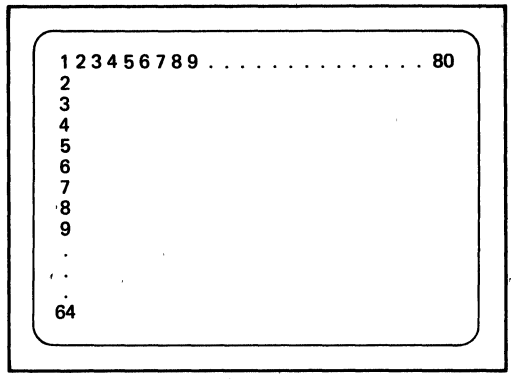


Figure 8. Screen Format

The 8276 can also be programmed to blank alternate rows. In this mode, the first row is displayed, the second blanked, the third displayed, etc. Display data is not requested for the blanked rows.

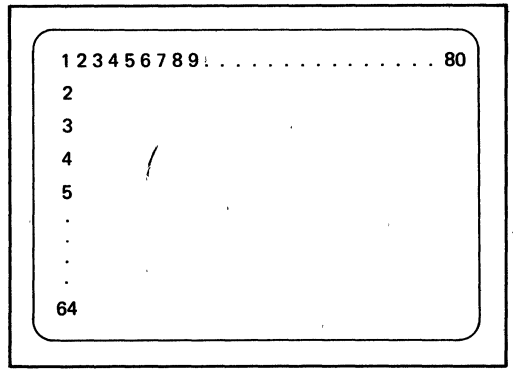


Figure 9. Blank Alternate Rows Mode

ROW FORMAT

The 8276 is designed to hold the line count stable while outputting the appropriate character codes during each horizontal sweep. The line count is incremented during horizontal retrace and the whole row of character codes are output again during the next sweep. This is continued until the entire character row is displayed.

The number of lines (horizontal sweeps) per character row is programmable from 1 to 16.

The output of the line counter can be programmed to be in one of two modes.

In mode 0, the output of the line *counter* is the same as the line *number.*

In mode 1, the line *counter* is offset by one from the line *number*.

Note: In mode 1, while the *first* line (line number 0) is being displayed, the *last* count is output by the line counter (see examples).

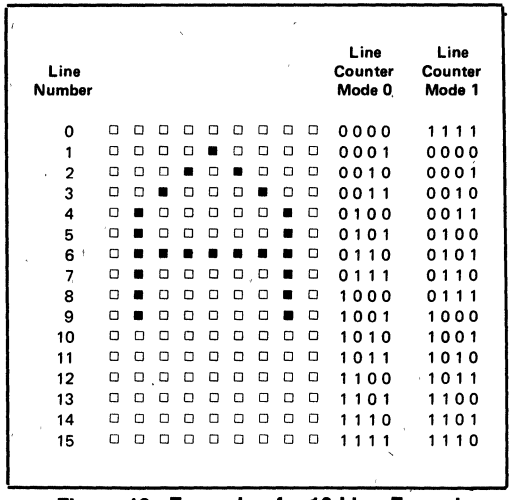


Figure 10. Example of a 16-Line Format

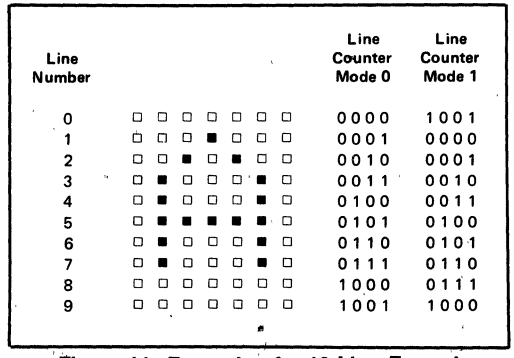


Figure 11. Example of a 10-Line Format

Mode 0 is useful for character generators that leave address zero blank and start at address 1. Mode 1 is useful for character generators which start at address zero.

Underline placement is also programmable (from line *number* 0 to 15). This is independent of the line *counter* mode.

If the line *number* of the underline is greater than 7 (line *number* MSB = 1), then the top and bottom lines will be blanked.

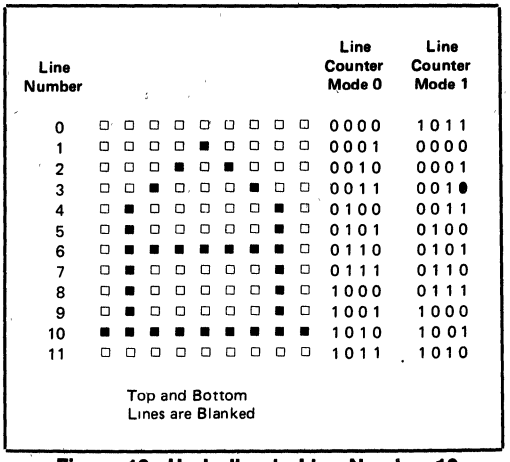


Figure 12. Underline in Line Number 10

If the line *number* of the underline is less than or equal to 7 (line *number* MSB = 0), then the top and bottom lines will *not* be blanked.

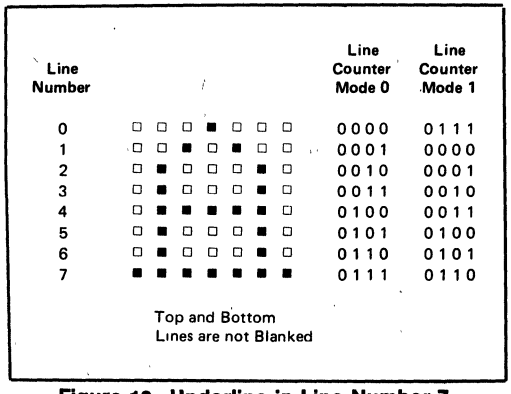


Figure 13. Underline in Line Number 7

If the line *number* of the underline is greater than the maximum number of lines, the underline will not appear.

Blanking is accomplished by the VSP (Video Suppression) signal. Underline is accomplished by the LTEN (Light Enable) signal.

DOT FORMAT
Dot width and character width are dependent upon the external timing and control circuitry.

Dot level timing circuitry should be designed to accept the parallel output of the character generator and shift it out serially at the rate required by the CRT display.

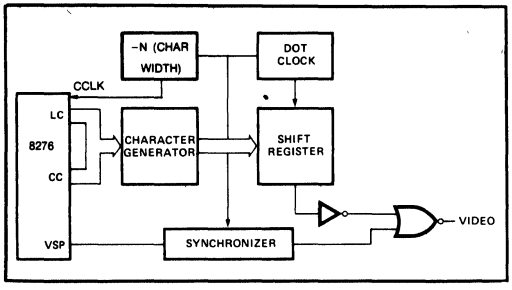


Figure 14. Typical Dot Level Block Diagram

Dot width is a function of dot clock frequency.

Character width is a function of the character generator width.

Horizontal character spacing is a function of the shift register length.

Note: Video control and timing signals must be synchronized with the video signal due to the character generator access delay.

Raster Timing

The character counter is driven by the character clock input (CCLK). It counts out the characters being displayed (programmable from 1 to 80). It then causes the line counter to increment, and it starts counting out the horizontal retrace interval (programmable from 2 to 32). This process is constantly repeated.

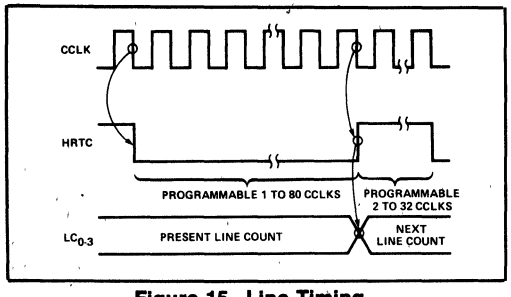


Figure 15. Line Timing

The line counter is driven by the character counter. It is used to generate the line address outputs (LC_{0-3}) for the character generator. After it counts all of the lines in a character row (programmable from 1 to 16), it increments the row counter, and starts over again. (See Character Format Section for detailed description of Line Counter functions.)

The row counter is an internal counter driven by the line counter. It controls the functions of the row buffers and counts the number of character rows displayed.

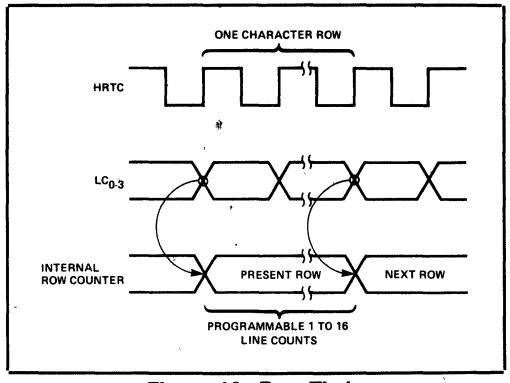


Figure 16. Row Timing

After the row counter counts all of the rows in a frame (programmable from 1 to 64), it starts counting out the vertical retrace interval (programmable from 1 to 4).

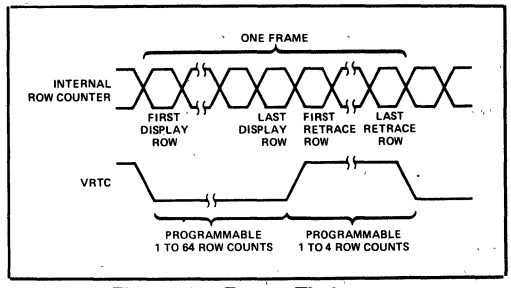


Figure 17. Frame Timing

The Video Suppression Output (VSP) is active during horizontal and vertical retrace intervals.

Dot level timing circuitry must synchronize these outputs with the video signal to the CRT Display.

Interrupt Timing

The 8276 can be programmed to generate an interrupt request at the end of each frame. If the 8276 interrupt enable flag is set, an interrupt request will occur at the *beginning* of the *last display row*.

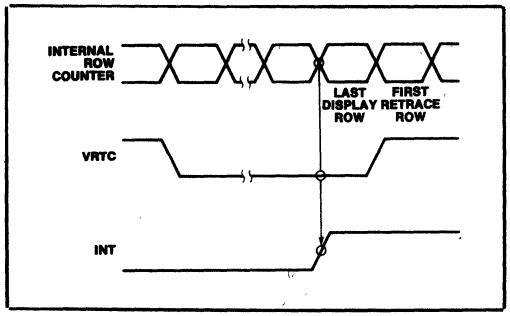


Figure 18. Beginning of Interrupt

INT will go inactive after the status register is read.

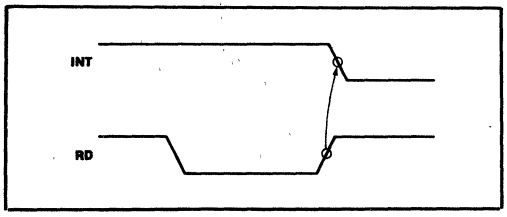


Figure 19. End of Interrupt

A reset command will also cause INT to go inactive, but this is not recommended during normal service.

Note: Upon power-up, the 8276 Interrupt Enable Flag may be set. As a result, the user's cold start routine should write a reset command to the 8276 before system interrupts are enabled.

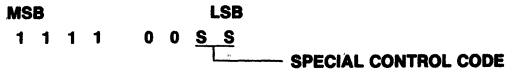
VISUAL ATTRIBUTES AND SPECIAL CODES

The characters processed by the 8276 are 8-bit quantities. The character code outputs provide the character generator with 7 bits of address. The Most Significant Bit is the extra bit and it is used to determine if it is a normal display character (MSB = 0), or if it is a Field Attribute or Special Code (MSB = 1).

Special Codes

Four special codes are available to help reduce bus usage.

SPECIAL CONTROL CHARACTER



S	S	FUNCTION
0	0	End of Row
0	1	End of Row-Stop Buffer Loading
1	0	End of Screen
1	1	End of Screen-Stop Buffer Loading

The End of Row Code (00) activates VSP and holds it to the end of the line.

The End of Row-Stop Buffer Loading (BRDY) Code (01) causes the Buffer Loading Control Logic to stop buffer loading for the rest of the row upon being written into the Row Buffer. It affects the display in the same way as the End of Row Code (00).

The End of Screen Code (10) activates VSP and holds it to the end of the frame.

The End of Screen-Stop Buffer Loading (BRDY) Code (11) causes the Row Buffer Control Logic to stop buffer loading for the rest of the frame upon being written. It affects the display in the same way as the End of Screen Code (10).

If the Stop Buffer Loading feature is not used, all characters after an End of Row character are ignored, except for the End of Screen character, which operates normally. All characters after an End of Screen character are ignored.

Note: If a Stop Buffer Loading is not the last character in a row, Buffer Loading is not stopped until after the next character is read. In this situation, a dummy character must be placed in memory after the Stop Buffer Loading character.

Field Attributes

The field attributes are control codes which affect the visual characteristics for a field of characters, starting at the character following the code up to, and including, the character which precedes the *next* field attribute code, or up to the end of the frame. The field attributes are reset during the vertical retrace interval.

The 8276 can be programmed to provide visible field attribute characters; all field attribute codes will occupy a position on the screen. These codes will appear as blanks caused by activation of the Video Suppression output (VSP). The chosen visual attributes are activated after this blanked character.

There are six field attributes:

1. *Blink*—Characters following the code are caused to blink by activating the Video Suppression output (VSP). The blink frequency is equal to the screen refresh frequency divided by 32.
2. *Highlight*—Characters following the code are caused to be highlighted by activating the Highlight output (HGLT).
3. *Reverse Video*—Characters following the code are caused to appear with reverse video by activating the Reverse Video output (RVV).
4. *Underline*—Characters following the code are caused to be underlined by activating the Light Enable output (LTEN).

5,6. *General Purpose*—There are two additional 8276 outputs which act as general purpose, independently programmable field attributes. GPA_{0-1} are active high outputs.

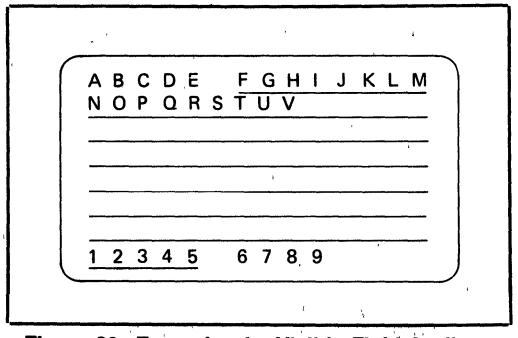
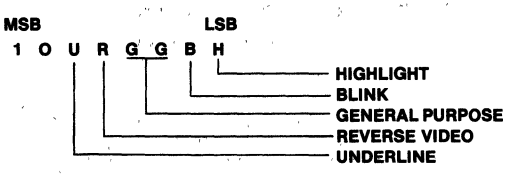


Figure 20. Example of a Visible Field Attribute (Underline Attribute)

FIELD ATTRIBUTE CODE



H = 1 FOR HIGHLIGHTING
B = 1 FOR BLINKING
R = 1 FOR REVERSE VIDEO
U = 1 FOR UNDERLINE
GG = GPA_1, GPA_0

Note: More than one attribute can be enabled at the same time. If the blinking and reverse video attributes are enabled simultaneously, only the reversed characters will blink.

Cursor Timing

The cursor location is determined by a cursor row register and a character position register which are loaded by command to the controller. The cursor can be programmed to appear on the display as:

1. a blinking underline
2. a blinking reverse video block
3. a non-blinking underline
4. a non-blinking reverse video block

The cursor blinking frequency is equal to the screen refresh frequency divided by 16.

If a non-blinking reverse video *cursor* appears in a non-blinking reverse video *field*, the cursor will appear as a normal video block.

If a non-blinking underline *cursor* appears in a non-blinking underline *field*, the cursor will not be visible.

Device Programming

The 8276 has two programming registers, the Command Register and the Parameter Register. It also has a Status Register. The Command Register can only be written into and the Status Register can only be read from. They are addressed as follows:

C/$\overline{P}$	OPERATION	REGISTER
0	Read	Reserved
0	Write	Parameter
1	Read	Status
1	Write	Command

The 8276 expects to receive a command and a sequence of 0 to 4 parameters, depending on the command. If the proper number of parameter bytes are not received before another command is given, a status flag is set, indicating an improper command.

Instruction Set

The 8276 instruction set consists of 7 commands.

COMMAND	NO. OF PARAMETER BYTES
Reset	4
Start Display	0
Stop Display	0
Load Cursor	2
Enable Interrupt	0
Disable Interrupt	0
Preset Counters	0

In addition, the status of the 8276 can be read by the CPU at any time.

1. RESET COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Reset Command	0 0 0 0 0 0 0 0
Parameters	Write	0	Screen Comp Byte 1	S H H H H H H H
	Write	0	Screen Comp Byte 2	V V R R R R R R
	Write	0	Screen Comp Byte 3	U U U U L L L L
	Write	0	Screen Comp Byte 4	M 1 C C Z Z Z Z

Action—After the reset command is written, BRDY goes inactive, 8276 interrupts are disabled, and the VSP output is used to blank the screen. HRTC and VRTC continue to run. HRTC and VRTC timing are random on power-up

As parameters are written, the screen composition is defined.

Parameter—S Spaced Rows

S	FUNCTIONS
0	Normal Rows
1	Spaced Rows

Parameter—HHHHHHH Horizontal Characters/Row

H H H H H H H	NO. OF CHARACTERS PER ROW
0 0 0 0 0 0 0	1
0 0 0 0 0 0 1	2
0 0 0 0 0 1 0	3
. . .	.
. . .	.
. . .	.
1 0 0 1 1 1 1	80
1 0 1 0 0 0 0	Undefined
. . .	.
. . .	.
. . .	.
1 1 1 1 1 1 1	Undefined

Parameter—VV Vertical Retrace Row Count

V V	NO. OF ROW COUNTS PER VRTC
0 0	1
0 1	2
1 0	3
1 1	4

Parameter—RRRRRR Vertical Rows/Frame

R R R R R R	NO. OF ROWS/FRAME
0 0 0 0 0 0	1
0 0 0 0 0 1	2
0 0 0 0 1 0	3
. . .	.
. . .	.
. . .	.
1 1 1 1 1 1	64

Parameter—UUUU Underline Placement

U U U U	LINE NUMBER OF UNDERLINE
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
. . .	.
. . .	.
. . .	.
1 1 1 1	16

Parameter—LLLL Number of Lines per Character Row

L L L L	NO. OF LINES/ROW
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
. . .	.
. . .	.
. . .	.
1 1 1 1	16

Parameter—M Line Counter Mode

M	LINE COUNTER MODE
0	Mode 0 (Non-Offset)
1	Mode 1 (Offset by 1 Count)

Parameter—CC Cursor Format

C C	CURSOR FORMAT
0 0	Blinking reverse video block
0 1	Blinking underline
1 0	Non-blinking reverse video block
1 1	Non-blinking underline

Parameter—ZZZZ Horizontal Retrace Count

Z Z Z Z	NO. OF CHARACTER COUNTS PER HRTC
0 0 0 0	2
0 0 0 1	4
0 0 1 0	6
.	.
.	.
.	.
1 1 1 1	32

Note: uuuu MSB determines blanking of top and bottom lines (1 = blanked, 0 = not blanked).

2. START DISPLAY COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Start Display	0 0 1 0 0 0 0 0
No parameters				

Action—8276 interrupts are enabled, BRDY goes active, video is enabled, Interrupt Enable and Video Enable status flags are set.

3. STOP DISPLAY COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Stop Display	0 1 0 0 0 0 0 0
No parameters				

Action—Disables video, interrupts remain enabled, HRTC and VRTC continue to run, Video Enable status flag is reset, and the "Start Display" command must be given to reenable the display.

4. LOAD CURSOR POSITION

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Load Cursor	1 0 0 0 0 0 0 0
Parameters	Write	0	Char. Number	(Char. Position in Row)
	Write	0	Row Number	(Row Number)

Action—The 8276 is conditioned to place the next two parameter bytes into the cursor position registers. Status flag not affected.

5. ENABLE INTERRUPT COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Enable Interrupt	1 0 1 0 0 0 0 0
No parameters				

Action—The interrupt enable flag is set and interrupts are enabled.

6. DISABLE INTERRUPT COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Disable Interrupt	1 1 0 0 0 0 0 0
No parameters				

Action—Interrupts are disabled and the interrupt enable status flag is reset.

7. PRESET COUNTERS COMMAND

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Write	1	Preset Counters	1 1 1 0 0 0 0 0
No parameters				

Action—The internal timing counters are preset, corresponding to a screen display position at the top left corner. Two character clocks are required for this operation. The counters will remain in this state until any other command is given.

This command is useful for system debug and synchronization of clustered CRT displays on a single CPU. After this command, two additional clock cycles are required before the first character of the first row is put out.

Status Flags

	OPERATION	C/P	DESCRIPTION	DATA BUS MSB LSB
Command	Read	1	Status Word	0 IE IR X IC VE BU X

IE — (Interrupt Enable) Set or reset by command. It enables vertical retrace interrupt. It is automatically set by a "Start Display" command and reset with the "Reset" command.

IR — (Interrupt Request) This flag is set at the beginning of display of the last row of the frame if the interrupt enable flag is set. It is reset after a status read operation.

IC — (Improper Command) This flag is set when a command parameter string is too long or too short. The flag is automatically reset after a status read.

VE — (Video Enable) This flag indicates that video operation of the CRT is enabled. This flag is set on a "Start Display" command, and reset on a "Stop Display" or "Reset" command.

BU — (Buffer Underrun) This flag is set whenever a Row Buffer is not filled with character data in time for a buffer swap required by the display. Upon activation of this bit, buffer loading ceases, and the screen is blanked until after the vertical retrace interval.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

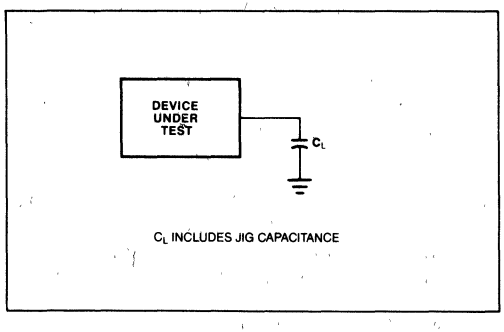
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ±5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to 0.45V
I_{CC}	V_{CC} Supply Current		160	mA	

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
C_{IN}	Input Capacitance		10	pF	f_C = 1 MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to V_{SS}

A.C. TESTING LOAD CIRCUIT



A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V)

Bus Parameters

READ CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AR}	Address Stable Before READ	0		ns	
t_{RA}	Address Hold Time for READ	0		ns	
t_{RR}	READ Pulse Width	250		ns	
t_{RD}	Data Delay from READ		200	ns	C_L = 150pF
t_{DF}	READ to Data Floating		100	ns	

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{AW}	Address Stable Before WRITE	0		ns	
t_{WA}	Address Hold Time for WRITE	0		ns	
t_{WW}	WRITE Pulse Width	250		ns	
t_{DW}	Data Setup Time for WRITE	150		ns	
t_{WD}	Data Hold Time for WRITE	0		ns	

CLOCK TIMING

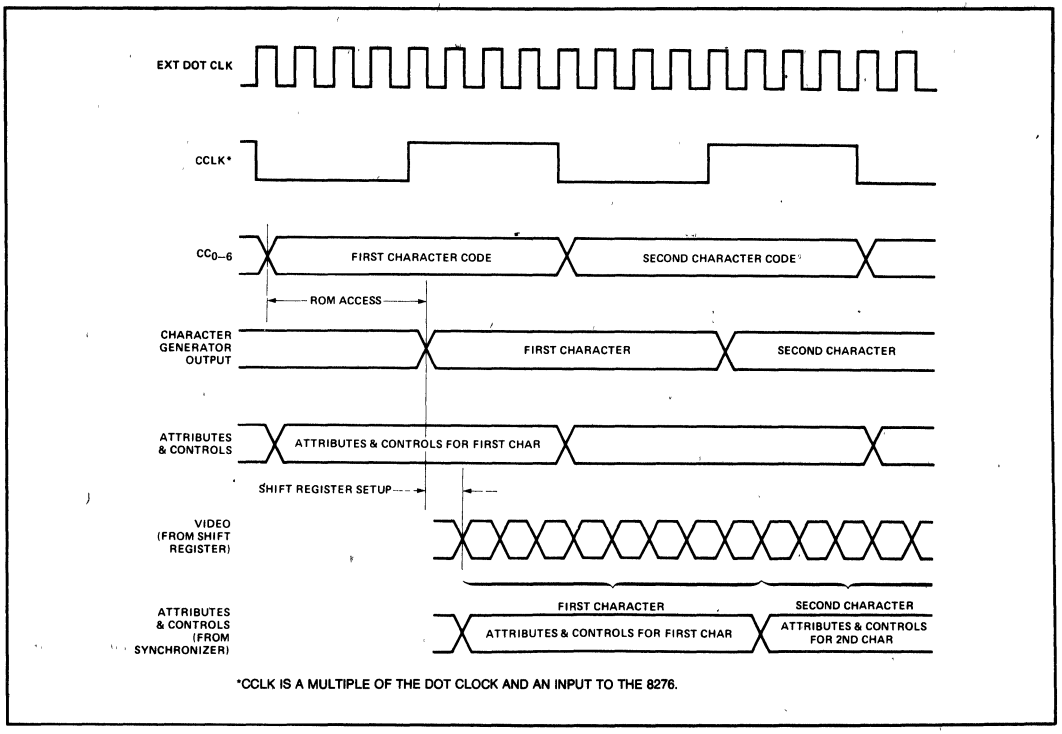
		8276		8276-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
t_{CLK}	Clock Period	480		320		ns	
t_{KH}	Clock High	240		120		ns	
t_{KL}	Clock Low	160		120		ns	
t_{KR}	Clock Rise	5	30	5	30	ns	
t_{KF}	Clock Fall	5	30	5	30	ns	

OTHER TIMING

		8276		8276-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
t_{CC}	Character Code Output Delay		150		150	ns	CL = 50 pF
t_{HR}	Horizontal Retrace Output Delay		200		150	ns	C_L = 50 pF
t_{LC}	Line Count Output Delay		400		250	ns	C_L = 50 pF
t_{AT}	Control/Attribute Output Delay		275		250	ns	C_L = 50 pF
t_{VR}	Vertical Retrace Output Delay		275		250	ns	C_L = 50 pF
t_{RI}	INT↓ from $\overline{RD}$↑		250		250	ns	C_L = 50 pF
t_{WQ}	BRDY↑ from $\overline{WR}$↑		250		250	ns	C_L = 50 pF
t_{RQ}	BRDY↓ from $\overline{WR}$↓		200		200	ns	C_L = 50 pF
t_{LR}	$\overline{BS}$↓ to $\overline{WR}$↓	0		0		ns	
t_{RL}	$\overline{WR}$↑ to $\overline{BS}$↑	0		0		ns	

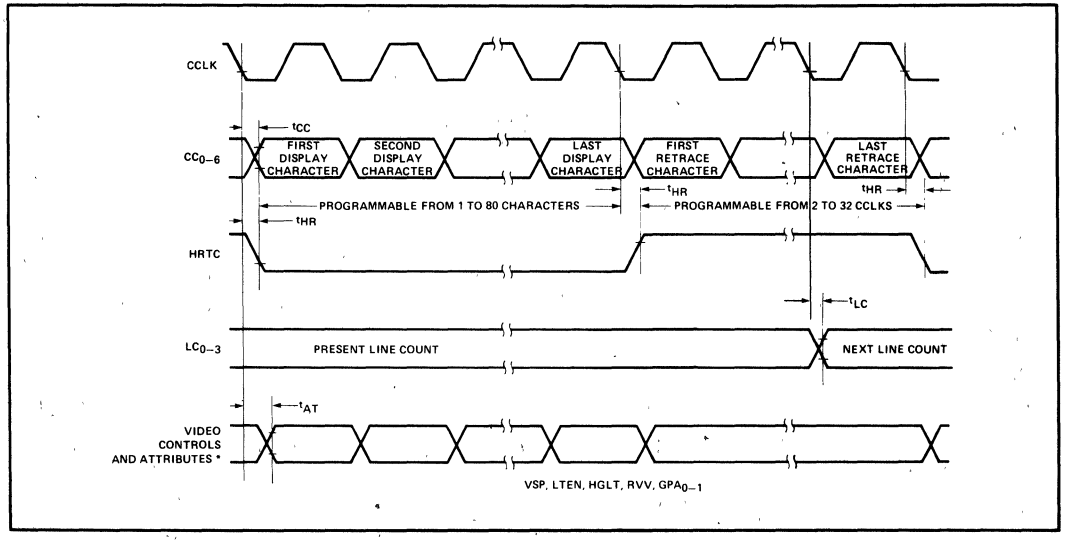
WAVEFORMS

Typical Dot Level Timing

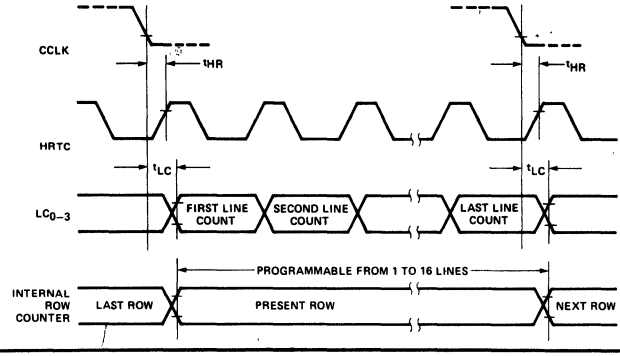


*CCLK IS A MULTIPLE OF THE DOT CLOCK AND AN INPUT TO THE 8276.

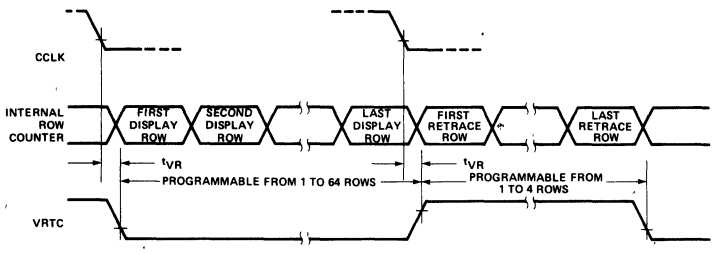
Line Timing



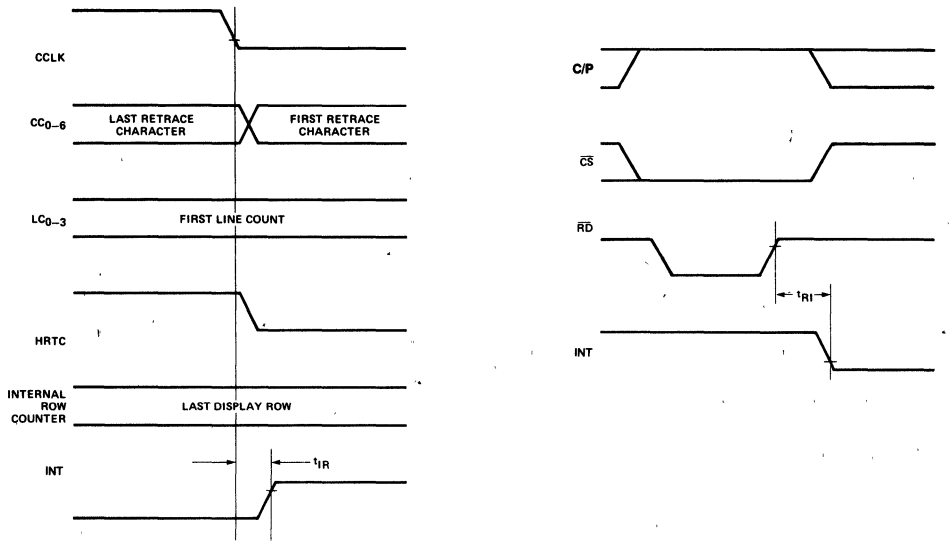
Row Timing



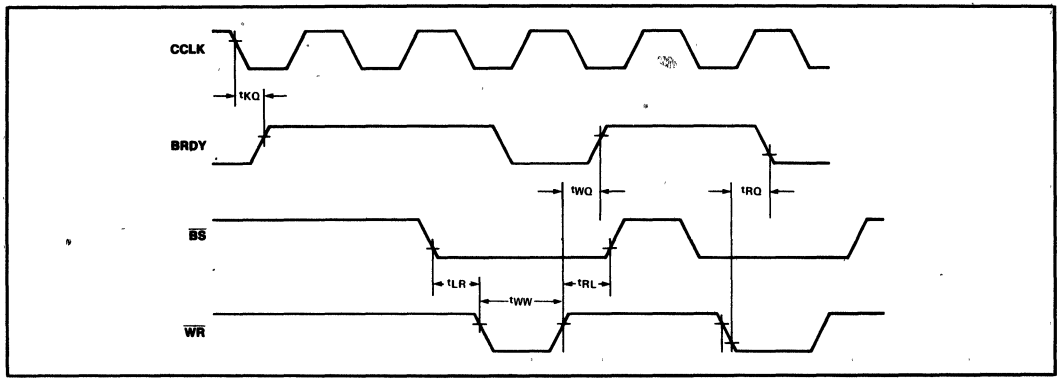
Frame Timing



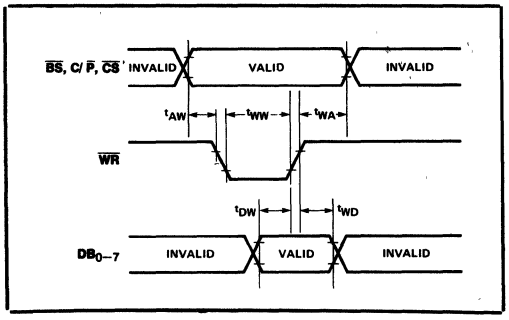
Interrupt Timing



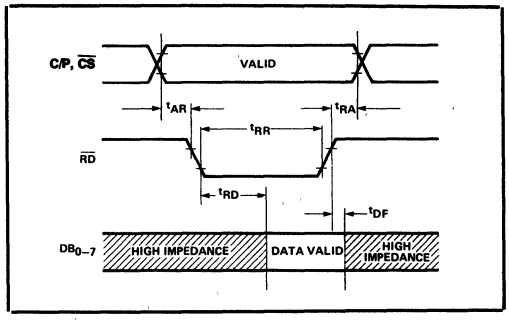
Timing for Buffer Loading



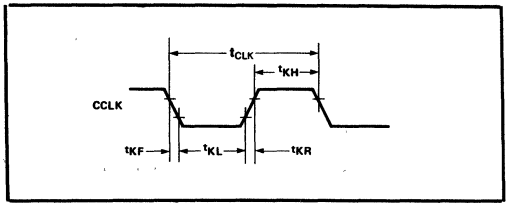
Write Timing



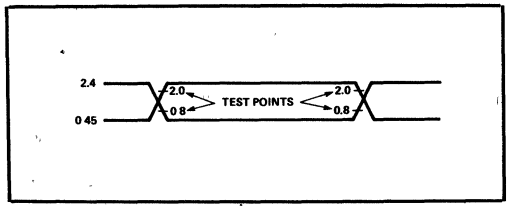
Read Timing



Clock Timing



Input and Output Waveforms for A.C. Tests



FOR A.C TESTING, INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0 " TIMING MEASUREMENTS FOR INPUT AND OUTPUT SIGNALS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0 "

intel

ARTICLE REPRINT

AR-255

By managing tasks like graphics generation and CRT refreshing, a dedicated VLSI display controller simplifies the design of intelligent graphics work stations.

Dedicated VLSI chip lightens graphics display design load

The role of graphics is becoming increasingly important for unscrambling the communications traffic between people and computers. Thanks to microprocessors and dedicated control ICs, designing high-reliability graphics work stations is now easier and less expensive than in the days of small-scale integration and expensive discrete-circuit CRT technology. Microprocessors simplify workstation design by transferring some graphics control tasks from hardware to software. However, a dedicated VLSI controller such as the 82720—with an on-board graphics processor—can push another step forward toward fast and economical design of high-quality intelligent graphics systems.

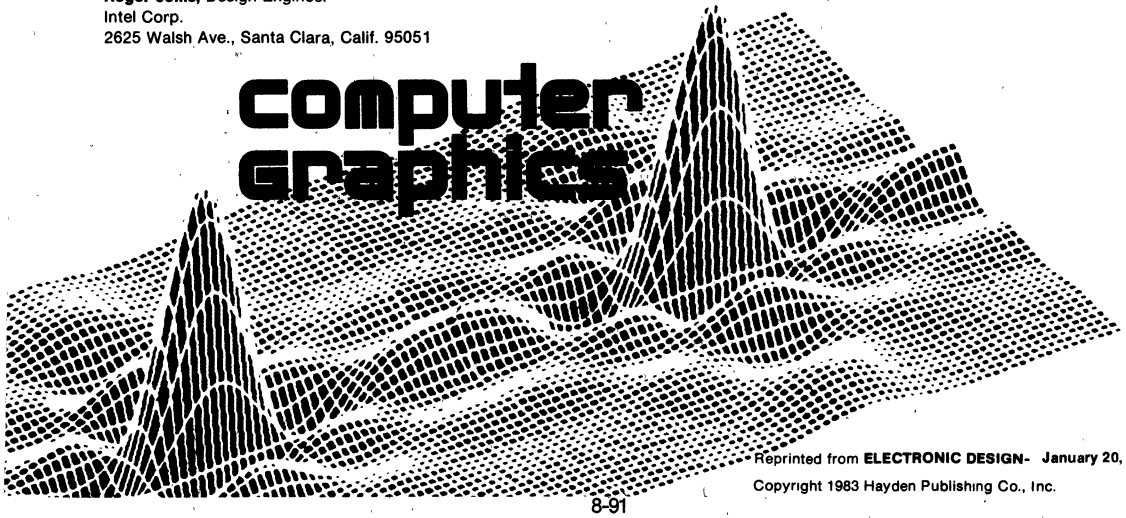
A typical application for the controller is a graphics work station aimed at high-end business and low-end engineering systems. Since such a station usually fits on the top of a desk, all of the electronics must be contained within a single printed-circuit board. This type of system requires a resolution of about 512 by 512 pixels and is frequently called on to display three-dimensional objects in various perspectives. To minimize the distortion of rotating objects, horizontal and vertical pixels should be equally spaced.

A typical display (500 vertices) must be drawn on the screen in less than 1 second to provide satisfactory interaction with the operator. The display may consist of lines, arcs, filled areas, and colors—seven colors are acceptable (see "A Look into Graphics Fundamentals").

Serial link interfaces station

An intelligent work station usually interfaces with a mainframe host via a serial communications link, a keyboard, and a serial link with an optional graphics tablet. This type of graphics input/output subsystem is diagrammed in Fig. 1. Two 5¼-in. floppy disks can satisfy the mass-storage needs of the system. Disk formatting must be compatible with the requirements of an IBM personal computer. Moreover, general-purpose software written for

Gary DePalma, Field Applications Engineer
Mark Olson, Product Marketing Engineer
Roger Jollis, Design Engineer
Intel Corp.
2625 Walsh Ave., Santa Clara, Calif. 95051

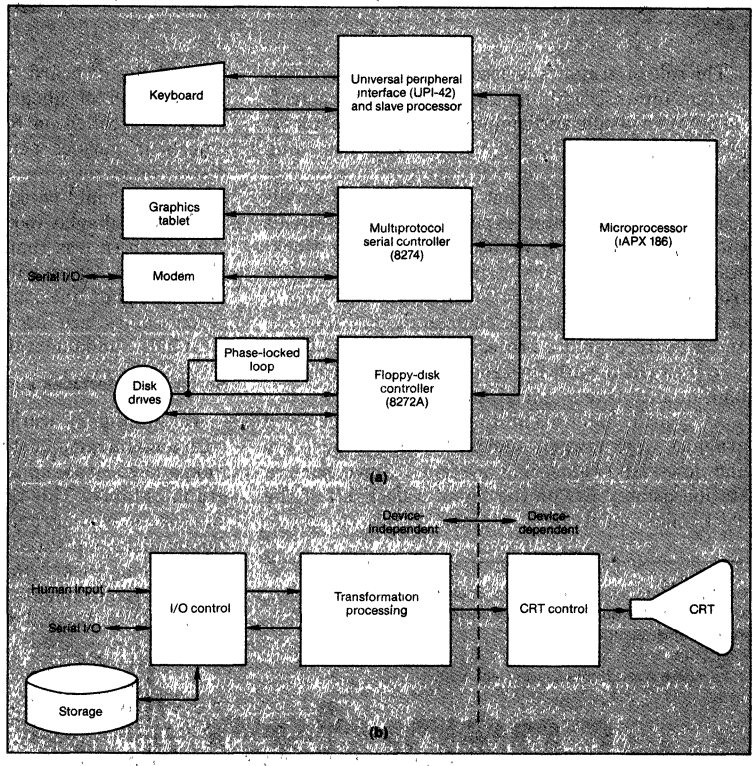


Reprinted from **ELECTRONIC DESIGN- January 20, 1983**

this computer must also be able to run on the work station.

Two of the most basic functions of a graphics system are generating and refreshing images on the CRT screen. Information pertaining to the images is stored in the bit-map memory, where monochrome pixels are represented by single bits and color pixels by groups of bits. Lines and arcs defined in normalized screen coordinates must be converted into images of the physical object.

In a bit-mapped raster graphics system, lines described by a transformed display list are reduced to a series of dots and placed in the image memory. The selection of the dots that will be activated is achieved through a scanning conversion algorithm, which must create lines that appear very smooth, start and end as expected, and look symmetrical no matter in which direction they are drawn. The algorithm is repeated thousands of times to draw a single picture and thus must operate as quickly as possible. At the same time, the image in memory must be repainted on the screen 30 times/s for

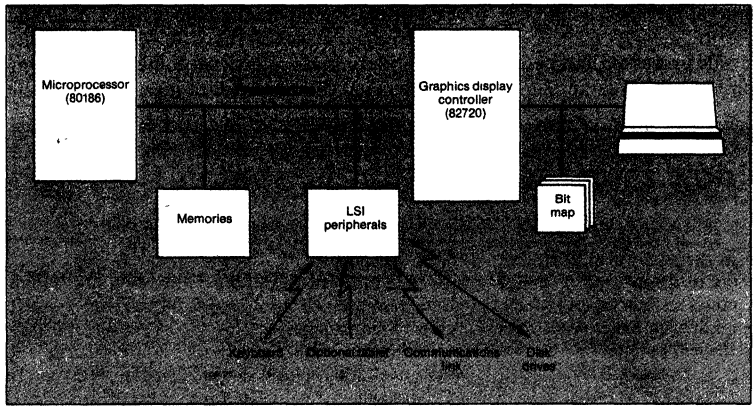


1. A graphics I/O subsystem for an intelligent work station consists of input peripherals (a keyboard and tablet), a serial communications link, and mass storage (floppy disks). Intelligence is provided by the microprocessor and the peripheral and memory controllers (a). The three basic tasks performed—I/O, transformation processing, and CRT control—all require data in the form of display lists stored in a data base (b).

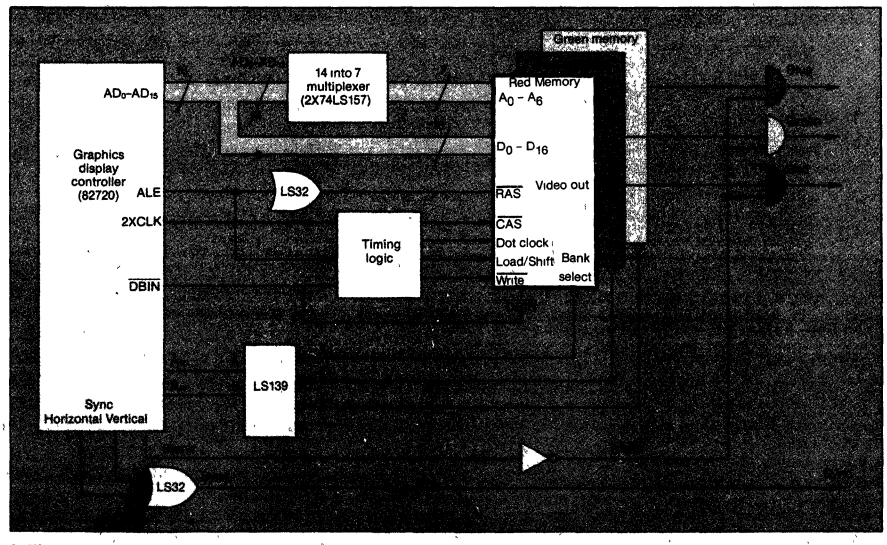
interlaced frames and 60 times/s for noninterlaced frames. Simple tasks, they nevertheless demand a high memory bandwidth.

Unlike other system control tasks, generating graphics figures requires both bit-manipulation and mathematics capabilities. Integer addition and multiplication operations calculate the coordinates of points on a line or a circle. But since pixels generally are neither complete words nor bytes, logical operations must be performed on the bits within the word that contains the selected pixel.

The inner loop of a so-called Bresenham line-drawing algorithm requires two or three addition operations, two comparisons or tests, and the masking of the correct value into the word for each pixel. Algorithms for drawing circles or filling areas are even more complex. In the inner loop of a filling algorithm, for example, the old word must be read from the bit map to determine whether some, all, or none of the pixels are within the area to be filled. If they are, the algorithm tests whether the pixels must be modified and then returns the word to the



2. The 82720 graphics display controller separates the tasks of graphics generation and CRT refreshing from other system tasks. That permits much greater system bandwidth, leading to graphics work stations that not only draw sharp pictures, but also offer color.



3. Three memory planes are implemented in the interface between the bit map and the graphics display controller. Three primary colors—red, green, and blue—are provided, with the controller's upper address bits responsible for selecting the memory planes during read/modify/write cycles.

bit map. Because such algorithms are heavily exercised, they must execute at extremely high speeds to avoid an adverse impact on the system's overall efficiency.

Memory bandwidth is the most precious commodity in a graphics system. In this application, screen refreshing requires that 750,000 bits be read 60 times/s, equating to a bandwidth of almost 6 Mbytes/s. The picture refreshing, therefore, has the highest-priority access to memory because any missed readings show up as noise in the picture, a situation that sometimes occurs with simple systems possessing a single-microprocessor, single-memory scheme.

In the latter type of design, one processor handles all functions except refreshing, which is implemented by a discrete counter arrangement or a simple CRT controller chip. Nevertheless, the refresh memory bandwidth always slows down the microprocessor. That loss of speed can be eliminated simply by separating the processor's memory system from the bit map, a process that effectively doubles system memory bandwidth.

The 82720 graphics display controller can provide the means of separating graphics generation and CRT refreshing from the other tasks and also perform the two tasks quickly and concurrently with the others. Residing between the microprocessor and the bit-map memory and video logic, the controller refreshes the CRT like other CRT controllers, converts high-level commands into images by placing the proper data into the correct bit

A look into graphics fundamentals

The graphics data found in graphics display lists typically describes objects in the real-world Cartesian coordinate system conforming to the axes X, Y, and Z (see the figure). Graphics data does not take the form of one bit for every point on a line; rather it represents higher-level forms such as the end points of a line and

```
WALL: OBJECT
      MOVE TO [X, Y, Z]
      DRAW LINE TO [X2, Y2, Z2]
      DRAW LINE TO [X3, Y3, Z3]
      DRAW LINE TO [X4, Y4, Z4]
      DRAW LINE TO [X1, Y1, Z1]
END WALL
ROOF: OBJECT
      .
      .
      .
HOUSE: OBJECT
      WALL AT [T1]
      WALL AT [T2]
      WALL AT [T3]
      WALL AT [T4]
      ROOF AT [T5]
      WINDOW AT [T6]
      WINDOW AT [T7]
      DOOR AT [T8]
END HOUSE
```

the starting, ending, and center points of an arc.

The coordinate system handles physical measurement units such as inches, feet, or meters, which are typically represented in a computer by 16- or 32-bit integers or by floating-point formats. Ultimately, complex graphics structures are stored in a data base in a hierarchical form consisting of lists of X, Y, and Z coordinates.

The first step in designing a CRT subsystem involves selecting the resolution and scanning rates. All conventional raster-scanning monitors have a display area that is wider than it is high in the ratio of 4 to 3 (called the aspect ratio). For pixels to be square—equally spaced in both the X and the Y direction—the number of horizontal pixels must be 4/3 the number of vertical pixels. This is expressed as 4H-3V, where H and V represent the number of horizontal and vertical pixels respectively. Resolution depends on the total quantity of pixels, which must be a power of two. If it is not, the number of pixels must be rounded to the next highest power of two, in which case some bits will be wasted. Furthermore, the number of horizontal pixels must be organized as an even number of 16-bit words.

To prevent wasted bits, the number of vertical and horizontal pixels are chosen as large as possible without exceeding a power of two. For the display in question, 512H by 512V = 2^{18} = 262,144 pixels. A screen format of 576H by 432V normally meets all requirements. The total number of pixels is then 248,832, and the ratio of horizontal to vertical pixels (576/432) is correct. Furthermore, the number of horizontal pixels makes exactly 36 16-bit words.

After figuring the aspect ratio, the format of the bit-map memory is the next item to be considered. The screen contains about 250,000 pixels, each of which can be either black or one of seven colors. These eight shades can be represented by three bits/pixel (2^3 = 8), meaning that the bit-map memory must handle about 750,000 bits. The organization of the memory, however, must be determined according to the various tradeoffs.

The entire memory must be accessed 60 times/s since that is the rate at which the image must be painted to prevent flickering. That equates to a refresh rate of 16.7 ms. As a rule of thumb, the monitor displays information 75% of the time and is blanked for retracing operations 25% of the time. Thus the whole memory must be read and sent to the CRT during a 12.5-ms interval (16.7 × 0.75), which constitutes the active

map, and interfaces easily and simply with proprietary microprocessors.

The 82720 accepts high-level commands (such as DRAW LINE, DRAW ARC, and FILL RECTANGLE) and executes them at much faster speeds than general-purpose microprocessors, primarily because it is a dedicated graphics hardware processor. Burst drawing rates as high as 1 pixel every 800 ns can be achieved. Screen refreshing is handled directly by the controller. The displayed portion of the bit-map memory can be configured to allow the display to be scrolled through memory in any direction. The horizontal and sync periods both are fully programmable, as is the position of the sync pulse in the blanking interval. Furthermore, the controller can be programmed to refresh low-cost dynamic RAMs. In the design being considered, the 82720 offloads the microprocessor from low-level graphics tasks, as shown in Fig. 2.

For the bit-map interface, the memory is implemented as three planes, each 16 kwords by 16 bits, with each plane driving red, green, or blue (Fig. 3). The upper address bits—A_{16} and A_{17}—select the memory planes during read/modify/write cycles but are ignored during screen refreshing cycles.

The graphics display controller generates the Row Address Strobe (RAS) signal for the dynamic RAMs, but the remaining timing signals must be supplied by external devices. These signals are produced by a state-machine timing generator consisting of a 4-bit counter and two flip-flops. The state machine synchronizes itself with RAS after

portion of a frame.

To meet these requirements, it is helpful to break the bit map into three planes of 432 by 576 bits. While the screen is being refreshed, data is read from the same address in each of the three planes and sent serially to the screen. The memories can then be arranged as three 16-kword-by-16-bit arrays, requiring a memory cycle time of 800 ns and consequently permitting the use of relatively slow, low-cost 16-kbit dynamic RAM chips.

When drawing graphics figures, memory can be treated as a single large plane divided into three primary colors: red, green, and blue. Thus the low-order memory could represent the color red; the middle-order memory, green; and the high-order memory, blue. Each primary color requires the setting of just 1 bit/pixel. However, a secondary color—cyan, yellow, or magenta—necessitates setting 2 bits/pixel. Therefore, drawing in a secondary color takes two memory cycles/pixel and is slower than drawing primary colors. If this creates system problems, additional hardware can be used to draw more than one plane at a time. However, in the system example, drawing speeds are not only met, but also exceeded without relying on extra hardware.

Starting with the vertical refresh rate of 16.67 ms/frame, the basic timing can be analyzed. From the 16.67-ms figure, subtract the 1.25 ms required by the monitor for its vertical blanking. That leaves 15.42 ms for scanning the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives 33.5 μs/line, equivalent to a horizontal scan rate of 28 kHz.

Vertical retracing requires 7 μs/line, and the active portion of each line is 35.7 μs. Subtracting 7 μs from 35.7 μs leaves 28.7 μs. During this time 576 pixels are displayed, for a pixel period of 28.7 μs/576 or 49.8 ns. This corresponds to a dot clock rate of 20.07 MHz, which is chosen as the system's basic clock rate.

Display lists and commands pass from the I/O subsystem to a unit that executes the transformation tasks. Transformations are primarily mathematical operations performed on the display units. Depending on the command, this module edits display lists, organizes them for display on the screen, or edits the display-list data base. By editing a display list, objects in the physical coordinate system can be created, destroyed, moved, or changed. Transforming a display list into a form compatible with the display is necessary, as the data base can have an unlimited real-world coordinate system in three dimensions, but the CRT screen is limited to only two dimensions.

Transformation tasks place a heavy burden on a microprocessor. For instance, in a typical transformation, a matrix multiplication is performed for every point on an object's display list. In three dimensions, each point requires multiplying a 4-element vector by a 4-by-4 matrix. Some of the elements are always zero, but the operation still takes 13 multiplications and 9 additions. A two-dimensional display list requires 4 additions and 4 multiplications. A typical display can contain hundreds or thousands of lines, each of which has two end points. Therefore the speed of matrix multiplication significantly influences system performance.

The coordinate system supported by the design example is three-dimensional and employs 32-bit integers. The system CPU executes 32-bit integer matrix multiplications at high speed. In conjunction with the graphic display controller, the drawing task is offloaded from the CPU, which in turn maximizes the central-processor time that can be allocated for those matrix multiplications. Waiting time is now much lower than in conventional systems. However, if a system requires floating-point transformations, the best high-speed performance is achieved with the addition of a numerical coprocessor.

the 82720 has been initialized. Figure 4 shows the complete schematic for each plane of the bit-map interface.

The remainder of the hardware design interfaces the graphics display processor, the processor memory, and the other peripherals with the 80186 microprocessor. The task is simplified by the processor's on-board chip-selection logic and wait-state generators. Furthermore, because of the processor's highly integrated architecture, the size of the overall hardware is quite small.

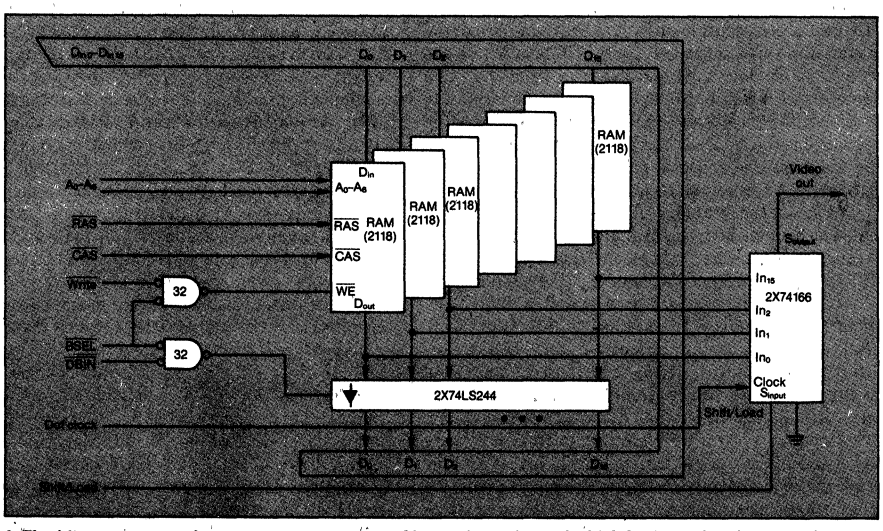
Joining processor and controller

Connecting the graphics display controller to the microprocessor is a simple task, as the processor's Data, Read, and Write signals are completely compatible with those of the 82720. However, because the controller has no chip-selection input, the Read or Write signals must be qualified through external hardware.

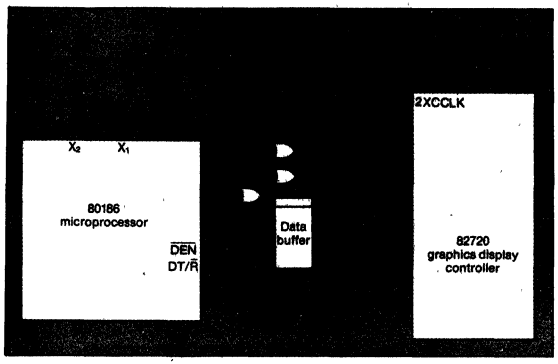
A number of chip-selection lines on the microprocessor can be programmed to place peripherals either in memory or in the processor's I/O space. Two gates are added to qualify the Read and Write signals. The DMA channel on the 80186 uses a second chip-select input as the Acknowledge signal, and data buffers are used to prevent bus contention at the end of a processor read cycle (Fig. 5).

Without buffers, the display controller must remove its data from the multiplexed address and data lines before the processor puts out the next address. At an 8-MHz clock rate, the processor requires that peripherals and memory vacate the bus in less than 85 ns; however, the standard speed of the controller is 100 ns. A faster version, the 82720-1, can be used, but it requires faster memory chips. A more cost-effective solution is simply adding buffers, if board space permits.

Serial communications to both the host and the optional tablet are handled by a multiprotocol serial controller (the 8274), which takes care of the host's synchronous and the tablet's asynchronous



4. The bit-map memory interface contains three address planes (one of which is shown here) to complete the graphics system. The $\overline{RAS}$ signal for the RAMs is generated by the graphics display controller.

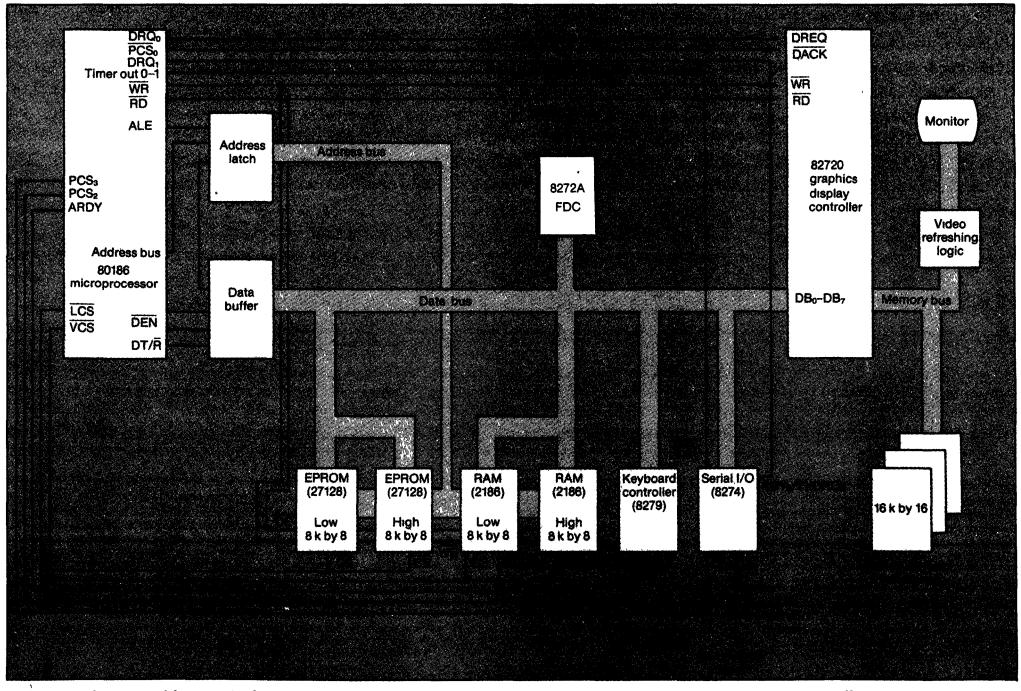


5. The interface between the 82720 and the system microprocessor is simple to implement because all of the processor's signals are compatible with the controller. It is necessary, however, to use external gates to qualify the RD or WR signals.

requirements. Interfacing is accomplished simply by connecting the buffered data bus, the latched lower-address lines, the Read and Write signals, and the chip-select. A final link brings the microprocessor's counter-timer output into the multiprotocol serial controller as a baud-rate clock. No buffering of the TTL support circuitry is necessary.

Universal chip interfaces keyboard

A universal peripheral interface chip (the UPI-42) serves as the keyboard interface and is programmed to scan the keyboard and interrupt the processor only on detection of a valid debounced keystroke. Mass-storage subsystems are managed by the 8272A floppy-disk controller. An external phase-locked loop circuit generates all of the timing signals reequired to connect a 5¼-in. drive to the system. On the microprocessor side, a DMA channel provides the link to the floppy-disk controller. Thus



6. A complete graphics control system is centered around an 80186 microprocessor and the 82720 controller. Local storage is provided by 32 kbytes of EPROM and 16 kbytes of RAM. The system comprises 85 chips and is housed on a single 12-by-12-in. printed-circuit board.

the processor has a high-speed disk interface, which loads it lightly.

To complete the graphics system illustrated in Fig. 6, 32 kbytes of EPROM and 16 kbytes of RAM support the microprocessor's program and display lists. The two EPROMs (27128s) come in 28-pin packages, thereby saving board space.

Hooking up the RAM chips is almost as straightforward. Since the microprocessor is a fully byte-addressable device, it can write bytes as well as words to the RAM. The chip-select input for the low (even) address RAM must be qualified with address A_0 at a logic zero, and the high (odd) address RAM must be qualified by the processor's Byte High Enable signal (BHE). The RAMs, designated 2186, have built-in controllers.

Since dynamic RAMs latch addresses on the leading edge of the chip-select signal, they must be qualified with the processor's Address Latch Enable signal to ensure that selection is made only after the address is valid. Then, a RAM latches the data to be written on the leading edge of the write pulse. The microprocessor's write signal must be delayed by one-half of a clock cycle to guarantee that data is valid at the correct time.

At this point, the design meets all of its performance goals. The system draws lines and circles at about 120,000 bits/s. That is approximately 82,000 pixels for a display consisting of even amounts of the three primary colors, as well as three secondary colors, and white. The 500 vectors of 25 pixels each can be drawn in about 0.15 s, six times faster than the 1-s requirement. The worst case—drawing all lines in white—can be accomplished in about one-third of a second. These specifications are satisfied when the graphics display controller is running from a 2.5-MHz clock. Drawing is performed only during retracing and the 82720 is programmed to use three memory cycles of each horizontal retrace for memory refreshing.

All of the components fit on a board measuring 12 by 12 in., so that the desktop size requirements are satisfied. The electronic components occupy about 100 in.^{2} of the low-cost, double-sided printed-circuit board. □

Bibliography:

Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," IBM System Journal, 1965, 4(1) pp. 25-30.

intel

ARTICLE REPRINT

AR-298

Graphics Chip Makes Low-Cost, High Resolution Color Displays Possible

by Mark Olson and Brad May

The making of displays that are both high-resolution and low-cost is the key to producing equipment for both the automated office and the engineering workstation. Through the introduction of 16-bit μPs such as Intel's iAPX 8088, 80186 and 80286, the processing power has been made available to perform very sophisticated functions for the user while making the human interface very simple.

That processing power can be unnecessarily drained, however, if the μP is burdened with the entire task of graphics display. Such a burden can fill up a significant part of the processor's I/O bandwidth, slow down the refresh rate of the display, and decrease the computational power of the CPU.

Intelligent peripheral ICs offload processing tasks from the CPU.

The logical way to avoid such limitations is to dedicate a specialized processor to the handling of display function. It should be capable of accepting high-level commands to minimize the burden on the CPU, as well as optimizing the execution of such commands through raster operations implemented in hardware at the device level.

Such a chip is Intel's 82720 Graphics Display Controller (GDC). It has features that give systems a fast drawing speed while reducing graphics display costs by 60% or more. It achieves these results by taking over the drawing and refresh functions from the CPU, by allowing the use of dynamic RAM's instead of static RAM's, and by reducing the overall parts count needed to create a complete graphics system.

The implementation of the drawing task is a major feature of the GDC. Other graphics chips perform only the display refresh function, leaving the more complicated drawing function entirely to the CPU. Since the CPU is doing every pixel of the drawing function on these systems, they also require faster bit map RAM than with the GDC. The GDC, on the other hand, is capable of handling the drawing function itself, drawing such objects as characters, slanted characters, points, lines, arcs, rectangles, and slanted rectangles based only upon lengths, slopes, and arc centers supplied by the CPU. The GDC's processing, moreover, takes place concurrently with the processing of the CPU.

2048 × 2048 Resolution

With its 4 Megapixel addressability, one GDC can handle a monochrome display with resolution as high as 2048 × 2048, and multiple GDC's can be linked to provide even higher resolution, such as color displays at 2048 × 2048. The chances are, however, that the GDC's full power will not be used in most applications. The typical

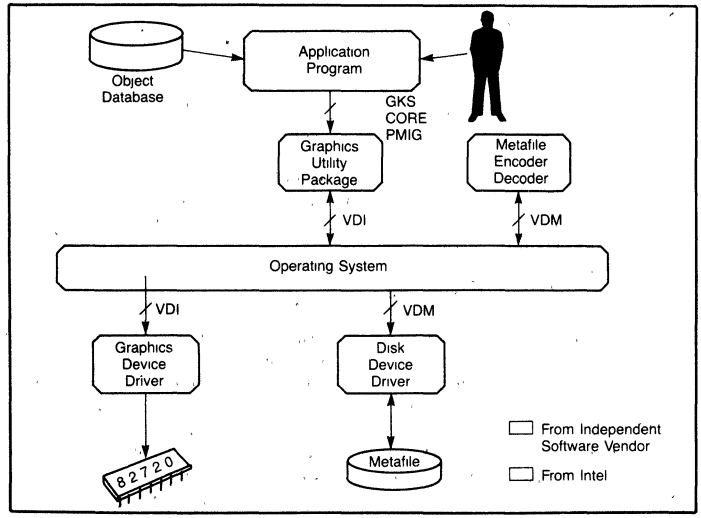


Figure 1: General graphics commands are translated into the VDI interface level and then into driver device commands.

Mark Olson and Brad May are Product Marketing Engineers for Peripheral Components Operation, Intel Corp., Santa Clara, CA 95051.

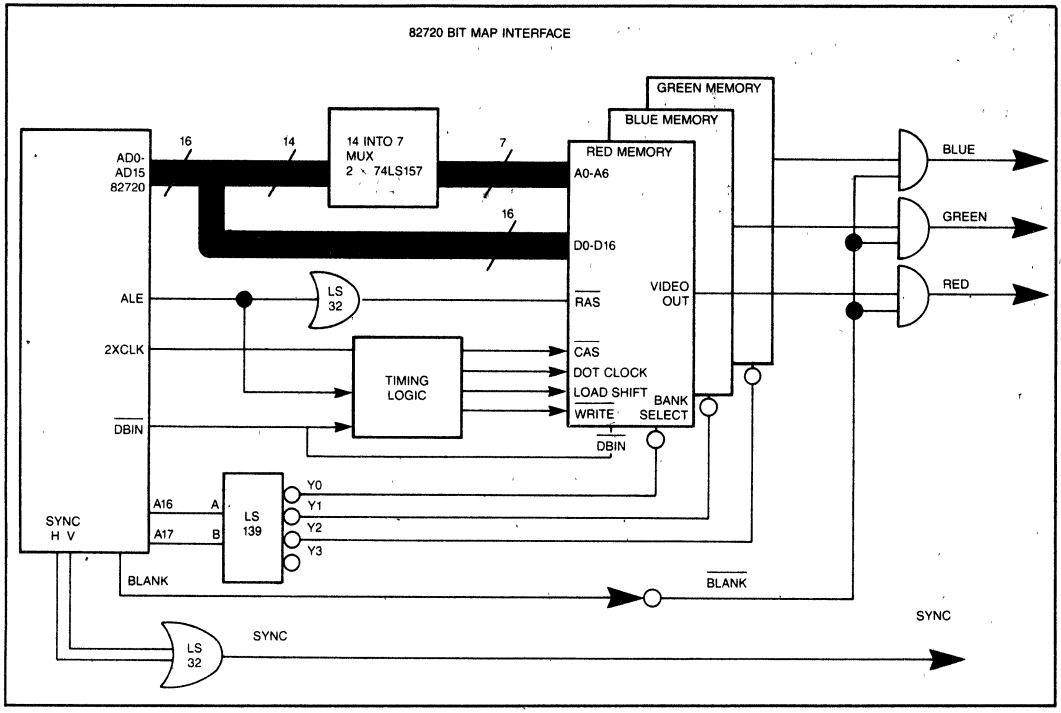


Figure 2: The memory is broken up into three planes, with each plane feeding one of the primary color guns of the CRT.

product considered high resolution for office automation applications is a 512 × 512 pixel monochrome or color display.

These latter restrictions are not imposed by the GDC, but rather have more to do with the cost of display monitors, the amount of RAM memory needed to support such displays, and the adequacy of such displays for most applications. It is possible to build "super graphics" boards with a GDC, such as the 1K by 1K pixel by 8 color plane graphics display designed by Phoenix Computer Graphics (Lafayette, LA). Such a display is capable of rendering 256 different colors on a high resolution screen.

Even higher performance can be achieved through the use of multiple GDC's to support multiple display windows, increased drawing speed, or increased bits per pixel. For multiple display windows, each GDC can be used to control one window of the display. For increased drawing speed, multiple GDC's can be operated in parallel. For increased bits/pixel, each GDC can contribute a portion of the number of bits necessary for a pixel.

Although the GDC is intended primarily for raster-scan graphics, it can also be used as a character display controller. It is capable of supporting up to four screens of data containing 25 rows by 80 columns, or one screen containing up to 100 rows by 256 characters.

Office Automation Display

High performance applications can stretch the usage of the GDC from low-end to high-end engineering displays, but research has shown that for office automation products, a 512 × 512 pixel display is quite acceptable, and that color is often a requirement. These requirements mesh with a major factor in display—the cost of the CRT. In OEM quantities, for example, one could expect to find a 512 × 512 monochrome display for under $100, a 256 × 256 color display (TV quality) for about $150, a 512 × 512 color CRT in the $300 range, and a 1K × 1K color display in the $800–$1000 category.

To give an example of the type of display that can be built for new office products using the GDC, consider a 512 × 512 pixel by 3 color plane combination CPU and graphics display on a single 12″ by 12″ board. Such a display is capable of generating 8 colors.

The list of parts (**Table 2**) comes to about $175 for 85 IC's taking up 104 square inches of board space. Even that parts count could be reduced by replacing the 48 16K DRAMs with 12 64K DRAMs—if a 4K × 16 bit DRAM were available. A very important note about the parts list is that the design is implemented with inexpensive 2118 dynamic RAMs. The design does

not require the faster, more expensive, and less dense static RAMs.

The parts count is low enough so that the processor and graphics controller can be placed together in a single 12″ by 12″ board. This is important because small overall size and footpad are selling points for desktop workstations. System speed is also enhanced when the graphics controller and CPU are on the same board, because their communication need not take up bus, inter-board bandwidth or experience any additional delays.

Pipelining Transformations

More important than putting the graphics display on the same board as the CPU is the level of communication between the CPU and graphics controller. If the burden of transformation processing is left entirely to the CPU while the graphics chip is used only as a CRT controller, then the CPU must communicate one bit per pixel to update a display. With the GDC, the CPU input takes higher level forms such as the slope and length of a line, the length and center point of an arc, or the key coordinates of a rectangle. Since the average line on a screen is about 25 pixels, that means that 25 times fewer CPU bus cycles are required to draw a graphical object with the GDC. These CPU cycles (an average of 50 μs each to calculate the graphical object and communicate it to the GDC) are the determining factor in drawing rate.

Viewed from a larger perspective, there are four tasks that must be performed by a CPU-graphics chip combination:

(1.) The CPU must calculate the higher-level graphics operations. This is done by the CPU and it involves the processing of macro-operations such as the CORE, GKS, PMIG or other graphics protocols. These general graphics commands are translated into an intermediate level, the VDI interface level (**Figure 1**) and then into device driver commands by software in the CPU.

(2.) Then, these lower-level graphical objects such as the key parameters for lines, arcs, characters, and rectangles, must be trans-

VLSI Takes Aim At Text Processing

The concept of co-processing is not a new one. Intended as a way of offloading computationally intensive tasks from a host CPU, it has been around at Intel since the introduction of the 8087 numerics processor and the 8089 I/O machine. A more recently developed product, the 82720 Graphics Display Controller is designed to bolster system performance by offloading graphics control chores from the CPU. The chip accepts high level commands from the CPU and, using its own drawing processor, accesses the required positions in the bit-map and handles the processing and display control functions.

Building on the success of these parts come two new co-processors designed to partition system intelligence even further. The 82586 is a communications co-processor designed to bridge the characteristics of CPU and network data rates. Its FIFO buffer and DMA facilities make it possible for a CPU to operate at the full Ethernet 10 Mbits/s transfer rate even in the face of continuous bursts of network data traffic.

Intel's most recent introduction is the 82730 text co-processor. Printers and other hard copy peripherals have supported additional text processing features such as proportional spacing and simultaneous superscript and subscript for some time. Implementing these features on the display screen has traditionally been a costly procedure. Thus, it is typically not done and screen displays often are not identical to their hard-copy printouts. Aimed to solve this designers headache, the 82730 has its own DMA capability and communicates asynchronously with the CPU via shared memory messages. It supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. In addition, when coupled with the 82720 Graphics Display Controller (**Figure 1**) the 82730 provides flexible mixing of text and graphics simultaneously on the same display.

—*Wilson*

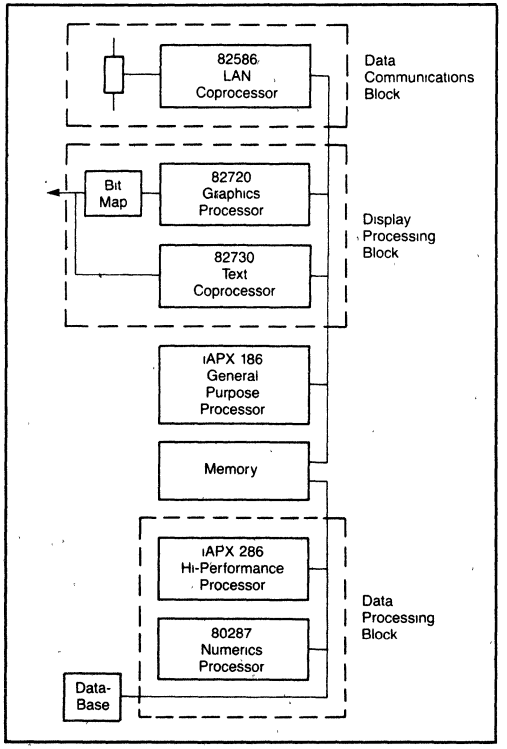


Figure 1: Offloading system tasks is simplified by new VLSI devices.

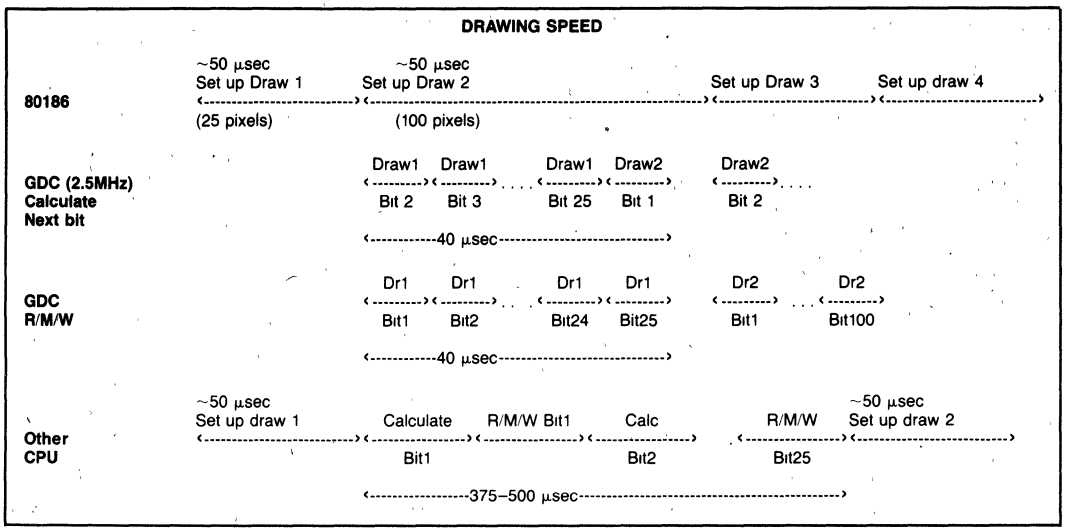


Table 1: The 80186 and the GDC work together to accomplish the drawing function.

formed into changes in the actual bits. This function is performed in hardware in the GDC concurrently with any level one processing done by the CPU. Other graphics controllers leave this task to the CPU to execute in software. The contrast is that, in such systems, the CPU must resolve the graphical object down to every point on a line, while with the GDC it need only designate the endpoints.

(3.) With the actual bits for the bit map calculated, they must be placed in the bit map memory. This involves a read-modify-write operation that requires three CPU cycles using other methods. With the GDC these operations are not the responsibility of the CPU. The GDC pipelines its execution so that it is calculating the next bit to change while it is executing the read-modify-write cycles.

(4.) Finally, the bit map memory must be dumped into the CRT. This is the refresh function performed by other graphics chips as well as the GDC.

The summation is that other systems require the CPU to process steps one to three serially, leaving only step four for the graphics controller. Systems with the GDC require the CPU to process only step one, with the GDC concurrently processing steps two through four. The GDC has another advantage in that during the transformation process in step three, the GDC executes the algorithms in hardware while a CPU must execute the algorithms in software. The algorithms are exactly the same in both cases. They are the Bresenam algorithms from IBM, in which the next pixel to be drawn becomes a binary decision between two pixels.

The execution of these algorithms is a crucial drawing time factor, because they are invoked many times for each updated screen. Consider that, in the inner loop of Bresenam's "line drawing algorithm," there are two or three additions, two comparisons or tests, and the masking of the proper value into the word for each pixel. The algorithms for drawing circles or filling areas are even more complex. In the inner loop of a fill algorithm, the old word must be read from the bit map, then tested to see if all, some, or none of the pixels are within the area to be filled. Next, it tests whether some or all of the pixels must be modified. Finally, the word must be returned to the bit map.

These algorithms are heavily used and the speed with which they can be executed has a direct effect upon the overall system efficiency. If they must be executed by a μP, the instruction fetching process slows down the calculations to a drawing rate of 15–20 μs per pixel. With a hardware implementation of these algorithms in the GDC, the calculations can be speeded up to achieve a drawing rate of 1600 ns (2.5 MHz version) or 800 ns (5 MHz version) per pixel.

Methods Of Refresh

In the fourth step, the dumping of bit map memory into the CRT, there are some differences between graphics controller chips. Motorola's MC6845 CRT controller, for example, uses a split-cycle refresh method in which each refresh cycle is alternated with a drawing cycle in which the μP updates the bit map. This gives the MC6845 a 50% drawing bandwidth.

With the GDC there are two drawing modes. The first is a "draw anytime" mode which replaces CRT refresh cycles with drawing cycles. This is the fastest mode, but it does result in on-screen disruptions. The second mode, which does not disrupt the on-screen display, draws only during the vertical and horizontal retracing periods. This gives the GDC about a 25%

1	80186	1	74LS04	1	20 MHz Clock
1	82720	1	74LS73	2	27128
2	74LS157	9	74LS244	2	2186
1	74LS139	8	74LS166	1	8274
1	74LS161	3	74LS32	1	8042
1	74LS11	2	8286	3	Connectors
1	74LS00	1	8 MHz Crystal	1	12×12 2 Layer PC

SUMMARY:

4 VLSI Controllers	80186	Processor
	82720	Graphics
	8274	Serial Link
	8042	Keyboard
4 VLSI Memory	27128	EPROM
	2186	IRAM
4816K DRAMs	2118	DRAM
29 MSI/SSI	—	Buffers/Glue

TOTAL: 85 IC'S →104 Sq. Inches

Parts Cost →About $175

Table 2: Parts list for 512×512×4 Color Display.

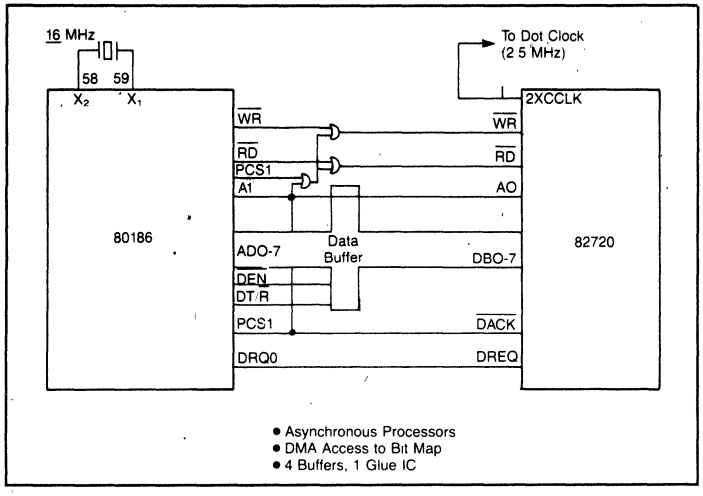


Figure 3: The two chip selects are OR'd together to qualify the R/W signals.

drawing bandwidth. At first glance that gives the GDC a disadvantage in drawing rate, but the fact is, with its pipelining and hardware execution of transformations, the GDC makes much more efficient use of its bandwidth. The critical timing factor is the amount of CPU participation in the drawing process, not the refresh bandwidth of the graphics controller. Another tradeoff is that, with its split-cycle architecture, the MC6845 requires RAM memory that is twice as fast as that required by the GDC in the same application.

Inexpensive RAM Is Fast Enough

Applying this perspective, one can begin to build the display with parts listed in **Table 2**. First one notes that a square display, as indicated by the 512×512 pixel initial specification, is not pleasing to the eye. It is much more appealing to have an aspect ratio of about 4:3, in which the number of pixels horizontally is 4/3 the number vertically. If the resolution is such that the total number of pixels is not a power of two, it will be necessary to round up to the next power of two and waste the extra bits.

The pixel arrangement which best meets this requirement is one with a 432 × 576 pixel format. It also meets the requirement that the number of pixels horizontally be an even number of 16-bit words. With three color bits per pixel (red, blue, and green), the total display memory is then about 500×500×3, or 750k bits.

It makes the most sense to break the memory up into three planes, with each plane feeding one of the primary color guns of the CRT (**Figure 2**). This leads to a memory arrangement of 16K × 16 × 3, using 16K dynamic RAMs with a 1K × 16 architecture. When drawing graphics figures, the memory can be treated as one large plane, split into the three primary colors. Drawing in low-order memory could represent red, middle-order could be used for green, and high-order for blue.

One advantage of this 3D memory is that drawing with a primary color requires setting only one bit per pixel. Drawing with a secondary color such as cyan, yellow, or magenta would take two GDC cycles, and creating white from all three colors would take three GDC cycles. If this were an issue, additional hardware could be used to draw more than one plane at a time. As the results will show, however, the drawing speed requirements can be exceeded without any added hardware.

Calculate The Drawing Rate

To see if the proposed design is practical, one should first calculate the drawing rate to see what the user interface will be like. Then one should check the refresh rate to make sure the design is uninterrupted and without flicker.

The proof of the assumption that CPU participation is the dominating factor lies in the 50 μs average time that it takes the CPU to calculate a graphical object and communicate its key parameters to the GDC. Assume that the graphical object is an average line containing

25 pixels, and that there are about 500 vectors on the average screen display.

The GDC's normal clock rate is 2.5 MHz, giving it a 400 ns period (the maximum clock rate is 5 MHz, with a 200 ns period.) It takes four GDC cycles to execute a read-modify-write on a bit (because two read cycles are required), so that the GDC's normal drawing rate is one pixel per 1600 ns. To draw the 25 pixels involved in the average line, then, would take 25 × 1600 ns, or 40 μs. Since this operation is done concurrently with CPU processing, the GDC will be waiting for the next graphical object by the time the CPU is ready.

If the screen were filled with nothing but 25-pixel vectors, then the drawing rate would be determined by the 50 μs average CPU calculation and transfer cycle, averaging about 2 μs per pixel. If all the vectors were white (worst case), then it would take 1.5 secs of drawing time to update the white screen. Since, in the undisturbed-screen mode, drawing is only done during the 25% of the time that the screen is undergoing horizontal or vertical blanking, this would mean 6 secs between updates.

In reality, however, the screen will not be filled with vectors. It will have an average of 500 vectors, and the color distribution could be presumed to be evenly distributed as one-third primary colors, one-third secondary colors, and one-third white. The 500 vectors will require the drawing of 12.5K pixels in monochrome, or 25K pixels with distributed colors. At a drawing rate of 2 μs per pixel, this takes 50 ms to draw. Drawing only during blanking, the screen would be updated in 200 ms.

Under these conditions, it would not help to use the maximum clock rate GDC (5 MHz), but if in some applications the average vector length is 100 pixels, then the CPU calculation-and-bus cycle (50 μs) would remain the same and the GDC's drawing cycle (1600 ns × 100 = 160 μs) would become a limiting factor. Using the 5 MHz GDC would cut that drawing time down to 800 ns/pixel, or 80 μs/vector. The 500 vector average screen would then contain 100K pixels with distributed colors and could be drawn in 80 ms. Multiplying by four because the drawing is done during blanking (25% of the time), that is 320 ms. That is a screen update in less than one-third second for a "busy" screen.

Calculate The Refresh Rate

These calculations are of little importance if the display flickers due to lack of refresh. This exercise is actually a demonstration of how the basic GDC clock rate was derived. Assume a non-interlaced display that must be refreshed 60 times per second. That gives a screen refresh rate of 16.67 ms, but on a typical CRT some 4.27 ms of that is blanked, leaving 12.4 ms of active display time. The dot sweep period is the 12.4 ms divided by the number of pixels (432 × 576 = 248.8K), or 49.8 ns. The inverse gives a 20.07 MHz dot clock.

Since the GDC dumps 16 bits from the bit map memory into the

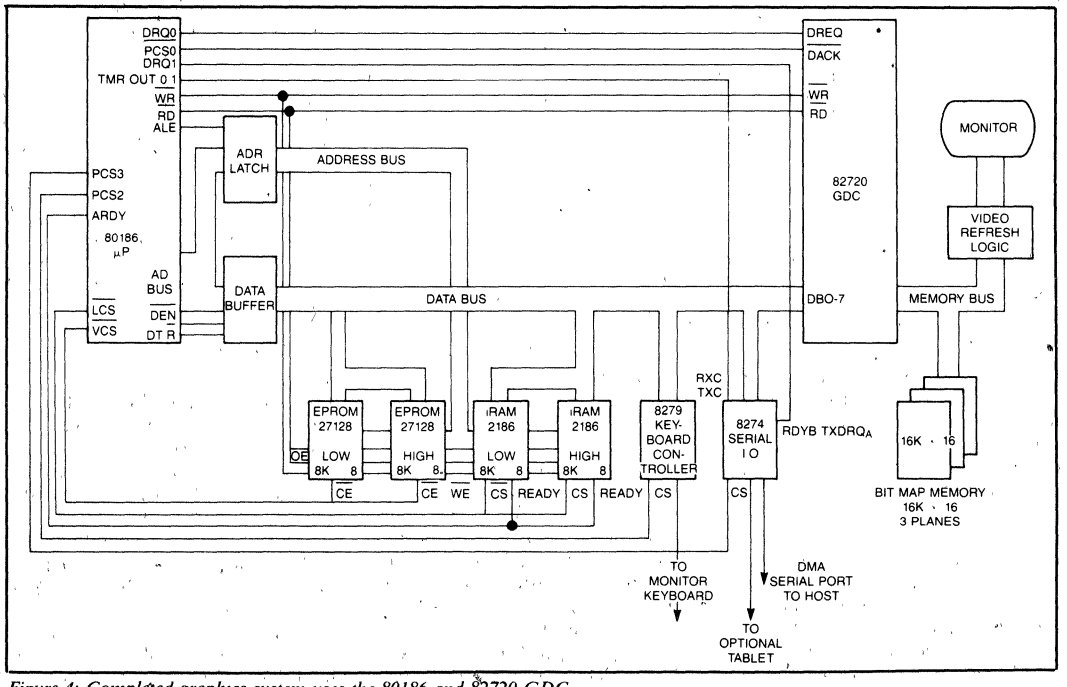


Figure 4: Completed graphics system uses the 80186 and 82720 GDC.

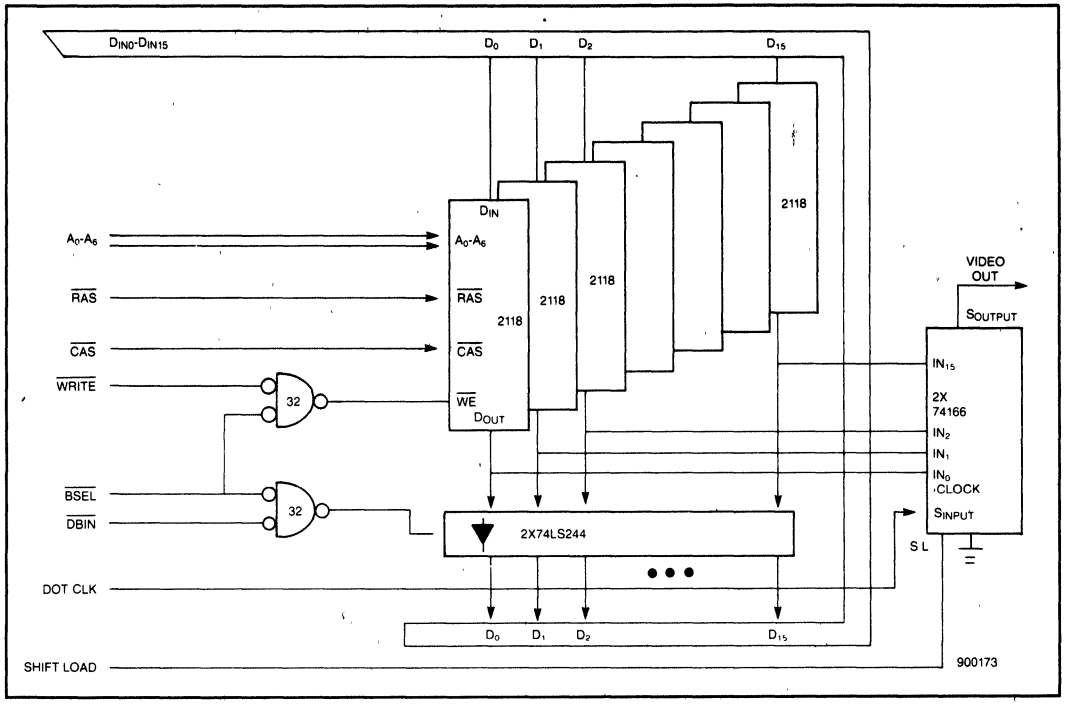


Figure 5: Since the 186 is a fully byte addressable machine, it is possible to write bytes as well as words into the RAMs.

16-bit shift register during each read, and since the shift register then feeds these bits out serially to the CRT, it makes sense that the GDC's read period should be 16 times the dot sweep period. That gives a GDC read period of about 800 ns. With each GDC read taking two cycles, the basic GDC clock period is then 400 ns, or 2.5 MHz. This gives a rock-solid display, and one would only want to go to the 5 MHz GDC to improve drawing rate.

For those who want to examine the blanking intervals to see if the CRT is indeed "typical," the blanking can be further broken down. The vertical blanking interval is 1.25 ms, leaving 15.42 ms to scan the 432 lines on the active portion of the display. Dividing 15.42 ms by 432 lines gives a 35.7 μs period per line, or a horizontal sweep rate of 28 KHz. Time is also needed for horizontal retrace, in this case, 7 μs of horizontal blanking per line. This leaves 28.7 μs to scan the 576 pixels on each line, resulting in the dot sweep period of 49.8 ns. Using a 20 MHz CRT helps keep the costs down, but the GDC can use CRT displays as fast as 80 MHz when higher resolution is required.

Mixed Mode

While it is possible to generate 8×8 characters and slanted characters in the graphics mode, the GDC also offers a mixed mode memory organization to display both characters and graphics drawn from separate windows in the display memory. The advantage of this mode is that it allows characters to be manipulated as 8-bit entities instead of the 64 bits that each would require in graphics mode. Of necessity, the graphics window display memory is reduced in this mode (64K 16-bit words instead of 256K), but even the reduced maximum graphics memory is still a megapixel and quite sufficient for both office automation and engineering display purposes.

In the character window, the GDC operates as it does in the pure character mode, with the exception that the line counter must be implemented externally. In addition to the two windows used for graphics and characters in the mixed mode, two other windows can be supported. These can be designated as either character or graphics windows by a selection on the A17 line.

Panning, Zooming, Light Pen

As special features, the GDC allows both panning and zooming in either graphics, character, or mixed modes. The zoom is accomplished by effectively increasing the size of the dots on the screen. Vertically, this is done by repeating the same display line. The number of repeat times is determined by the display zoom parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor.

82720
GRAPHICS DISPLAY CONTROLLER

- **Displays Low-to-High Resolution Images**
- **Draws Characters, Points, Lines, Arcs, and Rectangles**
- **Supports Monochrome, Gray Scale, or Color Displays**
- **Zooms, Pans and Windows Through a 4 Mpixel Display Memory**
- **Extremely Flexible Programmable Screen Display, Blanking, and Sync Formats**
- **Compatible with Intel's Microprocessor Families**
- **High-Level Commands Off Load Host Processor from Bit Map Loading and Screen Refresh Tasks**
- **Supports Graphics, Character, and Mixed Display Modes**

FUNCTIONAL DESCRIPTION

Introduction

The 82720 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to drive high-performance raster-scan computer graphics and character CRT displays. Positioned between the video display memory and Intel microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory. Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory directly supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words. The display can be zoomed and partitioned screen areas can be independently scrolled and panned. With its light pen input and multiple controller capability, the GDC is ideal for most computer graphics applications. Systems implemented with the GDC can be designed to be compatible with standards such as VDI, NAPLPS, GKS, Core, or custom implementations.

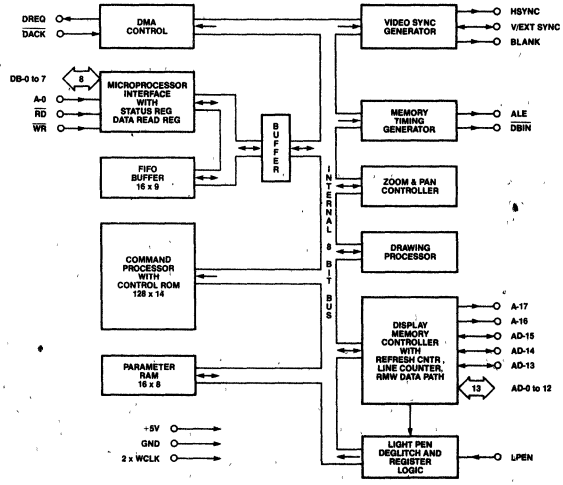


Figure 1. Block Diagram

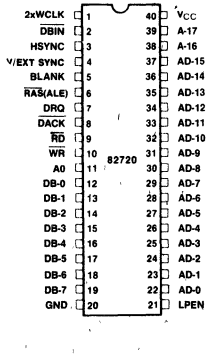


Figure 2. Pin Configuration

JUNE 1983
ORDER NUMBER: 210655-002

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Description
2XWCLK	1	I	**Clock Input**
$\overline{DBIN}$	2	O	**Display Bus Input:** Read strobe output used to read display memory data into the GDC.
HSYNC	3	O	**Horizontal Sync:** Output used to initiate the horizontal retrace of the CRT display.
V/EXT SYNC	4	I/O	**Vertical Sync:** Output used to initiate the vertical retrace of the CRT display. In slave mode, this pin is an input used to synchronize the GDC with the master raster timing device.
BLANK	5	O	**Blank:** Output used to suppress the video signal.
$\overline{RAS}$ (ALE)	6	O	**Row Address Strobe (Address Latch Enable):** Output used to start the control timing chain when used with dynamic RAMs. When used with static RAMs, this signal is used to demultiplex the display address/data bus.
DRQ	7	O	**DMA Request:** Output used to request a DMA transfer from a DMA controller (8237) or I/O processor (8089).
$\overline{DACK}$	8	I	**DMA Acknowledge:** Input used to acknowledge a DMA transfer from a DMA controller or I/O processor.
$\overline{RD}$	9	I	**Read:** Input used to strobe GDC Data into the microprocessor.
$\overline{WR}$	10	I	**Write:** Input used to strobe microprocessor data into the GDC.
A0	11	I	**Register Address:** Input used to select between commands and data read or written.
DB0	12	I/O	**Bidirectional Microprocessor Data Bus Line:** Input enabled by $\overline{WR}$. Output enabled by $\overline{RD}$.
DB1	13		
DB2	14		
DB3	15		
DB4	16		
DB5	17		
DB6	18		
DB7	19		
GND	20		**Ground.**
V_{CC}	40		**+5V Power Supply**
A_{17}	39	O	**Graphics Mode:** Display Address Bit 17 Output **Character Mode:** Cursor and Line Counter Bit 4 Output **Mixed Mode:** Cursor and Image Mode Flag
A_{16}	38	O	**Graphics Mode:** Display Address Bit 16 Output **Character Mode:** Line Counter Bit 3 Output **Mixed Mode:** Attribute Blink and Line Counter Reset
AD_{15}	37	I/O	**Graphics Mode:** Display Address/Data Bits 13–15 **Character Mode:** Line Counter Bits 0–2 Output **Mixed Mode:** Display Address/Data Bits 13–15
AD_{14}	36		
AD_{13}	35		
AD_{12}	34	I/O	**Display Address/Data Bits 0–12**
AD_{11}	33		
AD_{10}	32		
AD_9	31		
AD_8	30		
AD_7	29		
AD_6	28		
AD_5	27		
AD_4	26		
AD_3	25		
AD_2	24		
AD_1	23		
AD_0	22		
LPEN	21	I	**Light Pen Detect Input**

FUNCTIONAL DESCRIPTION (Continued)

Microprocessor Bus Interface

Control of the GDC by the system microprocessor is achieved through an 8-bit bidirectional interface. The status register is readable at any time. Access to the FIFO buffer is coordinated through flags in the status register.

Command Processor

The contents of the FIFO are interpreted by the command processor. The command bytes are decoded, and the succeeding parameters are distributed to their proper destinations within the GDC. The bus interface has priority over the command processor when both access the FIFO simultaneously.

DMA Control

The DMA Control circuitry in the GDC coordinates data transfers when using an external DMA controller. The DMA Request and Acknowledge handshake lines interface with an 8257 or 8237 DMA controller or 8089 I/O processor, so that display data can be moved between the microprocessor memory and the display memory.

Parameter RAM

The 16-byte RAM stores parameters that are used repetitively during the display and drawing processes. In character mode, the RAM holds the partitioned display area parameters. In graphics mode, the RAM also holds the drawing pattern and graphics character.

Video Sync Generator

Based on the clock input, the sync logic generates the raster timing signals for almost any interlaced, non-interlaced, or "repeat field" interlaced video format. The generator is programmed during the idle period following a reset. In video sync slave mode, it coordinates timing between the GDC and another video source.

Memory Timing Generator

The memory timing circuitry provides two memory cycle types: a two-clock period refresh cycle and the read-modify-write (RMW) cycle which takes four clock periods. The memory control signals needed to drive the display memory devices are easily generated from the GDC's $\overline{\text{RAS}}$(ALE) and $\overline{\text{DBIN}}$ outputs.

Zoom and Pan Controller

Based on the programmable zoom display factor and the display area parameters in the parameter RAM, the zoom and pan controller determines when to advance to the next memory address for display refresh and when to go on to the next display area. A horizontal zoom is produced by slowing down the display refresh rate while maintaining the video sync rates. Vertical zoom is accomplished by repeatedly accessing each line a number of times equal to the horizontal repeat. Once the line count for a display area is exhausted, the controller accesses the starting address and line count of the next display area from the parameter RAM. The system microprocessor, by modifying a display area starting address, allows panning in any direction, independent of the other display areas.

Drawing Processor

The drawing processor contains the logic necessary to calculate the addresses and positions of the pixels of the various graphics figures. Given a starting point and the appropriate drawing parameters, the drawing processor needs no further assistance to complete the figure drawing.

Display Memory Controller

The display memory controller's tasks are numerous. Its primary purpose is to multiplex the address and data information in and out of the display memory. It also contains the 16-bit logic units used to modify the display memory contents during RMW cycles, the character mode line counter, and the refresh counter for dynamic RAMs. The memory controller apportions the video field time between the various types of cycles.

Light Pen Debouncer

Only if two rising edges on the light pen input occur at the same point during successive video fields are the pulses accepted as a valid light pen detection. A status bit indicates to the system microprocessor that the light pen register contains a valid address.

System Operation

The GDC is designed to work with Intel microprocessors to implement high-performance computer graphics systems. System efficiency is maximized through partitioning and a pipelined architecture. At the lowest level, the GDC generates the basic video

raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory address are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters. At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored and communicated. The GDC takes care of the high-speed and repetitive tasks required to implement graphics systems.

GENERAL OVERVIEW

In order to minimize system bus loading, the 82720 uses a private video memory for storage of the video image. Up to 512K bytes of video memory can be directly supported. For example, this is sufficient capacity to store a 2048 × 2048 pixel × 1 bit image. Images can be generated on the screen by:

—Drawing Commands
—Program-Controlled Transfers
—DMA Transfers from System Memory

The 82720 can be configured to support a wide variety of graphics applications. It can support:

—High Dot Rates
—Color Planes
—Horizontal Split Screen
—Character-oriented Displays
—Multiplexed Graphic and Character Display

GRAPHIC DISPLAY CONFIGURATIONS

The 82720 provides the flexibility to handle a wide variety of graphic applications. This flexibility results from having its own private video memory for storage of the graphics image. The organization of this memory determines the performance, the number of bits/pixel and the size of the display. Several different video memory organizations are examined in the following paragraphs.

In the simplest 82720 system, the memory can store up to a 2048 × 2048 × 1 bit image. It can display a 1024 × 1024 × 1 bit section of the image at a maximum dot rate of 44 MHz, or 88 MHz in wide mode. In this configuration, only 1 bit/pixel is used.

By partitioning the memory into multiple banks, color, gray scale and higher bandwidth displays can be supported. By adding various amounts of external logic, many cost/performance tradeoffs for both display and drawing are realizable.

The video memory can be partitioned into 4 banks, each 1024 x 1024 bits. By selecting all 4 memory banks during display, 4 bits/pixel can be provided by a single 82720. Each bank of video memory contributes 1 bit to each pixel. This configuration can support color monitors, again with a maximum dot shift rate of 44 or 88 MHz.

Higher performance may be achieved by using multiple 82720s. Multiple 82720s can be used to support mutliple display windows, increased drawing speed, or increased bits per pixel. For display windows, each 82720 controls one window of the display. For increased drawing speed, multiple 82720s are operated in parallel. For increased bits/pixel, each 82720 contributes a portion of the number of bits necessary for a pixel.

CHARACTER DISPLAY CONFIGURATION

Although the 82720 is intended primarily for raster-scan graphics, it can be used as a character display controller. The 82720 can support up to 8K by 13 bits of private video memory in this configuration (1 character = 13 bits). This is sufficient memory to store 4 screens of data containing 25 rows by 80 characters. The 82720 can display up to 256 characters per row. Smooth vertical scrolling of each of 4 independent display partitions is also supported.

MIXED DISPLAY CONFIGURATION

The GDC can support a mixed display system for both graphic and character information. This capability allows the display screen to be partitioned between graphic and character data. It is possible to switch between one graphic display window and one character display window with raster line resolution. A maximum of 256K bytes of video memory is supported in this mode: half is for graphic data, half is for character data. In graphic mode, a one megapixel image can be stored and displayed. In character mode, 64K, 16-bit characters can be stored.

DETAILED OPERATIONAL DESCRIPTION

The GDC can be used in one of three basic modes —Graphics Mode, Character Mode and Mixed Mode. This section of the data sheet describes the following for each mode:

1. Memory organization
2. Display timing
3. Special Display functions
4. Drawing and writing

Graphics Mode Memory Organization

The Display Memory is organized into 16-bit words (32-bit words in wide mode). Since the display memory can be larger than the CRT display itself, two width parameters must be specified: display memory width and display width. The Display width (in words) is selected by a parameter of the Reset command. The Display memory width (in words) is selected by a parameter of the Pitch command. The height of the Display memory can be larger than the display itself. The height of the Display is selected by a parameter of the Reset command. The GDC can directly address up to 4Mbits (0.5Mbytes) of display RAM in graphics mode.

Graphics Mode Display Timing

All raster blanking and display timings of the GDC are a function of the input clock frequency. Sixteen or 32 bits of data are read from the RAM and loaded into a shift register in each two clock period display cycle. The Address and Data busses of the GDC are multiplexed. In the first part of the cycle, the address of the word to be read is latched into an external demultiplexer. In the second part of the cycle the data is read from the RAM and loaded into the shift register. Since all 16 (32) bits of data are to be displayed, the dot clock is 8 × (16 ×) the GDC clock or 16 × (32 ×) the Read cycle rate.

Parameters of the Reset or Sync command determine the horizontal and vertical front porch, sync pulse, and back porch timings. Horizontal parameters are specified as multiples of the display cycle time, and vertical parameters as a multiple of the line time.

Another Reset command parameter selects interlaced or non-interlaced mode. A bit in the parameter RAM can define Wide Display Mode. In this mode, while data is being sent to the screen, the display address counter is incremented by two rather than one. This allows the display memory to be configured to deliver 32 bits from each display read cycle.

The V Sync command specifies whether the V Sync Pin is an input or an output. If the V Sync Pin is an output, the GDC generates the raster timing for the display and other CRT controllers can be synchronized to it. If the V Sync pin is an input, the GDC can be synchronized to any external vertical Sync signal.

Graphics Mode Special Display Functions:

WINDOWING
The GDC's Graphics Mode Display can be divided into two windows on the screen, upper and lower. The windows are defined by parameters written into the GDC's parameter RAM. Each window is specified by a starting address and a window length in lines. If the second window is not used, the first window parameters should be specified to be the same as the active display length.

ZOOMING
A parameter of the GDC's zoom command allows zooming by effectively increasing the size of the dots on the screen. This is accomplished vertically by repeating the same display line. The number of times it is repeated is determined by the display zoom factor parameter. Horizontally, zoom is accomplished by extending each display word cycle and displaying fewer words per line, according to the zoom factor. It is the responsibility of the microprocessor controlling the GDC to provide the shift register clock circuitry with the zoom factor required to slow down the shift registers to the appropriate speed. The frequency of the 2XWCLK should not be changed. The zoom factor must be set to a known state upon initialization.

PANNING
Panning is accomplished by changing the starting address of the display window. In this way, panning is possible in any direction, vertically on a line by line basis and horizontally on a word by word basis.

Graphics Mode Drawing and Writing

The GDC can draw solid or patterned lines, arcs, circles, rectangles, slanted rectangles, characters, slanted characters, filled rectangles. Direct access to the bit map is also provided via the DMA Commands and the Read or Write data commands.

MEMORY MODIFICATION
All drawing and writing functions take place at the location in the display RAM specified by the cursor. The cursor is not displayed in Graphics Mode. The cursor location is modified by the execution of drawing, reading or writing commands. The cursor will move to the bit following the last bit accessed.

Each bit is drawn by executing a Read-Modify-Write cycle on the display RAM. These R/M/W cycles normally require four 2XWCLK cycles to execute. If the display zoom factor is greater than two, each R/M/W cycle will be extended to the width of a display cycle. Write Data (WDAT), Read Data (RDAT), DMA write (DMAW) and DMA read (DMAR) commands can be used to examine or modify one to 16 bits in each word during each R/M/W cycle. All other graphics drawing commands modify one bit per R/M/W cycle.

An internal 16-bit Mask register determines which bit(s) in the accessed word are to be modified. A one in the Mask register allows the corresponding bit in the display RAM to be modified by the R/M/W cycle. A zero in the Mask register prevents the GDC from modifying the corresponding bit in the display RAM.

The mask must be set by the Mask Command prior to issuing the WDAT or DMAW command. The Mask register is automatically set by the CURS command and manipulated by the graphics commands.

The display RAM bits can be modified in one of four ways. They can be set to 1, reset to 0, complemented or replaced by a pattern.

When replace by a pattern mode is selected, lines, arcs and rectangles will be drawn using the 16-bit pattern in parameter RAM bytes 8 and 9.

In set, reset, or complement mode, parameter RAM bytes 8 and 9 act as another level of masking for line arc and rectangle drawing. As each 16-bit segment of the line or arc is drawn, it is checked against the pattern in the parameter RAM. If the pattern RAM bit is a one, the display RAM bit will be set, reset, or complemented per the proper modes. If the pattern RAM bit is a zero, the display RAM bit won't be modified.

When replace by pattern mode is selected, the graphics character and fill commands will cause the 8 x 8 pattern in parameter RAM bytes 8 to 15 to be written directly into the display RAM in the appropriate locations.

In set, reset, or complement mode, the 8 x 8 pattern in parameter RAM bytes 8 to 15 act as a mask pattern for graphics character or fill commands. If the appropriate parameter RAM bit is set, the display RAM bit will be modified. If the parameter RAM bit is zero, the display RAM bit will not be modified. These modes are selected by issuing a WDAT command without parameters before issuing graphics commands. The pattern in the parameter RAM has no effect on WDAT, RDAT, DMAW, or DMAR operations.

READING AND DRAWING COMMANDS

After the modification mode has been set and the parameter RAM has been loaded, the final drawing parameters are loaded via the figure specify (FIGS) command. The first parameter specifies the direction in which drawing will occur and the figure type to be drawn. This parameter is followed by one to five more parameters depending on the type of character to be drawn.

The direction parameter specifies one of eight octants in which the drawing or reading will occur. The effect of drawing direction on the various figure types is shown in Figure 9.

RDAT, WDAT, DMAR, and DMAW Operations move through the Display memory as shown in the "DMA" Column.

The other parameters required to set up figure reading or drawing are shown in Figure 3.

DRAWING TYPE	DC	D	D2	D1	DM
INITIAL VALUE*	0	8	8	−1	−1
LINE	|Δl|	2|ΔD| − |Δl|	2(|ΔD| − |Δl|)	2|ΔD|	−
ARC**	rsin $\phi\uparrow$	r−1	2(r−1)	−1	rsin $\theta\downarrow$
RECTANGLE	3	A−1	B−1	−1	A−1
AREA FILL	B−1	A	A	−	−
GRAPHIC CHARACTER***	B−1	A	A	−	−
WRITE DATA	W−1	−	−	−	−
DMAW	D−1	C−1	−	−	−
DMAR	D−1	C−2	(C−2)/2†	−	−
READ DATA	W	−	−	−	−

*INITIAL VALUES FOR THE VARIOUS PARAMETERS ARE LOADED WHEN THE FIGS COMMAND BYTE IS PROCESSED.
**CIRCLES ARE DRAWN WITH 8 ARCS, EACH OF WHICH SPAN 45°, SO THAT SIN ϕ = 1/√2 AND SIN θ = 0.
***GRAPHIC CHARACTERS ARE A SPECIAL CASE OF BIT-MAP AREA FILLING IN WHICH B AND A ≤8. IF A = 8 THERE IS NO NEED TO LOAD D AND D2.

WHERE:
−1 = ALL ONES VALUE.
ALL NUMBERS ARE SHOWN IN BASE 10 FOR CONVENIENCE. THE GDC ACCEPTS BASE 2 NUMBERS (2s COMPLEMENT NOTATION WHERE APPROPRIATE).
− = NO PARAMETER BYTES SENT TO GDC FOR THIS PARAMETER.
Δl = THE LARGER OF Δx OR Δy.
ΔD = THE SMALLER OF Δx OR Δy.
r = RADIUS OF CURVATURE, IN PIXELS.
ϕ = ANGLE FROM MAJOR AXIS TO END OF THE ARC. $\phi \leq 45°$.
θ = ANGLE FROM MAJOR AXIS TO START OF THE ARC. $\theta \leq 45°$.
↑ = ROUND UP TO THE NEXT HIGHER INTEGER.
↓ = ROUND DOWN TO THE NEXT LOWER INTEGER.
A = NUMBER OF PIXELS IN THE INITIALLY SPECIFIED DIRECTION.
B = NUMBER OF PIXELS IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
W = NUMBER OF WORDS TO BE ACCESSED.
C = NUMBER OF BYTES TO BE TRANSFERRED IN THE INITIALLY SPECIFIED DIRECTION. (TWO BYTES PER WORD IF WORD TRANSFER MODE IS SELECTED.)
D = NUMBER OF WORDS TO BE ACCESSED IN THE DIRECTION AT RIGHT ANGLES TO THE INITIALLY SPECIFIED DIRECTION.
DC = DRAWING COUNT PARAMETER WHICH IS ONE LESS THAN THE NUMBER OF RMW CYCLES TO BE EXECUTED.
DM = DOTS MASKED FROM DRAWING DURING ARC DRAWING.
† = NEEDED ONLY FOR WORD READS.

Figure 3. Drawing Parameter Details

After the parameters have been set, line, arc, circle, rectangle or slanted rectangle drawing operations are initiated by the Figure Draw (FIGD) command. Character, slanted character, area fill and slanted area fill drawing operations are initiated by the Graphics Character Draw (GCHRD) command. DMA transfers are initiated by the DMA Read or Write (DMAR or DMAW) commands. Data Read Operations are initiated by the Read Data (RDAT) Command. Data Write Operations are initiated by writing a parameter after the WDAT command.

The area fill operation steps and repeats the 8 × 8 graphics character pattern draw operation to fill a rectangular area. If the size of the rectangle is not an integral number of 8 x 8 pixels, the GDC will automatically truncate the pattern at the edges furthest from the starting point.

The Graphics Character Drawing capability can be modified by the Graphics Character Write Zoom Factor (GCHR) parameter of the zoom command. The zoom write factor may be set from 1 to 16 (by using from 0 to 15 in the parameter). Each dot will be repeated in memory horizontally and vertically (adjusted for drawing direction) the number of times specified by the zoom factor.

The WDAT command can be used to rapidly fill large areas in memory with the same value. The mask is set to all 1's, and the least significant bit of the WDAT parameter replaces all bits of each word written.

Character Mode Memory Organization

In character mode, the Display memory is organized into up to 8K characters of up to 13 bits each. Wide mode is also available for characters of up to 26 bits.

The display memory can be larger than the display itself. The display width (in characters) is a parameter of the reset command. The display memory width (in characters) is a parameter of the Pitch Command. The height of the display (in lines) is a parameter of the Reset Command. The display memory height is determined by dividing the number of display memory words by the pitch.

In character mode, the display works almost exactly as it does in graphics mode. The differences lie in the fact that data read from the display RAM is used to drive a character generator as well as attribute logic if desired. In Character mode, address bits 13–16 become line counter outputs used to select the proper line of the character generator, and the address 17 output becomes the cursor and line counter MSB output.

Character Mode Display Timing

In character mode, the display timing works as it does in graphics mode. In addition, the Address 17 output becomes cursor output. The characteristics of the cursor are defined by parameters of the cursor and Character Characteristics (CCHAR) command. One bit allows the cursor output to be enabled or disabled. The height of the cursor is programmable by selecting the top and bottom line between which the cursor will appear. The blink rate is also programmable. The parameter selects the number of frame times that the cursor will be inactive and active, resulting in a 50% duty cycle cursor blinking at 2× the period specified by the parameter.

The cursor output pin also provides the line counter bit 4 signal, which is valid 10 clocks after the trailing edge of HSYNC.

Character Mode Special Display Functions

WINDOWING

The GDC's Character Mode display can be partitioned into one to four windows on the screen. The windows are defined by parameters written into the GDC's Parameter RAM. Each window is specified by a starting address and a window length in lines.

If windowing is not required, the first window length should be specified to be the same as the active display length.

ZOOMING AND PANNING

In character mode, zooming and pan handling commands function the same way as in Graphics Mode.

Character Mode Drawing and Writing

The GDC can read or write characters of up to 13 bits into or out of the Display RAM.

All reading and writing functions take place at the display RAM location specified by the cursor. The cursor location can be read by issuing the CURD command. The cursor can be moved anywhere within the display memory by the CURS command. The cursor location is also modified by the execution of character read or write commands.

Each character is written or read via a Read/Modify/Write cycle. The mask register contents determine which bit(s) in the character are modified. The mask register can be used to change character codes without modifying attribute bits or vice-versa. The Replace with pattern, Set, Reset and Complement

modes work exactly as they do in graphics mode, with the exception that the parameter RAM Pattern is not used. The pattern used is a parameter of the WDAT command.

The Figure Specify (FIGS) command must be set to Character Display mode, as well as specify the direction the cursor will be moved by read or write data commands.

In character mode, the FIGD and GCHRD commands are not used.

Mixed Mode Memory Organization

In mixed mode, the display memory is organized into two banks of up to 64K words of 16 bits each (32 bits in wide mode).

The display height and width are programmable by the same Reset or Sync command parameters as in the graphics and character modes. The display memory width (in words) is a parameter of the Pitch Command and the height of the display memory is determined by dividing the number of display memory words by the pitch.

An image mode signal is used to switch the external circuitry between graphics and character modes in two display windows.

In a graphics window, the GDC works as it does in pure graphics mode, but on a smaller total memory space (64K words vs 512K words).

In a character window, the GDC works as it does in pure character mode, but the line counter must be implemented externally. The counter is clocked by the horizontal sync pulse and reset by a signal supplied by the GDC.

In mixed mode, the GDC provides both a cursor and an attribute blink timing signal.

Mixed Mode Display Timing

In mixed mode, each word in a graphic area is accessed twice in succession. The AW parameter of the Reset or Sync command should be set to twice its normal value, and the video shift register load signal must be suppressed during the extra access cycle.

In addition, A16 becomes a Multiplexed Attribute and Clear Line Counter signal and A17 becomes a multiplexed cursor and image mode signal. A16 provides an active high line counter reset signal which is valid 10 clocks after the trailing edge of HSYNC. During the active display line time, A16 provides blink timing for external attribute circuitry. This signal blinks at 1/2 the blink rate of the cursor with a 75% on, 25% off duty cycle. A17 provides a signal which selects between graphics or character display, which is also valid 10 clocks after the trailing edge of HSYNC. During the active display time, A17 provides the cursor signal. The cursor timing and characteristics are defined in exactly the same way as in pure character mode.

Mixed Mode Special Display Functions

WINDOWING

The GDC supports two display windows in mixed mode. They can independently be programmed into either graphics or character mode determined by the state of two bits in the parameter RAM. The window location in display memory and size are also determined by parameters in the parameter RAM.

ZOOMING AND PANNING

In mixed mode, zooming and panning commands function the same as in graphics and character mode.

Mixed Mode Drawing and Writing

In mixed mode, the GDC can write or draw in exactly the same ways as in both graphics and character modes. In addition, the FIGS command has a parameter GD (Graphics Drawing Flag) which sets the image mode signal to select the proper RAM bank.

DEVICE PROGRAMMING

The GDC occupies two addresses on the system microprocessor bus through which the GDC's status register and FIFO are accessed. Commands and parameters are written into the GDC FIFO and are differentiated by address bit A0. The status register or the FIFO can be read as selected by the address line.

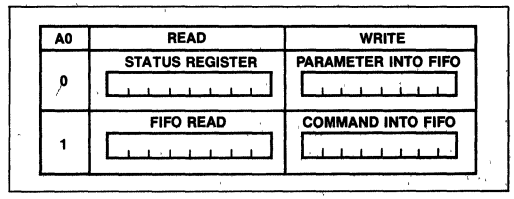


Figure 4. GDC Microprocessor Bus Interface Registers

Commands to the GDC take the form of a command byte followed by a series of parameter bytes as needed for specifying the details of the command. The command processor decodes the commands, unpacks the parameters, loads them into the appropriate registers within the GDC and initiates the required operations.

The commands available in the GDC can be organized into five categories as described in figure 5.

VIDEO CONTROL COMMANDS
1. RESET: RESETS THE GDC TO ITS IDLE STATE.
2. SYNC: SPECIFIES THE VIDEO DISPLAY FORMAT.
3. VSYNC: SELECTS MASTER OR SLAVE VIDEO SYNCHRONIZATION MODE
4. CCHAR: SPECIFIES THE CURSOR AND CHARACTER ROW HEIGHTS.

DISPLAY CONTROL COMMANDS
1. START: ENDS IDLE MODE AND UNBLANKS THE DISPLAY.
2. BCTRL: CONTROLS THE BLANKING AND UNBLANKING OF THE DISPLAY.
3. ZOOM: SPECIFIES ZOOM FACTORS FOR THE DISPLAY AND GRAPHICS CHARACTERS WRITING.
4. CURS: SETS THE POSITION OF THE CURSOR IN DISPLAY MEMORY.
5. PRAM: DEFINES STARTING ADDRESSES AND LENGTHS OF THE DISPLAY AREAS AND SPECIFIES THE EIGHT BYTES FOR THE GRAPHICS CHARACTER.
6. PITCH: SPECIFIES THE WIDTH OF THE X DIMENSION OF DISPLAY MEMORY.

DRAWING CONTROL COMMANDS
1. WDAT: WRITES DATA WORDS OR BYTES INTO DISPLAY MEMORY.
2. MASK: SETS THE MASK REGISTER CONTENTS.
3. FIGS: SPECIFIES THE PARAMETERS FOR THE DRAWING PROCESSOR.
4. FIGD: DRAWS THE FIGURE AS SPECIFIED ABOVE.
5. GCHRD: DRAWS THE GRAPHICS CHARACTER INTO DISPLAY

MEMORY DATA READ COMMANDS
1. RDAT: READS DATA WORDS OR BYTES FROM DISPLAY MEMORY.
2. CURD: READS THE CURSOR POSITION.
3. LPRD: READS THE LIGHT PEN ADDRESS.

DMA CONTROL COMMANDS
1. DMAR: REQUESTS A DMA READ TRANSFER.
2. DMAW: REQUESTS A DMA WRITE TRANSFER.

Figure 5. GDC Command Summary

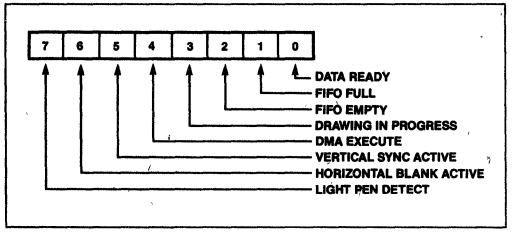


Figure 6. Status Register (SR)

Status Register Flags

SR-7: Light Pen Detect: When this bit is set to 1, the light pen address (LAD) register contains a deglitched value that the system microprocessor may read. This flag is reset after the 3-byte LAD is moved into the FIFO in response to the light pen read command.

SR-6: Horizontal Blanking Active: A 1 value for this flag signifies that horizontal retrace blanking is currently underway.

SR-5: Vertical Sync: Vertical retrace sync occurs while this flag is a 1. The vertical sync flag coordinates display format modifying commands to the blanked interval surrounding vertical sync. This eliminates display disturbances.

SR-4: DMA Execute: This bit is a 1 during DMA data transfers.

SR-3: Drawing In Progress: While the GDC is drawing a graphics figure, this status bit is a 1.

SR-2: FIFO Empty: This bit and the FIFO Full flag coordinate system microprocessor accesses with the GDC FIFO. When it is 1, the Empty flag ensures that all the commands and parameters previously sent to the GDC have been processed.

SR-1: FIFO Full: A 1 at this flag indicates a full FIFO in the GDC. A 0 ensures that there is room for at least one byte. This flag needs to be checked before each write into the GDC.

SR-0: Data Ready: When this flag is a 1, it indicates that a byte is available to be read by the system microprocessor. This bit must be tested before each read operation. It drops to a 0 while the data is transferred from the FIFO into the microprocessor interface data register.

FIFO Operation & Command Protocol

The first-in, first-out buffer (FIFO) in the GDC handles the command dialogue with the system microprocessor. This flow of information uses a half-duplex technique, in which the single 16-location FIFO is used for both directions of data movement, one direction at a time. The FIFO's direction is controlled by the system microprocessor through the GDC's command set. The microprocessor coordinates these transfers by checking the appropriate status register bits.

The command protocol used by the GDC requires the differentiation of the first byte of a command sequence from the succeeding bytes. This first byte contains the operation code and the remaining bytes carry parameters. Writing into the GDC causes the FIFO to store a flag value alongside the data byte to signify whether the byte was written into the command or the parameter address. The command processor in the GDC tests this bit as it interprets the entries in the FIFO.

The receipt of a command byte by the command processor marks the end of any previous operation. The number of parameter bytes supplied with a command is cut short by the receipt of the next command byte. A read operation from the GDC to the microprocessor can be terminated at any time by the next command.

The FIFO changes direction under the control of the system microprocessor. Commands written into the GDC always put the FIFO into write mode if it wasn't in it already. If it was in read mode, any read data in the FIFO at the time of the turnaround is lost. Commands which require a GDC response, such as RDAT, CURD and LPRD, put the FIFO into read mode after the command is interpreted by the GDC's command processor. Any commands and parameters behind the read-evoking command are discarded when the FIFO direction is reversed.

Read-Modify-Write Cycle

Data transfers between the GDC and the display memory are accomplished using a read-modify-write (RMW) memory cycle. The four clock period timing of the RMW cycle is used to: 1) output the address, 2) read data from the memory, 3) modify the data, and 4) write the modified data back into the initially selected memory address. This type of memory cycle is used for all interactions with display memory including DMA transfers, except for the two clock period display and RAM refresh cycles.

The operations performed during the modify portion of the RMW cycle merit additional explanation. The circuitry in the GDC uses three main elements: the Pattern register, the Mask register, and the 16-bit Logic unit. The Pattern register holds the data pattern to be moved into memory. It is loaded by the WDAT command or, during drawing, from the parameter RAM. The Mask register contents determine which bits of the read data will be modified. Based on the contents of these registers, the Logic unit performs the selected operations of REPLACE, COMPLEMENT, SET, or CLEAR on the data read from display memory.

The Pattern register contents are ANDed with the Mask register contents to enable the actual modification of the memory read data, on a bit-by-bit basis. For graphics drawing, one bit at a time from the Pattern register is combined with the Mask. When ANDed with the bit set to a 1 in the Mask register, the proper single pixel is modified by the Logic Unit. For the next pixel in the figure, the next bit in the Pattern register is selected and the Mask register bit is moved to identify the pixel's location within the word. The Execution word address pointer register, EAD, is also adjusted as required to address the word containing the next pixel.

In character mode, all of the bits in the Pattern register are used in parallel to form the respective bits of the modify data word. Since the bits of the character code word are used in parallel, unlike the one-bit-at-a-time graphics drawing process, this facility allows any or all of the bits in a memory word to be modified in one RMW memory cycle. The Mask register must be loaded with 1s in the positions where modification is to be permitted.

The Mask register can be loaded in either of two ways. In graphics mode, the CURS command contains a four-bit dAD field to specify the dot address. The command processor converts this parameter into the one-of-16 format used in the Mask register for figure drawing. A full 16 bits can be loaded into the Mask register using the MASK command. In addition to the character mode use mentioned above, the 16-bit MASK load is convenient in graphics mode when all of the pixels of a word are to be set to the same value.

The Logic unit combines the data read from display memory, the Pattern register, and the Mask register to generate the data to be written back into display memory. Any one of four operations can be selected: REPLACE, COMPLEMENT, CLEAR or SET. In each case, if the respective Mask bit is 0, that particular bit of the read data is returned to memory unmodified. If the Mask bit is 1, the modification is enabled. With the REPLACE operation, the modify data simply takes the place of the read data for modification enabled bits. For the other three operations, a 0 in the modify data allows the read data bit to be returned to memory. A 1 value causes the specified operation to be performed in the bit positions with set Mask bits.

Figure Drawing

The GDC draws graphics figures at the rate of one pixel per read-modify-write (RMW) display memory cycle. These cycles take four clock periods to complete. At a clock frequency of 5 MHz, this is equal to 800 ns. During the RMW cycle the GDC simultaneously calculates the address and position of the next pixel to be drawn.

The graphics figure drawing process depends on the display memory addressing structure. Groups of 16 horizontally adjacent pixels form the 16-bit words

which are handled by the GDC. Display memory is organized as a linearly addressed space of these words. Addressing of individual pixels is handled by the GDC's internal RMW logic.

During the drawing process, the GDC finds the next pixel of the figure which is one of the eight nearest neighbors of the last pixel drawn. The GDC assigns each of these eight directions a number from 0 to 7, starting with straight down and proceeding counterclockwise.

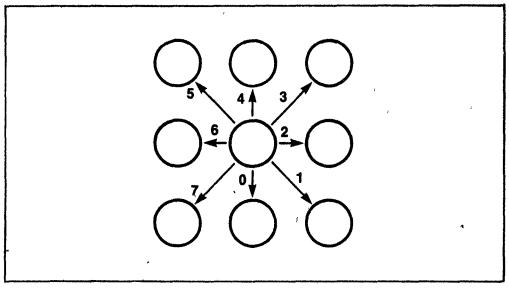


Figure 7. Drawing Directions

Figure drawing requires the proper manipulation of the address and the pixel bit position according to the drawing direction to determine the next pixel of the figure. To move to the word above or below the current one, it is necessary to subtract or add the number of words per line in display memory. This parameter is called the pitch. To move to the word to either side, the Execute word address cursor, EAD, must be incremented or decremented as the dot address pointer bit reaches the LSB or the MSB of the Mask register. To move to a pixel within the same word, it is necessary to rotate the dot address pointer register to the right or left.

Figure 8 summarizes these operations for each direction.

Whole word drawing is useful for filling areas in memory with a single value. By setting the Mask register to all 1s with the MASK command, both the LSB and MSB of the dAD will always be 1, so that the EAD value will be incremented or decremented for each cycle regardless of direction. One RMW cycle will be able to affect all 16 bits of the word for any drawing type. One bit in the Pattern register is used per RMW cycle to write all the bits of the word to the same value. The next Pattern bit is used for the word, etc.

DIR	ADDRESS OPERATION(S)
0	EAD = EAD + P
1	EAD = EAD + P If dAD.MSB = 1 then EAD = EAD + 1 dAD = LR(dAD)
2	If dAD.MSB = 1 then EAD = EAD + 1 dAD = LR(dAD)
3	EAD = EAD – P If dAD.MSB = 1 then EAD = EAD + 1 dAD = LR(dAD)
4	EAD = EAD – P
5	EAD = EAD – P If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)
6	If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)
7	EAD = EAD + P If dAD.LSB = 1 then EAD = EAD – 1 dAD = RR(dAD)

WHERE
P = PITCH, LR = LEFT ROTATE, RR = RIGHT ROTATE
CAD = CURSOR ADDRESS
dAD = DOT ADDRESS
LSB = LEAST SIGNIFICANT BIT
MSB = MOST SIGNIFICANT BIT

Figure 8. Address Calculation Details

For the various figures, the effect of the initial direction upon the resulting drawing is shown in figure 9.

Note that during line drawing, the angle of the line may be anywhere within the shaded octant defined by the DIR value. Arc drawing starts in the direction initially specified by the DIR value and veers into an arc as drawing proceeds. An arc may be up to 45 degrees in length. DMA transfers are done on word boundaries only, and follow the arrows indicated in the table to find successive word addresses. The slanted paths for DMA transfers indicate the GDC changing both the X and Y components of the word address when moving to the next word. It does not follow a 45 degree diagonal path by pixels.

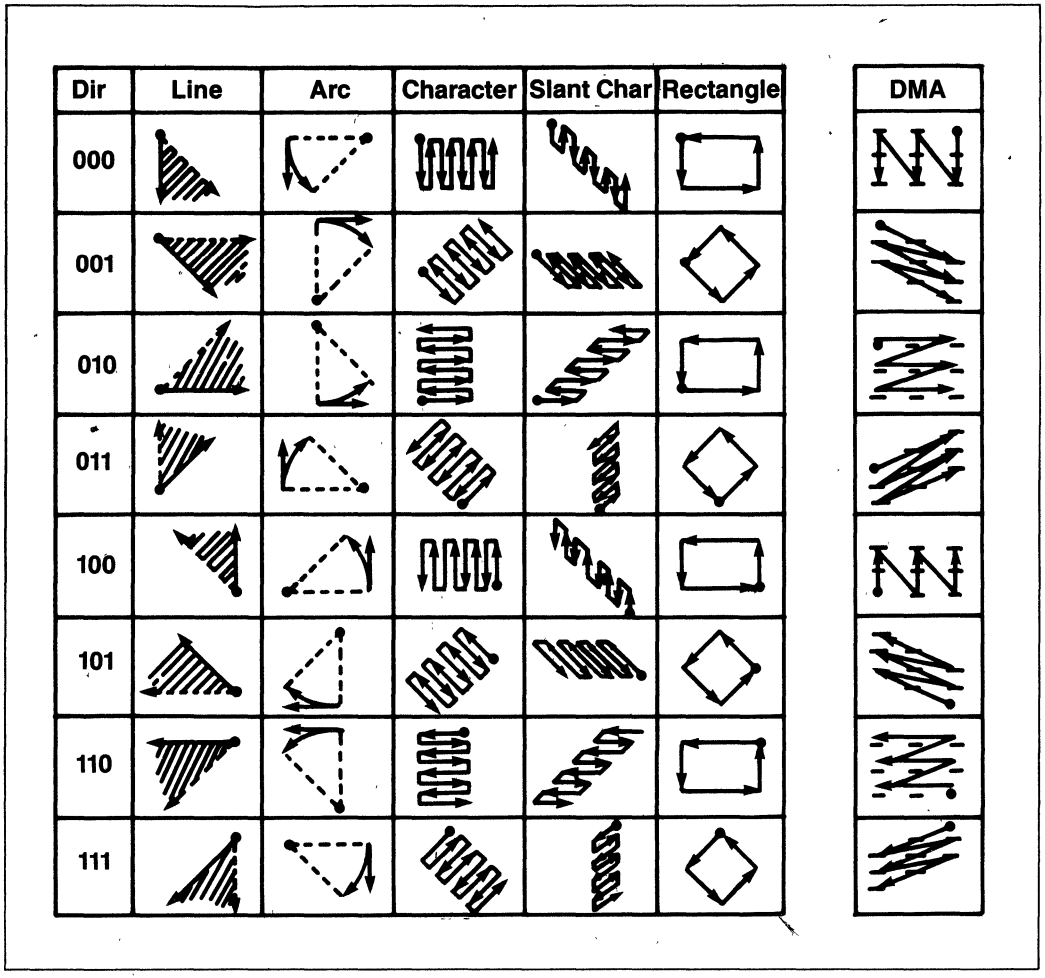


Figure 9. Effect of the Direction Parameter

Drawing Parameters

In preparation for graphics figure drawing, the GDC's Drawing Processor needs the figure type, direction and drawing parameters, the starting pixel address, and the pattern from the microprocessor. Once these are in place within the GDC, the Figure Draw command, FIGD, initiates the drawing operation. From that point on, the system microprocessor is not involved in the drawing process. The GDC Drawing Processor coordinates the RMW circuitry and address registers to draw the specified figure pixel by pixel.

The algorithms used by the processor for figure drawing are designed to optimize its drawing speed. To this end, the specific details about the figure to be drawn are reduced by the microprocessor to a form conducive to high-speed address calculations within the GDC. In this way the repetitive, pixel-by-pixel calculations can be done quickly, thereby minimizing the overall figure drawing time. Figure 3 summarizes the parameters.

Graphics Character Drawing

Graphics characters can be drawn into display memory pixel-by-pixel. The up to 8-by-8 character is loaded into the GDC's parameter RAM by the system microprocessor. Consequently, there are no limitations on the character set used. By varying the drawing parameters and drawing direction, numerous drawing options are available. In area fill applications, a character can be written into display memory as many times as desired without reloading the parameter RAM.

Once the parameter RAM has been loaded with up to eight graphics character bytes by the appropriate PRAM command, the GCHRD command can be used to draw the bytes into display memory starting at the cursor. The zoom magnification factor for writing, set by the zoom command, controls the size of the character written into the display memory in integer multiples of 1 through 16. The bit values in the PRAM are repeated horizontally and vertically the number of times specified by the zoom factor.

The movement of these PRAM bytes to the display memory is controlled by the parameters of the FIGS command. Based on the specified height and width of the area to be drawn, the parameter RAM is scanned to fill the required area.

For an 8-by-8 graphics character, the first pixel drawn uses the LSB of RA-15, the second pixel uses bit 1 of RA-15, and so on, until the MSB of RA-15 is reached. The GDC jumps to the corresponding bit in RA-14 to continue the drawing. The progression then advances toward the LSB of RA-14. This snaking sequence is continued for the other 6 PRAM bytes. This progression matches the sequence of display memory addresses calculated by the drawing processor as shown in figure 9. If the area is narrower than 8 pixels wide, the snaking will advance to the next PRAM byte before the MSB is reached. If the area is less than 8 lines high, fewer bytes in the parameter RAM will be scanned. If the area is larger than 8 by 8, the GDC will repeat the contents of the parameter RAM in two dimensions.

Parameter RAM Contents

The parameters stored in the parameter RAM, PRAM, are available for the GDC to refer to repeatedly during figure drawing and raster-scanning. In each mode of operation the values in the PRAM are interpreted by the GDC in a predetermined fashion. The host microprocessor must load the appropriate parameters into the proper PRAM locations. PRAM loading command allows the host to write into any location of the PRAM and transfer as many bytes as desired. In this way any stored parameter byte or bytes may be changed without influencing the other bytes.

The PRAM stores two types of information. For specifying the details of the display area partitions, blocks of four bytes are used. The four parameters stored in each block include the starting address in display memory of each display area, and its length. In addition, there are two mode bits for each area which specify whether the area is a bit-mapped graphics area or a coded character area, and whether a normal or wide display cycle is to be used for that area.

The other use for the PRAM contents is to supply the pattern for figure drawing when in a bit-mapped graphics area or mode. In these situations, PRAM bytes 8 through 16 are reserved for this patterning information. For line, arc, and rectangle drawing (linear figures) locations 8 and 9 are loaded into the Pattern register to allow the GDC to draw dotted, dashed, etc. lines. For area filling and graphics bit-mapped character drawing locations 8 through 15 are referenced for the pattern or character to be drawn.

Details of the bit assignments are shown on the following pages for the various modes of operation.

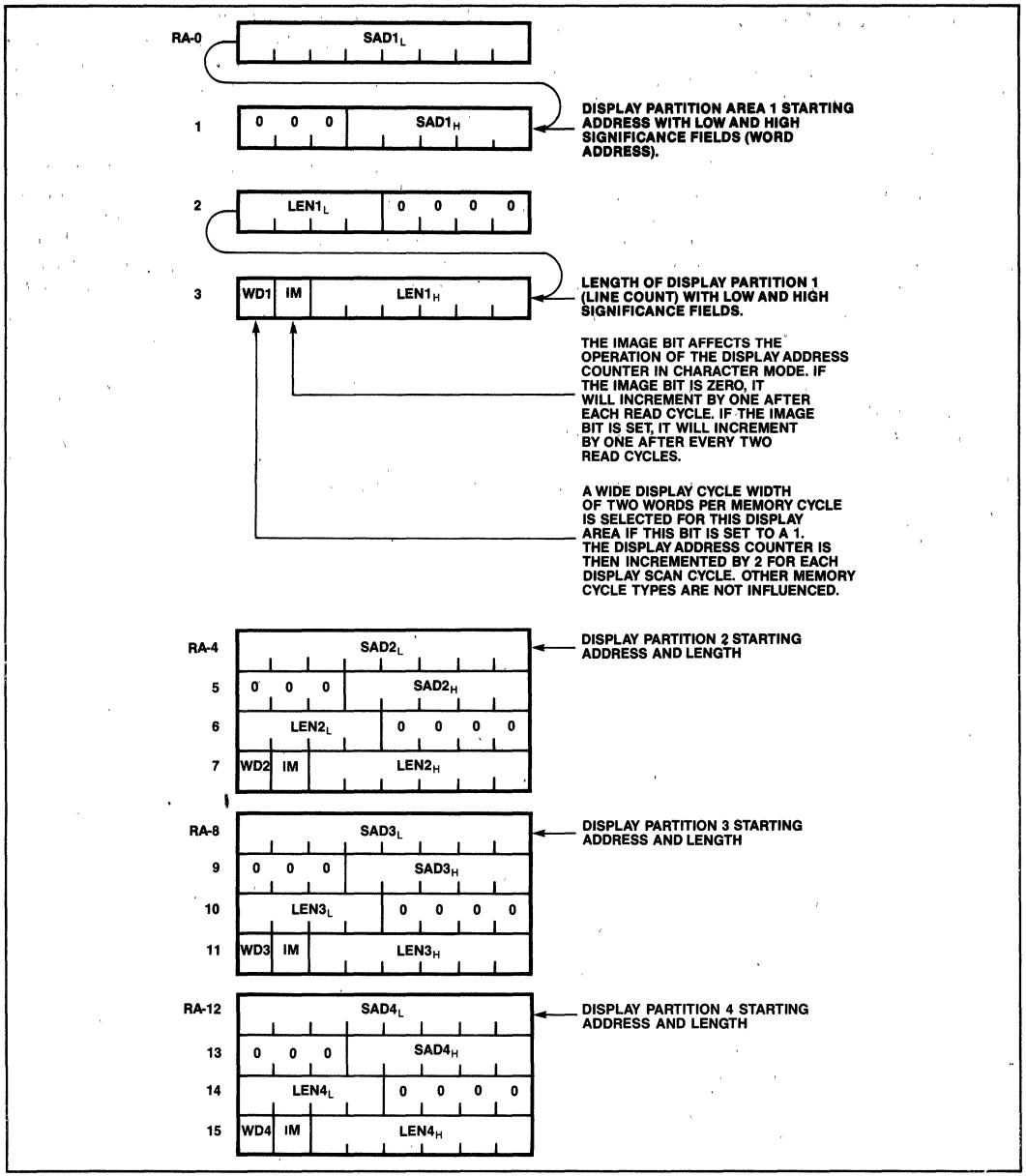


Figure 10. Parameter RAM Contents—Character Mode

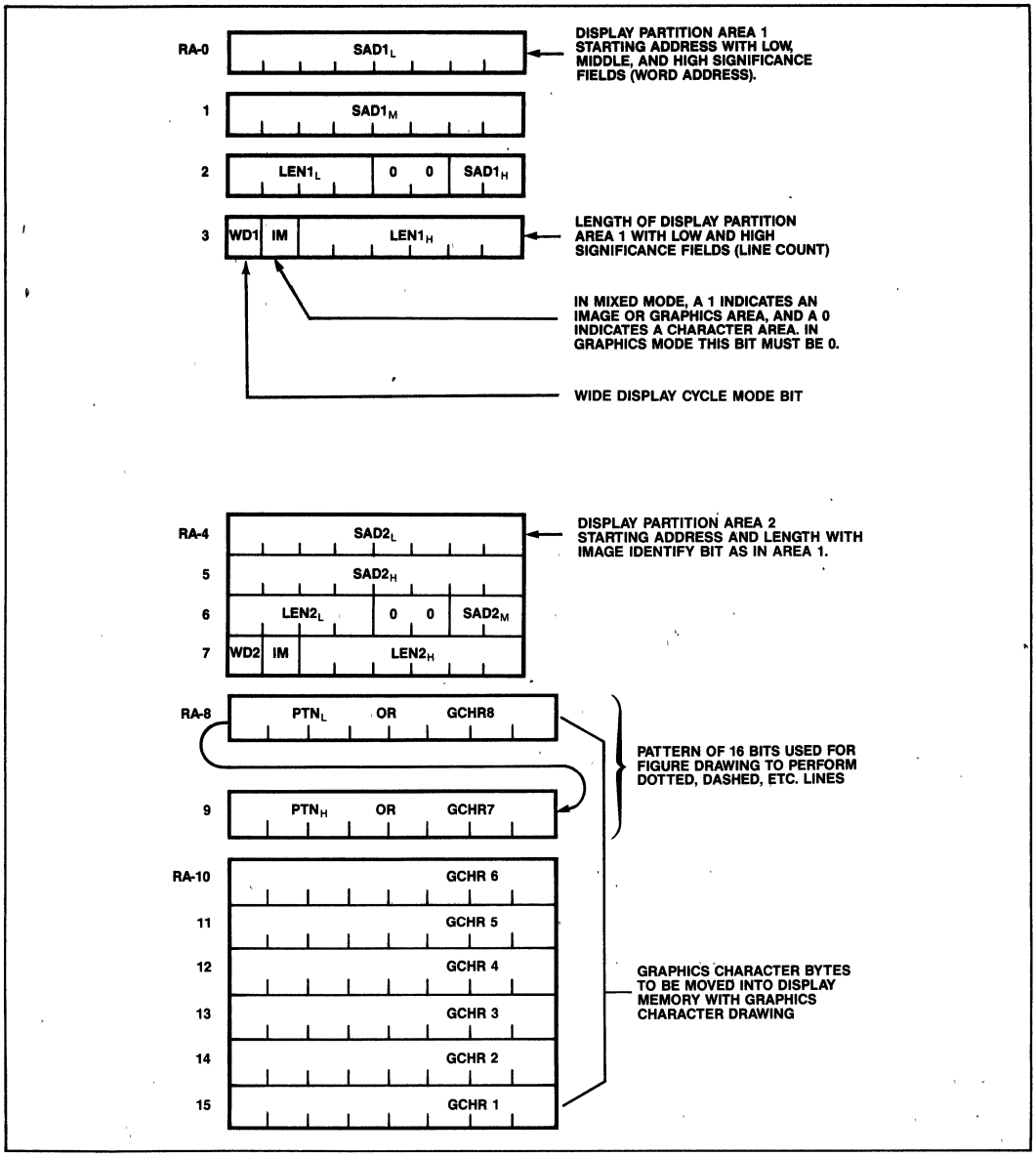


Figure 11. Parameter RAM Contents—Graphics and Mixed Graphics and Character Modes

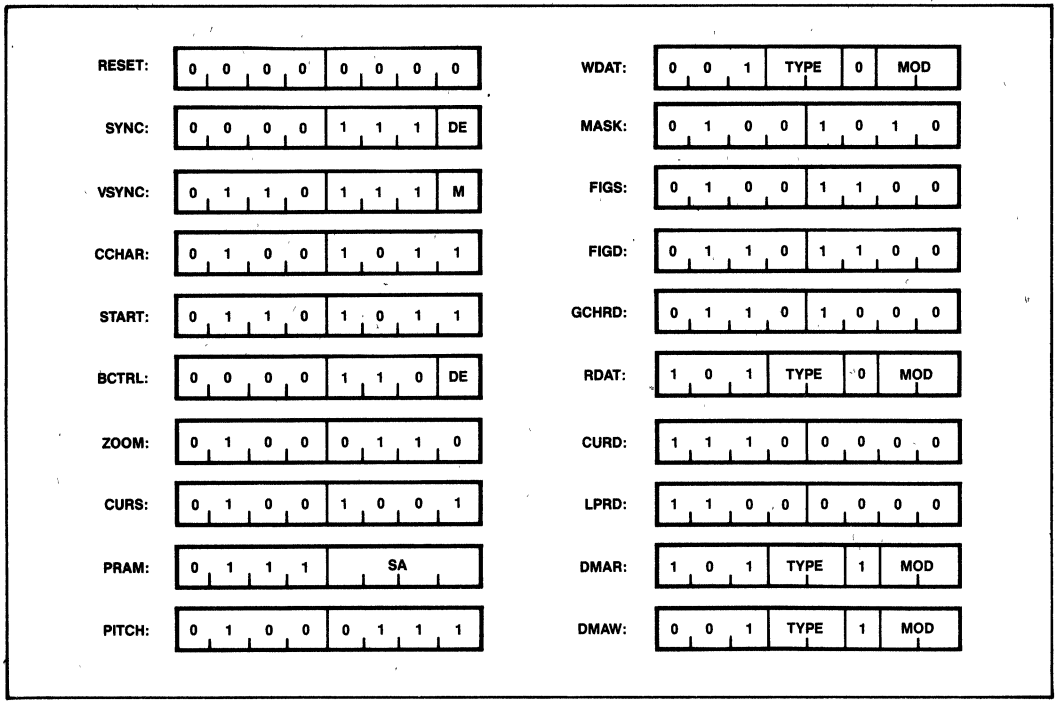


Figure 12. Command Bytes Summary

VIDEO CONTROL COMMANDS

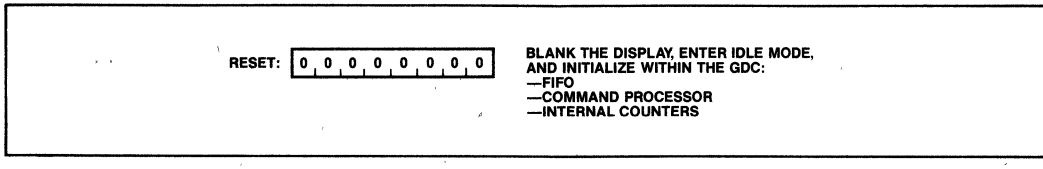


Figure 13. Reset Command

RESET COMMAND

This command can be executed at any time and does not modify any of the parameters already loaded into the GDC.

If followed by parameter bytes, this command also sets the sync generator parameters as described below. Idle mode is exited with the START command.

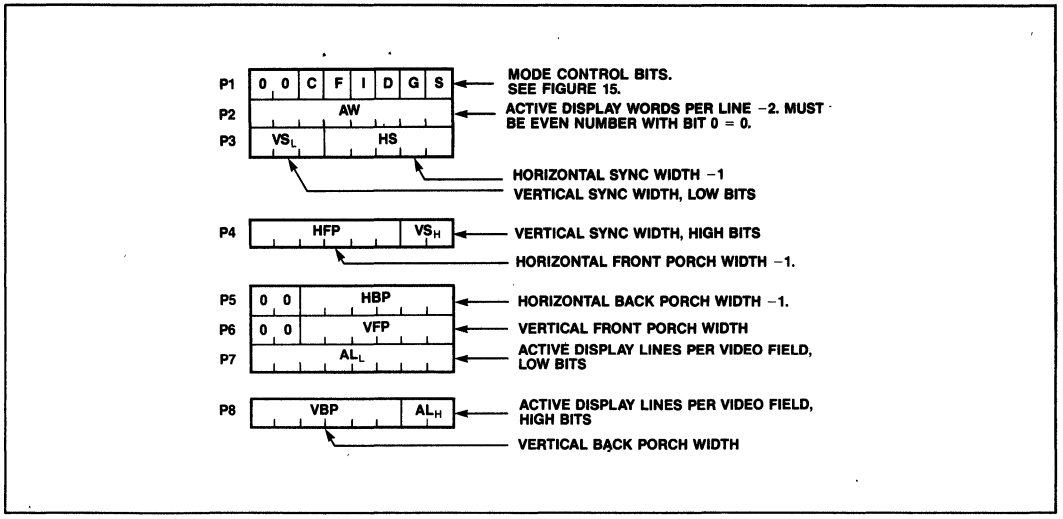


Figure 14. Optional Reset Parameters

In graphics mode, a word is a group of 16 pixels. In character mode, a word is one character code and its attributes, if any.

The number of active words per line must be an even number from 2 to 256.

An all-zero parameter value selects a count equal to 2^n where n = number of bits in the parameter field for vertical parameters.

All horizontal widths are counted in display words. All vertical intervals are counted in lines.

Sync Parameter Constraints

HORIZONTAL FRONT PORCH CONSTRAINTS

1. In general:
   HFP ≥2 words
2. If DMA is used, or the display zoom factor is greater than one in interlaced display mode:
   HFP ≥3 words
3. If the GDC is used in slave mode:
   HFP ≥4 words
4. If the light pen input is used:
   HFP ≥6 words

HORIZONTAL Sync CONSTRAINTS

1. If dynamic RAM refresh is used:
   HS ≥2 words
2. If interlaced display mode is used:
   HS ≥5 words

HORIZONTAL BACK PORCH CONSTRAINTS

1. In general:
   HBP ≥3 words
2. If interlaced display mode is used, or the IMAGE or WIDE mode bits change within one video field:
   HBP ≥ 5 words

MODE CONTROL BITS (FIGURE 15)

Repeat Field Framing:	2 Field Sequence with ½ line offset between otherwise identical fields.
Interlaced Framing:	2 Field Sequence with ½ line offset. Each field displays alternate lines.
Noninterlaced Framing:	1 field brings all of the information to the screen.

Total scanned lines in interlace mode is odd. The sum of VFP + VS + VBP + AL should equal one less than the desired odd number of lines.

Dynamic RAM refresh is important when high display zoom factors or DMA are used in such a way that not all of the rows in the RAMs are regularly accessed during display raster generation and for otherwise inactive display memory.

Access to display memory can be limited to retrace blanking intervals only, so that no disruptions of the image are seen on the screen.

C G	DISPLAY MODE
0 0	MIXED GRAPHICS & CHARACTER
0 1	GRAPHICS MODE
1 0	CHARACTER MODE
1 1	INVALID

I S	VIDEO FRAMING
0 0	NONINTERLACED
0 1	INVALID
1 0	INTERLACED REPEAT FIELD FOR CHARACTER DISPLAYS
1 1	INTERLACED

D	DYNAMIC RAM REFRESH CYCLES ENABLE
0	NO REFRESH—STATIC RAM
1	REFRESH—DYNAMIC RAM

F	DRAWING TIME WINDOW
0	DRAWING DURING ACTIVE DISPLAY TIME AND RETRACE BLANKING
1	DRAWING ONLY DURING RETRACE BLANKING

Figure 15. Mode Control Bits

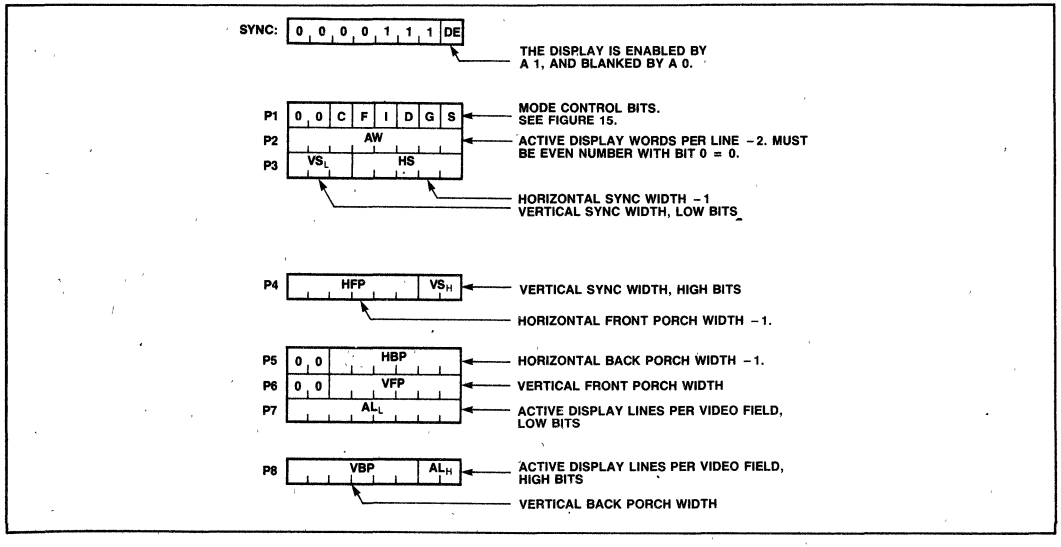


Figure 16. Sync Command

SYNC Format Specify Command

This command loads parameters into the sync generator. The various parameter fields and bits are identical to those at the RESET command. The GDC is not reset nor does it enter idle mode.

Vertical Sync Mode Command

When using two or more GDCs to contribute to one image, one GDC is defined as the master sync generator, and the others operate as its slaves. The VSYNC pins of all GDCs are connected together.

Slave Mode Operation

A few considerations should be observed when synchronizing two or more GDCs to generate overlayed video via the VSYNC INPUT/OUTPUT pin. As mentioned above, the Horizontal Front Porch (HFP) must be 4 or more display cycles wide. This is equivalent to eight or more clock cycles. This gives the slave GDCs time to initialize their internal video sync generators to the proper point in the video field to match the incoming vertical sync pulse (VSYNC). This resetting of the generator occurs just after the end of the incoming VSYNC pulse, during the HFP interval. Enough time during HFP is required to allow the slave GDC to complete the operation before the start of the HSYNC interval.

Once the GDCs are initialized and set up as Master and Slaves, they must be given time to synchronize. It is a good idea to watch the VSYNC status bit of the Master GDC and wait until after one or more VSYNC pulses have been generated before the display process is started. The START command will begin the active display of data and will end the video synchronization process, so be sure there has been at least one VSYNC pulse generated for the Slaves to synchronize to.

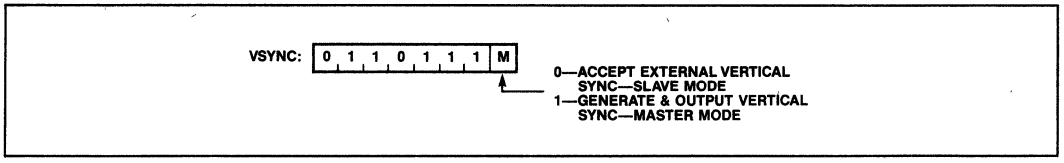


Figure 17. Vertical Sync Mode Command

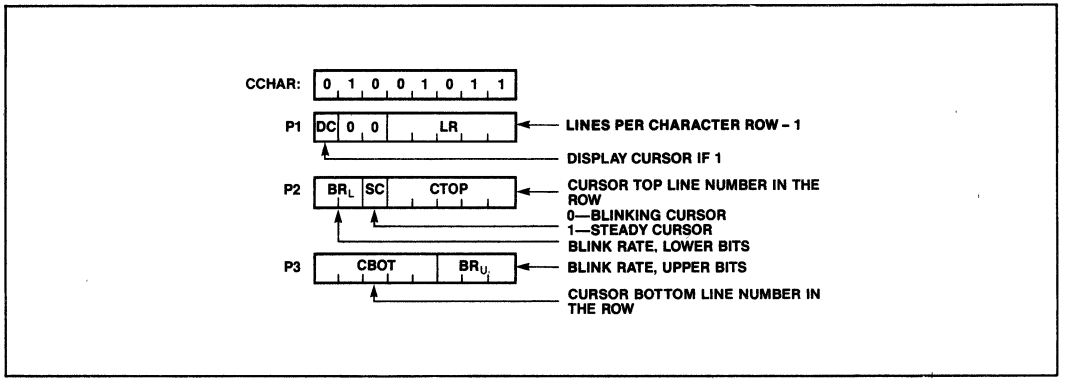


Figure 18. Cursor & Character Characteristics Command

Cursor and Character Characteristics Command

In graphics mode, LR should be set to 0. For interlaced displays in graphics mode, BR should be set to 3. The blink rate parameter controls both the cursor and attribute blink rates. The cursor blink-on-time = blink-off-time = 2 × BR (video frames). The attribute blink rate is always ½ the cursor rate but with a ¾ on-¼ off duty cycle.

DISPLAY CONTROL COMMANDS

Zoom Factors Specify Command

Zoom magnification factors of 1 through 16 are available using codes 0 through 15, respectively.

Cursor Position Specify Command

In character mode, the third parameter byte is not needed. The cursor is displayed for the word time in which the display scan address (DAD) equals the cursor address. In graphics mode, the cursor word address specifies the word containing the starting pixel of the drawing; the dot address value specifies the pixel within that word.

Parameter RAM Load Command

From the starting address, SA, any number of bytes may be loaded into the parameter RAM at incrementing addresses, up to location 15. The sequence of parameter bytes is terminated by the next command byte entered into the FIFO. The parameter RAM stores 16 bytes of information in predefined locations which differ for graphics and character modes. See the parameter RAM discussion for bit assignments.

Pitch Specification Command

This value is used during drawing by the drawing processor to find the word directly above or below the current word, and during display to find the start of the next line.

The Pitch parameter (width of display memory) is set by two different commands. In addition to the PITCH command, the RESET (or SYNC) command also sets the pitch value. The "active words per line" parameter, which specifies the width of the raster-scan display, also sets the Pitch of the display memory. In situations in which these two values are equal there is no need to execute a PITCH command.

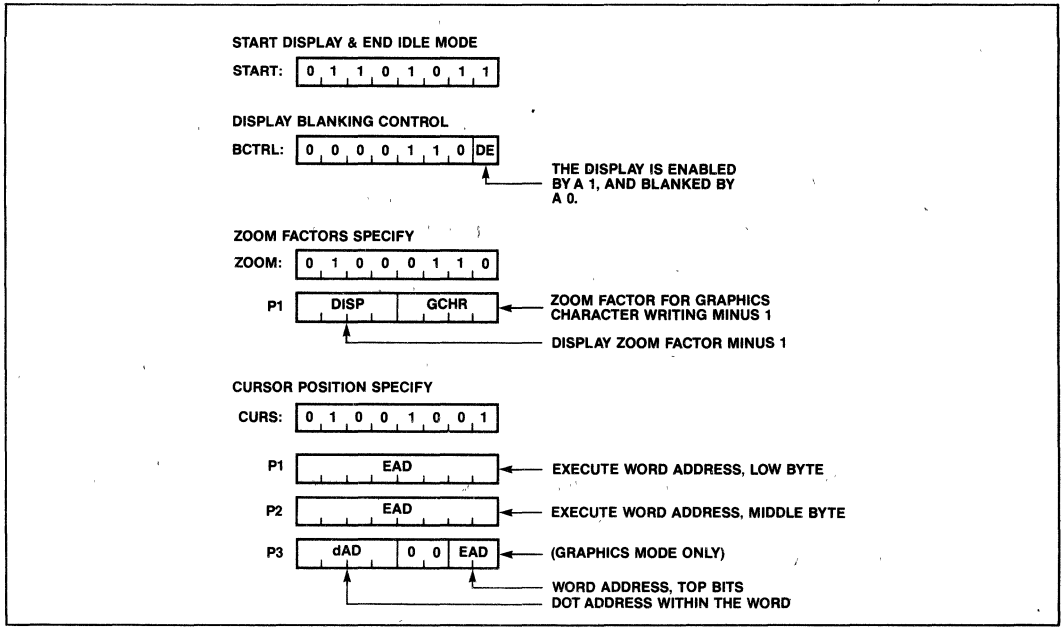


Figure 19. Display Control Commands

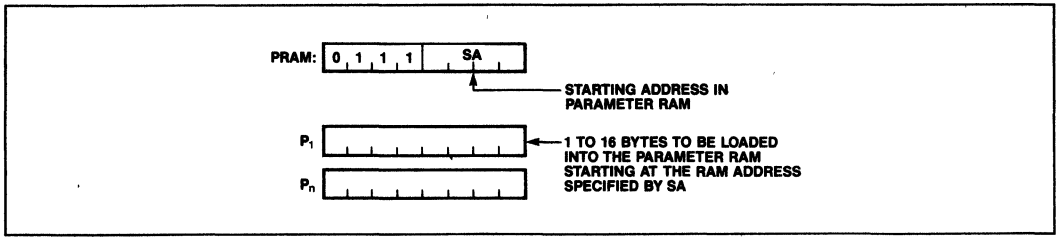


Figure 20. Parameter RAM Load Command

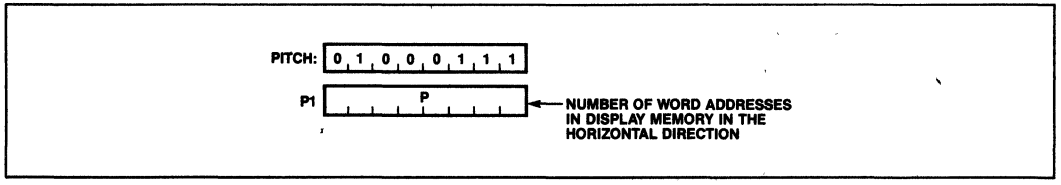


Figure 21. Pitch Specification Command

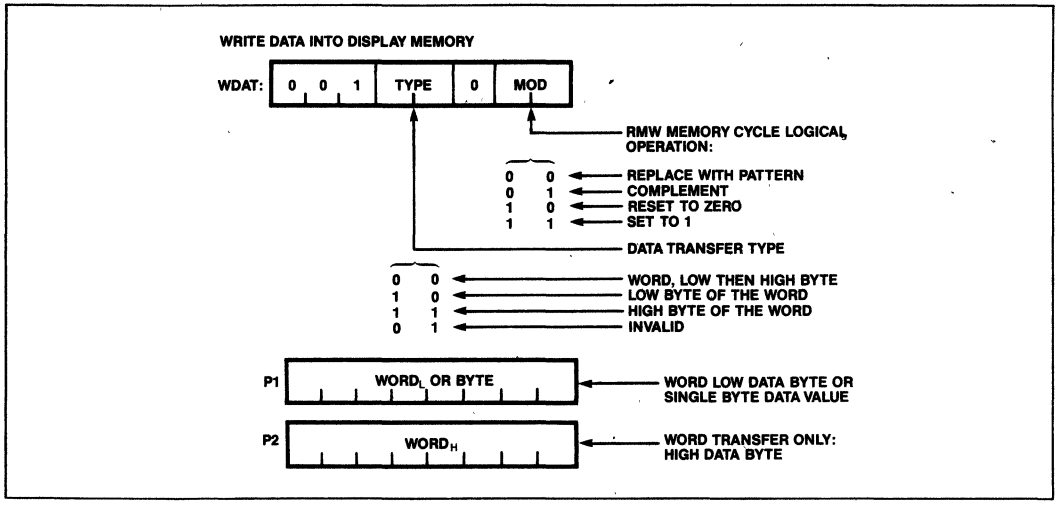


Figure 22. Write Data Command

DRAWING CONTROL COMMANDS

Write Data Command

Upon receiving a set of parameters (two bytes for a word transfer, one for a byte transfer), one RMW cycle into Video Memory is done at the address pointed to by the cursor EAD. The EAD pointer is advanced to the next word, according to the previously specified direction. More parameters can then be accepted.

For byte writes, the unspecified byte is treated as all zeros during the RMW memory cycle.

In graphics bit-map situations, only the LSB of the WDAT parameter bytes is used as the pattern in the RMW operations. Therefore it is possible to have only an all ones or all zeros pattern. In coded character applications all the bits of the WDAT parameters are used to establish the drawing pattern.

The WDAT command operates differently from the other commands which initiate RMW cycle activity. It requires parameters to set up the Pattern register while the other commands use the stored values in the parameter RAM. Like all of these commands, the

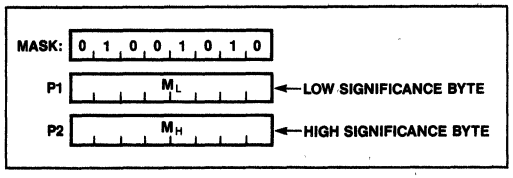
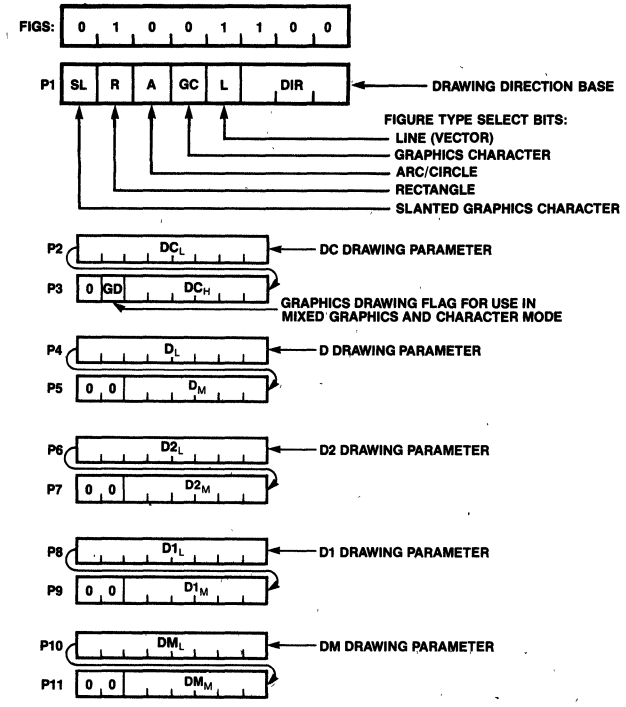


Figure 23. Mask Register Load Command

WDAT command must be preceded by a FIGS command and its parameters. Only the first three parameters need be given following the FIGS opcode, to set up the type of drawing, the DIR direction, and the DC value. The DC parameter +1 will be the number of RMW cycles done by the GDC with the first set of WDAT parameters. Additional sets of WDAT parameters will see a DC value of 0 which will cause only one RMW cycle to be executed.



VALID FIGURE TYPE SELECT COMBINATIONS

SL	R	A	GC	L	OPERATION
0	0	0	0	0	CHARACTER DISPLAY MODE DRAWING, INDIVIDUAL DOT DRAWING, DMA, WDAT, AND RDAT
0	0	0	0	1	STRAIGHT LINE DRAWING
0	0	0	1	0	GRAPHICS CHARACTER DRAWING AND AREA FILLING WITH GRAPHICS CHARACTER PATTERN
0	0	1	0	0	ARC AND CIRCLE DRAWING
0	1	0	0	0	RECTANGLE DRAWING
1	0	0	1	0	SLANTED GRAPHICS CHARACTER DRAWING AND SLANTED AREA FILLING

ONLY THESE BIT COMBINATIONS ASSURE CORRECT DRAWING OPERATION

Figure 24. Figure Drawing Parameters Specify Command

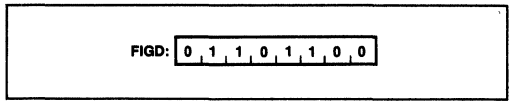


Figure 25. Figure Draw Start Command

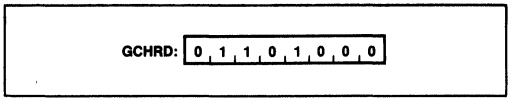


Figure 26. Graphics Character Draw and Area Filling Start Command

Mask Register Load Command

This command sets the value of the 16-bit Mask register of the figure drawing processor. The Mask register controls which bits can be modified in the display memory during a read-modify-write cycle.

The Mask register is loaded both by the MASK command and the third parameter byte of the CURS command. The MASK command accepts two parameter bytes to load a 16-bit value into the MASK register. All 16 bits can be individually one or zero, under program control. The CURS command on the other hand, puts a "1 of 16" pattern into the Mask register based on the value of the Dot Address value, dAD. If normal single-pixel-at-a-time graphics figure drawing is desired, there is no need to do a MASK command at all since the CURS command will set up the proper pattern to address the proper pixels as drawing progresses. For coded character DMA, and screen setting and clearing operations using the WDAT command, the MASK command should be used after the CURS command if its third parameter byte has been output. The Mask register should be set to all ones for any "word-at-a-time" operation.

Figure Draw Start Command

On execution of this instruction, the GDC loads the parameters from the parameter RAM into the drawing processor and starts the drawing process at the pixel pointed to by the cursor, EAD, and the dot address, dAD.

Graphics Char. Draw and Area Fill Start Command

Based on parameters loaded with the FIGS command, this command initiates the drawing of the graphics character or area filling pattern stored in Parameter RAM. Drawing begins at the address in display memory pointed to by the EAD and dAD values.

DATA READ COMMANDS

Read Data Command

Using the DIR and DC parameters of the FIGS command to establish direction and transfer count, multiple RMW cycles can be executed without specification of the cursor address after the initial load (DC = number of words or bytes).

As this instruction begins to execute, the FIFO buffer direction is reversed so that the data read from display memory can pass to the microprocessor. Any commands or parameters in the FIFO at this time will be lost. A command byte sent to the GDC will immediately reverse the buffer direction back to write mode, and all RDAT information not yet read from the FIFO will be lost. MOD should be set to 00.

Cursor Address Read Command

The Execute Address, EAD, points to the display memory word containing the pixel to be addressed.

The Dot Address, dAD, within the word is represented as a 1-of-16 code.

Light Pen Address Read Command

The light pen address, LAD, corresponds to the display word address, DAD, at which the light pen input signal is detected and deglitched.

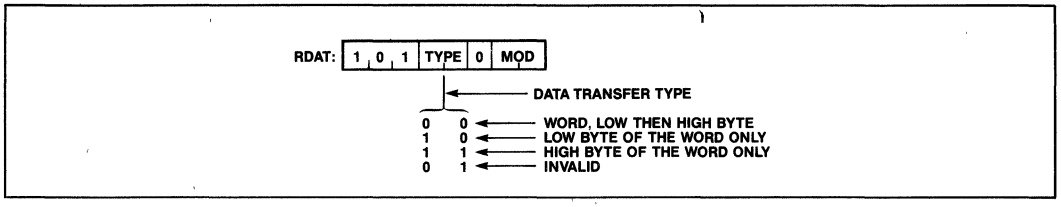


Figure 27. Read Data from Display Memory Command

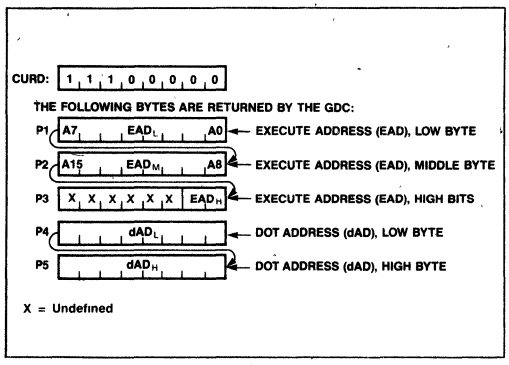


Figure 28. Cursor Address Read Command

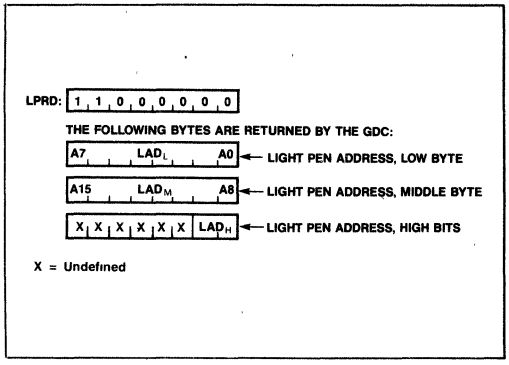


Figure 29. Light Pen Address Read Command

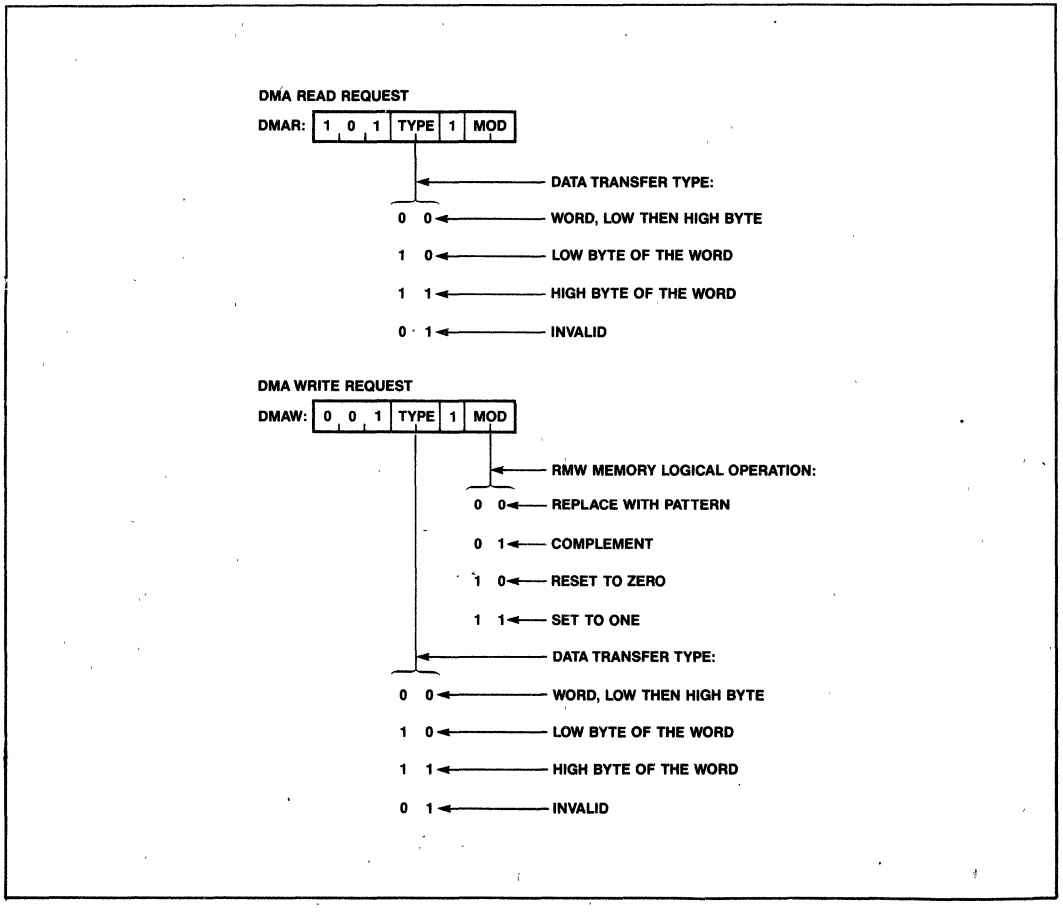


Figure 30. DMA Control Commands

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Voltage on any Pin with Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

COMMENT: Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS

T_A = 0°C to 70° C; V_{CC} = 5V ± 10%; GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{OZ}	Output Leakage Current		±10	μA	V_{SS}+0.45 ≤ V_I ≤ V_{CC}
I_{IL}	Input Leakage Current		±10	μA	V_{SS} ≤ V_I ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	0.6	V	
V_{CH}	Clock Input High Voltage	3.5	V_{CC} + 0.5	V	
I_{CC}	V_{CC} Supply Current		270	mA	Typical = 150 mA

CAPACITANCE

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Limits		Unit	Conditions
		Min.	Max.		
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz V = 0
C_{IO}	I/O Capacitance		20	pF	
C_{OUT}	Output Capacitance		20	pF	
C_O	Clock Input Capacitance		20	pF	

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{SS} = 0V, V_{CC} = +5V ±10%)

DATA BUS READ CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AR}	A_0 setup to $\overline{RD}\downarrow$	0		0		0		ns	
T_{RA}	A_0 hold after $\overline{RD}\uparrow$	0		0		0		ns	
T_{RR}	$\overline{RD}$ Pulse Width	$T_{RD}+20$		$T_{RD}+20$		$T_{RD}+20$		ns	
T_{RD}	$\overline{RD}\downarrow$ to Data Out Delay		120		80		70	ns	CL = 50pF
T_{DF}	$\overline{RD}\uparrow$ to Data Float Delay	0	120	0	100	0	90	ns	
T_{RV}	$\overline{RD}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DATA BUS WRITE CYCLE

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{AW}	A_0 Setup to $\overline{WR}\downarrow$	0		0		0		ns	
T_{WA}	A_0 Hold after $\overline{WR}\uparrow$	0		0		10		ns	
T_{WW}	$\overline{WR}$ Pulse Width	120		100		90		ns	
T_{DW}	Data Setup to $\overline{WR}\uparrow$	100		80		70		ns	
T_{WD}	Data Hold after $\overline{WR}\uparrow$	0		0		10		ns	
T_{RV}	$\overline{WR}$ Recovery Time	4 T_{CY}		4 T_{CY}		4 T_{CY}		ns	

DISPLAY MEMORY TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CA}	Address/Data Delay from 2XWCLK↑	30	160	30	130	30	110	ns	CL = 50pF
T_{AC}	Address/Data Hold Time	30	160	30	130	30	110	ns	CL = 50pF
T_{DC}	Data Setup to 2XWCLK↓	0		0		0		ns	
T_{CD}	Data Hold Time	$T_{IE}+20$		$T_{IE}+20$		$T_{IE}+20$		ns	
T_{IE}	2XWCLK↓ to $\overline{DBIN}$	30	120	30	90	30	80	ns	CL = 50pF
T_{CAH}	2XWCLK↑ to ALE↑	30	125	30	100	30	90	ns	CL = 50pF
T_{CAL}	2XWCLK↓ to ALE↓	30	100	30	80	30	70	ns	CL = 50pF
T_{AL}	ALE Low Time	$T_{CY}+30$		$T_{CY}+30$		$T_{CY}+30$		ns	
T_{AH}	ALE High Time	$T_{CH}-20$		$T_{CH}-20$		$T_{CH}-20$		ns	
T_{CO}	Video Signal Delay from 2XWCLK↑		150		120		100	ns	

A.C. CHARACTERISTICS (Continued)

OTHER TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{PC}	LPEN or VSYNC Input Setup to 2XWCLK↑	30		20		15		ns	
T_{PP}	LPEN or VSYNC Input Pulse Width	T_{CY}		T_{CY}		T_{CY}		ns	

CLOCK TIMING

Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{CY}	Clock Period	250	2000	200	2000	180	2000	ns	
T_{CH}	Clock High Time	105		80		70		ns	
T_{CL}	Clock Low Time	105		80		70		ns	
T_R	Rise Time		20		20		20	ns	
T_F	Fall Time		20		20		20	ns	

DMA TIMING

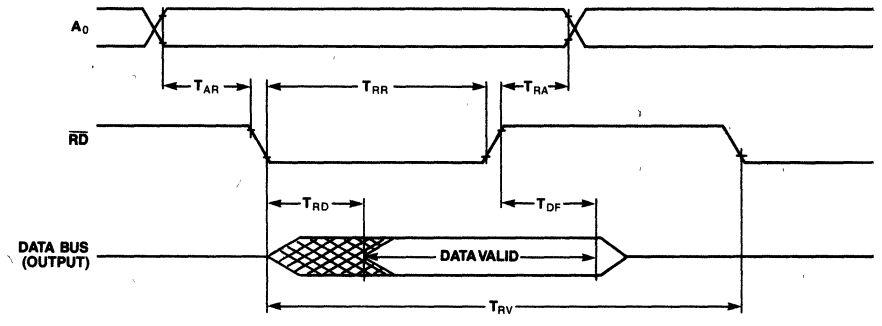
Symbol	Parameter	82720		82720-1		82720-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
T_{ACC}	$\overline{DACK}$ Setup to $\overline{RD}$↓ or $\overline{WR}$↓	0		0		0		ns	
T_{CAC}	$\overline{DACK}$ Hold from $\overline{RD}$↑ or $\overline{WR}$↑	0		0		0		ns	
T_{RR1}	$\overline{RD}$ Pulse Width	$T_{RD1}+20$		$T_{RD1}+20$		$T_{RD1}+20$		ns	
T_{RD1}	$\overline{RD}$↓ to Data Out Delay		$1.5\ T_{CY}+120$		$1.5\ T_{CY}+80$		$1.5\ T_{CY}+70$	ns	CL = 50pF
T_{KQ}	2XWCLK↑ to DREQ Delay		150		120		100	ns	CL = 50pF
T_{RQAK}	DREQ Setup to DACK↓	0		0		0		ns	
T_{AKRQ}	$\overline{DACK}$↓ to DREQ↓ Delay		$T_{CY}+150$		$T_{CY}+120$		$T_{CY}+100$	ns	CL = 50pF
T_{AKH}	$\overline{DACK}$ High Time	T_{CY}		T_{CY}		T_{CY}		ns	
T_{AK1}	$\overline{DACK}$ Cycle Time, Word Mode	$4\ T_{CY}$		$4\ T_{CY}$		$4\ T_{CY}$		ns	
T_{AK2}	$\overline{DACK}$ Cycle Time, Byte Mode	$5\ T_{CY}$		$5\ T_{CY}$		$5\ T_{CY}$		ns	

A.C. TEST CONDITIONS

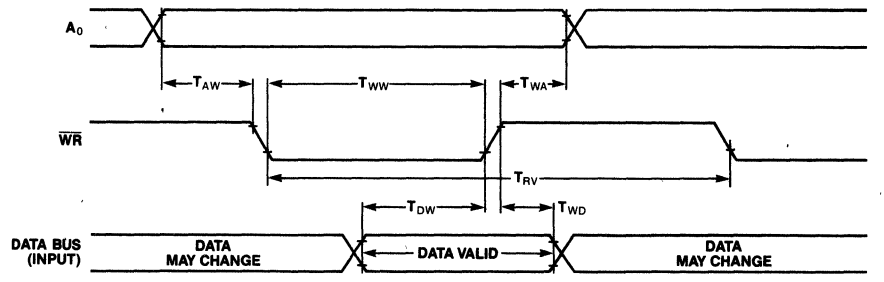
Input Pulse Levels (except 2XWCLK) ..... 0.45V to 2.4V
Input Pulse Levels (2XWCLK) ..... 0.3V to 3.9V
Timing Measurement Reference Levels (except 2XWCLK) ..... 0.8V to 2.0V
Timing Measurement Reference Levels (2XWCLK) ..... 0.6V to 3.5V

WAVEFORMS

DATA BUS TIMING READ CYCLE

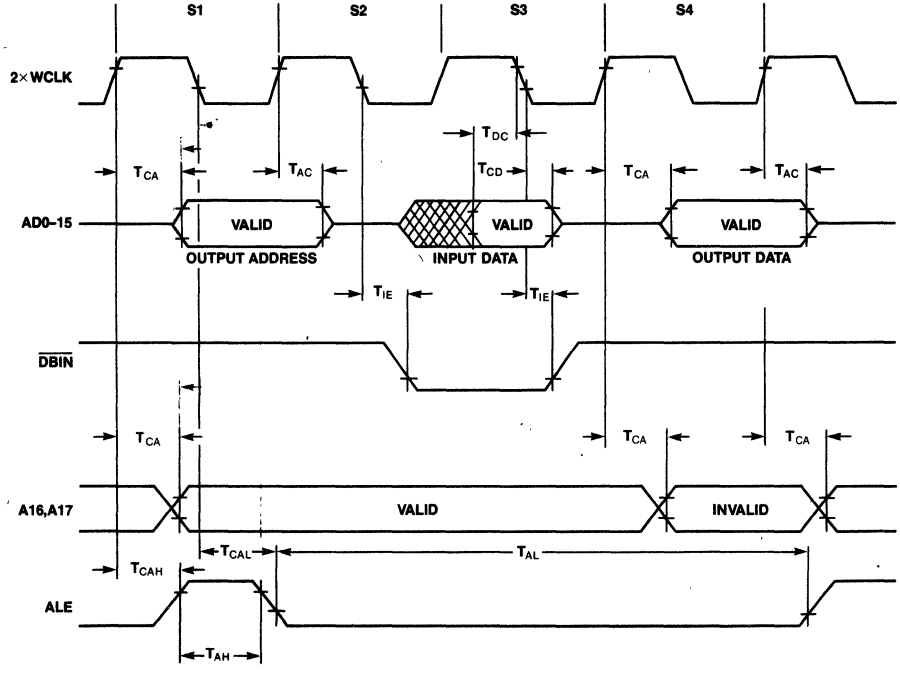


WRITE CYCLE



WAVEFORMS (Continued)

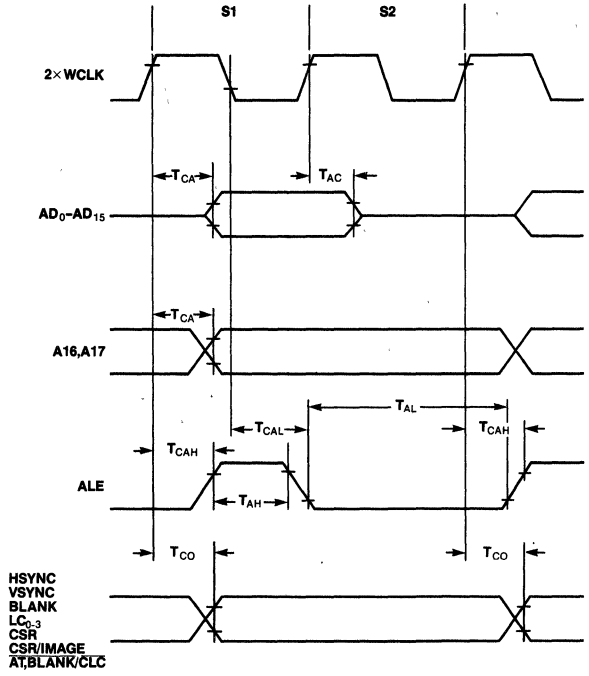
DISPLAY MEMORY TIMING
READ/MODIFY/WRITE CYCLE



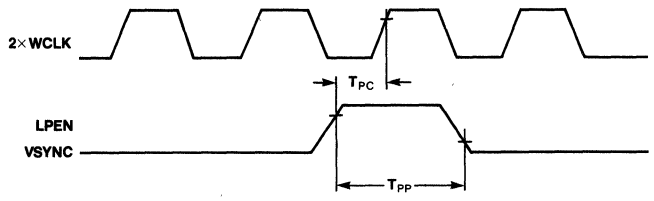
WAVEFORMS (Continued)

DISPLAY MEMORY TIMING (Continued)

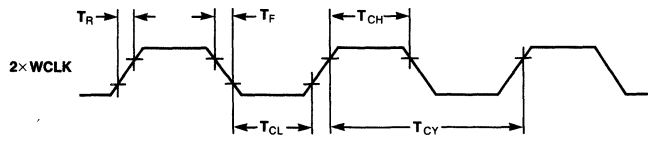
READ CYCLE



OTHER TIMING



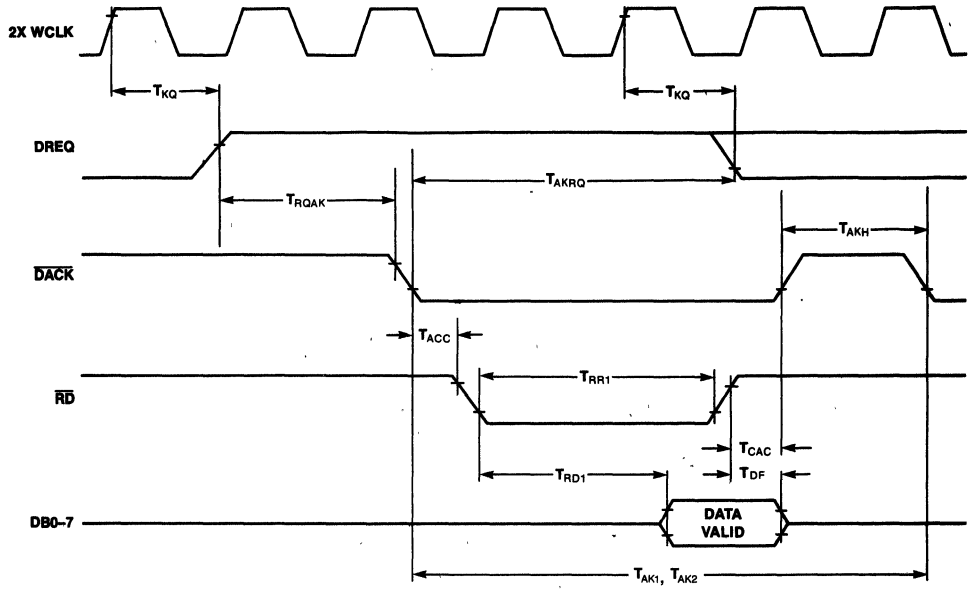
CLOCK TIMING



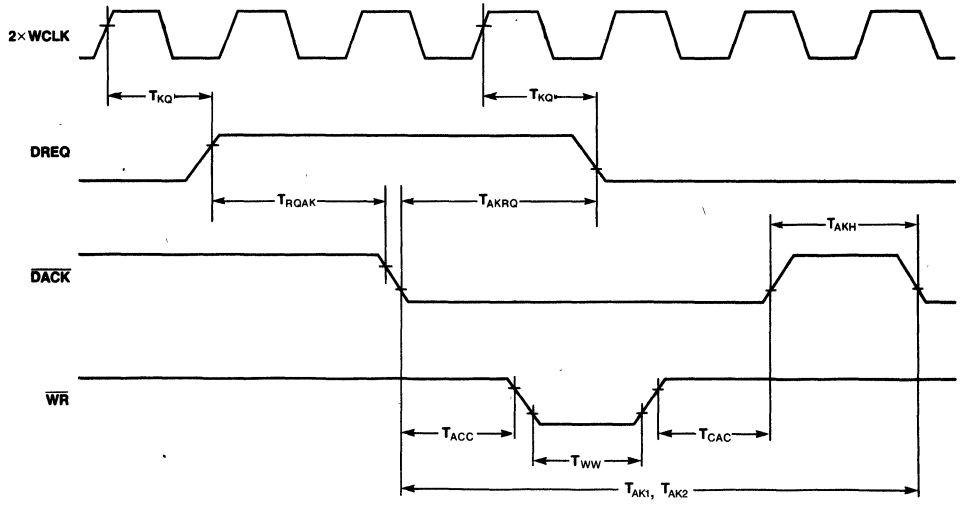
WAVEFORMS (Continued)

DMA TIMING

READ



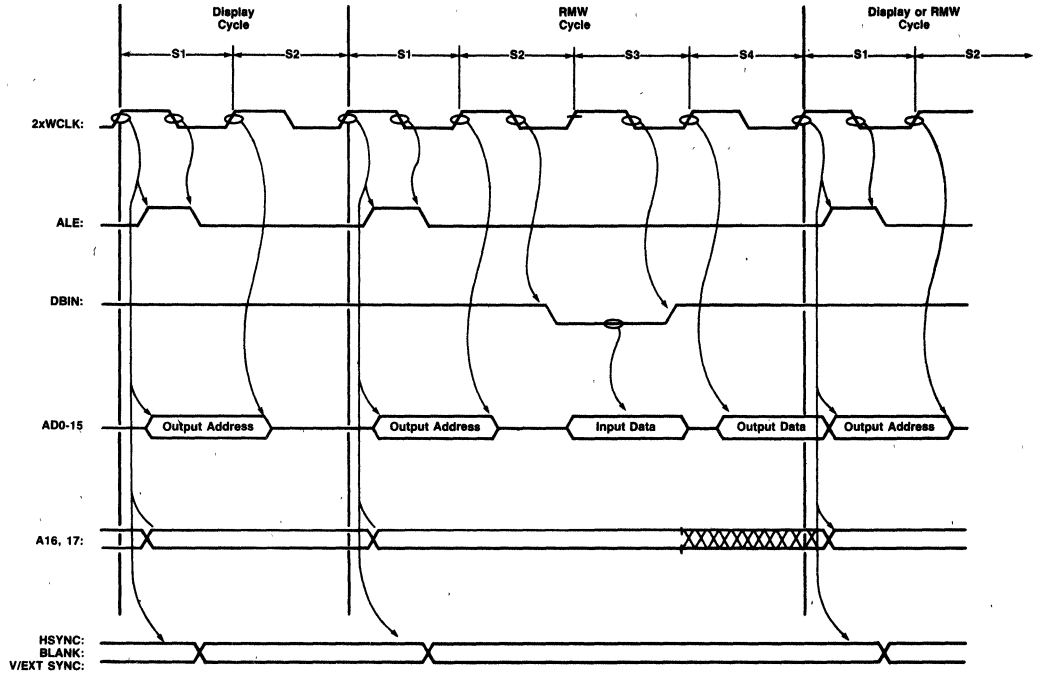
WRITE



210655-002

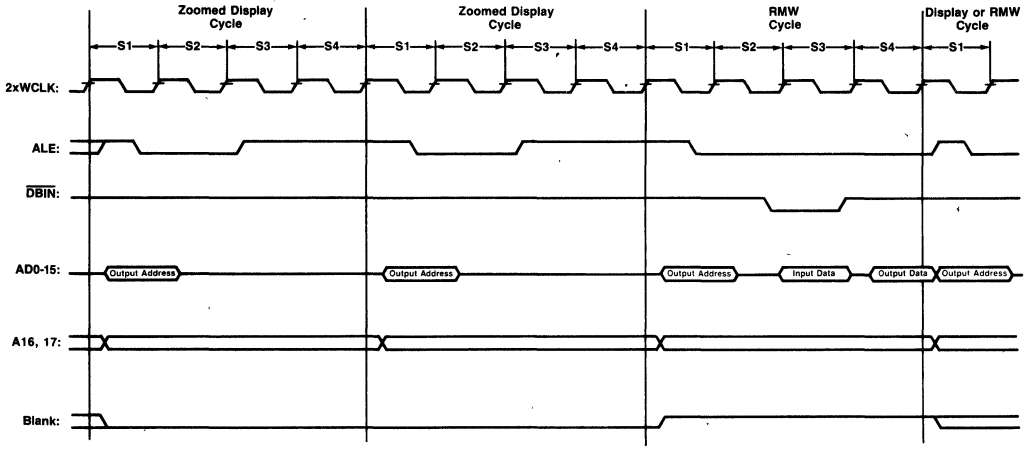
WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (1X ZOOM)

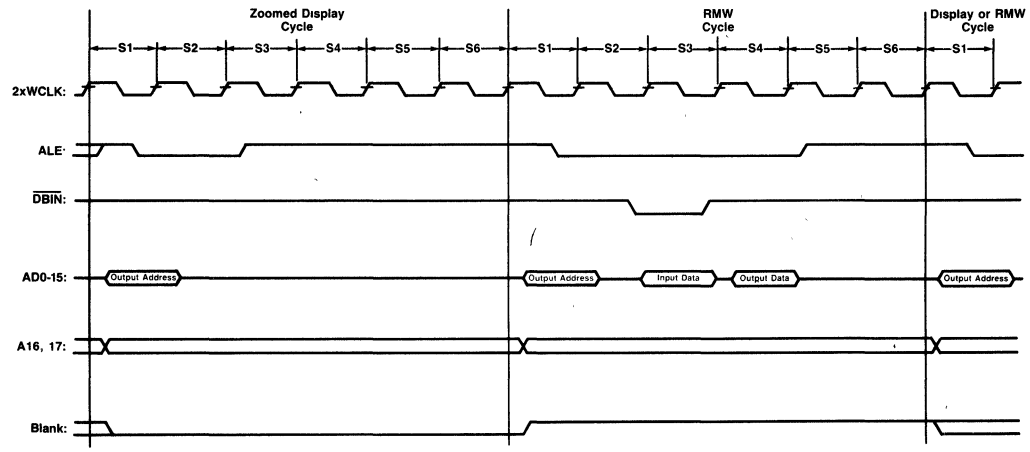


WAVEFORMS (Continued)

DISPLAY AND RMW CYCLES (2X ZOOM)



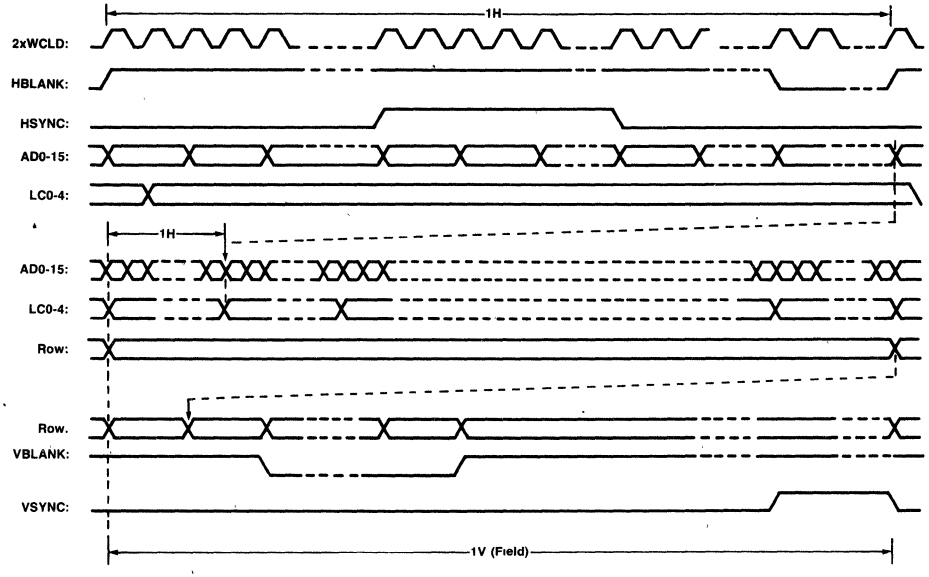
ZOOMED DISPLAY OPERATION WITH RMW CYCLE (3X ZOOM)



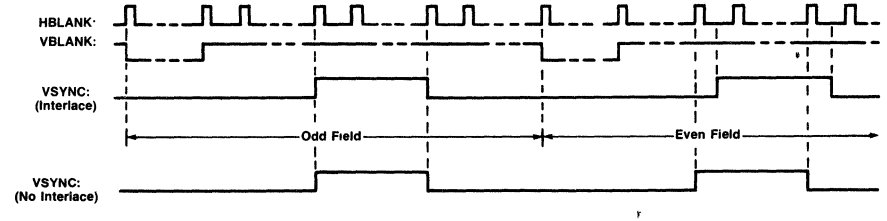
210655-002

WAVEFORMS (Continued)

VIDEO SYNC SIGNALS TIMING

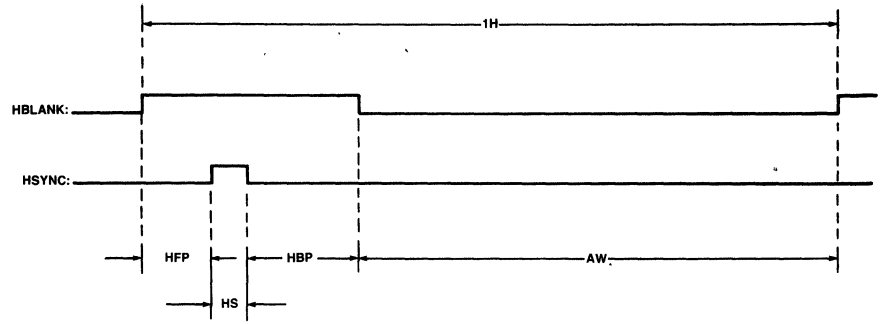


INTERLACED VIDEO TIMING

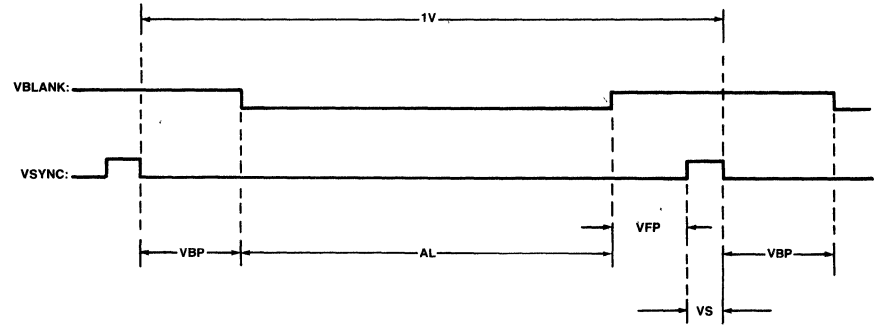


WAVEFORMS (Continued)

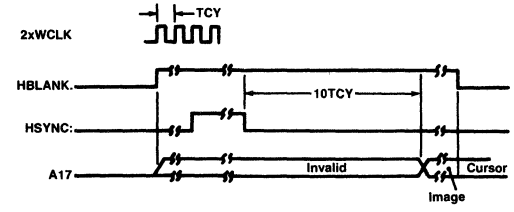
VIDEO HORIZONTAL SYNC GENERATOR PARAMETERS



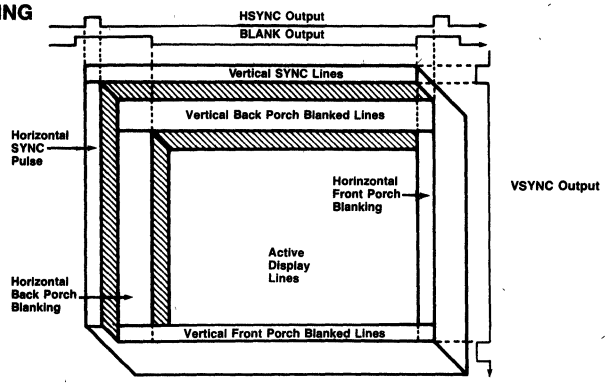
VIDEO VERTICAL SYNC GENERATOR PARAMETERS



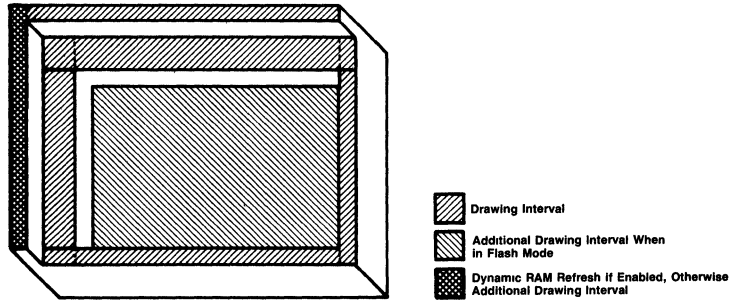
CURSOR—IMAGE BIT FLAG



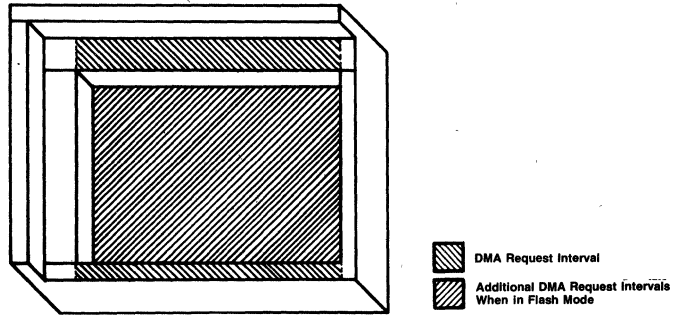
VIDEO FIELD TIMING



DRAWING INTERVALS



DMA REQUEST INTERVALS



A HAYDEN PUBLICATION

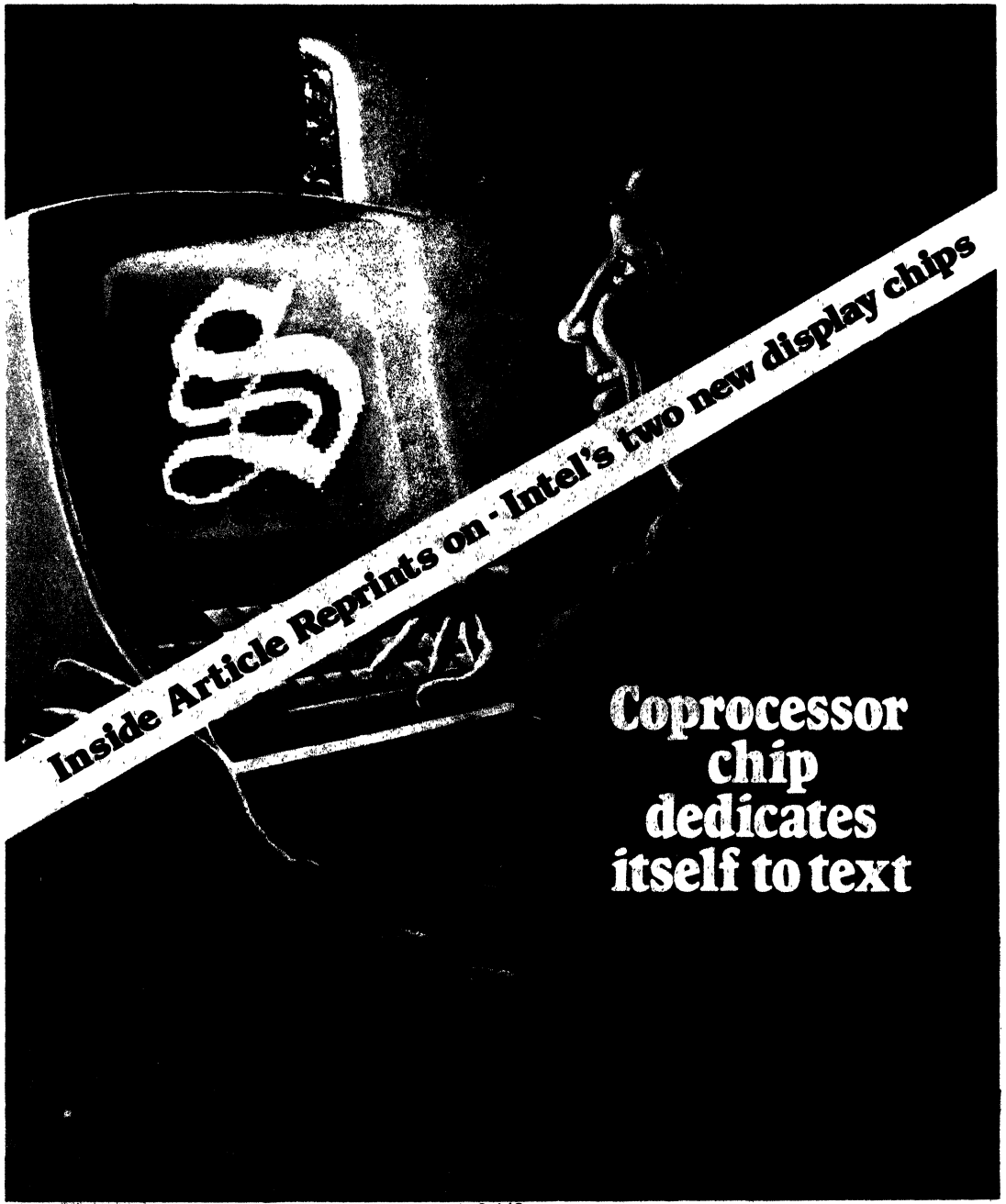
ElectronicDesign®

FOR ENGINEERS AND ENGINEERING MANAGERS—WORLDWIDE

FEBRUARY 17, 1983

COMPONENTS SPECIAL

• Capacitors • Precision resistors • Shielding materials
• Selecting electrolytics • Power MOSFETs for switchers



The first chip dedicated to text manipulation, the 82730 operates as a coprocessor to a host CPU and executes many high-level commands that reduce the software needed for processing text.

Text coprocessor brings quality to CRT displays

The quality of text in raster-scanning CRT displays has always been a tradeoff against the complexity, performance, and cost of the associated video system. By allocating many of the complex display functions to firmware, a dedicated text coprocessor chip, the first of its kind, replaces printed-circuit boards that contain more than 100 ICs while increasing system performance by relieving many of the host processor's text manipulation tasks. The chip thus makes possible the high quality and high performance sought, without the need to compromise because of high design complexity and high cost of text-processing hardware.

Though its speed makes the 82730 text coprocessor beneficial on its own, its utility can be enhanced considerably when working with the 82731 video interface controller. Together they provide proportional spacing, simultaneous subscript and superscript displays, dual cursors, dynamically reloadable character fonts, and user-programmable field and character attributes. By adding still another chip, the 82720 graphics display controller, the device can display high-resolution graphics and text at the same time.

Housed in a 68-pin package, the 82730 text coprocessor combines a direct memory access channel and a processor bus interface that permit it to fetch its own instructions and data from the host system's memory, independent of and in parallel with the host CPU.

The two processors communicate through messages—commands, parameters, and status words—which are placed in a communication block inside a shared memory. The host issues commands by preparing messages, storing them in the communication block, and directing the coprocessor's attention to them by activating a Channel Attention signal, which is implemented in hardware. In return, the coprocessor sets a flag in the shared memory that notifies the host when it has executed the command.

The 29 high-level commands built into the 82730 break down into two groups: channel commands, which work at the system level to start and stop the display and to communicate status and similar information, and data-stream commands, which are incorporated directly into the display-data strings to govern the DMA process and control row

Anand Balaram, Product Marketing Engineer
Andrew Volk, Project Manager
Intel Corp.
3065 Bowers Ave., Santa Clara, Calif. 95051

Reprinted from ELECTRONIC DESIGN - February 17, 1983

ORDER NUMBER: 210832

Text coprocessor

characteristics, character attributes, and so on.

The 82730 resides on a local system bus with the host microprocessor, such as the 80186 CPU, and therefore provides the same address, data, and control signals as the main processor. By handling several of the tasks typically done by the host processor—like DMA control and display formatting—it leaves the host free for other tasks.

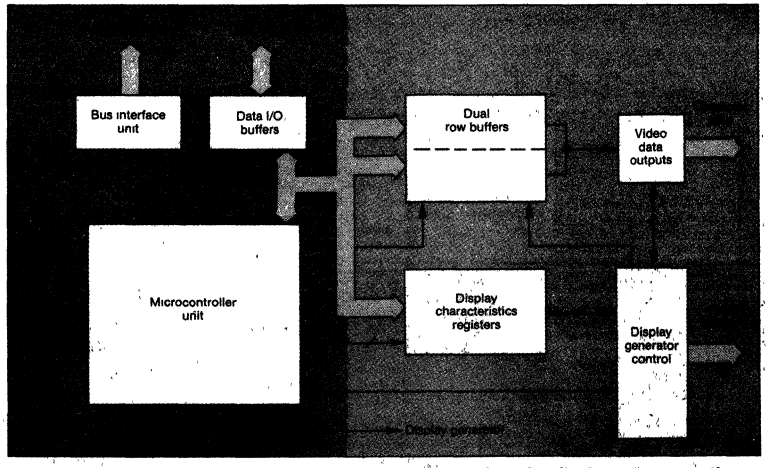
For example, when the coprocessor is configured to share the CPU bus, a portion of the host microprocessor bus bandwidth must be devoted to the DMA process that refreshes the CRT. However, the 82730's high-speed intelligent DMA controller (operating at a maximum data rate of 4 Mbytes/s) helps minimize the time spent executing the refresh operation, while simultaneously handling the formatting of the display data. A different approach involves a dual-ported memory architecture, which places the memory between the CPU and the coprocessor. That completely frees the processor bus of the refresh activity, allowing the host more time to execute other tasks. It has become a more cost-effective method, as some dynamic memory controllers now contain dual-ported arbitration logic on chip.

Inside the chip

The basic architecture of the coprocessor is divided into two main parts: a memory interface and a display generator section (Fig. 1). The memory interface lets the coprocessor and the system processor communicate via the shared memory. The display generator, in turn, responds to the data provided by the memory interface and carries out the display operations.

The memory interface actually comprises two smaller subsections, a bus interface unit and a microcontroller unit. The bus interface provides an intelligent connection from the 82730 to the host processor bus and also buffers the data transfer requests from the microcontroller. Upon initialization, the bus interface can be programmed for 8- or 16-bit data and 16- or 32-bit addresses. Furthermore, the host interface can be configured for 8- or 16-bit-wide data buses, making the coprocessor compatible with 8- or 16-bit host processors, like the 8088/80188 and the 8086/80186. Running at 8 MHz maximum in 16-bit systems, the 82730 handles the maximum DMA rate of 4 Mbytes/s.

The microcontroller unit stores the microinstructions for the 82730's high-level operations. The microcontroller's internal processor manages the memory transfers, interprets the commands embedded in the data stream, and executes those commands by sending data to the appropriate control registers or display data buffers. To optimize the transfer of data between the system and the CRT interface, the coprocessor uses three clocks—one for the host interface, the other two for video data. The memory interface section runs from the bus clock, the CRT interface operates from a reference and a character clock.



1. Divided into two main sections—a memory interface unit and a display generator—the 82730 text coprocessor can operate at optimum speed since each section can function independently at a different clock speed.

Although the coprocessor packs a considerable amount of processing power on a single NMOS chip, it cannot handle the high video dot rate needed to deliver high character counts to the CRT display. For that, it needs the 82731 video interface controller, which gains its high speed and drive capability from bipolar technology. In addition, the combination of the 82730 and 82731 succeeds in reducing the video interface to just a few latches and a software character generator residing in RAM or ROM (Fig. 2).

Inside the 82731 are the reference- and character-clock generators, a video shift register, and all attribute logic (Fig. 3). Housed in a 40-pin package, the circuit offers TTL-compatible inputs and outputs except for the video output, which is ECL-compatible and provides a dot-shift clock rate of 50 MHz maximum on characters up to 16 dots wide. The circuit proportionally spaces characters by accepting the width sent from the character generator and sending an appropriate character-clock output whose period determines the variable width of the character to be displayed.

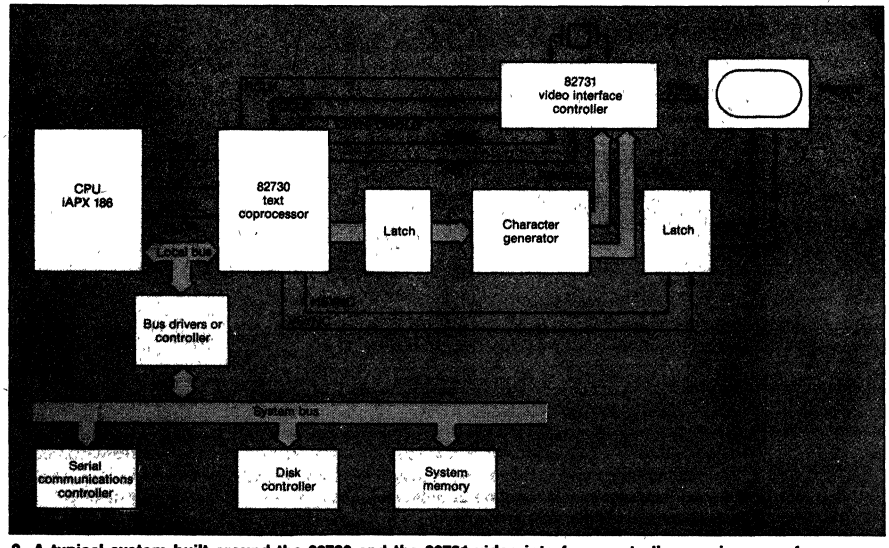
The video interface controller can employ an inexpensive, low-frequency crystal and internally multiply that frequency to generate the high-frequency dot clock. It also supports control functions such as screen reverse video, synchronized character field, and tabbing operations. The dot clock drives the internal video shift register, the character clock controls the unloading of data from the row buffers in the 82730, and the reference clock establishes the raster and screen formats. The reference clock also supplies the basic timing for the horizontal sync, blanking, border, and active display time. The corresponding vertical attributes—except border—are driven by the horizontal line time. All seven of these screen parameters are programmable by the system designer with the 82730.

System interfaces are simple

As a coprocessor, the 82730 has the same bus-control signals as an 80186 host processor and thus can share the system-bus controllers, drivers, and latches. The host processor and the 82730 arbitrate for control of the local bus through the Hold and Hold Acknowledge lines (HLD/HLDA). The Channel Attention (CA) and System Interrupt (SINT) control lines complete the wired interface. With this configuration, the 82730 has access to all the memory that the 80186 CPU has available.

Anytime the CPU wants to send a message to the 82730, it writes the command and busy flag into the 82730 command block and then pulses the coprocessor's CA input to inform it that a message is waiting. The 82730 then raises the HOLD output and waits for access to the bus. When the CPU relinquishes the bus, it raises the HLDA input of the 82730.

Once the 82730 becomes active, it transmits the command block address that was stored in its

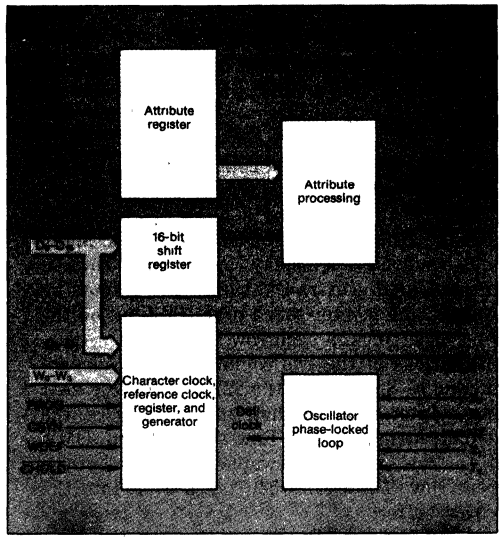


2. A typical system built around the 82730 and the 82731 video interface controller requires very few additional ICs to mate with a host processor like the 80186. Only the system bus drivers, some latches, and a character generator are needed.

registers during initialization. That address, in conjunction with the appropriate memory control signals—including read or write strobes, lower or upper address latch enables, upper address output, or data enable output—will either read the command block or write to it. All these signals are coordinated by the bus clock.

Whenever the 82730 must send status information to the host CPU, it gains control of the bus and places the data into the status location in the command block. The bus is then released and the coprocessor notifies the CPU through the SINT signal. When the coprocessor is using a dual-ported memory to communicate with the 82730, the HOLD and HLDA signals are not employed. Instead, the 82730 accesses the dual-ported memory directly rather than acquiring the bus from the CPU.

When the display process is activated, the coprocessor uses its built-in DMA capability to fetch display data from the memory. The data consists of character data mixed with data-stream commands; embedded data-stream commands provide the flexibility to manipulate data on the fly.



3. The 82731 video interface controller is manufactured with bipolar technology, enabling it to handle video dot rates of 50 MHz and higher, which are needed by high-character-count displays. The controller serializes the parallel character outputs from the coprocessor and adds the desired attributes to each character.

Soft fonts loaded

The 82730 also permits soft fonts to be automatically loaded into RAM-based character generators. Addresses and data stored in the system memory are then loaded into the row buffers of the coprocessor. During blanked rows (generally during the vertical retrace), address information is loaded into a latch and data is written to the character generator.

The 82730's dual row buffers help reduce the bandwidth and access time requirements for the system memory. The data stored in one buffer is being used to display a row on the screen while the second buffer is being loaded, by the microcontroller, with the next display row from the system memory. Up to 200 characters can be stored and displayed by each row buffer. Furthermore, since the display generator section operates asynchronously with the microcontroller unit, each can operate at optimal speed. Processing is synchronized by internal flags and shared internal storage, and data that will be displayed is exchanged through the row buffers.

The coprocessor's display generator handles the data that defines the timing and the operation of the CRT interface. That data, which is stored in the display characteristics registers of the chip, controls every aspect of the display—from the screen's format to the blink rates of the characters and cursors. All the parameters that define the initial display characteristics can be set by one command—MODEST—thus reducing the time and effort required to establish a screen format.

Beneath the simplicity of the hardware shown in Fig. 2 are the high-level instructions—channel commands—and the data-stream commands. When combined with a table-driven linked-list data structure, they ease software development.

Central to the software is the command block, through which all channel commands are transferred between the coprocessor and the host. This block is located within the shared memory, and its exact position is set during the 82730's initialization routine (Fig. 3a). Once established, it contains all the information needed to start the display-data fetch; to communicate status, interrupt, and cursor position information; and to give the location of the mode block, which contains all the parameters for setting up the display. The START DISPLAY channel command begins the sequence (Fig. 3b).

Since the display data is set up within linked lists, the coprocessor can rapidly change any of the lists without shifting huge amounts of data. The display fetch starts with the value of the list-switch bit which selects one of two list-base pointers in the command block. The pointer points to its string pointer list; the pointers in that list direct the on-chip DMA to the data strings containing the desired display data and data-stream commands. The programmer can modify one pointer list while

displaying from the other, and can also switch screens merely by changing the list-switch bit, thus eliminating time-critical data manipulations.

Two data-stream commands—End of String (EOS) and End of Row (EOR)—are key to the linked list and DMA activities. Strings are a logical concept: they contain blocks of contiguous data stored anywhere in memory. In contrast, rows are a physical concept and represent a block of characters that make up a physical row on the screen. Many strings can exist in a display row, or many rows in a string. (Only the extra DMA overhead of fetching the new string pointer sets a practical limit on the number of strings in each row.)

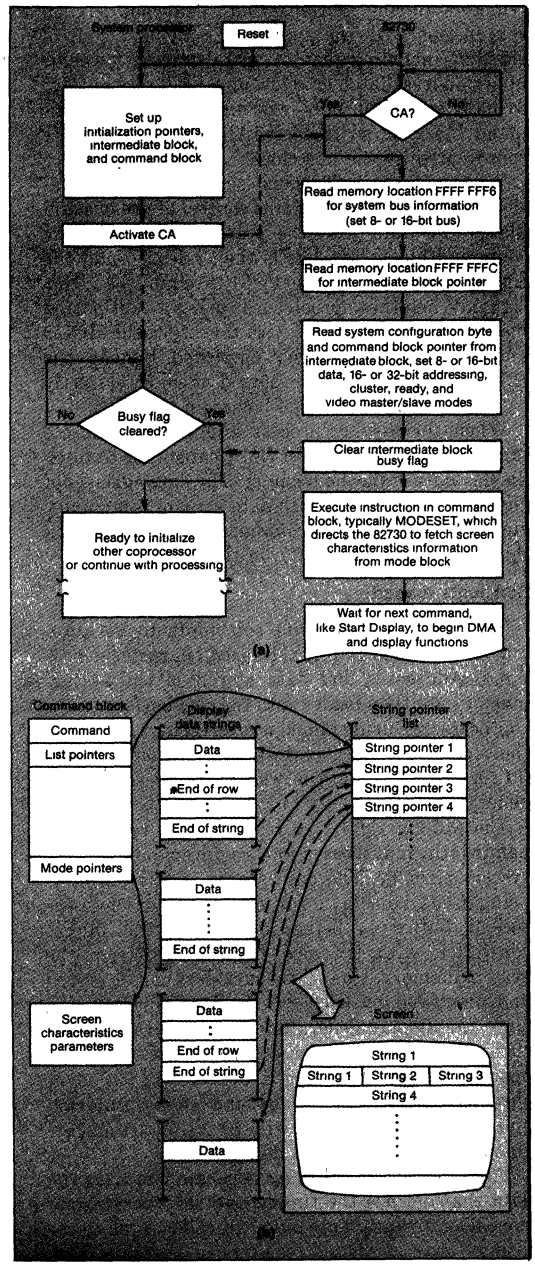
The actions of the two commands are independent. End of String tells the 82730 to get the next string pointer from the list, and from there, the next data string. End of Row suspends the DMA until the row buffers are swapped at the end of the current row. The DMA then takes over, into the new row buffer.

String manipulation fosters high speed

Strings are commonly the next level of text organization above single characters. With the 82730, a string can be as small as a character or it can be a word, row, sentence, paragraph, or a page of characters. These high-level entities can be moved merely by manipulating a small string pointer table (Fig. 5). The heart of the algorithm for word wraparound, a common feature in text processors, can easily be accommodated by a single command such as the String Compare command of the 80186. Word wraparound is then achieved by scanning the data (not moving it) and manipulating a few pointers. Earlier system designs would have required a multiple-instruction software loop that scanned and moved every individual character.

An extension of the linked list allows programmers to set up several independent data windows on the CRT screen in a virtual screen mode. That feature is especially helpful if a user wants both a menu window and one or more work-space windows. Such a scheme saves a lot of time for the end user—eliminating the back-and-forth movement between menus and working text. To set this up, several data structures, each with its own command block, can be accessed in a table-driven sequence to put data in a given window on the screen (Fig. 6).

The string list and data strings are the same for regular or virtual modes; only the structure of their command blocks differs. Thus, each virtual window can be an independent software entity in the system, and the 82730 can present these independent data bases simultaneously.



4. Both the host CPU and the coprocessor go through an initialization sequence when the computer system is reset (a). The coprocessor then looks for a START DISPLAY command so that it can load the various data strings from the system memory into the display generator section, attach attributes, and display the data on the CRT (b).

Text coprocessor

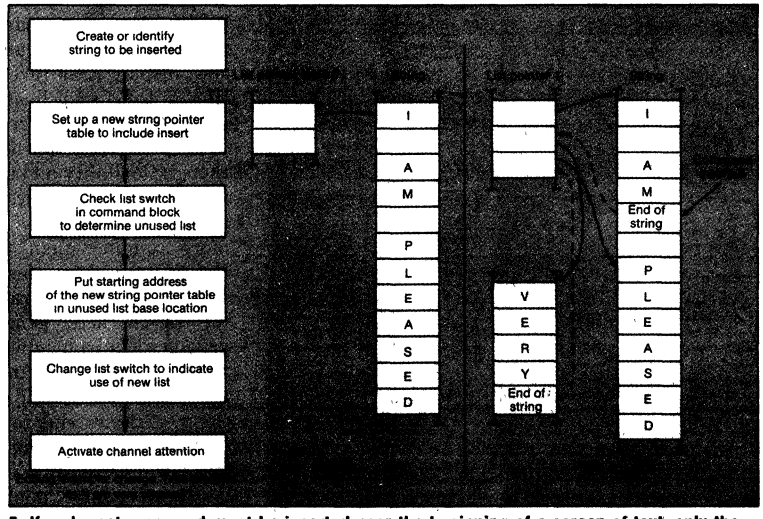
Multiple 82730s can also be used in a single system. Up to four devices can be clustered in a single system, with one serving as a system master and the others as slaves. The data for this cluster can be interleaved, permitting the cluster to work from one data base to get more characters per screen or more bits per character. Also, in the slave mode, the 82730's video outputs can be synchronized to an external video signal, giving the system such capabilities as mixed text and graphics, broadcast subtitling (text overlay), and overlays for video recording.

Attributes enhance display quality

The designers of the 82730 have given it the ability to highlight various areas of an on-screen document through the use of character and field attributes. In the 16-bit data word, for example, only the most significant bit is committed; it serves as the command or data designator. If set to 1, the word is a data-stream command, with the remaining 15 bits becoming one of the predefined instructions. However, if the MSB is 0, the other bits are at the discretion of the designer, who may choose which and how many are needed for character codes, attributes, or user-defined functions.

The 82730's six predefined attributes—reverse video, invisible, blinking character, two underlines, and a special graphics character—can be programmed to respond to any of the 15 bits, or they can be completely disabled. In addition, they can be set character by character or through a field-attribute mask. All can be attached to any character. The blinking character can be programmed for a wide range of duty cycles and blink rates. The two underlines can be independently positioned anywhere in the row height, and the position can be changed from row to row. Thus the underline can be doubled or serve as a strike-through line, a fraction line, or an overbar. One of the underlines can also be programmed to blink at the same rate as a blinking character.

The graphics character is relatively important, since it permits character information to be displayed to the full height of the row. It causes the chip's line-counter output to count from zero at the top of the display row continuously through to the bottom of the row. When used with special characters, this attribute allows business forms and graphs to be easily constructed.



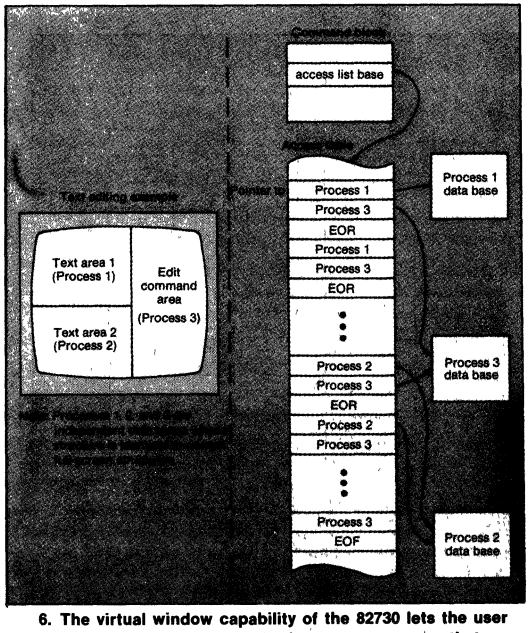
5. If a character or word must be inserted near the beginning of a screen of text, only the list pointers must be changed to add the item. In older systems, all the characters following the insertion or deletion were shifted in the memory to revise the display.

Text coprocessor

Another capability of the 82730 is subscript and superscript characters, done by manipulating the line-counter outputs. The SUB SUP N data-stream command declares which and how many pairs of characters will be displayed simultaneously as subscripts and superscripts.

Proportionally spaced displays could cause some subscript and superscript characters to have different widths and thus disrupt the vertical alignment of a character pair. A special output of the 82730 called Width Defeat prevents that misalignment by causing the 82731 video interface controller to enforce a predefined width—programmed upon system initialization—during the display of subscript and superscript characters.

The proportional spacing is performed by the reference and the character clock. Used to shift out the character and attribute data, the character clock operates during the display field. Its frequency can vary character by character, up to a rate of 10 MHz, to set the width of the character currently being displayed. The video interface controller takes the character width information that has been supplied by the character generator and produces a variable width character clock that supports the proportional spacing. This approach also greatly reduces system complexity and cost compared with previous designs.



6. The virtual window capability of the 82730 lets the user arrange independent areas in the system memory that can be displayed simultaneously on the CRT monitor.

Screen and row formats are flexible

The reference clock signal in a system that contains the 82730 and 82731 chips is a constant-frequency clock that forms the time base to generate the horizontal scan lines and vertical frame periods. One scan line can last for 256 reference clock periods, and one frame can contain up to 2048 scan lines. Within these periods, the respective synchronization pulses and the border and character fields can be set anywhere within that range. All these timing relationships, including the scan and frame periods, can be changed on a frame-by-frame basis to suit changing applications.

The screen format is flexible all the way down to the row level. For instance, the height of a row (up to 32 scan lines) and the vertical position of the characters within that row can be changed from row to row by a single data-stream command called FULROWDESCRPT. In addition, the command lets the programmer set the starting and ending scan lines within the row for the normal, subscript, and superscript character fields and the two cursors.

The same data-stream command that defines the row characteristics can also be used to blank the row, display it as reverse video, double its height (for up to 64 scan lines per row), or eliminate the proportional spacing.

Graphics, too, can be handled by the 82730, although flexibility and resolution are not as high as with the 82720 graphics display processor. Business applications typically need graphics that are no more complex than two- or three-dimensional charts or business forms. These formats can be stored as special characters in a standard font set for the character generator. Even more complex graphics can be handled through the use of mosaic graphic cells, which can be stored in RAM to permit alterations. Of course, as in most systems using floppy-disk systems for main storage, the desired fonts or graphics forms can be saved on the disks and downloaded as needed for the display.

There are many applications that also require a simple graphic display along with text—signature verification on banking terminals and general-purpose credit verification, for example.□

intel

ARTICLE REPRINT

AR-296

September 1983

MIGHTY CHIPS

By James Fawcette
PC World Vol 1, #5

Order Number 230810-001

Something exciting is going on. But like most significant events, it is not happening quickly. Spurred on by developments in integrated circuit technology, a new generation of personal computers is taking shape, and the IBM PC and its clones are at the forefront.

As IBM PC users, it's sometimes hard to remember that the inanimate metal boxes in front of us are susceptible to evolution. But occasionally a product is introduced that forces the complete redesign of our personal computers.

Integrated circuits (ICs), the devices that bring intelligence to our machines, have reached a new level of technological achievement, and now the computers that use them must advance as well. Strange as it seems, these small silicon chips are setting the guidelines for the next generation of personal computers.

THE CHIP MAKERS

Now that personal computers have caught on, the semiconductor manufacturers who make ICs are eyeing the swelling market for personal computer ICs.

Dozens of newly developed semiconductor chips are being aimed at the personal computer market. These chips range from hard disk controllers that speed access time to linear predictive coding processors for speech recognition. With these new ICs driving personal computer design, we'll soon see machines we once only reasoned would exist: diskless computers running a wide array of software loaded over telephone lines; computers that display text exactly as it will be printed, with justified margins, superscripts and subscripts, and bold and italic typefaces on screen; and systems with greater, more accessible graphics.

As computer design is simplified by these advanced ICs, product differentiation will become greater. This portends the death of those PC clones capable only of basic spreadsheet and word processing operations. Instead, to survive in the increasingly cost-competitive, standardized personal computer market, small-system manufacturers will tailor their products for niche markets.

BIG BLUE

Intel Corporation, located in Northern California's renowned Silicon Valley, is one of the largest and most innovative chip manufacturers in the industry. IBM has been committed to Intel products for years; the PC is built around Intel's 8088 microprocessor and, as recently as late 1982, IBM invested $225 million in a minority share of Intel stock. A commitment this size is a good indicator of IBM's faith in Intel products. IBM's good faith and multimillion-dollar investment is guaranteed by Intel's long-standing promise that software written for the 8088 will run on all its future processors.

By taking a close look at the Intel ICs, we can gain valuable insight into the capabilities of the IBM PCs that will be built around them. The design philosophy of Intel's IC family differs radically from that of competitors Motorola, National Semiconductor, and Zilog. Diverse chip designs mean that the system designs of the IBM PC and its competitors, such as Apple's Lisa (based on the Motorola 6800 microprocessor), will also be radically different.

THE MICROPROCESSORS

Of the many Intel chips being produced, some will have a greater impact on the computer industry than others. In the vanguard will be the new microprocessors.

Design of the PC was shaped by IBM's surprising selection of the 8088. This choice caught most industry observers off guard since IBM, also the world's largest semiconductor manufacturer, had traditionally used its own designs for computer logic. Once Big Blue settled on the 8088, Intel's design philosophy was firmly implanted in the PC—from the 8088's segmented memory scheme to its 16-bit registers and 8-bit bus.

Like the 8088, each of the four microprocessors Intel is now readying for production could dramatically influence the design and performance of tomorrow's PCs.

The 80186. The recipe for putting an entire central processing unit (CPU) board on one chip is easy. Take an; 8086 (the 16-bit bus big brother of the 8088), speed it up, and then add most of the support chips essential to making the 8086 run in a personal computer. Reduce the size with the help of computer-aided design until all the chips fit onto one sliver of silicon, and *voila,* you have the 80186 (186), an entire motherboard on a chip.

While firming up plans for full-scale production of the 186, Intel is currently providing samples of the chip to computer manufacturers, including MAD Computer and Durango Systems. The rewards for using this newest chip are many: manufacturing costs are cut since a single IC is less expensive to buy than a boardful of them; physical CPU size is reduced, opening the way to shrink overall computer size or to put more power in the same box; and development time is cut for computer designers, which means considerable savings for system makers.

The 80188. If the 186 is too rich for your taste, the 80188 (188) may be more suitable. As with the 186; the 188's core CPU and support chips are melded on a single IC; like the 8088, however, the 188 has an 8-bit interface to the outside world (the 186 has a 16-bit interface). The 188 decreases costs by allowing computer manufacturers to use less expensive 8-bit peripherals. Although the 186 has received more publicity so far, the 188, aimed squarely at the massive 8-bit computer market, is expected to be used in greater numbers, at least in the short term.

The 80286. Powerful multiuser systems will benefit the most from the 80286 (286), possibly the most powerful microprocesor commercially available to date. Squeezing 150,000 transistors on a chip, the 286's designers have integrated a pair of HMOS-III (Intel's own proprietary process technology) 8086s and numerous other very large scale integration (VLSI) components. The resultant chip is two to three times faster than the Motorola 68000 even though both chips can address about the same amount of

memory. The 286 has very high speed (1.5 million instructions per second, five to six times faster than the 8086), about 16 megabytes worth of addressable physical memory, the ability to address a virtual memory of 1 gigabyte per task (equal to the capacity of 100 IBM XT Winchester drives), and the ability to provide several layers of muiltiuser security on chip.

The 80386. Not yet built, the 80386 (386) is promised for 1984, but the release date may slide to 1985. If the 286 is vastly more powerful than the 8088 or 8086, then the 386's potential is staggering. Complementary metallic oxide semiconductor (CMOS) process technology, which lowers power consumption, is being used to build this 32-bit chip. Intel, Motorola, and National Semiconductor are already jockeying for position in what will be an intense competition for the 32-bit market. Motorola is claiming that its 68020 will be the first widely available 32-bit microprocessor when it is introduced later this year, although NCR has already scooped the industry with its 32-bit chip. Hewlett-Packard, not to be outdone, has put 450,000 transistors on a single proprietary 32-bit microprocessor, which is used in the $20,000 to $30,000 HP 9000 work station.

How will these processors impact the personal computers that use them? The most obvious effect will be faster performance. Even the budget model 188 boasts two to five times the instruction and execution speed of the 8088 in today's PC. A 286 is about twice again as fast as the 188, and next year's data-gobbling 386 will have more speed than anyone can immediately use.

Since the 188 is ideal for low-priced portable computers, it ceates the ironic probability that a PC-compatible portable may soon by available that will run the IBM PC's full line of software and run it faster than the full-sized PC.

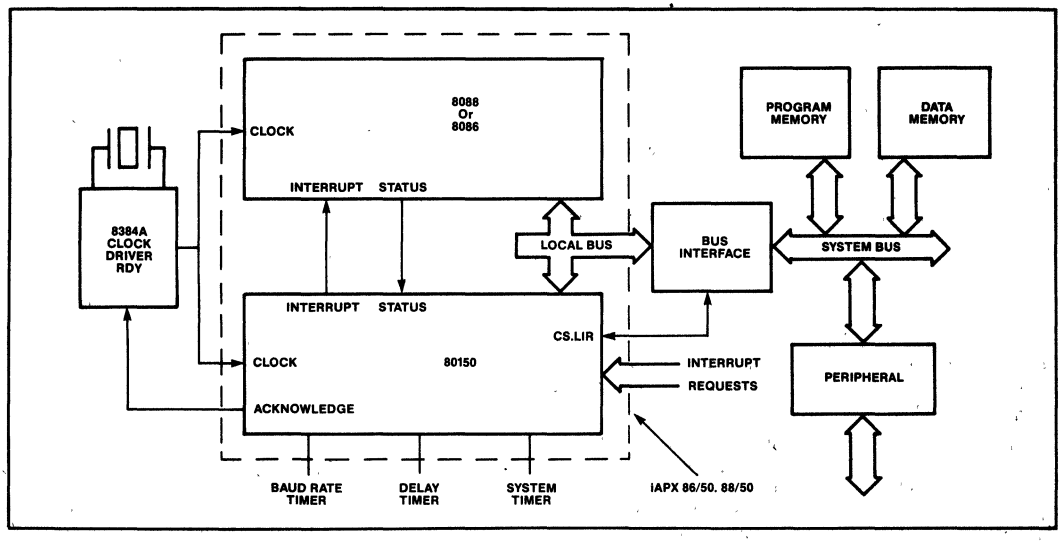
SOFTWARE ON SILICON

One chip ready to plug into the next generation of personal computers is the 80150 (150) CP/M software-in-silicon operating system. A complete CP/M-86 operating system is stored in ROM on this chip, along with drivers for input and output devices.

Use of a 150 CP/M chip will eliminate the traditional booting up procedure of loading an operating system disk and reading its contents into operating RAM. Instead, the user will simply turn on the computer and press a CP/M-86 button. Again and more importantly, this chip lowers overall computer production costs since a disk drive and attendant control circuits are replaced by a solitary chip.

Another chip, similar to the 150, has Intel's proprietary RMX operating system in silicon. This little-known RMX chip is also suitable for present and future IBM PCs.

Many people question the wisdom of putting software in silicon. "Software should be soft," says Bill Gates, chairman of the board at Microsoft. He points out that operating systems are constantly updated; for instance, Microsoft will soon offer a revised version of MS-DOS that supports networking. Such updates can't readily be added to a hardware production line and certainly won't help the ROM chips already in users' computers.



BLOCK DIAGRAM OF INTEL'S 80150 CP/M ON A CHIP WITH THE 8088 OR 8086 MICROPROCESSOR

Still, Intel argues that its choice of CP/M makes the 150 practical. "We picked CP/M because it is a mature operating system," Says Intel's product marketng engineer for software on silicon, Carl Buck. "We'd have more difficulty with a less developed product." The many versions of MS-DOS helped eliminate that operating system from consideration. Yet according to Digital Research President John Rowley, Intel left some room on the 150 chip to add to CP/M in the future.

Also, use of the 150 CP/M chip doesn't preclude the use of other operating systems. PC-DOS could still be loaded into a system and run, making use of the 150's input/output drivers.

PORTABLES

Having software on silicon opens the way for very powerful diskless portable computers. The minimum configuration for a 188-based unit with the 150 CP/M operating system could include one or two BASIC applications programs in ROM, providing spreadsheet and word processing power in a unit the size of a keyboard with a small flip-up screen. Intel Product Marketing Engineer Tony Zingale suggests we may soon see truly usable portables selling for around $500.

More ambitious and expensive portables could accept applications software over telephone lines, loading them into a variety of media. Several memory technologies will compete for room in portable computers, including magnetic bubble memories, already being used in the Grid and Teleram computers. Commercially available bubbles have 4 megabit capacity, while 10- to 16-megabit bubbles are projected for the near future. Japan's NEC reported a major breakthrough that within 5 years will allow bubbles to store 1 gigabit of data. Of course, 8 of those bits are needed to store 1 byte of data.

Vying with bubbles in some applications and complementing them in others are electronically programmable read-only memories, or EPROMs. Like ROM, EPROMs are nonvolatile chips. Unlike ROM, EPROMs can be reprogrammed. Intel now offers 256K EPROMs, and it is anticipated that other companies will offer 256K EPROMs before the year's end.

GRAPHICS

The space created on the motherboard by the 186 and friends will enable computer designers to add more graphics capability to their systems. Like the 150 there are co-processor chips ready for the task.

A pair of Intel ICs, the 82720 (720) graphics display controller and the 82730 (730) text co-processor, are touted as providing vastly enhanced and simplified displays. With the 730, text can be displayed on the computer screen as it will be printed out. Italics can be mixed with straight text, and superscripts and subscripts are shown without the annoying and often misleading arrows common in today's software.

Editing can be speeded up by the 730's support for split screens, multiple windows, dual cursors, smooth scrolling, and table-driven linked lists. Displays of up to 200 characters per row and 128 lines per screen can be supported, and unique character sets, such as Arabic or Japanese, can be built.

Even more capability can be added though the 720, an IC that works with or without the 730. Introduced in September 1982, the 720, a joint effort between Intel and NEC, is said to be integral to graphics plans for NEC's 8086-based Advanced Personal Computer.

One application in which the 720 and 730 will shine is opening windows on-screen. Most computer users are familiar with the ability of Apple's Lisa to link spreadsheets, graphics, and word processing through multiple displays, or windows, on one screen. Lisa uses memory-hungry software and dedicated hardware. Apple's initial release uses 1 full megabyte of RAM, and Lisa will soon be offered with 4M of internal memory in addition to a mandatory 5M hard disk.

For comparison, the IBM PC, limited by the range of the 8088, can address 1M tops. VisiCorp's *Visi/ON* promises Lisa-like graphics and program-linking capabilities for the IBM PC, with lower memory demands and no dedicated hardware other than a mouse. Although *Visa/ON* supposedly runs faster with an 8087 math co-processor, VisiCorp will not comment on whether its software will make use of the 720 or the 730.

BIT-MAPPED GRAPHICS

Both Lisa and *Visa/ON* use bit-mapping, a process that the 720 and the 730 are said to simplify. In plain words, to create an image on-screen, the electron gun that illuminates the screen must be positioned and then turned on and off. Data to do this is stored in RAM as a bit-map memory corresponding to positions of pixels lit on the screen. For one-level monochrome displays, 1 memory bit describes each pixel; for color and levels of grey, several bits must be used to describe each pixel.

Creating images is a lengthy chain of simple operations. In a system that uses the 8088 alone, the microprocessor is heavily burdened and the software runs slowly. Using complementary chips to take up part of the processing chore will speed up the process considerably. This is where the 720 and the 730 come in By doing tasks such as looking up and manipulating a library of commonly used figures, quickly accessing the bit-map memory, and rewriting the bit map, both chips speed text and graphics operations.

FLAT VS. SEGMENTED MEMORY

Use of the 720 and the 730 demonstrates Intel's design philosophy and how this philosophy impacts the IBM PC. Computers such as Lisa that are based on the Motorola 68000 have a flat memory, while computers based on the 8088 or 8086 use segmented memory. According to Intel,

segmented memory (see "How the PC thinks," *PC World,* Vol. 1, No. 1) works better for text and graphics manipulation than its flat counterpart. Ordinarily in processing any string of characters, changing a single letter in a string of text means repositioning every character in a document. But since segmentation uses pointers to locate data in memory, only the pointers locating the beginning and the end of a passage of text have to be changed. Similarly, pointers in memory can be used to position bit-map data corresponding to multiple windows on-screen, eliminating the need to recalculate and manipulate the entire bit map. Segmented vs. flat memory has become somewhat of a religious issue in the semiconductor industry.

Intel and Motorola also differ on how much burden to put on the CPU. Motorola's 68000 is faster than the 8088 and the 8086 and can address more memory than either of those chips or the 188 or the 186. But the 186 and the 286 are substantially faster than the 68000. Also, the 286's ability to address 16M opens the way to using large memory segments, strengthening Intel's case for segmented memory.

In many 68000-based high-end systems the computer designers have decided to use a co-processor, either bit slice, or in one case, an 8086, to do graphics. Many people are skeptical of Intel's graphics approach, but Intel maintains that its approach will allow computer designers greater flexibility. In an ultimate system, multiple 720s and 730s could be combined to handle interactive windows under the direction of a 286 processor, while more complex imagery (beyond the practical ability of bit-mapping) could be managed by an 80287 math co-processor, the next generation cousin of the 8087. The creation of three-dimensional graphics that can be rotated on screen for advanced computer-aided design and manufacturing systems, for instance, is best handled by Vector Graphics rather than bit-mapping.

SOFTWARE DEMANDS

Yet there is more to computer design than hardware. Software must be written to take advantage of the new IC's promise. In the case of the 286, no operating system yet exists that can take full advantage of its operating capabilities. Plug-ins currently on the market that add the 286 to the IBM PC provide little more than a faster 8086. Only new operating software will use the new chips to their fullest potential.

One solution on the horizon is a 286 version of XENIX due to be introduced mid-1983. XENIX, a multiuser operating system with a visual shell similar to MS-DOS, is a takeoff on Bell Labs' UNIX operating system. A licensing agreement among Intel, Bell Labs, and Microsoft, the author of XENIX and MS-DOS, is reported being negotiated. Negotiations between Intel and Digital Research to provide a CP/M variant for the 286 have been underway for some time but have reportedly stagnated.

For lower-end systems Microsoft is said to be upgrading MS-DOS to accommodate networking. This advance comes at the right time, as the 188 and 186 open up sockets that could be used for local area network chips such as the programmable Ethernet chip from Intel.

As long as software and hardware keep growing rapidly together, PC users will be offered a continuing stream of improved computers and ever more capable plug-in boards. The variety seems endless and next year's crop exciting.

PROCESSING

VLSI Coprocessor Delivers High Quality Displays

Many microprocessor-based systems today use VLSI technology in processing and memory components. However, designers of subsystems have, up until now, not been able to incorporate this technology into their products because of the lack of available ICs. When, in 1981, NEC introduced the 7220 graphics display controller, users found that they could bolster system performance by off-loading graphics control chores from the system CPU. Second-sourced by Intel as the 82720, the chip uses its own drawing processor to access the required positions in the bitmap and handles both processing and display functions.

Now, Intel is poised to introduce a text coprocessor, the 82730, which is specifically tailored to execute data manipulation and display tasks. Lucio Lanza of Intel explains, "In an intelligent terminal or workstation, the CPU spends a lot of its time manipulating both graphics and text. We have identified these areas in terms of CPU use and we have distributed these blocks so that the CPU is not overburdened."

Coprocessors fall into two categories based on their architecture and operation. One type expands the microprocessor's own architecture by adding additional hardware and instructions. This type of tightly coupled coprocessor can be thought of as a transparent expansion of the microprocessor's architecture and works in sychronization with the CPU. Intel's first such coprocessor, the 8087, was designed for numerics processing and increased the microprocessor's math performance as much as 100 times.

The second type of coprocessor independently fetches its own data and sends instructions in parallel to the microprocessor. It therefore allows the microprocessor to process the tasks it handles best and delegate to the coprocessor the task it is best equipped to handle. In this cate-

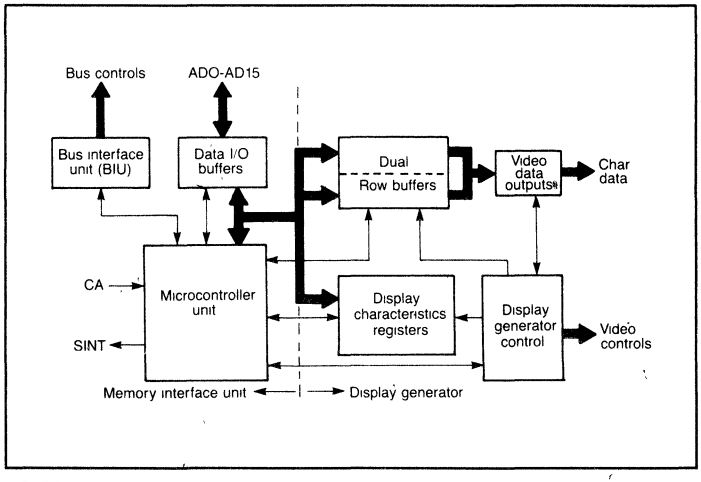


FIGURE 1: Block diagram of the 82730.

Andrew Wilson
Technical Editor

gory are I/O channel coprocessors and others that deal with communications and text processing tasks.

"The 82720 is not yet at this level," Lanza said, "since it does not have the capability of going to memory and extracting its own instruction and executing it—it needs something to spoon feed it."

Coprocessors of the second category do not monitor the CPU instruction stream. Instead, they are linked to the CPU via messages prepared and stored in shared memory. The CPU will prepare data and high level directives and then place them in memory. Upon completion of this control block, the CPU will alert the coprocessor by signaling it through a common channel attention line. From that point on, the coprocessor works on its own, fetching required data and instructions and then executing those instructions.

It is not synchronized with the CPU but works asynchronously and independently. When the coprocessor completes its task, it informs the CPU by signaling on the CPU's interrupt line.

The rationale for designing a coprocessor with one or the other architectures depends on the application requirement. Tightly coupling the coprocessor with the CPU gives the advantage of a short coprocessor preparation time but has the drawback of consuming the CPU's bus bandwidth.

In the case of numeric processing, the speed of executing the floating point algorithm is of paramount importance. Reducing the preparation time of the coprocessor task is the key because of the number of microseconds it takes to execute the task. Rapid algorithmic execution requires tight coupling. In the application of the I/O related coprocessor, the task execution time is much longer and the requirement for bus time can be much higher. And, for I/O operations the preparation time is not critical. A shared memory coupling is preferred for those types of applications because it provides greater flexibility in the design of the bus structure.

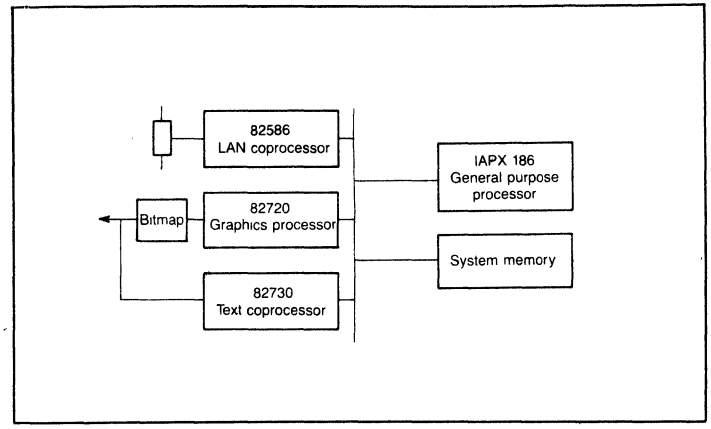


FIGURE 2: Building block approach.

Text coprocessing

"In the design of the 82730," said Lanza, "we have tried to eliminate all the known differences between what is visible on the screen and what is obtained on the printed page. In word processing systems today, even the length of a row on the CRT is sometimes not the same as the length seen in print. Clearly, when you are editing text this becomes a major problem."

The 82730 supports the generation of text displays through features which include proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. Editing capabilities are further enhanced by features such as split screen, virtual windows, smooth scrolling and table-driven linked lists.

Figure 1 shows a block diagram of the 82730. The chip is divided into two main sections—the memory interface unit and the display generator. The memory interface unit provides the communication between the 82730 and the system processor, while the display generator acts on the display data and carries out the display operation.

Communication between the 82730 and the CPU takes place through messages placed in communication blocks in shared memory. The processor issues channel commands by preparing these message blocks and directing the 82730's attention to them by activating a hardware channel attention signal (CA). The memory interface unit fetches and executes these commands. When the display process is activated, the 82730 repeatedly fetches display data and embedded datastream commands from memory utilizing its built-in DMA capability, executes any datastream commands as encountered on the fly, and loads the row buffers with the display data. After executing these commands, the 82730 clears a busy flag in memory, to inform the host CPU that it is ready for the next command.

The memory interface unit is divided into two sections—the bus unit and the microcontroller unit. The bus interface unit provides the electrical interface to the system bus and the timing signals required for the microcontroller unit operations, making these operations transparent to the microcontroller unit. The 82730 can be programmed during initialization to provide 8 or 16 bit data, and 16 or 32 bit addressing.

The microcontroller unit contains the microinstruction store and the associated circuitry required for the execution of all channel and datastream commands. It uses the bus interface unit in carrying out its memory access tasks such as loading the row buffers with display data.

The interaction between the microcontroller unit and the display generator takes place through shared internal storage. The microcon-

"The device provides the ability to independently maximize the performance of the CPU."

troller unit fetches data from memory and writes it in the internal storage, while the display generator reads from the internal storage and carries out the display operation. The microcontroller unit and display generator operate asynchronously with respect to each other. Synchronization is accomplished through communication via internal flags and display timing signals generated by the display generator. The internal shared storage consists of row buffers which store the display data and internal registers which store display parameters. There are two row buffers each capable of storing up to 200 characters. The data in one row buffer is used by the display generator to display one complete character row on the screen, while the microcontoller unit is loading the second row buffer with display data fetched from memory. At the end of the row being displayed, the buffers are swapped and the microcontroller unit and display generator resume their tasks.

The display characteristics registers contain all the information used to control every aspect of display characteristics from screen size to blink rates. A major portion of this register set is the three content addressable memory (CAM) arrays that allow flexible timing control for row and screen characteristics. The user has the power to set the parameters for the entire screen by invoking a single high-level command.

By separating the video interface clocks from the bus interface clock, the 82730 provides the designer with the ability to independently maximize the performance of the CPU and video sections of the system.

The video interface consists of two independent clocks: the Reference Clock (RCLK) and the Character Clock (CCLK). While the RCLK controls the raster timing and defines the screen layout, the CCLK independently shifts character and attribute information out of the 82730, which allows proportional spacing to be achieved.

Combining text and graphics

A major requirement in the design of engineering workstations is the simultaneous display of both text and graphics. In terms of graphics requirements, the designer of such systems needs a drawing processor for fast geometric primitives, a math processor for fast transformations and a general purpose processor for access to the graphics database.

For text, string processing is needed for manipulation of text primitives and database processing is needed for access to the document files. The solution to this problem can be solved by using both the 720 graphics coprocessor and the 730 text coprocessor (Figure 2).

Both coprocessors work with Intel's new 82586 communications coprocessor. This works in conjunction with a CPU and the appropriate software to provide local area network (LAN) control capabilities. Message data to be placed on the network by a microprocessor-based work station is stored in shared memory in transmit blocks. Pointers (starting address information) to these blocks are stored along with processing instructions in other shared memory blocks. Status information and overall directives are stored in system control blocks which serve as the mailbox between the CPU and the 82586.

When alerted by a channel attention signal, the 82586 will perform a host of tasks involved in accessing data to be transmitted from its location in memory, framing the message packets containing the data and seeing to the transmission on the network medium. In addition, the 82586 receives and buffers incoming data which it then stores in shared memory for the CPU to collect. It is the CPU's job to allocate the blocks of memory for the LAN coprocessor to store the received packet data.

82730
TEXT COPROCESSOR

- **High Quality Display for Text Applications**
- **Provides Proportional Spacing, Alphamosaic Graphics, Simultaneous Superscript/Subscript and Soft Font Support**
- **High Performance Text Manipulation provided by 4 Mbytes/sec DMA and on-chip Dual Row Buffers (up to 200 characters each)**
- **Programmable Bus Interface Handles 8 or 16 Bit Data and 16 or 32 Bit Addressing; iAPX 86/88/186/188 Compatible**
- **On-Chip Processing Unit Simplifies Software Design by Executing High Level Commands and Supporting Linked List Data Structures**
- **Extremely Flexible; Programmable Features Include Screen and Row Formats, Two Cursors, Character and Field Attributes and Smooth Scrolling**
- **Simultaneous Display of Independent Data Bases Through Programmable Virtual Screen Mode**
- **High Resolution Display; Up to 200 Characters per Row and 2048 Scan Lines per Frame**
- **Separate Bus and Video Clocks Allow Optimization of Overall System Performance**
- **Provides a Complete LSI Solution for Display Control when Used in Conjunction with the 82731 Video Interface Controller**

The 82730 Text Coprocessor is a high performance VLSI solution for raster scan text oriented displays. The 82730 works as a coprocessor and has processing capabilities specifically tailored to execute data manipulation and display tasks. It provides the designer the ability to functionally partition his system thereby offloading the system CPU and achieving maximum performance through concurrent processing. The 82730 supports the generation of high quality text displays through features like proportional spacing, simultaneous superscript/subscript, dynamically reloadable fonts and user programmable field and character attributes. An intelligent system interface and efficient software capabilities makes 82730 based systems easy to design. In addition, when coupled with the 82720 Graphics Display Controller, the 82730 provides flexible mixing of high quality text and graphics simultaneously on the same display.

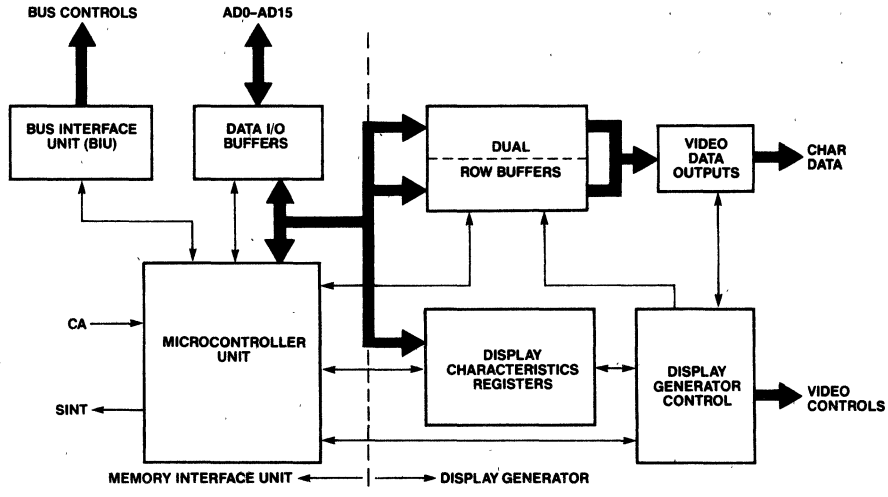


Figure 1. 82730 Block Diagram

NOVEMBER 1983
ORDER NUMBER: 2100924-002

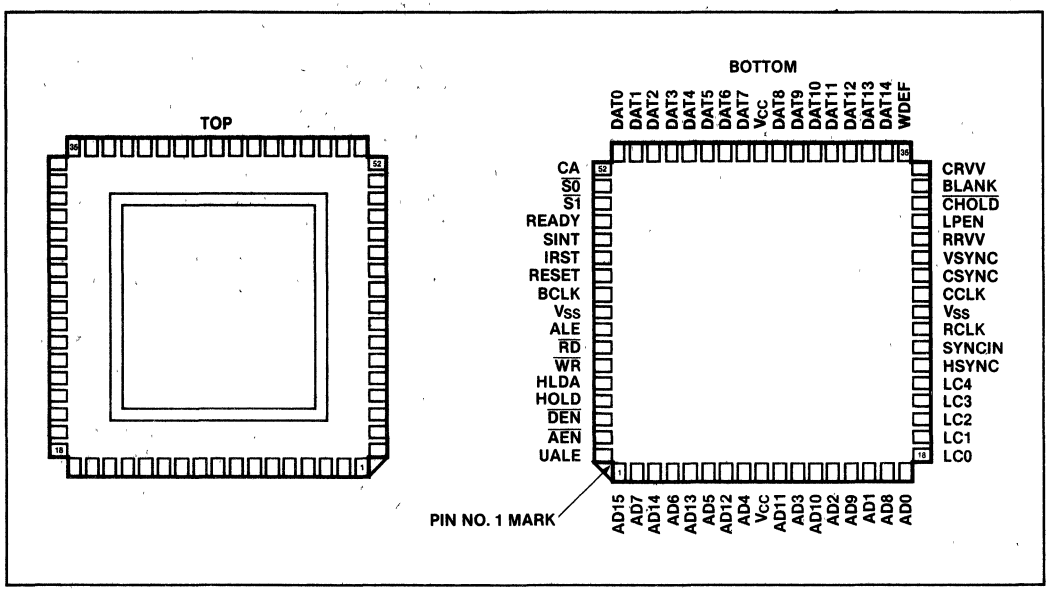


Figure 2. 82730 Pinout Diagram

Table 1. 82730 Pin Description

The 82730 is packaged in a 68 pin JEDEC Type A ceramic package.

Symbol	Pin Number	Type	Name and Function
AD15-AD0	1-8 10-17	I/O	Address Data Bus; these lines output the time multiplexed address (TU, T1 states) and data (T2, T3, T4 and TW) bus. The bus is active HIGH and floats to 3-state OFF when the 82730 is not driving the bus (i.e. HOLD is not active or when HOLD is active but not acknowledged, or when RESET is active).
BCLK	59	I	Bus clock; provides the basic timing for the Memory Interface Unit.
$\overline{RD}$	62	O	Read strobe; indicates that the 82730 is performing a memory read cycle on the bus. $\overline{RD}$ is active low for T2, T3 and TW of any read cycle and is guaranteed to remain high in T2 until the address is removed from the bus. $\overline{RD}$ is active low and floats to 3-state OFF when 82730 is not driving the bus. $\overline{RD}$ will return high before entering the float state and will not glitch low when entering or leaving float.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
$\overline{WR}$	63	O	Write strobe; indicates that the data on the bus is to be written in a memory device. $\overline{WR}$ is active for T2, T3 and TW of any write cycle. It is active LOW and floats when 82730 is not driving the bus. $\overline{WR}$ will return high before entering the float state and will not glitch low when entering or leaving float.
ALE	61	O	Lower Address Latch Enable; provided by the 82730 to latch the address into an external address latch such as 8282/8283 (active HIGH). Addresses are guaranteed to be valid on the trailing edge of ALE.
UALE	68	O	Upper Address Latch Enable; it is similar to ALE except that it occurs in upper address output cycle (TU).
$\overline{AEN}$	67	O	Address Enable; $\overline{AEN}$ is active LOW during the entire period when 82730 is driving the bus. It can be used to unfloat the outputs of the Upper and Lower Address latches.
$\overline{DEN}$	66	O	Data enable; provided as a data bus transceiver output enable for transceivers like the 8286/8287. $\overline{DEN}$ is active LOW during each bus cycle and floats when 82730 is not driving the bus. $\overline{DEN}$ will not glitch when entering or leaving the float state.
$\overline{S0}$, $\overline{S1}$	53, 54	O	Status pins; encoded to provide bus-transaction information: $\overline{S1}$ / $\overline{S0}$ / Bus Cycle Initiated 0 / 0 / - - - (Reserved) 0 / 1 / Memory Read 1 / 0 / Memory Write 1 / 1 / Passive (No bus cycle) These pins are directly compatible with iAPX 86, 186 status outputs $\overline{S1}$ and $\overline{S0}$. The status pins are floated when 82730 is not driving the bus. They will not glitch when entering or leaving the 3-state condition.
READY	55	I	READY; signal to inform the 82730 that the data transfer can be completed. Immediately after RESET, READY is asynchronous (internally synchronized) but can be programmed during initialization to bus synchronous.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
HOLD	65	O	HOLD; indicates that the 82730 wants bus access. HOLD stays active HIGH during the entire period when 82730 is driving the bus.
HLDA	64	I	Hold Acknowledge; indicates to 82730 that it is granted the bus access as requested. HLDA may be asynchronous to 82730 clock. If HLDA goes inactive (LOW) in the middle of an 82730 bus cycle, the 82730 will complete the current bus cycle first, then it will drop HOLD and float address and bus control outputs.
CA	52	I	Channel Attention; used to notify 82730 that a command in the command block is waiting to be processed. CA is latched on its falling edge.
SINT	56	O	Status Interrupt; used to inform the processor that an unmasked interrupt has been generated in the 82730 status register.
IRST	57	I	Interrupt Reset; SINT is cleared by activating the IRST pin.
RESET	58	I	Reset; causes 82730 to immediately terminate its present activity and enter a dormant state. The signal must be active HIGH for at least 4 BCLK cycles and is internally synchronized to the bus clock.
CCLK	27	I	Character clock; input used to clock row buffer data, attribute, cursor and line count out of 82730. When more than one 82730 is connected in cluster mode, CCLK is used to synchronize output from both master and slave chips. A character data word will be output at every rising edge of CCLK.
RCLK	25	I	Reference clock; input used to generate timings for the screen layout and to define screen columns for data formatting. All raster output signals are specified relative to the rising edge of RCLK.
DAT0-DAT14	36-42 44-51	O	Video data bus output; the least significant 15 bits of the character data words are passed through the 82730 row buffer and made available on the pins DAT0-DAT14. The user has the flexibility to partition the data word into character and attribute bits per his requirements. The bits that are assigned for internally generated attributes may also be available at pin DAT0-DAT14. New character data will be shifted to these output pins at every rising edge of the CCLK. Together with LC0-LC4, they may be used to address the character generator or as attribute controls.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
WDEF	35	O	Width Defeat; is used to indicate when the character is allowed to be a variable width or must be of fixed width. WDEF is LOW if the character being output is normal, but is HIGH if it is a superscript/subscript character or visible attribute (TAB or GPA). Optionally, WDEF can be held high by user command.
LC0-LC4	18-22	O	Line count outputs; used to address the character generator for the line positions in a row. The line number output is a function of the display mode and character attributes programmed by the user.
CSYNC	28	O	CCLK synchronization output; used to synchronize external character clock generator to reference clock timing. This output is active (high) outside the display field.
$\overline{\text{CHOLD}}$	32	O	CCLK Inhibit output; used by external logic to inhibit CCLK generation. This output is active (low) during the tab and end-of-row function.
SYNCIN	24	I	Synchronization input; used to synchronize the vertical timing counters to an externally generated VSYNC signal. Used by slave mode 82730 to synchronize to a master mode 82730 and by the master 82730 to lock the frame to an external source such as the power line frequency.
HSYNC	23	O (MASTER) I (SLAVE)	Horizontal Sync; in master mode, it is used to generate the CRT monitor's horizontal sync signal. It is active HIGH during the programmed horizontal sync interval. In interlace slave mode it is used in conjunction with SYNCIN to indicate the start of the even field for timing counter reset. At RESET, pin is set as an output in the LOW state.
VSYNC	29	O	Vertical Sync; active HIGH during the programmed vertical sync interval and used to generate the CRT monitor's vertical sync signal.
BLANK	33	O	Blanking output; used to suppress the video signal to the CRT. BLANK is clocked by CCLK.
CRVV	34	O	Character Reverse Video (CCLK output); used to externally invert video data output. CRVV is clocked by CCLK.
RRVV	30	O	Reference Reverse Video (RCLK output); to externally invert video in the field and border area if so programmed by user. It is LOW outside the border area, RRVV is clocked by RCLK.

Table 1. 82730 Pin Description (Continued)

Symbol	Pin Number	Type	Name and Function
LPEN	31	I	Light Pen Input; used to latch the position of a light pen. At the rising edge of this input, the column position and the row position of the 82730 will be loaded into the LPENROW and LPENCOL locations in the Command block.
V_{CC}	9, 43		Power; +5 volts nominal potential.
V_{SS}	26, 60		Power; ground potential.

FUNCTIONAL DESCRIPTION

Figure 1 shows a basic block diagram of the 82730 Text Coprocessor. The chip is divided into two main sections, the Memory Interface Unit and the Display Generator. The Memory Interface Unit controls fetching of the data and commands and handles interrupts and status. The Display Generator takes the data fetched by the Memory Interface Unit and presents it to the Video Interface logic which in turn drives the CRT monitor.

Memory Interface Unit

The Memory Interface Unit is divided into two sections: the Bus Interface Unit and the Microcontroller Unit. The Bus Interface Unit does the actual interfacing to the memory bus. It fetches or writes data under the control of the Microcontroller Unit. The Microcontroller Unit is a microprogrammed controller which is designed to efficiently fetch data from memory (up to 4 Mbytes/sec), and decode and execute various control and data handling commands. The Bus Interface Unit may be configured for 8 or 16 bit bus operation. With 8 bit bus selection, the user may specify either 8 or 16 bit character data. It also handles address manipulation automatically after being loaded from the Microcontroller Unit.

Display Generator

The Display Generator takes the data fetched from memory plus the modes programmed into it at initialization and produces all the video timing and the data transfers to support the CRT monitor at the character level. The 82730 works with an external character generator and the 82731 Video Interface Controller. The data is passed to the Display Generator from the Memory Interface Unit through the dual row buffers (similar in operation to the one in the 8275 CRT controller). The row buffers allow the user to use cheaper and slower main memory for display needs, provide on-chip attribute and display function generation, and avoid the conflict of access to the display memory (that would otherwise take place) by using an ordinary DMA access mechanism.

SYSTEM BUS INTERFACE

The Memory Interface Unit provides communication with system processor as well as memory interactions. Communication between the processor and the 82730 is performed via messages placed in communication blocks in shared memory. The processor can issue commands by preparing message blocks and directing the 82730's attention to them by asserting a hardware channel attention. The 82730 can cause interrupts on certain conditions, if enabled by the processor by activating its System Interrupt output, with status and error reporting taking place through the communication block in memory.

BUS INTERFACE UNIT:

The 82730 Bus Interface Unit provides an 8086 compatible bus interface which consists of:

- a 16/32 bit multiplexed Address/Data Bus: AD_0 - AD_{15}
- A complete set of local bus control signals compatible with 8086 min mode: $\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and READY
- Two status signals $\overline{S0}$ and $\overline{S1}$, compatible with 8086 max mode so that a bus controller (8288) can be shared for Multibus® access.
- Local bus arbitration through HOLD/ HLDA
- Two upper Address Latch controls: UALE and $\overline{AEN}$

The BUS INTERFACE UNIT (BIU) utilizes the same Bus structure as the 80186 or basically the same bus structure as the 8086 in both Min. and Max. mode, (with the exception of RQ/GT) and it performs a bus cycle only on demand (e.g., to fetch a command from the command block, or fetch a character from display data memory). The same set of T-states (T1, T2, T3, T4 and TW) of 8086 are used to handle the time multiplexed address/data bus. However, adaptations are made to handle 32 bit addresses as explained in the following sections where specific details of the BIU operation are described. Those details not mentioned can be assumed to be the same as those of the 80186.

ADDRESS BUS

The 82730 can be programmed during initialization to operate on either 16 bit or 32 bit (including any length between 17 and 32) physical addresses. Note that the 82730 does not use memory segmentation. The programmer must calculate physical addresses from segment and offset values to manipulate data structures.

To support 32 bit physical addresses with a 16 bit physical bus, multiplexing is again used. An upper address output cycle, TU, is inserted between T4 and T1 to output the upper 16 bits of address. The upper address latch enable, UALE, is used to latch the upper addresses during TU. Figure 3 shows the configuration of a 32 bit address bus.

TU occurs only when the 32 bit mode is specified and the upper address register of BIU is reloaded by MCU. This may result from:

i) Initialization

ii) Manipulation of display data or command pointers, for example, when a new string pointer is loaded during the execution of the END OF STRING command.

iii) DMA address incrementing across a 64K byte segment boundary.

iv) Regaining the bus after losing it to a higher priority master.

Timing of UALE is identical to that of ALE. $\overline{AEN}$ is equivalent to the active period of 82730 driving the bus.

If 16 bit address mode is programmed, TU will never occur in any bus cycle since the MIU treats all display pointers as 16 bit quantities and loading of internal upper address register is bypassed during address calculation. UALE always stays inactive, but $\overline{AEN}$ still goes active to indicate the 82730 has control of the bus.

DATA BUS

The 82730 is capable of operating on either an 8 bit or a 16 bit Data bus, as programmed during initialization on the SYSBUS byte.

When an 8 bit data bus is specified, the address present on AD15 to AD8 Address/Data lines is maintained for the complete bus cycle. Therefore, compatibility with 80188, 8088, 8089 and 8085 multiplexed address peripherals is maintained. Since the internal processing of the 82730 generally operates on 16 bit data quantities, two Bus fetch cycles are performed for each 16 bit data item. The first cycle fetches the low order byte, the second cycle the high order byte. These 2 fetch cycles are always executed back to back. If HLDA drops during the first cycle, the 82730 will not respond until the second cycle is completed. An 8 bit data mode can be selected in an 8 bit bus system that requires only 8 bit character data be fetched.

In 16 bit bus system, the 82730 requires all 16 bit quantities to start on even address boundary. Word transfer to or from odd boundary is not allowed since this type of transfer not only doubles the use of bus bandwidth but also can be easily avoided in application software. All that is required is to make sure all address pointers be an even number (A0=0).

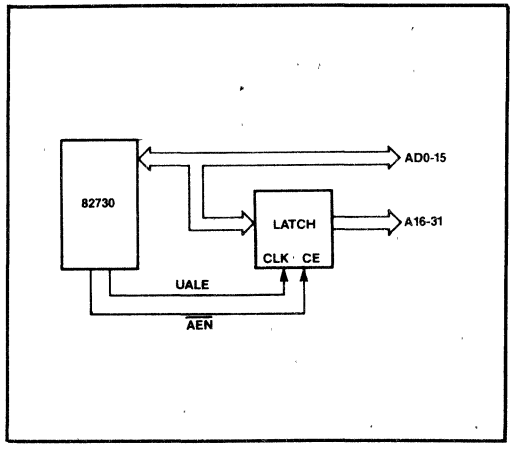


Figure 3. Address Extension up to 32 Bits

BUS CONTROLS

The 82730 BIU provides both the 8086 MIN. Mode (Local Bus Control) and MAX. mode bus control signals simultaneously in any bus cycle. By providing a complete set of Local Bus control signals, the component count of the Local processing module is minimized.

Because only two types of Bus operations, Memory Read and Memory Write, are executed in the 82730 BIU, the 8086's $\overline{S2}$ status signal is omitted from the Max. mode controls. S2 could be set to "1" during any 82730 Bus cycle. $\overline{AEN}$ can be used to produce $\overline{S2}$ since it stays active whenever 82730 is driving the bus. The status signals become valid at the middle of the cycle before T1 which could be either T4 or TU.

$\overline{BHE}$ is not provided on the 82730 because, the 82730 only writes words to even address boundaries and bytes to the upper byte position. For these writes $\overline{BHE}$ is always high. A pullup resistor or a three-state buffer controlled by $\overline{AEN}$ can provide this signal.

DT/R is also not provided on the 82730 because its function can be replaced with $\overline{S1}$, latched by ALE.

After RESET is applied, READY is set to be an asynchronous input. An on-board synchronization circuit provides reliable operation for any type of system. During initialization, READY may be programmed to be bus synchronous. For those systems that can meet the set-up time specifications, this mode provides more efficient bus utilization.

LOCAL BUS ARBITRATION

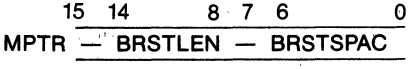
The 82730 BIU is designed to function as a bus master in a multimaster Local bus environment using the HOLD/HLDA protocol for Bus arbitration.

In the Self Contained Arbitration scheme, one processor and one 82730 share access to the local bus. The 82730 raises its HOLD request whenever it needs bus access. After HLDA is granted from the processor, the 82730 will not start driving the bus until 2 clock cycles later. This latency allows sufficient time for the 8086 or 80186 processor to get off the bus. When 82730 completes its bus accesses, it will first float its output drivers before dropping the hold request.

In a Local bus configuration with three or more bus masters, a higher priority DMA Peripheral device can preempt the HLDA from a 82730 which is the current bus master. The 82730 will complete its current bus cycle, then float its output drivers and drop the HOLD request. However, the 82730 may raise the HOLD request again 2 clock cycles later if it still needs the bus to complete the interrupted burst DMA activities.

DMA BURST AND SPACE

Some system configurations using the 82730 would be adversely affected by the long burst data transfers which the Memory Interface Unit (MIU) may occasionally desire. Since the 82730 will normally be configured as one of the higher priority bus masters, burst lengths must be limited for these systems. For this reason, the length of a burst transfer and the number of memory cycles between burst transfers are both programmable via the mode registers:



BRSTLEN- Burst Length.Determines the number of contiguous word-fetch cycles which may be requested. Programmable from 1 to 127. Note that in an 8 bit bus, 16 bit data system, the burst counter only increments once for the 2 bus cycles required to complete a word fetch. (Note: burst length = 0 is not defined and should not be programmed with a non-zero burst space)

BRSTSPAC - Burst Space. Determines a minimum number of bus clocks to occur between burst accesses. Programmable from 0-511 in increments of four. Zero space selects an infinite burst length.

A DMA burst could be terminated before the programmed burst length is reached in the following circumstances:

i) The MIU does not need any more bus accesses, for example, when the row buffer is filled.

ii) A datastream command is encountered and the MIU must execute the command first before it resumes data accessing.

iii) The bus is taken away by a higher priority device in multi-master bus configuration.

In these cases, the burst counter is cleared. The BIU must complete a full burst before it waits through the SPACE cycles. DMA Burst/Space will be set to zero space until the completion of the first MODESET command.

INITIALIZATION OF BIU

Upon activation of the RESET input, the 82730 BIU will stop all operations in progress and deactivate all outputs. It will stay in this quiescent state until memory access is requested by the MCU after MCU receives its first channel attention after RESET. The following table shows the state of all MIU outputs during and after reset.

Table 2. 82730 Bus During and After Reset

Signals	Condition
AD15-0	Three-state
$\overline{RD}$, $\overline{WR}$, $\overline{DEN}$	Driven to '1' then three-state
$\overline{S0}$, $\overline{S1}$	Driven to '1' then three-state
ALE, UALE	Low
$\overline{AEN}$	High
HOLD	Low
SINT	Low

82730 COMPATIBILITY ISSUES

82730 Bus Clock Compatibility

The 82730 uses the 50% duty cycle output of the iAPX-186 at 8 MHz or that generated by a clock generator such as the 82285. A different duty cycle clock may be used at lower frequencies, so the 82730 is also useable with the iAPX-86, 88 family.

82730 Bus Interface Compatibility

The bus interface compatibility between the 82730 and another bus master has four main issues: data bus width, addressability, control bus structure and local bus mastership arbitration.

Data Bus

Data Bus width compatibility with all 85/86 family processors (8085, 8086, 8088, 80188, 80186, and 80286) is being supported by the 8/16 data bit programmability already discussed. This allows interfacing to the above processors either directly or through a Multibus-like interface.

Address Bus

The 82730 uses real 32-bit addresses. The user's software must calculate real addresses; this general addressing scheme allows the 82730 to be used with any microprocessor.

Control Bus

The 82730 implements both 8086 minimum and maximum mode bus control structures. This was done to maximize compatibility with the 80186 which has the same structure. This allows the 82730 to be run locally (minimum mode) with a 8085, 8086, 8088, 80188, or 80186. The 80186/188 and 82730 can run together at 8MHz because of clock duty cycle considerations. The 82730 can only communicate to an 80286 via a system bus (such as MULTIBUS), bus interface, or dual-port RAM.

INITIALIZATION SEQUENCE

The first CA (Channel Attention) after Reset causes an Initialization Sequence to be executed. The system processor must set up the appropriate initialization information in memory and set the BUSY flag in the Intermediate Block to a non-zero value prior to issuing this CA.

Initially, 32-bit addressing and 8-bit data bus width are assumed until the corresponding information is fetched during the initialization. First the SYSBUS byte is fetched from memory location FFFF FFF6. (When the address bus is less than 32 bits wide, the higher order bits are unused.) The format for SYSBUS byte is shown in Figure 4 and is the same as that used for 8089. The data bus width is specified by the least significant bit w, with w=0 indicating an 8-bit bus and w=1 signifying a 16-bit bus.

A 32-bit real address pointer is then fetched from memory locations FFFF FFFC through FFFF FFFF, with lower bytes of the pointer residing in lower addresses. This pointer is used as an Intermediate Block Pointer (IBP).

The Intermediate Block Pointer (IBP) is incremented by two and is used to locate the Command Block Pointer (CBP). Four bytes are fetched irrespective of whether a 16-bit or 32-bit addressing option is used. The System Configuration byte (SCB) is then fetched from location (IBP+6).

The least significant bit, (U of the SCB) specifies 16 or 32-bit addressing option, with U=0 indicating 16 bit addressing and U=1 specifying 32-bit addressing. The SCB also contains information about cluster operation. Since up to four 82730's can be connected in a cluster with their respective data interleaved in memory, cluster information is needed for the data access task. The SCB specifies Cluster Number (CL NO), which is the number of 82730's connected in a cluster and Cluster Position (CL POS) which is the position

of this particular 82730 within the cluster. CL NO = 0, 1, 2 or 3 indicates a cluster containing 1, 2, 3 or 4 82730's respectively. Similarly, CL POS = 0, 1, 2 or 3 indicates 1st, 2nd, 3rd or 4th position respectively. Each 82730 adds an offset equal to 2 * CLPOS to the SPTR fetched from memory and increments the pointer by 2 * (CL NO + 1). The programming of CL NO and CL POS is independent. No checking is done for CL POS greater than CL NO on the 82730. Note that at least one 82730, in a cluster (even if it is a cluster of one), must be assigned as cluster position zero (CL POS = 0) for Virtual Display mode to work properly.

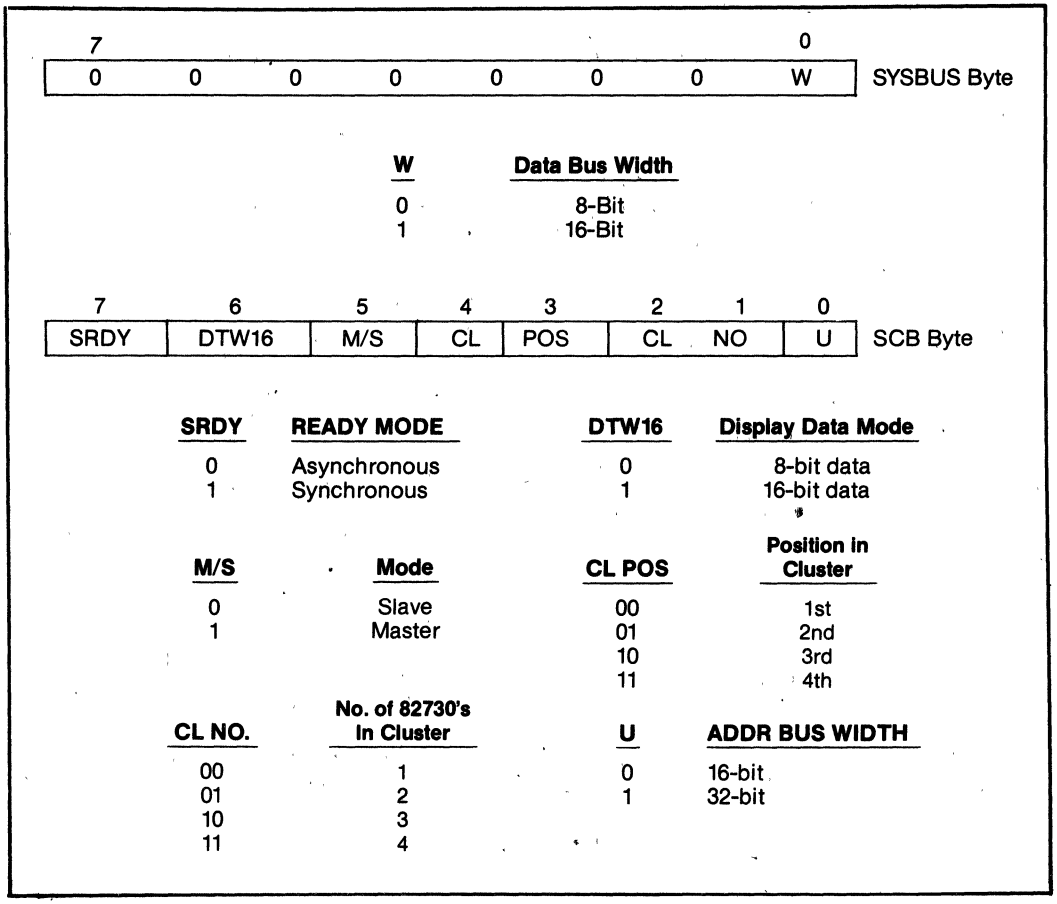


Figure 4. SYSBUS and SCB Encoding

The SCB also contains an $\overline{M}/S$ bit which specifies a master or slave mode. The $M/\overline{S}$ bit is stored internally for use by the Display Generator (DG). $M/\overline{S}$ = 1 indicates a master mode and $M/\overline{S}$ = 0 specifies a slave mode. The format for the System Configuration Byte (SCB) is shown in Figure 4. Following these actions, the BUSY flag in the Intermediate Block at address IBP is cleared and a normal Channel Attention sequence is then executed.

The last two bits in the SCB are DTW16 and SRDY. DTW16 specifies whether the display data in 8 bit bus mode (W=0) is 8 or 16 bit. If a 16 bit system is specified (W=1) then DTW16 is ignored and forced internally to a "one". SRDY specifies whether the clock synchronization circuit for the READY pin is internal (SRDY=0) or external (SRDY=1).

The Initialization Control Blocks in memory are illustrated in Fig. 5a. How these fit into the control structure of the 82730 is shown in Figure 5b.

Channel Attention Sequence

When the processor activates CA, an internal latch in 82730 is set on the falling edge of CA input and this latch is sampled by the MCU. The first CA activation after reset causes the 82730 to execute an initialization sequence. Any subsequent activation will cause the MCU to start processing the command block by fetching a channel command.

If a display is in progress, the MCU will sample CA at each end of frame, otherwise it will sample CA every cycle until it is found active. When CA is found active, the MCU will fetch the command byte from "COMMAND" location in the command block, execute the command and clear the BUSY flag upon completion. The internal CA latch is also cleared by the MCU. An invalid command code has the effect of NOP and the BUSY flag is cleared. It will also cause the Reserved Channel Command (RCC) status bit to be set.

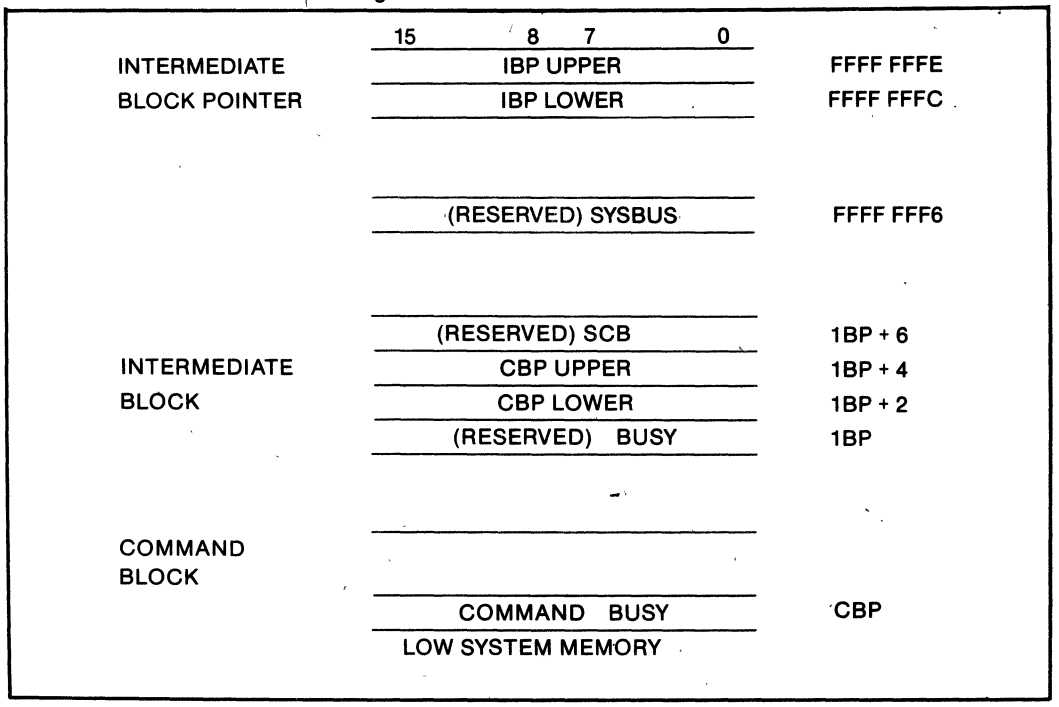


Figure 5a. Initialization Control Blocks

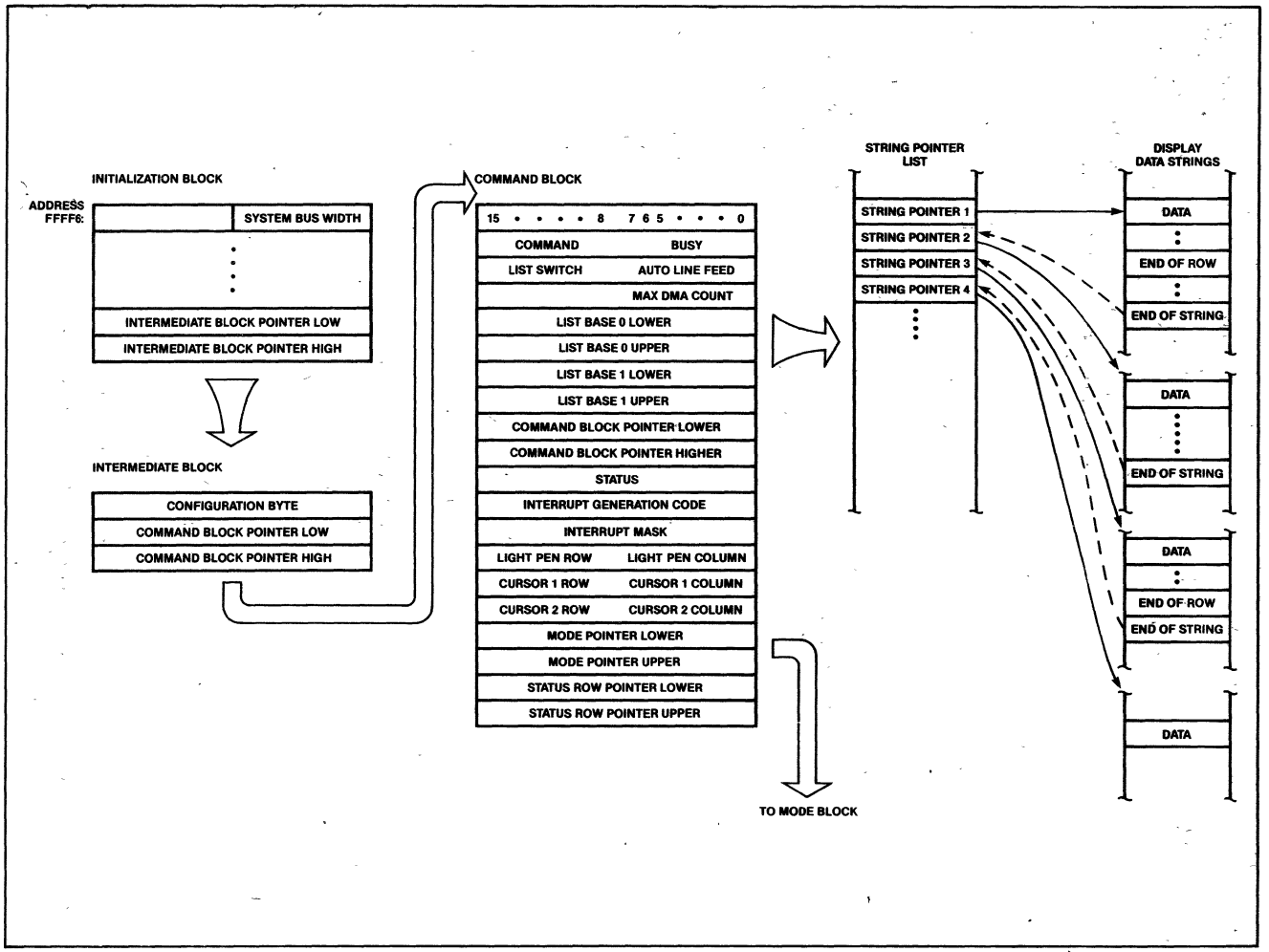


Figure 5b. Control Structure of the 82730

82730 CHANNEL COMMANDS

Table 3. Channel Commands

	COMMAND	OP CODE		
1	START DISPLAY	0000	0001	01 H
2	START VIRTUAL DISPLAY	0000	0010	02 H
3	STOP DISPLAY	0000	0011	03 H
4	MODE SET	0000	0100	04 H
5	LOAD CBP	0000	0101	05 H
6	LOAD INTMASK	0000	0110	06 H
7	LPEN ENABLE	0000	0111	07 H
8	READ STATUS	0000	1000	08 H
9	LD CUR POS	0000	1001	09 H
10	SELF TEST	0000	1010	0A H
11	TEST ROW BUFFER	0000	1011	0B H
12	NOP	0000	0000	00 H
13	(RESERVED)	From: 0000 To: 1111	1100 1111	0C H FF H

The system processor issues channel commands to 82730 via the Command Block. The processor first checks if the BUSY flag in the command block has been cleared. It should wait for the BUSY flag to be cleared before proceeding with the issuing of a command. When the BUSY flag is cleared, the processor places a command byte in the "COMMAND" location in command block, sets the BUSY flag to a non-zero value and asserts Channel Attention (CA), by activating the CA input to 82730. A Channel Attention should not be issued, if the BUSY flag has not been cleared.

START DISPLAY

0000 0001	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Position values are fetched from the Command Block and stored internally after this command is received. The BUSY flag is cleared and the normal display process is activated.

The MCU fetches strings of data from the memory, using the parameters LISTSWITCH, LBASE0 and LBASE1. The data fetched is interpreted as data-stream commands or character data to be displayed by the Display Generator. The MCU loads the data into one of the two Row Buffers in the CRT controller, while the Display Generator displays the data from the other buffer, the buffers being swapped at the end of the row. Any data-stream commands encountered during data fetch are immediately executed.

The display process is continued until it is deactivated by a STOP DISPLAY command or a Reset. Other channel commands can be issued while a display is in progress and they will be executed when CA is found active at one of the periodic samplings at each end of frame.

The DIP (Display in Progress) status bit is set and the VDIP (Virtual Display in Progress) is cleared upon receiving a START DISPLAY command. Both bits are reset upon receiving a STOP DISPLAY command or a Reset.

It is necessary to load in proper mode information through a MODESET command before activating the display. Following Reset, START DISPLAY command will not be executed, i.e., will result in a NOP until a MODESET command has been issued.

START VIRTUAL DISPLAY

0000	0010	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon receiving this command. The BUSY flag is cleared and the Virtual Screen display process is activated.

The operation of Virtual Screen display process is similar to that of a regular display process, except for following a different data access mechanism. The parameters LISTSWITCH, LBASE0 and LBASE1 in the command block represent ACCESS SWITCH, ACCESS BASE0 and ACCESS BASE1 respectively, in virtual screen display.

The VDIP (Virtual Display in Progress) status bit is set and the DIP status bit is cleared upon receiving a START VIRTUAL DISP command: Both DIP and VDIP are reset upon receiving a STOP DISPLAY command or a Reset.

START VIRTUAL DISPLAY command will not activate a display and results in a NOP until a MODESET command is issued after a Reset.

STOP DISPLAY

0000	0011	CMD Byte

The display process is deactivated upon receiving this command. The DIP and VDIP status bit are reset and the BUSY flag is cleared.

This command blanks the display. HSYNC and VSYNC are **not** affected.

MODESET

0000	0100	CMD Byte

The Mode Pointer contained in command block location (CBP + 30) is used to access the Mode Block and the modes are fetched sequentially and loaded into the corresponding internal registers in 82730. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally upon completion and the BUSY flag is cleared.

The organization of mode words in the mode block and the parameters supplied by them are shown below (See Figure 10). Some of these parameters which are critical to the operation of a text coprocessor are required to remain unchanged over most of normal operation. No provision is made to prevent MODESET from changing these parameters and it is left to the designer to insure that they are not changed.

The modes provide horizontal and vertical mode display parameters, interlace information, DMA burst and spacing specifications, cursor characteristics as well as attribute enables and bit-selects. Typically, this would be the first command issued after initialization. The Mode Block provides all the parameters needed for a complete initialization of the 82730 for display. Thus a single Modeset command can fully initialize the chip. Note that until the first Modeset command is sent, certain functions such as VSYNC and HSYNC are not enabled.

It is necessary to set up proper mode information, before activating a display. Therefore, a display activating commands should not be issued unless proper mode information has been loaded through a MODESET command. START DISPLAY and START VIRTUAL DISPLAY commands will result in a NOP if a MODESET command has not been issued since the most recent Reset.

LOAD CBP

0000	0101	CMD Byte

The address pointer "NEW CBP" contained in the command block is fetched and stored in the CBP register in the text coprocessor, replacing the old CBP. This effectively moves the command block in the memory. The Command byte from the new Command Block is fetched and the specified channel command is executed. The BUSY flag in the new Command Block is cleared upon completion.

LOAD INTMASK

0000	0110	CMD Byte

The interrupt mask contained in location "INT MASK" in the command block is fetched and stored internally in the CRT controller. When a particular mask bit is set, the interrupt is disabled for a status bit in the corresponding bit position. An interrupt is generated by the text coprocessor by activating the SINT pin, if a status bit is 1 and the corresponding bit in the interrupt mask is 0. The BUSY flag is cleared upon completion.

Interrupts can be enabled for the following status bits.

7	6	5	4	3	2	1	0	BIT
—	RDC	RCC	FDE	EOF	DBOR	LPU	DUR	STATUS WORD

RDC: Reserved Datastream Command Encountered
RCC: Reserved Channel Command Executed
FDE: Frame Data Error (Fetching characters past physical End of Frame)
EOF: End of "n" frames (Logical end of nth frame)
DBOR: Data Buffer Overrun (Row Buffer filled completely without encountering END OF ROW command)
LPU: Light Pen Update
DUR: Data Underrun (Buffer swap initiated before finishing Row Buf loading)

READ STATUS

0000	1000	CMD Byte

The internal status register is written to"STATUS" location in the command block. The status register is then cleared, however DIP and VDIP status bits are not cleared. LISTSWITCH, Auto Linefeed, Max DMA Count and Cursor Positions are fetched from the Command Block and stored internally. The BUSY flag is then cleared.

LD CUR POS

0000	1001	CMD Byte

The display row and column positions of cursors 1 & 2 as set in locations "CUR1 ROW," CUR1 COL," "CUR2 ROW" and "CUR2 COL" in the command block are loaded into internal registers in the CRT controller. Also LISTSWITCH Auto Linefeed and Max DMA Count are loaded from the Command Block and the BUSY flag is cleared. This command is used to change the cursors only. Note that the cursor positions are also updated with the execution of other channel commands.

The cursor characteristics for display are defined by the mode. During the display process, a cursor will be displayed accordingly at the position specified above.

STATUS WORD

15-9	8	7	6	5	4	3	2	1	0
—	VDIP	DIP	RDC	RCC	FDE	EOF	DBOR	LPU	DUR

LPEN ENABLE

0000	0111	CMD Byte

The Light Pen detection process is enabled to search for a rising edge on the LPEN pin. The BUSY flag is then cleared.

If the display process is active and a rising edge is detected on the LPEN input, the corresponding row and column position on the screen is stored internally. At the next end of frame, the LPEN position is written to locations "LPENROW" and "LPENCOL" in the command block and the LPU (Light Pen Update) status bit is set.

If the display process is not active, this command has no immediate effect. However, the LPEN detection process remains enabled and will take effect if a display is activated subsequently.

NOP

0000	0000	CMD Byte

LISTSWITCH, Auto Linefeed, Max DMA Count, and Cursor Positions are fetched from the command block and stored internally as in all other channel commands. The Busy flag is then cleared.

82730 DATASTREAM COMMANDS

Datastream Commands

Datastream Commands are commands embedded in the data fetched from memory by the data access task. These commands are differentiated from character data by the command bit. The most significant bit (MSB) of each data word is designated as the command bit. If the command bit is "1", the lower 15 bits of the data word are interpreted as a datastream command, while if the command bit is "0" the lower 15 bits (or 7 bits if DTW16=0) are interpreted as character data.

Datastream Command Operation

During the data access task, the Micro Controller Unit (MCU) examines the command bit of each data word fetched. If the command bit is 1, it executes the datastream command specified in the data word. Otherwise, it stores the lower 15 bits of the data word in the Row Buffer as character data. This process is repeated for each data word fetched.

Datastream commands can be used for changing Row Characteristics on a row by row basis, for carrying out editing functions and for formatting data into rows and frames. These commands are executed by the MCU immediately after they are encountered. As a convenience for the user, the set of all possible command codes starting with "11" in the two most significant bits has been designated as NOP commands. The user can use these command codes for any desired purpose. All other command codes which are not presently defined, are reserved for future expansion and should not be used by the user. The currently undefined codes cause the RDC (Reserved Datastream Command) status bit to be set and also generate an interrupt, if enabled. Reserved command codes should not be used.

Datastream Command List

Table 4. 82730 Datastream Commands

	COMMAND	COMMAND CODE		OP CODE
		OP CODE	PARAMETERS	
1	ENDROW	1000 0000	XXXX XXXX	80
2	EOF	1000 0001	XXXX XXXX	81
3	END OF STRING & END OF ROW	1000 0010	XXXX XXXX	82
4	FULROWDESCRPT	1000 0011	"n"	83
5	SL SCROLL STRT	1000 0100	XXX SCR LINE	84
6	SL SCROLL END	1000 0101	XXX END LINE	85
7	TAB TO n	1000 0110	"n"	86
8	LD MAX DMA COUNT	1000 0111	COUNT	87
9	ENDSTRG	1000 1000	XXXX XXXX	88
10	SKIP n	1000 1001	"n"	89
11	REPEAT n	1000 1010	"n"	8A
12	SUB SUP n	1000 1011	"n"	8B
13	RPT SUB SUP n	1000 1100	"n"	8C
14	SET GEN PUR ATTRIB	1000 1101	GPA OP	8D
15	SET FIELD ATTRIB	1000 1110	XXXX XXXX	8E
16	INIT NEXT PROCESS (Command process command)	1000 1111	XXXX XXXX	8F
17	(RESERVED)	10XX XXXX	XXXX XXXX	90-BF
18	NOP	11XX XXXX	XXXX XXXX	C0-FF

The preceding commands are recognized as valid datastream commands. The corresponding command codes are also indicated. It should be noted that the most significant bit of the command bit is always 1, in order for the word to be interpreted as command.

The "Init Next Process" command can be issued only through a command process in Virtual Screen Display. It is included in this list because its operation is analogous to a datastream command in a virtual screen access environment. Also, in virtual screen display certain datastream commands are interpreted differently, depending upon whether they are encountered in a process datastream or as command process commands. When a command is ignored (becomes a NO-OP) in a virtual display, any parameters that are associated with it are also ignored. The command process command operation is discussed separately. The operation of all other datastream commands is described below.

ENDROW

15	14	8	7	0
1	000	0000	XXXX	XXXX

This command signifies that no more characters will be loaded in the Row Buffer for this row and an End of Row indicator is stored accordingly. When the row currently being loaded is displayed, the Display Generator (DG) will blank the screen from the end of row character position until the physical end of row.

The Micro Controller Unit (MCU) stops fetching data and waits for DG to swap the Row Buffers. The data access task is resumed following the buffer swap. If a physical end of frame is reached while the MCU is waiting for a buffer swap the MCU ceases to wait and executes an EOF (End of Frame) command.

In virtual display, this command is interpreted as a VEOR (Virtual End of Row) if encountered in a virtual process datastream.

VEOR

ENDROW command in a virtual process datastream is interpreted as VEOR (Virtual End of Row) and it terminates a virtual row. The current LPTR is stored in the process header addressed by the "Process Addr" register. The Max Count register is also stored in the Max DMA Count location in the process header. Similarly, the Field Attribute Mask is also stored in the header. In addition, in auto linefeed mode (ALF = 1) other parameters characterizing the process state are also saved in the header. The "Process Addr" register is loaded with the address of the header of the next process fetched from the Access table. The "Access Tab Addr" register is post-incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. The data access task is then resumed for the next process.

EOF

15	14	8	7	0
1	000	0001	XXXX	XXXX

This command (End of Frame) signifies that no more characters will be loaded in the Row Buffers for this frame. The Micro Controller Unit (MCU) stops fetching data words and waits for the physical end of frame. If a virtual display is in progress, this command is interpreted as VEOS (Virtual End of Frame), if encountered in a virtual process datastream.

The Display Generator (DG) swaps the row buffers at the end of the current display row and starts displaying the row containing the EOF command. When the character preceding the EOF command is displayed, the DG blanks the screen until the physical end of frame. The MCU fetches the Status Row data then waits until its display is completed. It then performs the actions described below.

If LPEN has been enabled and a rising edge on the LPEN input has been detected, the LPENROW and LPENCOL positions in the command block are updated and the LPU status bit is set. If a Channel Attention has occurred, i.e., if CA has been activated, the command byte is fetched from command block and the specified channel command is executed. If the command issued is a "Stop Display" command, the MCU will terminate the display process and wait for the next channel attention. Otherwise, the MCU resumes the data access task by reinitializing pointers for the new frame and continues to fill the Row Buffers.

VEOF

EOF command in a virtual process datastream is interpreted as VEOF (Virtual End of Frame). It provides for reinitialization of LPTR using LISTSWITCH, LBASE0 and LBASE1 for each process, analogous to the automatic reinitialization of LPTR at each end of frame in a Normal Display.

LPTR for the current process is reinitialized using LISTSWITCH, LBASE0 and LBASE1 contained in the process header. The End of Display (EOD) bit in the header is set to 1. The current process is terminated as in a VEOR and the next process in Access Table is accessed.

EOL

15	14	8	7	0
1	000	0010	XXXX	XXXX

The EOL (End of Line) command has a combined effect of NXTROW and NXTSTRG commands. All the actions performed in a END OF ROW command are carried out. In addition a END OF STRING command is executed before resuming the data access task. Thus, following the end of row, the data access is continued with the next data string. In virtual process datastream, this command has the combined effect of VEOR and END OF STRING.

FULROWDESCRPT

15	14	8	7	0
1	000	0011	n	

The next "n" words fetched from memory are loaded into the Row Characteristics holding registers. "n" is specified by the lower order byte of the command word and should be between 0 and 7.

The parameters loaded by this command will be used to define the row characteristics at the time the row currently being loaded is displayed. The data words defining these characteristcs which follow the FULROWDESCRPT command must be ordered and organized in memory in a specific format. The format for FULROWDESCRPT parameters is shown below in Figure 6 starting with "Lines Per Row" as the first parameter loaded.

This command will be ignored if encountered in a virtual process datastream. The MSB of all the parameters must be zero for proper operation in virtual display.

	Upper Byte								Lower Byte							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					RVV	BLK	DBL	W								
Lines per row	—	—	—	—	ROW	ROW	HGT	DEF	—	—	—			LPR		
Normal Start/Stop	—	—	—				NRMSTRT		—	—	—			NRMSTOP		
Superscript Start/Stop	—	—	—				SUPSTRT		—	—	—			SUPSTOP		
Subscript Start/Stop	—	—	—				SUBSTRT		—	—	—			SUBSTOP		
Cursor 1 Start/Stop	—	—	—				CUR1 STRT		—	—	—			CUR1 STOP		
Cursor 2 Start/Stop	—	—	—				CUR2 STRT		—	—	—			CUR2 STOP		
Underline Line Selects	—	—	—				UL2 LINE SEL		—	—	—			UL1 LINE SEL		

RVV ROW, when this bit is set the CRVV pin will be inverted for the next full row.

BLK ROW, when this bit is set the row will be blanked (BLANK high).

DBLHGT, when the double height bit is set, all character are displayed with twice the scan lines per row.

WDEF, when the width defeat bit is set, the WDEF pin is activated for the entire row.

The following can be programmed from 0 to 31 yielding a range of 1 to 32 lines.

LPR specifies number of lines per row.

NRMSTRT, SUPSTRT, SUBSTRT specify line numbers in a display row which mark the start of normal, superscript and subscript characters respectively.

NRMSTOP, SUPSTOP, SUBSTOP specify line numbers in a row where normal, super script and subscript characters end respectively.

CUR1 STRT, CUR2 STRT specify the starting line numbers in a row for cursor 1 and cursor 2 respectively.

ULINE1 SEL, ULINE2 SEL specify the line numbers in a row where underline 2 will appear respectively.

All FULROWDESCRPT parameters affect the row in which they are programmed and stay in effect until changed by another FULROWDESCRPT command.

Figure 6. Format for FULROWDESCRPT

SL SCROLL STRT

15	14	8	7	5	4	0
1	000	0100	XXX		SCR LINE	

The Slow Scan register in 82C3 is loaded with the scroll line specified by the five least significant bits of the command word. When the row currently being loaded is displayed, the line count for that row will start with the value specified by the Slow Scan register. A "Margin" (MGN) parameter, loaded by MODESET, specifies the number of blank lines plus one to be added at the top of the slow scroll field on the screen. This ensures the availability of sufficient DMA time for fetching the next row, when only a small number of scan lines are displayed in the top row of slow scroll window. This command is used for starting a slow scroll. (Note: MGN = 0 results in no margin buffer lines)

This command will be ignored if encountered in a virtual process datastream or if a SL SCROLL END command is encountered later on the same row.

SL SCROLL END

15	14	8	7	5	4	0
1	000	0101	XXX		END LINE	

The scroll location in row characteristics holding registers is loaded with the number of lines specified by the five least significant bits of the command word. This number specifies the number of lines to be displayed when the row currently being loaded is displayed. This is used instead of the regular LPR (Lines Per Row) characteristics, for this particular row. This command is used in the last row of a slow scroll for terminating a slow scroll. The Margin (MGN) parameter, loaded by MODESET, is used in the same way as in slow scroll start except that the specified number of blank lines are inserted at the bottom of the slow scroll in this case. This command will be ignored if encountered in a virtual process datastream or if followed by a SL SCROLL STRT on the same row.

TAB TO n

15	14	8	7	0
1	000	0110	"n"	

The lower byte of the command word specifies the column (RCLK count) after SYNCSTRT at which a Tab should occur. At display time, after the character preceding the Tab command is displayed, the screen is blanked until the RCLK count specified by the command ("n") is reached. After reaching the specified count, display is resumed by displaying the character following the TAB command.

If the RCLK count specified by the Tab command has already occurred before beginning the blanking for Tab, the display will be blanked until the end of the row.

This command is ignored, if encountered in a virtual display process datastream.

LD MAX DMA COUNT

15	14	8	7	0
1	000	0111		MAX COUNT

The Max Count register in 82730 is loaded with the Max DMA Count specified by the lower byte of the command word. The DMA Counter is also reinitialized with the Max Count value in the Command Block after all channel commands.

MAX DMA Count is programmable in the range of 1 to 256 (MAX COUNT value 0 equals 256). However, counts greater than the row buffer capacity will cause row buffer overruns if the data strings depend on MAX DMA to terminate the fetching.

The DMA counter is decremented for each data word as the Row Buffer is being loaded. Datastream commands and words supplying parameters for datastream commands as in FULROWDESCRPT, are not counted. Superscript/Subscript characters are counted in pairs, i.e., a pair of characters causes only one count.

In virtual screen display, every time a new process is accessed, the DMA counter is initialized with the Max DMA Count contained in the process header. This value is also stored in a Max Counter register.

At virtual end of row (VEOR) the Max Count register is written to the process header. The "LD Max DMA Count" command is ignored if encountered in a virtual process datastream.

ENDSTRG

15	14	8	7	0
1	000	1000	XXXX	XXXX

The SPTR register in the 82730 is loaded with a new String Pointer (SPTR) value fetched from the memory location indexed by the List Pointer (LPTR), which is stored in the LPTR register. The

LPTR register is incremented by two if a 16-bit addressing option is used and by four if 32-bit addressing is used. When more than one 82730 is connected in a cluster, each of them adds an offset, determined by its position in the cluster, to the pointer fetched from memory, before storing it in its SPTR register.

This command directs the data access to the next data string in the list of strings indexed by LPTR. The operation of this command is identical for a Virtual or Normal Display. In virtual display, the next data string within the current display process is accessed.

SKIP n

15	14	8	7	0
1	000	1001	n	

The next "n" data words fetched from memory are ignored. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to 0 is specified, no words are skipped. Any datastream commands encountered in the data fetch are not counted towards these n words. Also parameters following the datastream command as in FULROWDESCRPT are not counted. All embedded datastream commands are executed. If the data words skipped include any superscript-subscript characters, they are skipped in pairs and a pair of characters is counted as only one count in "n". If another skip command is encountered its value of "n" is added to the present skip count and skipping continues.

If an EOF (End of Frame) datastream command is encountered, SKIP n is terminated. A ENDROW command causes termination of a SKIP n command in non-auto linefeed mode (ALF=0) in either normal or virtual dispaly mode. If ALF=1 the ENDROW is ignored, and not counted.

REPEAT n

15	14	8	7	0
1	000	1010	n	

The next data word (byte, if DTW16=0) fetched from memory is stored in the Row Buffer "n" times, where "n" is specified by the lower byte of the command word. "n" is programmable from 0 to 255. If n equal to 0 is specified no repetitions will occur, and the word following the Repeat n command will be ignored. This character will eventually be displayed n times. The DMA counter is also made to count n times. In non-auto linefeed mode (ALF = 0), reaching Max DMA Count before the n repetitions are completed will result in a termination of the Repeat n command. This command will also be terminated if the Row Buffer gets filled completely before the n repetitions are completed.

It should be noted that the data word immediately following the Repeat n command is treated as character data, irrespective of the value of its command bit.

SUP/SUB n

15	14	8	7	0
1	000	1011	n	

The next "n" pairs of data words (bytes, if DTW16 = 0) fetched from memory are treated as superscripts or subscript characters. "n" is specified by the lower byte of the command word. These n pairs are assumed to be ordered with the superscript preceding the subscript.

No datastream commands are permitted in the 2n words following this command. All of these words are interpreted as superscript-subscript pairs. The DMA counter is made to count only once for each pair of characters. In non-auto linefeed mode (ALF=0), reaching the Max DMA Count will result in a termination of this command. If n equal to zero is specified, no action will result.

RPT SUB/SUP n

15	14	8	7	0
1	000	1100	n	

The operation of this command is similar to that of the "Repeat n" command except that the pair of characters following the "RPT SUB/SUP n" command is repeated n times. "n" is specified by the lower byte of the command word and is programmable from 0 to 255. If n equal to zero is specified, no repetitions will occur, and the two data words following the "RPT Sub/Sup n" command will be ignored. The two data words (bytes, if DTW16=0) immediately following the command word are interpreted as a superscript-subscript pair and are repeated. The DMA counter is made to count only once for each repetition of the pair. In non-auto linefeed mode (ALF=0), reaching Max DMA Count prior to completion of n repetitions will cause a termination of this command.

SET GEN PUR ATTRIB

15	14	8	7	0
1	000	1101	GPA OPERAND	

This command provides control over the output pins assigned to General Purpose Attributes, GPA1 through GPA4.

	7	6	5	4	3	2	1	0
GPA OPERAND	GPA4 DATA	GPA4 EN	GPA3 DATA	GPA3 EN	GPA2 DATA	GPA2 EN	GPA1 DATA	GPA1 EN

ENCODING

GPAx DATA	GPAx EN	FUNCTION
0	0	ROW BUFFER DATA
1	0	ROW BUFFER DATA
0	1	GPA DATA = 0
1	1	GPA DATA = 1

In Virtual Display, everytime a display process is accessed, the state of the General Purpose Attributes is loaded from the header. The GPA in the Process Header is also updated each time a SET GPA command is executed. Thus the GPA state in the header is updated to reflect any changes caused by the "Set Gen Pur Attrib" command.

The encoding of the operand, specifying GPA operation, is shown below.

SET FIELD ATTRIB

15	14	8	7	0
1	000	1110	XXXX	XXXX
0	FIELD ATTRIBUTE MASK			

The word following this command is fetched. This word is used as a Field Attribute Mask in storing all subsequent display data words in row buffer. The bits in the data words fetched from memory corresponding to the bit positions containing a "1" in Field Attribute Mask are all set to 1 before storing the data word in row buffer. The Field Attribute Mask is used on all display data words fetched from memory. The mask register will contain all 0's upon reset and is cleared at the beginning of each frame.

NOP

15	14	8	7	0
1	1XX	XXXX	XXXX	XXXX

No action is taken. The data access task is resumed by fetching the next data word.

Datastream Command Conventions

The reaching of Max DMA Count, encountering of terminating commands such as ENDROW, EOF, etc. and occurrences of these while executing a "skip n" command give rise to various possible combinations of events. The behaviour of 82730 under these circumstances is described below:

i) When Max DMA Count is reached, it has the effect of a VEOR command if a Virtual Display is in progress or a ENDROW command if a Normal Display is in progress. It also causes an automatic end of string i.e., the effect of a NXTSTRG command in non-auto linefeed mode (ALF = 0).

ii) In non-auto linefeed mode, "Repeat n", "Sub/Sup n" and Rpt Sub/Sup n" commands are terminated upon reaching a max DMA count, even if "n" is not reached.

iii) "Skip n" command is terminated if EOF command is encountered. It is also terminated upon encountering a ENDROW command in non-auto linefeed mode (ALF = 0).

iv) "Repeat n" "Sub/Sup n" and "RPT Sub/Sup n" commands can be nested within a "Skip n" command. If superscript-subscript characters are skipped, each pair of characters counts as one skipped character. If the above commands are encountered during a "skip n" and if the specified count (n) in these commands is not reached by the end of execution of the "skip n" command, the execution of the nested command is continued beyond the termination of "skip n" command until the remaining portion of the count specified in the nested command is completed.

VIRTUAL SCREEN MODE

Command Process Commands

In Virtual Screen Display, 82730 accesses display processes and command processes through the Access table. The command processes enable the I/O Driver process to direct 82730 to execute certain data stream commands by inserting an appropriate command process address in the Access table. This capability enables the preservation of uniformity and consistency of operation between normal and virtual environments, by assigning different interpretations to the command according to the access environment. It is especially useful for termination and initialization commands. The operation of command process commands is analogous to that of data stream commands except for a different access environment.

Command Process Command List

The commands allowed in command processes can be divided into two subsets. The first subset consists of commands that can be issued only through a command process, while the second one consists of normal datastream commands that can also be issued through a command process. The command code for a datastream command issued through a command process is the same as that for the normal datastream command embedded in the data. However, certain datastream commands are interpreted differently when they are issued through a command process as opposed to embedding in the datastream of a virtual display process. The most significant bit (MSB) of the command word must be a "1". In the datastream, this bit distinguishes a command word from character data. In the process environment, this bit distinguishes a command process from a display process. The commands permitted in command processes are listed below. No other commands will be recognized if encountered in a command process and will result in a NOP. All undefined command codes apart from those designated as NOP are reserved and should not be used. Encountering an illegal command code causes the RDC (Reserved Datastream Command) status bit to be set and will generate an interrupt, if enabled.

Table 5. Command Process Command List

COMMAND	INTERPRETATION IN VIRTUAL PROCESS DATASTREAM	COMMAND CODE				OP CODE
		OP CODE		PARAMETERS		
Command Process Only Command:						
1 INIT NEXT PROCESS	NOP	1000	1111	XXXX	XXXX	8F
Command Process or Datastream Commands:						
2 ENDROW	VEOR	1000	1000	XXXX	XXXX	80
3 EOF	VEOR	1000	0001	XXXX	XXXX	81
4 EOL	VEOR + NXTSTRG	1000	0010	XXXX	XXXX	82
5 FULROWDESCRPT	NOP	1000	0011	"n"		83
6 SL SCROLL STRT	NOP	1000	0100	XXX	"SCR LINE"	84
7 SL SCROLL END	NOP	1000	0101	XXX	"END LINE"	85
8 TAB TO n	NOP	1000	0110	"n"		86
9 LD MAX DMA COUNT	NOP	1000	0111	"COUNT"		87
10 (RESERVED)	RESERVED	10XX	XXXX	XXXX	XXXX	90-BF
11 NOP	NOP	11XX	XXXX	XXXX	XXXX	C0-FF

INIT NEXT PROCESS

15	14	8	7	0
1	000	1111	XXXX	XXXX

This command can be used only in a command process to initiate a virtual display "window".

Upon receiving this command, the command process is terminated and the next process in Access Table is accessed by fetching the new process address. However, the LPTR register is not directly loaded from the LPTR location in the process header. Instead, LISTSWITCH in the process header is examined and LPTR is initialized with the value LBASE 0 or LBASE 1 depending upon whether LISTSWITCH is 0 or 1 respectively. Both LBASE0 and LBASE1 are contained in the header.

The process header format is shown in Figure 7. Also the End of Display Bit (EOD) in the header is reset.

The data access task for a virtual display is then resumed, with this value of LPTR.

	15	14	13	8	7	6	0	LOCATION
LS: LISTSWITCH ALF: AUTO LINE FEED	0	----			EOD		----	PROCESS ADDR
		----			LS ALF			PROC ADDR + 2
		----					MAX DMA COUNT	PROC ADDR + 4
					LBASE0 LOWER			PROC ADDR + 6
					LBASE0 UPPER			PROC ADDR + 8
					LBASE1 LOWER			PROC ADDR + 10
					LBASE1 UPPER			PROC ADDR + 12
	1	----					GPA	PROC ADDR + 14
	1				FIELD ATTRIBUTE MASK			PROC ADDR + 16
					LPTR LOWER			PROC ADDR + 18
					LPTR UPPER			PROC ADDR + 20
SAVE AREA					SPTR LOWER			PROC ADDR + 22
					SPTR UPPER			PROC ADDR + 24
	RPT S/S	S/S	RPT	— —		REPT COUNT		PROC ADDR + 26
	1				REPT CHAR			PROC ADDR + 28
	1				REPT CHAR 2			PROC ADDR + 30

PROCESS ADDR

15	14	8	7	0
1			COMMAND	

C/$\overline{D}$

Figure 7. Process Header for Display and Command Process

ENDROW

15	14	8	7	0
1	000	0000	XXXX	XXXX

The actions performed by a ENDROW datastream command in a Normal Display are carried out. The next process in Access Table is accessed and the data access task is resumed, after the next Row Buffer swap

EOF

15	14	8	7	0
1	000	0001	XXXX	XXXX

The actions performed by an EOF (End of Frame) data stream command in a Normal Display are carried out.

EOL

15	14	8	7	0
1	000	0010	XXXX	XXXX

This command is identical to ENDROW command in Virtual Display in Command Process environment. ENDSTRG, which is strictly a data operation within a display process is meaningless in the command process environment.

FULROWDESCRPT

15	14	8	7	0
1	000	0011	"n"	

The actions performed by the FULROWDESCRPT datastream command are carried out. The data access task is resumed by accessing the next process in the Access Table.

SL SCROLL STRT

15	14	8	7	5	4	0
1	000	0100	XXX		"SCR LINE"	

The same actions as the SL SCROLL STRT datastream command. The data access is resumed with the next process in Access Table.

SL SCROLL END

15	14	8	7	5	4	0
1	000	0101	XXX		"END LINE"	

The actions performed by a SL SCROLL END datastream command, in a Normal display, are carried out. The data access task is resumed with the next process in Access Table.

TAB TO n

15	14	8	7	0
1	000	0110	"n"	

The effect of this command process command is identical to that of the TAB TO n datastream command. The TAB can be used to establish the left edge of a virtual display "window".

LD MAX DMA COUNT

15	14	8	7	0
1	000	0111		MAX COUNT

The Max Count register on 82730 is loaded with the value specified by the lower byte of the command word. The DMA counter is also initialized with this Max Count Value.

The next process in the Access Table is accessed. However, the Max DMA Count value in the process header is not used for initializing the DMA counter. Instead, the DMA counter as initialized by the LD Max DMA Count command is used for this process. The virtual display data access task is then resumed normally. When the process is terminated, the new Max Count value is written to the process header. Thus the Max Count value in the header is updated as a result of this command.

NOP

15	14	8	7	0
1	1XX	XXXX	XXXX	XXXX

No action is taken. Data access task is resumed by fetching the next process address from Access Table.

ERROR AND STATUS HANDLING

Error Conditions

Since the MCU and DG function asynchronously with respect to each other, different relative timings in MCU and DG operation are possible, some of which result in error conditions. The lack of appropriate termination commands for row or frame data in the datastream also gives rise to certain error conditions. These types of situations occurring in display process operation are described below.

In normal operation, DG initiates a buffer swap at the physical end of a display row. If the MCU has not finished loading its row buffer by that time, a "Data Underrun" occurs. This results in

blanking of the screen until physical end of frame by DG and execution of an EOF (End of Frame) command by MCU. Data underrun also occurs when the first row of the frame has not finished loading by the start of the character field. The entire frame will be blanked in this case.

If a physical end of frame is reached prior to encountering an EOF datastream command, a "Frame Data Error" occurs, which results in the execution of an EOF command by MCU. (Note that this does not disrupt the visible display action, and may not constitute an error for certain data structures. The error indication is included as a flag where knowledge of this condition is desired.) Similarly, when the MCU fills up a row buffer completely, without encountering a END-ROW command, the "Data Buffer Overrun" flag is set.

All of the above conditions result in the setting of an appropriate status bit and generation of an interrupt if the corresponding interrupt has been enabled.

15 ... 9	8	7	6	5	4	3	2	1	0
(RESERVED)	VDIP	DIP	RDC	RCC	FDE	EOF	DBOR	LPU	DUR

VDIP: Virtual Display In Progress
DIP: Display In Progress
RCC: Reserved Channel Command
RDC: Reserved Datastream Command
FDE: Frame Data Error

DUR: Data Under Run

This status bit is set by Display Generator if the Microcontroller Unit (MCU) has not finished loading its Row Buffer when the DG initiates a buffer swap at the physical end of a display row. This condition is defined as data underrun and causes the MCU to execute an EOF command and the DG to blank the screen until the physical end of frame.

LPU: Light Pen Update

This status bit is set by the MCU after updating the LPENROW and LPENCOL locations in command block. The detection of LPEN input is enabled by the LPEN ENABLE channel command. The detection of a rising edge on the LPEN input causes the current row and column position to be stored internally. The MCU updates the LPEN ROW and LPEN COL locations in command block at the next end of frame and sets the LPU status bit. Further updates of these command block locations are inhibited until another LPEN ENABLE command is issued.

EOF: End of Frame
DBOR: End of Row
LPU: Light Pen Update
DUR: Data Under Run

DBOR: Data Buffer Over Run

This status bit is set when the MCU tries to fill a row buffer beyond its capacity. The MCU will stop fetching characters after this point and the display is blanked following the completion of the row currently being displayed.

Status and Interrupt Handling

A status word is maintained in an internal register by 82730 and it is written to the "STATUS" location in command block when the "Read Status" channel command is executed. The processor can thus read status information by issuing this command. the processor can also enable interrupts for certain status bits by specifying an interrupt mask which is loaded in 82730 as a result of a "Load Int Mask" channel command. This establishes a communication mechanism between 82730 and the processor for error and status reporting.

Status Word

The format for the status word is shown below. The function of each of the status bits is described below.

The status bits get set under the conditions described above. Interrupts can be enabled for all status bits except DIP and VDIP bits. The interrupt status bits are cleared at the beginning of each new display field. DIP and VDIP bits are cleared only after receiving a "STOP DISPLAY" command or a Reset.

All status bits are cleared by a Reset.

EOF: End of Frame

This bit is set by the DG at the physical end of the nth frame, where 'n' is specified by the MODESET parameter FRAME INTERRUPT COUNT. This provides the means for timing frame related events such as slow scrolls.

FDE: Frame Data Error

This status bit is set by the DG at the physical end of frame if no EOS datastream command has been encountered until then. This also results in the execution of the EOS command by the MCU.

RCC: Reserved Channel Command

This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

RDC: Reserved Datastream Command

This bit is set by the MCU upon encountering an illegal datastream or command process command. This can be used to trap software errors during program development.

DIP: Display In Progress

This bit is set by the MCU immediately after receiving a "Start Display" channel command. It remains set as long as the display process is active and is reset upon receiving a "Start Virtual Display" or "Stop Display" command or a Reset. Interrupts cannot be enabled for this status bit.

VDIP: Virtual Display In Progress

This bit is set by the MCU immediately after receiving a "Start Virtual Display" channel command and is reset upon receiving a "Start Display" or "Stop Display" command or a Reset. This bit remains active as long as the virtual display process is active. Interrupts cannot be enabled for this status bit.

Interrupt Processing

The system processor can enable interrupts on any of the status bits, with the exception of DIP and VDIP bits, by specifying an interrupt mask. A "1" in a bit position in the interrupt mask disables (masks out) interrupts on the status bit located in the corresponding bit position in the status word. The format for Interrupt Mask is shown below. The Int Mask can be loaded into 82730 from the INTMASK location in command block by a "Load Int Mask" channel command.

If the interrupt is enabled for a particular status bit by programming a "0" in the corresponding bit position in INTMASK and if the status bit gets set during the course of the display, an interrupt will be generated by 82730 at the next end of frame. At the end of frame, the 82730 will first perform the tasks of updating LPEN position (if required) and servicing the Channel Attention (if CA was activated). Then the status word in the internal register will be written to the INT GENERATION CODE location in the Command Block and the SINT output will be activated. The SINT pin is not deactivated until an interrupt reset signal is received at the IRST pin.

82730 continues to perform its normal display task after activating the SINT pin. If no interrupt reset is received until the next end of frame then any new interrupts that might have been generated at that end of frame will be lost. Therefore, it is essential for the system processor to issue an interrupt reset within a frame time after an interrupt is generated.

When the display is not activated, the only interrupt that can occur is the Reserved Channel Command interrupt. Upon receiving an invalid channel command, 82730 will write the status word to INT Generation Code location in the Command Block and activate SINT output, if that interrupt is enabled.

The processor can use the interrupt capability to get status information from 82730. A possible interrupt service routine for the system processor is shown in flow chart form in Figure 9.

15 7	6	5	4	3	2	1	0
(RESERVED)	RDC INT MASK	RCC INT MASK	FDE INT MASK	EOF INT MASK	DBOR INT MASK	LPU INT MASK	DUR INT MASK

INT MASK = 0 Enables the corresponding interrupt.
INT MASK = 1 Masks or disables the corresponding interrupt.

Figure 8. Interrupt Mask

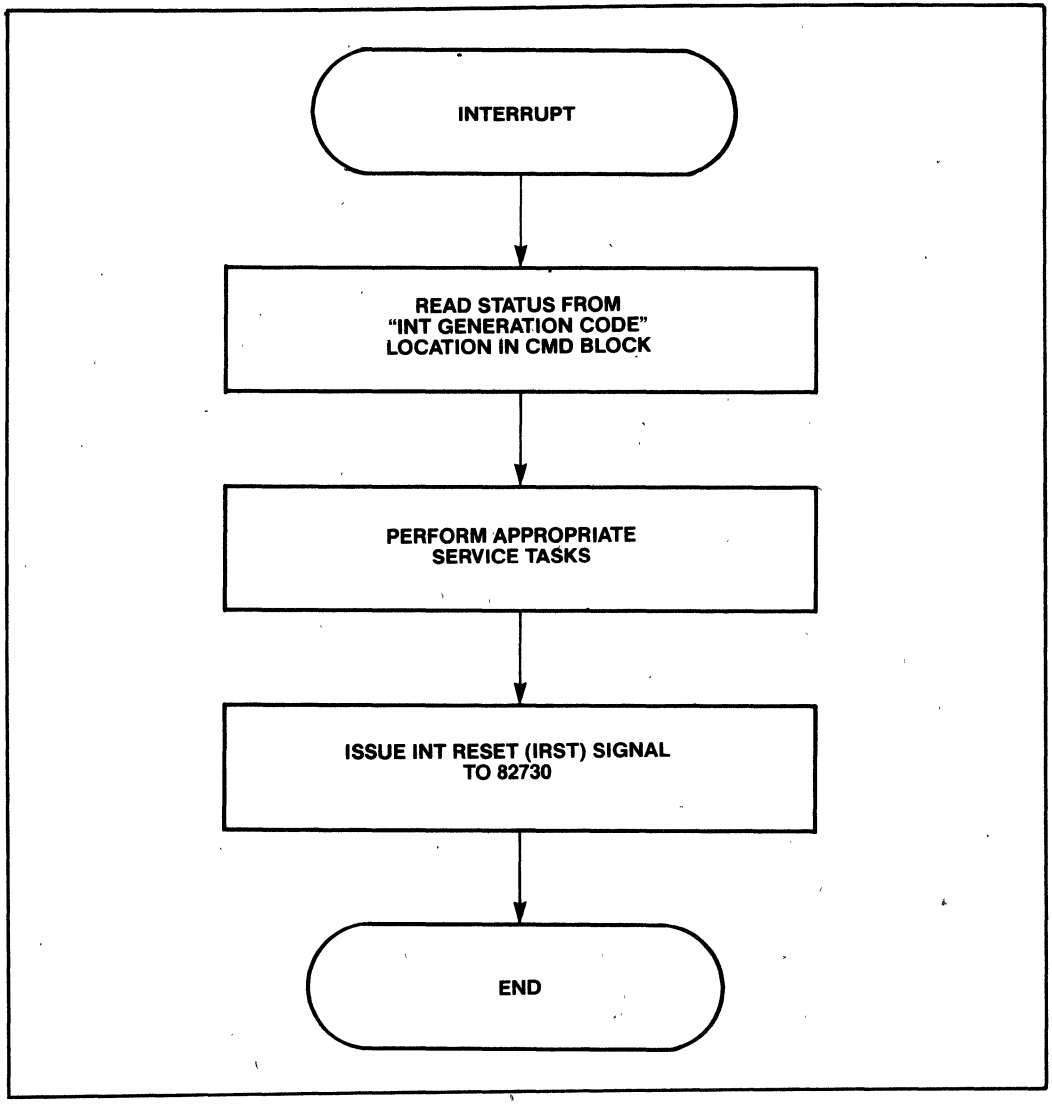


Figure 9. Interrupt Service Routine For System Processor

82730 VIDEO INTERFACE

The Mode Pointer in the Command Block points to a parameter block containing the Mode information required for the display. The organization of the mode words in the Mode Block is shown below.

FROM MODE POINTER IN COMMAND BLOCK

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	LOCATION
DMA	-	BURST LENGTH							-	BURST SPACE							MPTR
HORIZONTAL MODES	LINE LENGTH								HSYNCSTP								MPTR = 2
	HFLDSTRT								HFLDSTP								MPTR = 4
	HBRDSTRT								HBRDSTP								MPTR = 6
	-	-	-	-	-	-	-	-	-	-	-	SCROLL MARGIN					MPTR = 8
CHAR ROW CHARACTERISTICS	-	-	-	-	RVV ROW	BLK ROW	DBL HGT	W DEF	-	-	-	LPR					MPTR = 10
	-	-	-	NRMSTRT					-	-	-	NRMSTP					MPTR = 12
	-	-	-	SUPSTRT					-	-	-	SUPSTP					MPTR = 14
(FULROWDESCRPT)	-	-	-	SUBSTRT					-	-	-	SUBSTP					MPTR = 16
	-	-	-	CUR1STRT					-	-	-	CUR1STP					MPTR = 18
	-	-	-	CUR2STRT					-	-	-	CUR2STP					MPTR = 20
	-	-	-	U2 LINE SEL					-	-	-	U1 LINE SEL					MPTR = 22
	-	FIELD ATTRIBUTE MASK															MPTR = 24
VERTICAL MODES	-	-	-	-	-	FRAME LENGTH											MPTR = 26
	-	-	-	-	-	VSYNCSTP											MPTR = 28
	-	-	-	-	-	VFLDSTRT											MPTR = 30
	-	-	-	-	-	VFLDSTP											MPTR = 32
BLINK CONTROL	(RESERVED)																MPTR = 34
	(RESERVED)																MPTR = 36
	-	DUTY CYC		CURSOR BLINK					-	-	-	-	FRAME INT COUNT				MPTR = 38
	-	DUTY CYC		CHAR BLINK					ILE	RFE	B POL	BUE	CR2 CD	CR1 CD	CR2 BE	CR1 BE	MPTR = 40
ATTRIBUTE BIT SELECTS	REVERSE VIDEO			BLINKING CHAR					-	-	-	-	CR2 RVV	CR1 RVV	CR2 OE	CR1 OE	MPTR = 42
	ABS LINE COUNT			INVISIBLE CHAR					UNDERLINE 2				UNDERLINE 1				MPTR = 44

Figure 10. Mode Block Organization

CAM ARRAYS

Three Content Addressable Memory arrays are used for generating timing parameters to control the video display: the HORIZ MODE CAM, the VERT MODE CAM andthe CHAR ROW CAM. The user has the flexibility to define his own timing parameters by loading them into the CAM arrays via the MIU. All of these parameters can be modified at the end of every frame. All the parameters in the CHAR ROW CAM, except MARGIN, are changeable on a row by row basis. Each of the three CAM arrays is described separately below:

Timing Sources

RCLK and CCLK inputs are provided by the external video logic to the 82730. The RCLK is used to increment the HORIZ COL CNTR and hence generates all horizontal timing parameters. CCLK is used to clock the character and attribute data output from the 82730 to the external display dot logic. Data changes on the positive going edge of RCLK or CCLK.

Initialization

Upon activation of the RESET input, the 82730 display generator will stop all operations in progress and deactivate all outputs. It will stay in this quiscent state until the MIU executes the MODESET command. The following table shows the states of all the Display Generator outputs during and after RESET.

Pin Name	Condition
DAT0-14	Low
WDEF	Low
LC0-4	High
BLANK	Low
CSYNC	High
$\overline{\text{CHOLD}}$	High
HSYNC	Low
VSYNC	Low
CRVV	Low
RRVV	Low

After reset of the 82730, the CAM arrays are in undetermined states. The CAM arrays are set upon the execution by the MIU of the MODESET command. The HORIZ and VERT MODE CAM contents are especially critical since they are used to generate timing control signals to the external video logic. Without the generation of the timing signals, no display process can take place. Hence, START DISPLAY command cannot be executed before the first MODESET command after the device reset. The START DISPLAY command will be ignored if it precedes the MODESET command.

The row buffers also contain unknown information after power up and reset. In executing the START DISPLAY command, the MIU would first load the two row buffers with the first two rows of character data to be displayed. Upon completion of loading of both buffers, it will signal the DG to begin the display process. In this way, only valid character data will be output to the external video logic.

Timing Parameters

The timing parameters read from the MODESET Block and stored in the VERT MODE CAM and HORIZ MODE CAM are used to control the video display and they can be best illustrated in the Map of Timing Parameters shown below. All of these timings have to be defined after power up and reset and can be changed on a frame by frame basis during display.

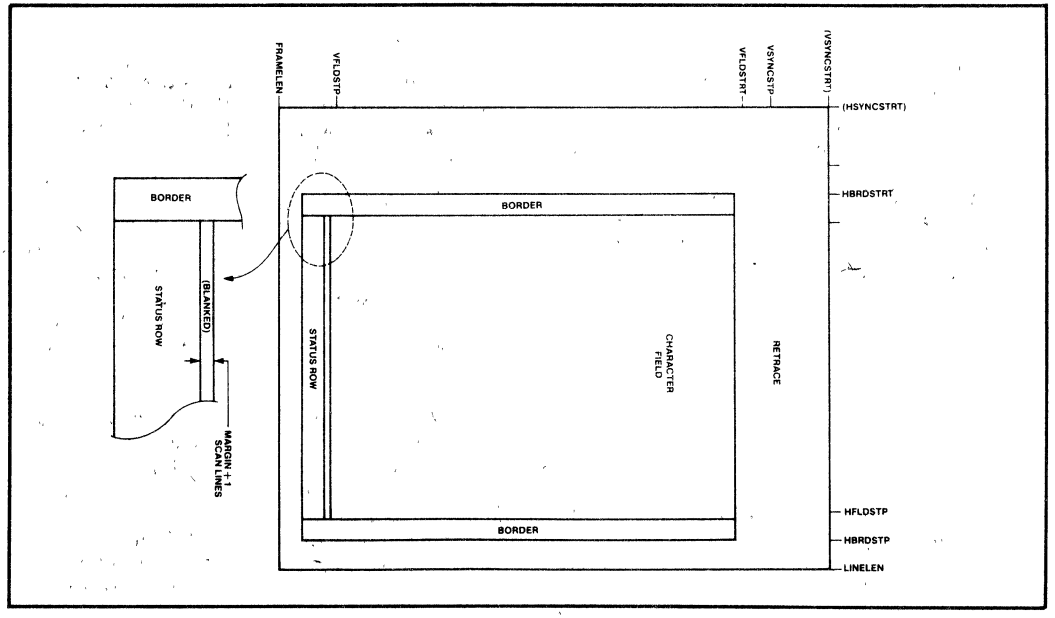


Figure 11. Timing Parameters

Row Timing Parameters

The row timing parameters are stored in HORIZ MODE CAM and are programmable from 0 to 255 RCLK times. These parameters are:

(a) HSYNCSTRT - Horizontal Sync Start. The RCLK count on each scan line where HSYNC pin is activated. This parameter is not programmable. The RCLK period that follows the rising HSYNC edge is defined as column zero. It is used as the reference for all other horizontal timing parameters.

(b) HSYNCSTP - Horizontal Sync Stop. The RCLK count on each scan line where the HSYNC pin is deactivated. The falling edge of HSYNC occurs at the leading edge of the programmed RCLK period.

(c) LINELEN - Line Length. This parameter defines the total number of RCLK's in each scan line including display time, border and horizontal retrace time. There are LINELEN + 1 RCLK periods per horizontal line scan.

(d) HBDRSTRT - Horizontal border start. The RCLK count on a scan line where the border begins. The border begins at the leading edge of the programmed RCLK period.

(e) HBDRSTP - Horizontal Border Stop. The RCLK count on a scan line where the border ends. The border terminates at the leading edge of the programmed RCLK period.

(f) HFLDSTRT - Horizontal Field Start. The RCLK count on a scan line where the character display field begins. If the row buffer is ready to be displayed, the CSYN pin will be deactivated at this point. This field begins at the leading edge of the programmed RCLK period.

(g) HFLDSTP - Horizontal Field Stop. The RCLK count on a line where the character display field stops. When this timing point is reached, CSYN will be activated. This field ends at the leading edge of the programmed RCLK period.

There is also one pseudo parameter, SYNCDLY. It is fixed at one half LINELEN and is used as the start and end timing for VSYNC in odd frames in interlaced displays. VSYNC starts at HSYNCSTRT in even frames for interlaced displays and all frames for non-interlaced displays.

There are certain restrictions in the programming of HFLDSTRT and HFLDSTP and those restrictions are best illustrated below. There has to be at least 4 RCLKS in between HFLDSTRT and HFLDSTP of the same scan line and 15 RCLKS in between HFLDSTP of one line and HFLDSTRT of the next. The minimum delay of 15 RCLKS is for the charging of the pipeline from the row buffer to the character data output DAT0-DAT14 as well as the setting of the correct value for the scan line output LC0-LC4.

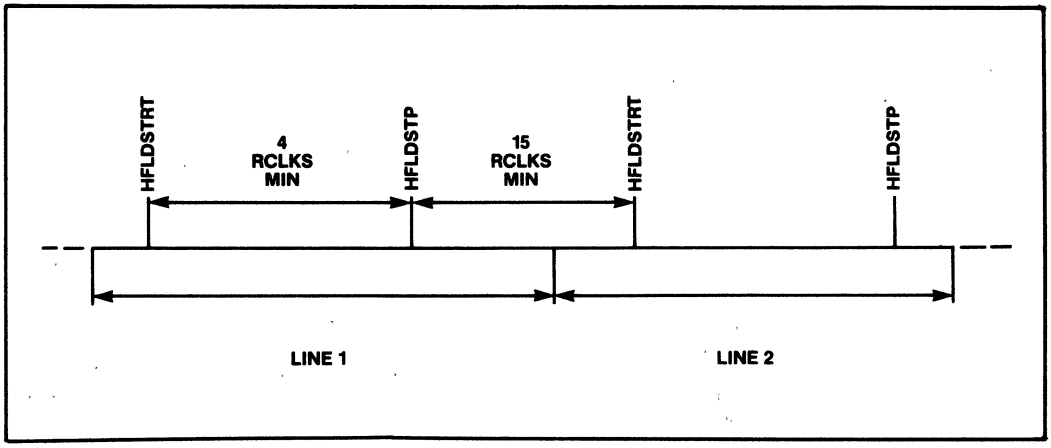


Figure 12. Horizontal Timing Restrictions

Frame Timing Parameters

Frame timing parameters are stored in the VERT MODE CAM and are programmable from 0-2047 scan lines. These parameters are:

(a) VSYNCSTRT - Vertical Sync Start. The line count where the VSYNC is activated. This occurs at the end of a field automatically. This parameter is not programmable. The rising edge of VSYNC occurs with the rising edge of HSYNC for all non-interlace fields and for odd fields in the interlace mode.

(b) VSYNCSTP - Vertical Sync Stop. The line count at which the VSYNC pin is normally deactivated. VSYNC changes at the rising edge of HSYNC normally. However it occurs at SYNCDLY at the beginning of odd fields of an interlaced display.

(c) FRAMELEN - Frame Length. This parameter defines the total number of scan lines per frame. It is used to reset the FRAME LINE CNTR. In an interlaced display, FRAMELEN must be an even number. If an odd number is programmed, one additional line will occur automatically. There will be FRAMELEN + 1 scan lines per frame. (Note that interlace mode contains two fields per frame).

(d) VFLDSTRT - Vertical Field Start. Programs the scan line count where the character display field begins.

(e) VFLDSTP - Vertical Field Stop. Programs the scan line count where the regular character display field ends. VFLDSTP times the beginning of the Status Row. The channel attention sequences, interrupt handling, row buffer swap and intialization for the next frame are started after the display of the Status Row is completed. See * below.

* (Character Field Boundry definition: The starting or ending event is defined to occur at HFLDSTP on the scan line following the programmed value. Thus the visible character field effectively begins two scan lines below the programmed start value and ends one scan line below the programmed stop value.)

Status Row

The Vertical Frame Timing Parameters have no border controls, unlike the Horizontal Row Timing Parameters. The top and bottom borders can be replaced with regular display rows that are video-reversed and contain no data. The top border is easily timed from VFLDSTRT. The bottom border is more difficult without help from the Vertical Timing generators. If there were no help, the user would have to keep track of the number of scan lines used in each row to know when to stop regular display and create the bottom border. This would also preclude his ending his regular display with an EOF command before the border.

The 82730 provides this help with the Status Row feature. The display of the Status row is timed from VFLDSTP and allows the user to display a row in a fixed position at the bottom of the screen that is independent of the regular data and any display errors (display ended by an EOF command or the DURN, DBOR, or FDE errors). (There is one dependency on the regular display data: the row format. The last FULROWDESCRPT (FRD) set in the regular data will be used on the Status Row unless a new command is issued for the row. It is recommended that the user include a new FRD command in the Status Row data to eliminate this dependency).

Status Row display starts SCROLL MARGIN plus one scan line after VFLDSTP. This margin is provided to insure enough DMA time if the regular display runs up to VFLDSTP. The user can create a bottom border or any end-of-display row that he chooses. A display status or system status line, or special programmable key function definition line can be implemented with this feature.

CHARACTER ATTRIBUTES

The 15 bits of the character word can be partitioned into character address and attribute bits.

Some common attributes may be individually defined and enabled or disabled by fields in the attribute parameter registers. Each attribute has two means of being enabled. The enable bits defined below are set during the MODESET channel command and are used as a global enable. The user does not have to enable the provided attributes. He may free more data bits for his own use this way. The second enable bit is contained in each character loaded to the row buffer to enable the attribute on a character by character basis. They are individually described in detail in the following sections.

Reverse Video

When a character with the reverse video attribute is displayed, the CRVV pin will be inverted during the time the character is being displayed. The reverse video affects the entire height of the row for that character space. For superscript/subscript pairs, the reverse video effect is controlled by superscript until SUBSTRT when the subscript attribute bit takes control. The parameter for this attribute is:

RVBS - Reverse Video Bit Select. This parameter selects one of the 15 bits of a character data word. Values 0 through 14 select the corresponding bit. Value 15 disables the Reverse Video attribute.

Blinking Character

When a character with the blinking character attribute is displayed, the BLANK pin will be activated and deactivated during the character display time according to programmable rate and duty cycle. The parameters for this attribute are:

(a) BCBS - Blinking Character Bit Select. Selects one of the 15 bits of a character data word as the blinking character attribute control. As with Reverse Video above, the value of the select determines the controlling bit or disables the attribute.

(b) CHAR BLNK FREQ - Selects one of the 32 blinking frequencies available for the blinking character and blinking underline. The character blink rate is calculated as below:

$$\text{Blink Rate} = \frac{\text{Frame Refresh Rate}}{4 \times \text{CHAR BLNK FREQ}}$$

(c) CHAR DUTY CYCLE - A 2-bit register to select 4 duty cycles available for blinking character and blinking underline. The selection logic is defined to be as follows:

00=100% always on
11= 75% on
10= 50% on
01= 25% on

Underline #1

When a character with underline is displayed, the BLANK Pin will be activated and the CRVV pin will be inverted during the time the scan line specified

by the underline select register is displayed. The parameters used to define underline #1 are:

(a) ULS1 - Underline Line Select 1. It determines which scan line of a character row will be used for the underline #1. This parameter is modifiable on a row by row basis by the FULROWDESCRPT command.

(b) ULBS1 - Underline Bit Select 1. This parameter can only be changed by MODESET. It selects one of the 15 bits of a character data word as the underline #1 attribute control. Again, a value of 15 in the select field disables this attribute.

Underline #2 (Blinking)

Underline #2 can be made to blink. When its blinking feature is deactivated, its visual effect is exactly the same as underline #1. When it is enabled to blink, its blink rate and blinking duty cycle are the same as those defined for blinking character. The parameters used to define this attribute are:

(a) UL2SEL - Underline Line Select 2. This parameter determines which scan line of a character will be the 2nd underline. It is changeable on a row by row basis by the FULROWDESCRPT command.

The next two parameters can only be modified by the MODESET Command.

(b) ULBS2 - Underline Bit Select 2. Selects one of the 15 bits of a character data word or GPA1 as the second underline attribute control. A bit select value of 15 disables the second underline.

(c) BUE - Blinking Underline Enable. Activation of this bit will cause the second underline attribute to start blinking.

Invisible

A character with this attribute will occupy its character position on the screen but will not be displayed (i.e. BLANK will be active). This attribute does not affect the Reverse Video attribute if they are programmed together. The parameter that is used to implement this attributes:

IBS - Invisible Bit Select. Selects one of the 15 bits of a character data word as the invisible attribute control. Value 15 disables the invisible attribute.

Absolute Line Cntr Attribute

This character attribute allows the display of special graphic characters, or may be used to upshift normal characters to implement displays with overlapping superscript and subscript fields. When a character with this character attribute enabled is being displayed, its LC0-LC4 pins will reflect the output from the CHAR ROW LNE CNTR which counts the absolute line count of a row. The activation of this attribute overrides the line count mode of both normal and subscript/ superscript characters. The parameter used to select the attribute is:

ABS LINE BIT SEL. This four bit register selects one of the 15 bits of a character data word as the absolute line counter output attribute control. Select value 15 disables the ABS Line attribute.

Cursor Generation

The cursor characteristic parameters are changeable on a frame by frame basis by MODESET.

(a) CUR FREQ - Cursor frequency. Selects the blinking frequency for both cursors. The selection logic is similar to CHAR BLNK FREQ

(b) CUR DUTY CYCLE - Cursor duty cycle. Selects the blinking duty cycle for both cursors. Its selection logic is similar to CHAR DUTY CYCLE.

(c) CR1RVV - Cursor 1 Reverse Video Enable selects a reverse video type cursor as opposed to a solid (blanking) cursor.

(d) CR1BE - Cursor 1 Blink Enable changes the cursor 1 block or underline to a blinking block or underline. Enabling this bit also causes DAT 14 pin to "blink" as well, if the CR10E bit is set.

(e) CR10E - Cursor 1 Output Enable reconfigures the DAT 14 pin to indicate when cursor 1 is active. CR20E enabled directs the cursor 2 signal to DAT 13 pin in a similar fashion.

(f) CR1CD - Cursor 1 Light Pen Cursor Detect directs the CCLK cursor #1 position to be translated to its nearest equivalent RCLK position through the LPEN facility.

An identical set of parameters (c) through (f) is available for the generation of CURSOR 2. The two cursors share the same FREQ and DUTY CYCLE parameters.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias .... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 3 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	Volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	$I_{OL} = 2$ mA [1]
V_{OH}	Output High Voltage	2.4		Volts	$I_{OH} = -400\ \mu A$
I_{CC}	Power Supply Current		400	mA	@$T_A = 0°C$
I_{LI}	Input Leakage Current		10	μA	$V_{IN} = 0 - V_{CC}$
I_{LO}	Output Leakage Current		10	μA	$V_{OUT} = 0.45 - V_{CC}$
V_{BLI}	Bus Clock Input Low Voltage	−0.5	0.8	Volts	
V_{BHI}	Bus Clock Input High Voltage	2.0	V_{CC} + 1.0	Volts	
V_{CLI}	Character Clock Input Low Voltage	−0.5	0.8	Volts	
V_{CHI}	Character Clock Input High Voltage	2.0	V_{CC} + 0.5	Volts	
V_{RLI}	Reference Clock Input Low Voltage	−0.5	0.8	Volts	
V_{RHI}	Reference Clock Input High Voltage	2.0	V_{CC} + 0.5	Volts	

NOTE:
1. $I_{OL} = 2.6$ mA on the $\overline{S1}$ and $\overline{S0}$ pins.

A.C. CHARACTERISTICS

82730 Bus Interface Input Timing Requirements
$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds.

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLCL	BCLK Cycle Period	125	2000	ns	
TCLCH	BCLK Low Time	52		ns	
TCHCL	BCLK High Time	52		ns	
TCH1CH2	BCLK Rise Time		30	ns	0.45V→2.4V
TCL1CL2	BCLK Fall Time		30	ns	2.4V→0.45V
TDVCL	Data in Set-Up Time	20		ns	

A.C. CHARACTERISTICS (Continued)

82730 Bus Interface Input Timing Requirements (Continued)

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds.

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLDX	Data on Hold Time	10		ns	
TARYHCH	Async. READY Active Set-Up Time	35		ns	
TSRYHCL	Sync. READY Active Set-Up Time	+20		ns	
TRYLCL	READY Inactive Set-Up Time	10		ns	
TCLRYX	READY Hold Time	20		ns	
TCTVCL	HLDA, RESET Set-Up Time	35		ns	
TCLCTX	HLDA, RESET Hold Time	10		ns	
TCAVCAX	CA Pulse Width	TCLCL		ns	

82730 Bus Interface Output Timing Response

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds. $C_L = 200$ pF except on ALE where $C_L = 100$ pF

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLAV	Address Valid Delay	0	70	ns	
TCLAX	Address Hold Time	0		ns	
TAVAL	Address Valid to ALE/UALE Inactive	TCLCH – 30		ns	
TLLAX	Address Hold to ALE Inactive	TCHCL – 10		ns	
TCLAZ	Address Float Delay	TCLAX	45	ns	
TAZRL	Address Float to $\overline{RD}$ Active	0		ns	
TLHLL	ALE/UALE Width	TCLCH – 10		ns	
TCLLH	ALE/UALE Active Delay	0	45	ns	
TCHLL	ALE/UALE Inactive Delay	0	45	ns	
TCVCTV	Control Active Delay ($\overline{DEN}$,$\overline{WR}$,$\overline{AEN}$)	0	70	ns	
TCVCTX	Control Inactive Delay ($\overline{DEN}$,$\overline{WR}$,$\overline{AEN}$)	0	80	ns	
TCLDOV	Data Out Valid Delay	0	55	ns	
TCLDOX	Data Out Hold Time	0		ns	
TDWHDOX	Data Out Hold Time After $\overline{WR}$	TCLCL – 60		ns	
TCLHV	Hold Output Delay	0	85	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL – 50		ns	
TCLRL	$\overline{RD}$ Active Delay	0	95	ns	
TCLRH	$\overline{RD}$ Inactive Delay	0	70	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL – 40		ns	

A.C. CHARACTERISTICS (Continued)

82730 Bus Interface Output Timing Response (Continued)

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds. $C_L = 200$ pF except on ALE where $C_L = 100$ pF

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLSIN	SINT Valid Delay	0	70	ns	
TRIHSIL	RINT Active to SINT Inactive		250	ns	
TCHSV	Status Active Delay	0	75	ns	
TCLSH	Status Inactive Delay	0	70	ns	
TWLWH	$\overline{WR}$ Width	2TCLCL – 40		ns	
TFLHL	Bus Float to HOLD Inactive	0		ns	

82730 Display Generator Input Timing Requirements

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds. $C_L = 100$ pF except where noted.

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TRCHRCH	RCLK Cycle Period	100	2500	ns	
TRCHRCL	RCLK High Time	40		ns	
TRCLRCH	RCLK Low Time	40		ns	
TRRCK	RCLK Rise Time		30	ns	0.45V→2.4V
TFRCK	RCLK Fall Time		30	ns	2.4V→0.45V
TCCHCCH	CCLK Cycle Period	100	None	ns	
TCCHCCL	CCLK High Time	30		ns	
TCCLCCH	CCLK Low Time	40		ns	
TRCCK	CCLK Rise Time		30	ns	0.45V→2.4V
TFCCK	CCLK Fall Time		30	ns	2.4V→0.45V
TSYVCR	SYNCIN Set-Up Time to RCLK in Slave Mode		30	ns	

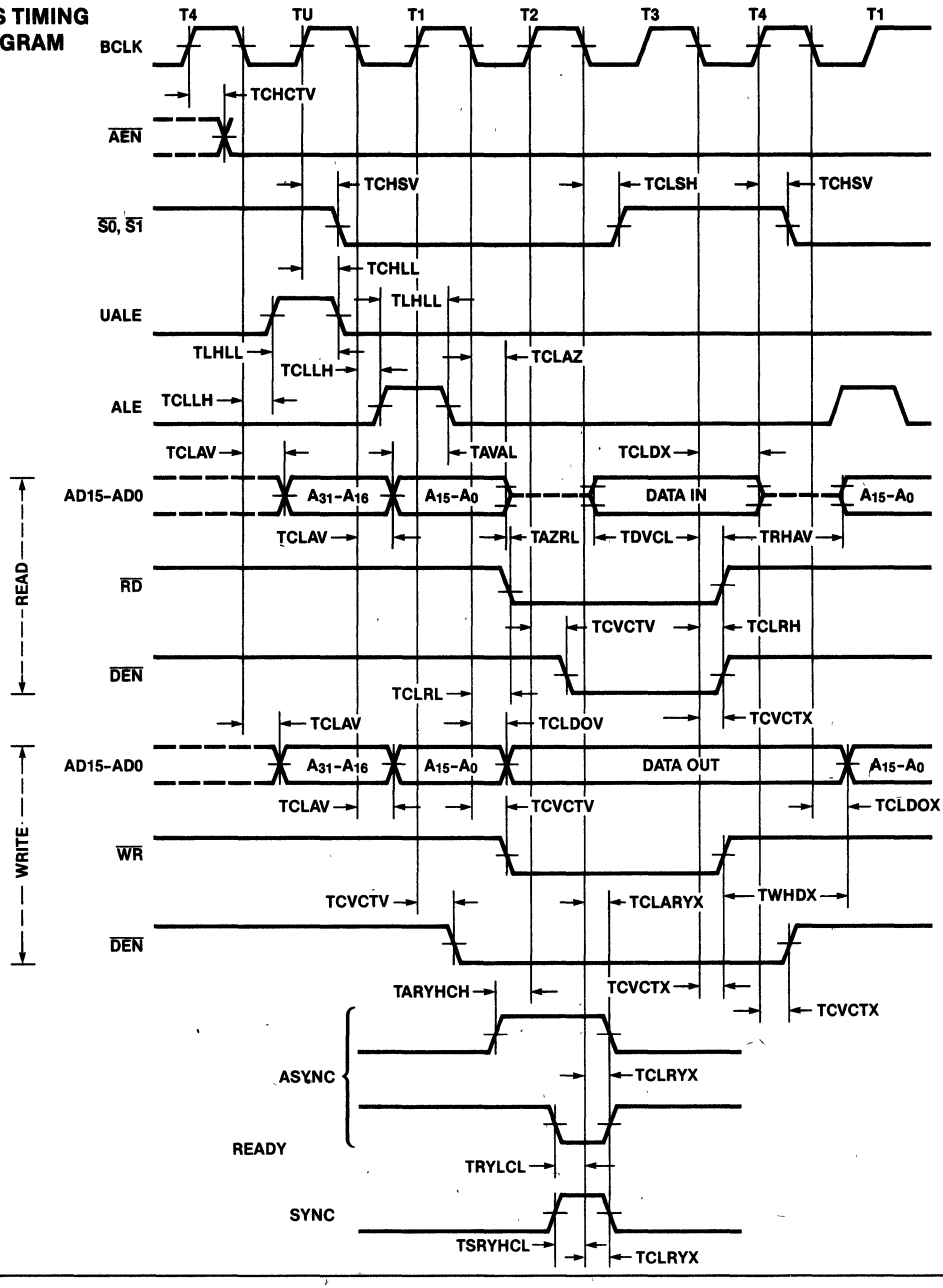
82730 Display Generator Output Timing Response

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$. All timings in nanoseconds.

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCCHDV	Data, Line Count and Attribute and Output Valid Delay from the Delay from the Rising Edge of CCLK		70	ns	$C_L = 100$ pF
TRCHCV	Delay of Outputs CSYNC, VSYNC, HSYNC or RRVV from the Rising Edge of RCLK		70	ns	$C_L = 100$ pF
TCCHCL	CCLK Rising to $\overline{CHOLD}$ Low		75	ns	$C_L = 50$ pF
TRCLCH	RCLK Falling to $\overline{CHOLD}$ High		60	ns	$C_L = 50$ pF

WAVEFORMS

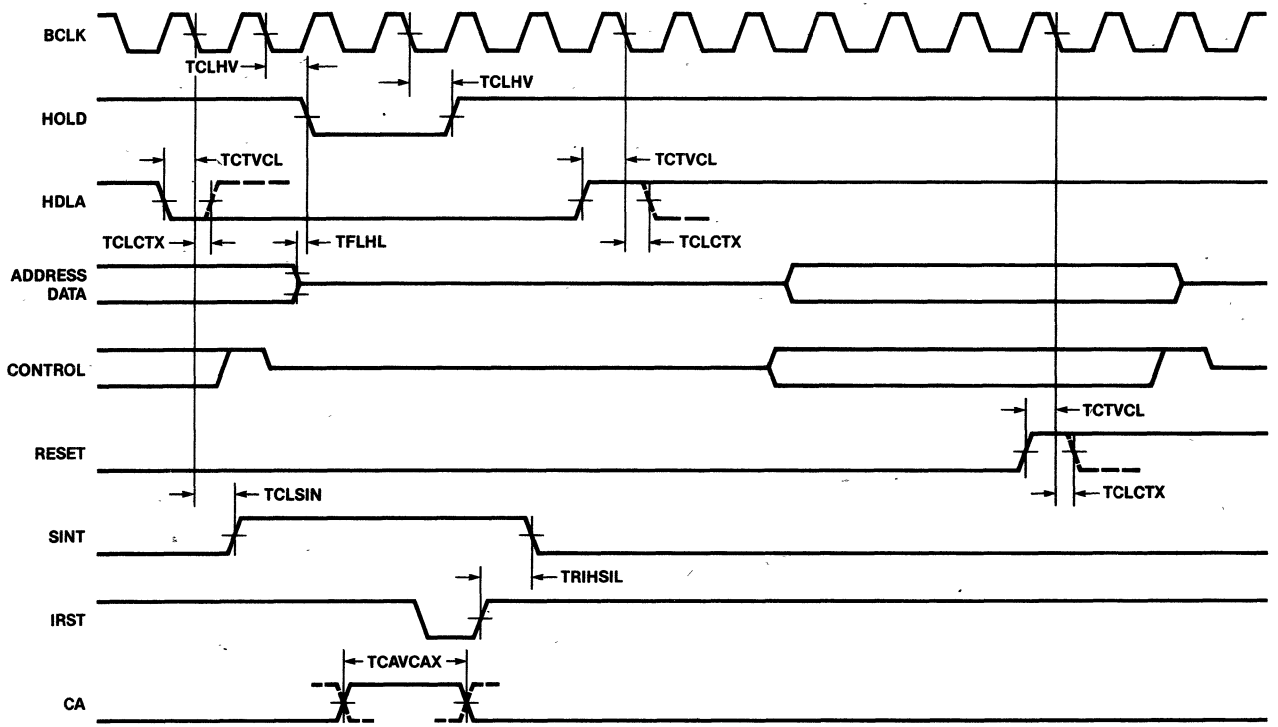
BUS TIMING DIAGRAM



210921-002

WAVEFORMS (Continued)

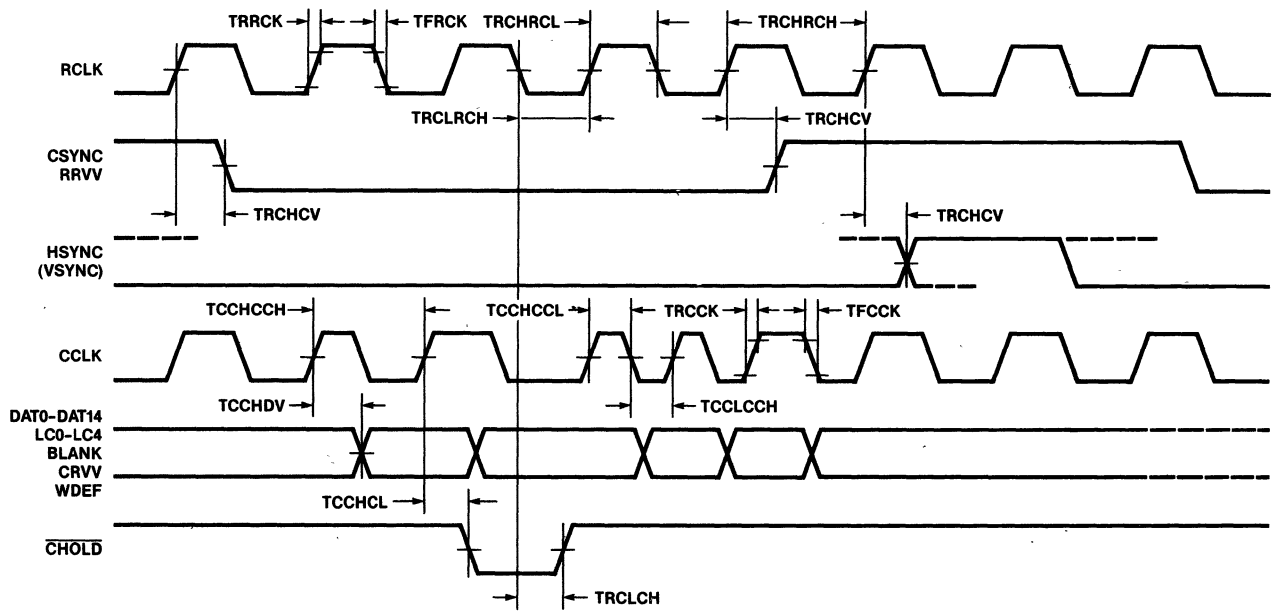
HOLD, RESET, SINT AND CA TIMING



210921-002

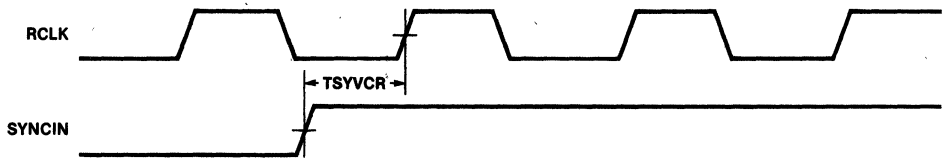
WAVEFORMS (Continued)

DISPLAY GENERATOR INTERFACE TIMING

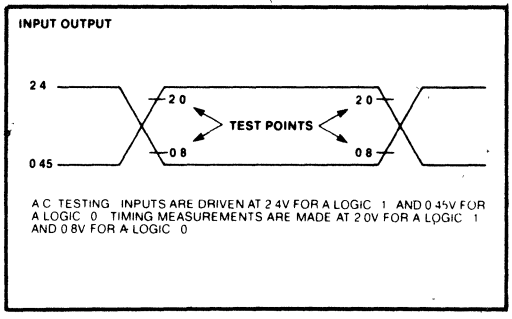


WAVEFORMS (Continued)

SYNCIN TIMING

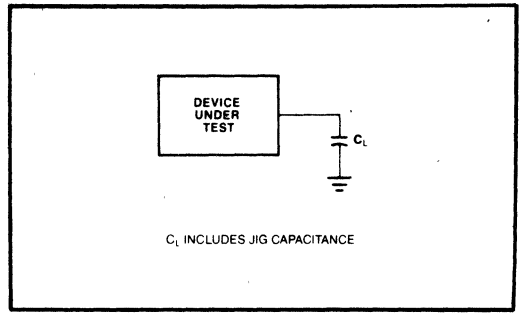


A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

82731
VIDEO INTERFACE CONTROLLER

- Parallel to Serial Data Conversion
- On-Chip Clock Generator
- High Video Dot Rates
  80 MHz—82731-2
  50 MHz—82731
- Character up to 16 Dots Wide
- Proportional Character Spacing
- On-Chip Character Attribute Processing
- Control Functions to Provide Screen Reverse Video, Video Clock, Synchronization and Tab Function
- Single 5V Power Supply
- 40 Pin DIP
- All Inputs and Outputs TTL Compatible Except Video Output which is ECL

The 82731 is a general purpose video interface which generates a serial video signal output from parallel character and attribute information coming from the character generator and the 82730 Text Coprocessor. With a character generator and minimal hardware, the 82731 will comprise a complete video interface system for the 82730 Text Coprocessor and the CRT monitor.

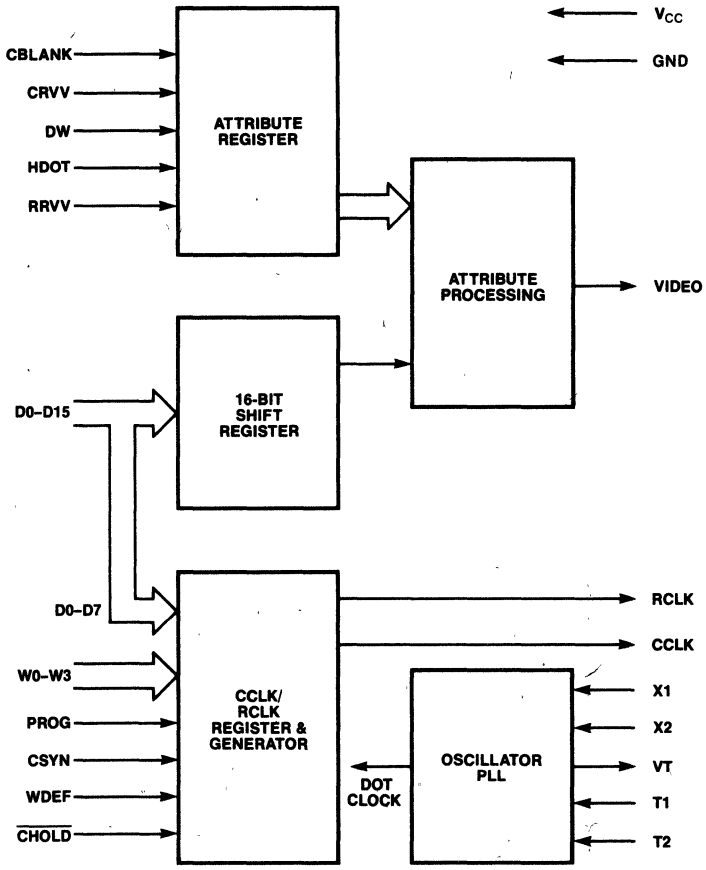


Figure 1. 82731 Block Diagram

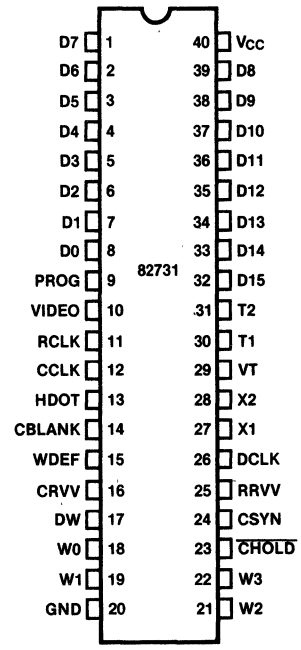


Figure 2. 82731 Pin Configuration

NOVEMBER 1983
Order Number: 210925-003

Table 1. 82731 Pin Description

Symbol	Pin Number	Type	Name and Function
D0-D15	8-1, 39-32	I	Character data parallel inputs.
PROG	9	I	Program control input; used to program default width values of CCLK and RCLK; these are latched into the 82731 via D0-D7 at the rising edge of CCLK (PROG is active high).
VIDEO	10	O	Video output; provides the dot information clocked by the internal dot clock
RCLK	11	O	Reference clock output; used to generate timings for the screen columns for data formatting and video signals. The period of RCLK is programmable from 6 to 21 times the period of the internal dot clock.
CCLK	12	O	Character clock output; used to clock character and attribute information out of the CRT controller. The period of CCLK is programmable from 3 to 18 times the period of the internal dot clock.
HDOT	13	I	Half dot shift input; the video signal at the video output will be delayed by half dot clock for character rounding (active high).
CBLANK	14	I	Character blank attribute input; the video output is blanked (active high).
WDEF	15	I	Width defeat attribute input; the CCLK period is set to a preprogrammed default value (active high).
CRVV	16	I	Character reverse video attribute input; inverts the character data from D0-D15 (active high).
DW	17	I	Double width attribute input; the internal dot clock frequency and the CCLK frequency are divided by two (active high). The RCLK frequency remains unchanged.
W0-W3	18, 19, 21, 22	I	Clock width inputs; they are used for programming the CCLK clock width on a character by character basis.
$\overline{\text{CHOLD}}$	23	I	CCLK inhibit input; this signal inhibits CCLK generation and is used for TAB function (active low).
CSYN	24	I	CCLK synchronization input; CCLK will be synchronized to RCLK and the video output signal is defined by RRVV (active high).
RRVV	25	I	Field reverse video input; the video signal at the video output will be inverted (active high).
DCLK	26	O	Dot clock output; ECL-level signal; must be connected to a 3.3k resistor to ground if used.
X1-X2	27, 28	I	Inputs for fundamental mode crystal; its frequency must be 1/8 of the required dot clock frequency.
VT	29	O	Tuning voltage for PLL-VCO; this output is used to tune the LC-circuit and thus control the oscillator frequency of the internal dot clock.
T1-T2	30, 31	I	LC-circuit inputs for PLL-VCO. T1 can be used to provide the 82731 with an external TTL-level clock at twice the dot clock frequency.
VCC	40	-	+5V power supply
6ND	20	-	Ground (0V)

FUNCTIONAL DESCRIPTION

The Video Interface Controller, 82731, in a typical CRT system shown in Figure 3, interfaces the Text Coprocessor to the CRT video terminal. It receives the parallel data along with the attribute and control information from the Text Coprocessor, processes it into a serial video signal which can be fed to a video CRT terminal. It also generates the basic dot clock (DCLK), character clock (CCLK) and the reference clock (RCLK) signals required by the Text Coprocessor.

CRT terminals requiring very high resolution, extremely stable and absolutely flicker-free picture place special demands on the dot rate generator. In such applications dot rates up to 80 MHz are necessary. This allows 12.5 ns per dot (pixel) for converting data, attribute and control information into serial form for the video terminal.

The functionality of the 82731 is largely determined by the complexity and the demands of the CRT controller it supports. Figure 1 shows the block diagram of the Video Interface Controller. The dot clock is generated by voltage controlled LC circuit connected at T1 and T2. Another clock is generated which is crystal controlled and has frequency 1/8 of the dot clock. This is used to stabilize the dot clock using an on-chip phase locked loop (PLL). This two-oscillator concept enables the use of low cost, fundamental mode crystals even for generating frequencies up to 80 MHz.

The 16 bit shift register receives parallel inputs from pins D0-D15. This allows a maximum character width of 16 dots. The minimum width is 3 dots. The character width is programmable through pins W0-W3 for proportional character spacing. This also determines the character clock (CCLK) frequency. Programming of the default character width and the reference clock (RCLK) is done through inputs D0-D7 and PROG. Signal WDEF can be used to switch between the default character width and the one specified dynamically through the lines W0-W3. When using variable character width, for example, in generating tables on the screen, it is essential that every entry in a column starts at the same dot distance (and not the character distance) from the start of line. The 82731 supports this requirement by providing a tab function using CSYN and $\overline{\text{CHOLD}}$ signals to synchronize with the reference clock (RCLK).

It is possible to shift any scan line of any character by half a dot using the HDOT signal. This feature, known as character rounding, further enhances the quality of high resolution character displays. Other features, like character blinking, reverse video etc., which improve the readability of text on screen are directly supported by the 82731 using signals CRVV and RRVV from the Text Coprocessor, processing them and affecting the final video signal to show the characters with the desired attributes.

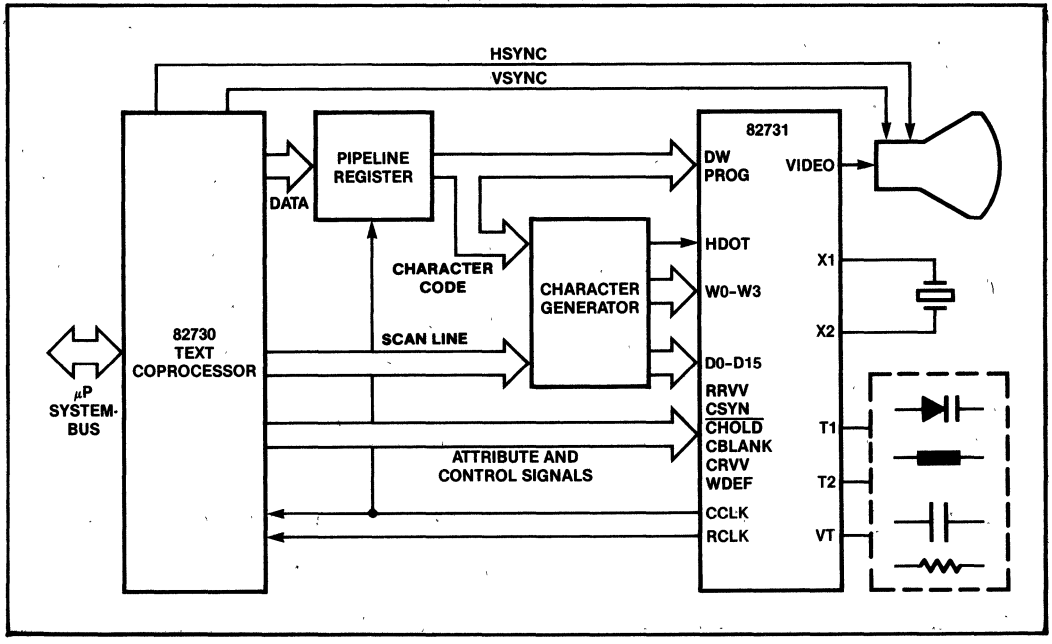


Figure 3. CRT System Block Diagram

Clock Generation

The most fundamental clock required to run the CRT display is the dot clock which provides the reference for the dot data to be shifted serially to the CRT. In addition, it is the basis for the character clock (CCLK) and the reference clock (RCLK) required by the 82730.

Dot Clock

The dot clock is derived from an on-chip oscillator which runs at twice the normal dot clock (DCLK) frequency. A voltage-controlled LC circuit is connected to the T1, T2 pins, to create a voltage-controlled oscillator (VCO). The 82731 compares the phase of this oscillator with another on-chip oscillator controlled by a crystal attached to the X1, X2 pins. This oscillator runs at 1/8 the normal DCLK frequency to allow using inexpensive low-frequency crystals. The on-chip PLL circuit produces an error voltage via the VT pin which locks the VCO to the 16th harmonic of the crystal frequency (see Figure 4a).

Alternatively the 82731 can be supplied with an external TTL-level clock at twice the normal DCLK frequency via the T1 pin, as shown in Figure 4b.

When the Double Width (DW) input is active, the DCLK frequency is divided to 1/2 its normal value. This affects the DCLK, CCLK, and VIDEO outputs, but not RCLK.

Designing the Oscillator Circuit

The whole external oscillator circuit consists of three parts:
—the crystal circuit,
—the voltage controlled LC-circuit, and
—the loop filter for the PLL.

Figure 5a shows the general crystal circuit. The crystal must be a fundamental mode series resonant type with a resonant frequency of 1/8 of the desired dot clock frequency. The capacitor C_X is necessary if a fine adjustment of the dot clock rate must be

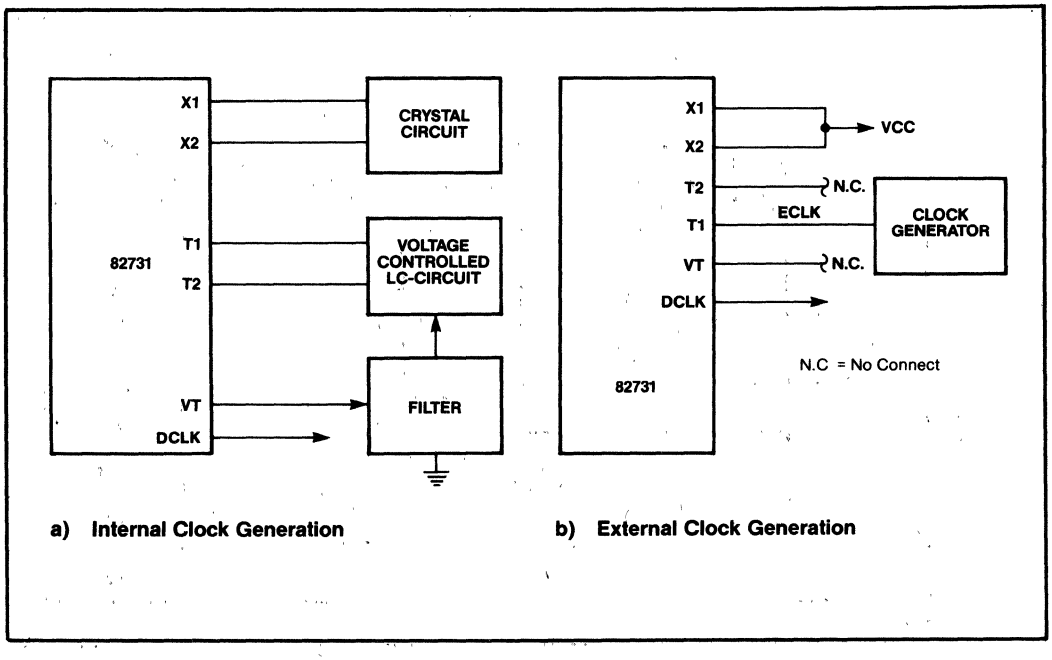
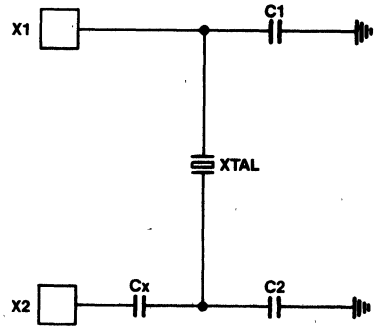


Figure 4. Clock Generation

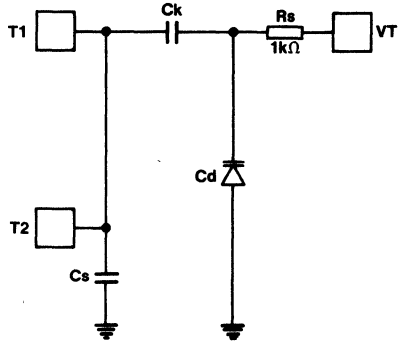


a) Crystal Circuit

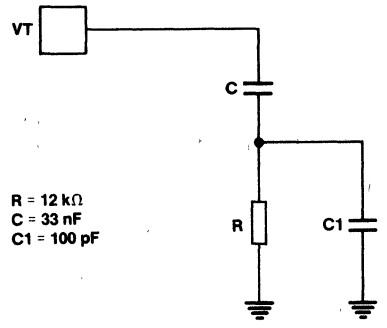
Cx (pF)	FDCLK (MHz)
2.2	64.053
6.8	64.016
15	63.987
33	63.966

(nominal crystal frequency 8 MHz)

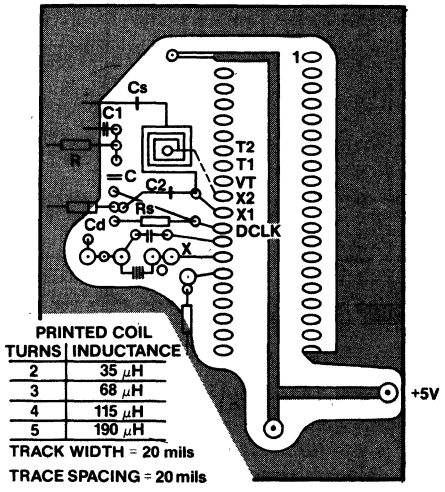
b) Example of the influence of Cx on the dot clock frequency



c) VCO circuit



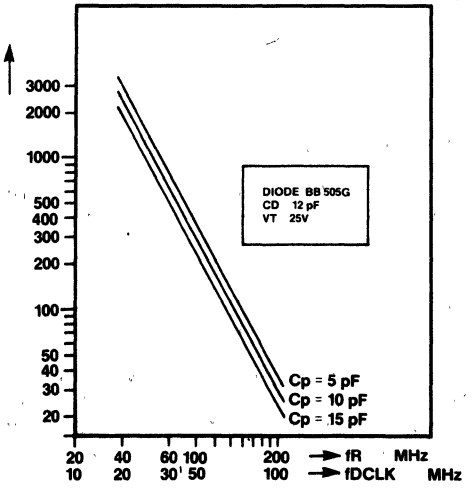
d) PLL - loop filter



PRINTED COIL TURNS	INDUCTANCE
2	35 μH
3	68 μH
4	115 μH
5	190 μH

TRACK WIDTH = 20 mils
TRACE SPACING = 20 mils

e) Example layout for printed circuit



f) L/f-diagram

Figure 5. Designing the Oscillator Circuit

made. Figure 5b shows an example how the dot clock frequency can vary with different values of Cx. The capacitors C1 and C2 may be necessary to suppress overtone oscillations if the crystal frequency is below 6 MHz. The exact values depend on the crystal used and must be determined empirically. The recommended ranges are 0 to 10 pF for C1 and 0 to 100 pF for C2.

The voltage controlled LC-circuit is shown in Figure 5c. The effective resonant circuit consists of the inductance L, the capacitance Cd of the varactor diode and the parasitic capacitance Cp. Its resonant frequency is

$$fR = \frac{1}{2\pi\sqrt{L \cdot (Cd + Cp)}}$$

where fR must be 2 × fDCLK. The value of Cp depends on many factors (e.g. layout, single/multilayer board . . .), thus it changes from application to application. However a value of 5 to 15 pF seems to be a good approximation.

The value of DC (varactor diode) should be determined at a control voltage of 2.5 V to get the lock-in-range as wide as possible. The variation of VT ranges from 1 V to VCC-1 which results in a minimum frequency shift of about 6-8% in relation to the center frequency at 2.5 V.

The value of the inductance L must be determined in such a way that the resulting center frequency lies as near as possible to the needed frequency fR = 2 × fDCLK to guarantee a stable dot clock under all operation conditions. Figure 5f shows a diagram that will help to find the needed inductance L. It is based on the use of a varactor diode (Siemens BB 505G) that has a capacitance of 12 pF at a control voltage of 2.5 V. The use of other diodes will of course lead to other diagrams.

At dot clock frequencies higher than 50 MHz the needed inductance becomes lower than 100 μH. In these cases it is better to integrate the inductance into the board layout. Figure 5e shows a possible layout for the external oscillator circuit and approximate (measured) values of the inductance of the printed coil (trace width and trace spacing 20 mils).

The loop filter converts the current pulses at the VT pin into the control voltage VT for the VCO. it is an essential part of the PLL and affects the lock-in-range and stability of the PLL. A second order filter that was found to work well under all operation conditions and over the full frequency range is shown in Figure 5d.

Reference Clock (RCLK)

RCLK is the reference clock output used to generate video timing and to define screen columns for data formatting and tabulor locations. In addition, it is used to clock the field attribute signals into the 82731. The period of RCLK is programmable from 6 to 21 times the period of the dot clock, i.e. the RCLK hightime is 3 dot clock periods and the RCLK lowtime is programmable from 3 to 18 dot clock periods. It is programmed via D4-D7 at the rising edge of CCLK, when PROG is active (see Table 1 and Figure 6).

The RCLK clock width should be programmed only once after a system reset.

Table 1. Programming RCLK

D7	D6	D5	D4	PROG	RCLK Period (dot clocks)
0	0	0	0	1	16
0	0	0	1	1	17
0	0	1	0	1	18
0	0	1	1	1	19
0	1	0	0	1	20
0	1	0	1	1	21
0	1	1	0	1	6
0	1	1	1	1	7
1	0	0	0	1	8
1	0	0	1	1	9
1	0	1	0	1	10
1	0	1	1	1	11
1	1	0	0	1	12
1	1	0	1	1	13
1	1	1	0	1	14
1	1	1	1	1	15

Character Clock (CCLK)

CCLK is the fundamental character clock output used to clock character and attribute information from the 87730.

It is a rising edge triggered clock and inside the active character field its period is programmable from 3 to 18 times the period of the dot clock, i.e. the

CCLK hightime is 2 dot clock periods and the CCLK low time is programmable from 1 to 16 dot clock periods.

When CSYN is active (normally outside the active character field) CCLK is forced to match RCLK. In this case the CCLK high time is 3 dot clock periods instead of 2.

In order to support proportional spacing, the period of CCLK can be reprogrammed at the beginning of each CCLK cycle (i.e. at the beginning of each character) if PROG is inactive.

Programming the character width is done via the clock width inputs W0-W3 according to Table 2. The W0-W3 input data is clocked into the 82731 at the rising edge of CCLK and defines the width of the currently displayed character (see Figure 7).

If the width defeat attribute (WDEF) is active, the period of CCLK will be set to the programmed default value ignoring the clock width inputs W0-W3. This value is programmable from 3 to 18 times the period of the dot clock via the D0-D3 inputs, when the PROG input is active (see Figure 6).

The default CCLK width should be programmed only once after a system reset.

The CCLK clock period will be doubled if the double width attribute (DW) is asserted at the rising edge of CCLK.

NOTE

If width of CCLK is programmed to 17 or 18, zeros are shifted out from the internal shift register after the 16 data bits and displayed according to the attribute signals.

Clock Initialization Sequence (PROG)

After power on the width of RCLK is a random value between 6 and 21 and the width of CCLK is a random value between 3 and 18.

The 82731 should be initialized in the following way:

- Activate the CSYN signal. CCLK is forced to match RCLK, which has a minimum clock width of 6 dot clock periods.
- Apply the clock width informations to D0-D3 and D4-D7 according to tables.
- Activate the PROG signal. The default width of CCLK and the width of RCLK are programmed at the next rising edge of CCLK (see Figure 6).
- Remove the PROG signal.

CSYN can be removed at the beginning of the next active data field.

Table 2. Programming CCLK

PROG = 1	D3	D2	D1	D0	CCLK Period (dot clocks)
PROG = 0	W3	W2	W1	W0	
	0	0	0	0	16
	0	0	0	1	17
	0	0	1	0	18
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

Note:
PROG = 1: Programming the CCLK default clock width during the initialization phase via D0-D3 at the rising edge of CCLK.
PROG = 0: Programming the clock width of the current CCLK cycle via W0-W3 at the rising edge of CCLK.

Character Data Signals

The character data signals are normally provided by the character ROM and clocked into the 82731 at the rising edge of CCLK.

The character data signals consist of:

- the character data lines (D0-D15),
- the character width information (W0-W3), and
- the half dot shift signal (HDOT).

Dot Data (D0-D15)

The dot data signals will be clocked into the 82731 via the D0-D15 inputs at the rising edge of CCLK. The actual character width is defined by the W0-W3 inputs or the default width information previously programmed. The dot data will be displayed dependent on the control signals and on the corresponding attribute information. The data bits are serially shifted out at the video output starting with D0.

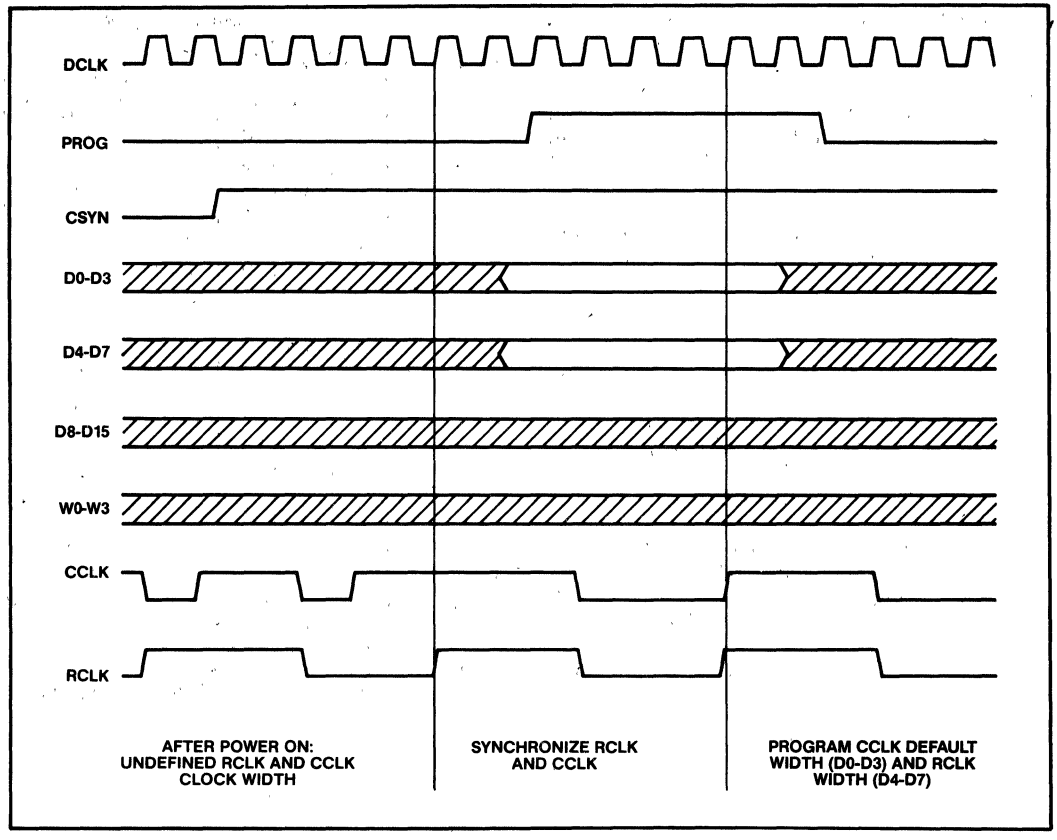


Figure 6. Clock Initialization

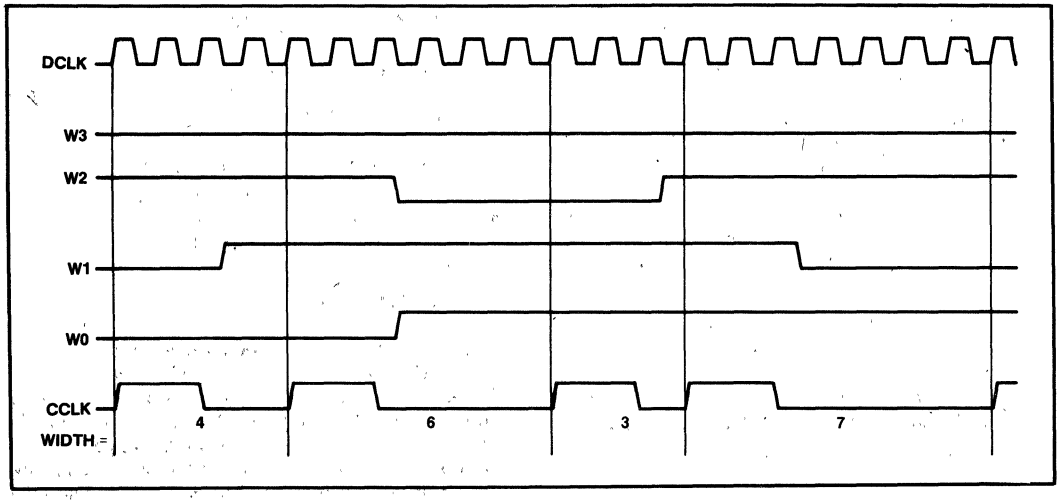


Figure 7. Action of Clock Width Inputs W0-W3 on CCLK

If CCLK width is greater than 16, zeros are shifted out for the rest of the dot clocks and displayed according to the attribute signals.

Character Width (W0-W3)

The W0-W3 inputs are clocked into the 82731 at the rising edge of CCLK and determine the width of the currently displayed character.

Half Dot Shift (HDOT)

The half dot shift signal is clocked into the 82731 at the rising edge of CCLK. When the half dot shift signal is active (high), the output of the video data will be delayed by half a dot time. The first dot of the character dot line is transmitted for one and a half dot clock period while the last dot of this character dot line is displayed for half a dot clock period. The remaining character dots are transmitted for one dot clock period and thus are shifted by half a dot.

The HDOT signal is not a character attribute signal, because it can change from scan line to scan line of a character. Thus it is reasonable to generate it from the character ROM, together with the dot data and the width information.

Character Attribute Signals

These signals are clocked into the 82731 at the rising edge of CCLK. Thus they are valid for the next character only.

The character attribute signals consist of:

- character blanking — CBLANK,
- character reverse video — CRVV,
- double width — DW, and
- width defeat — WDEF.

Outside the active character field (which is defined by the CSYN signal) all character attribute signals are ignored.

Character Blanking (CBLANK)

If CBLANK is active (high), the blank attribute will produce the effect of blanking the display of the character. When the CBLANK attribute is active, the corresponding dot data information D0-D15 will be as if all zeros were forced at the inputs. The video output can be inverted to all ones by simultaneously activating the CRVV attribute. Independent of these character oriented operations the video output signal is also affected by the RRVV field attribute signal.

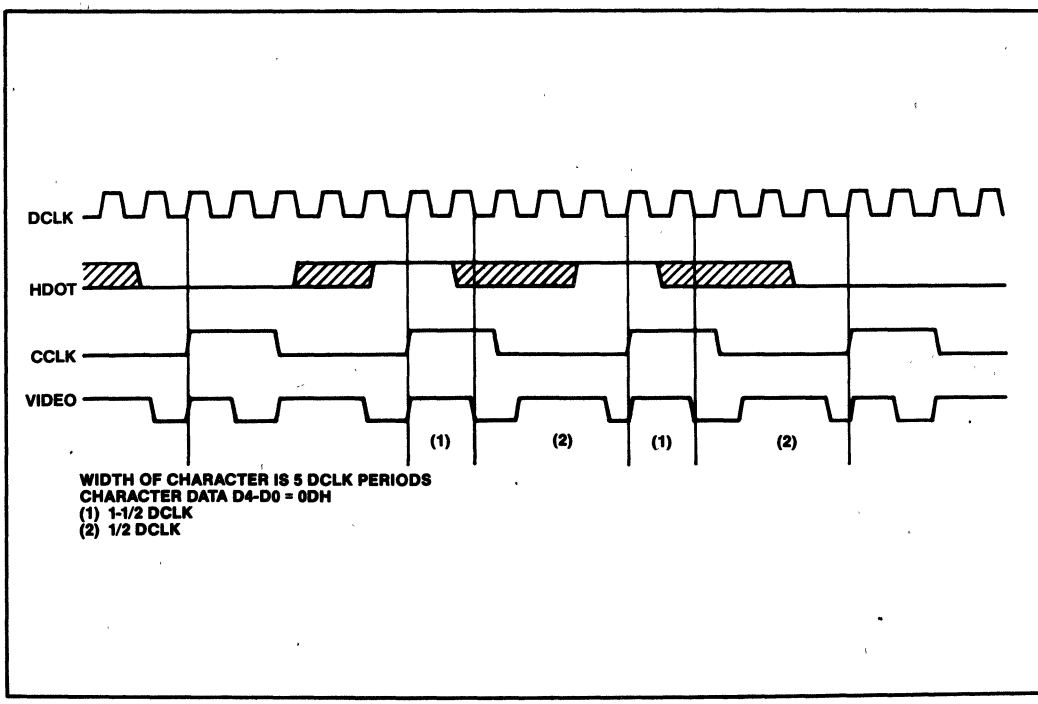


Figure 8. Function of HDOT on VIDEO

Character Reverse Video (CRVV)

CRVV is an active high signal. In the character field, the CRVV attribute will produce the effect of reversing the polarity of the display during the transmission of the current character. CRVV is also effective together with the CBLANK attribute (see CBLANK description) and the RRVV signal. Outside the character field, the CRVV attribute is ignored.

Although the CBLANK signal is normally a character attribute, it may change from dot line to dot line of a character. Thus one or more underlines or cursors can be generated by the CRT controller activating CBLANK and CRVV.

Double Width (DW)

The dot clock frequency and the CCLK frequency will be halved when the double width attribute is active (high), producing characters that are twice as wide. The period of RCLK is not changed (see Figure 9).

Width Defeat (WDEF)

The WDEF attribute signal is clocked into the SAB 82731 at the rising edge of CCLK. When the width defeat attribute is active (high), the width of CCLK will be set to a default width value previously programmed (see figure 10).

Field Attribute Signals

The field attribute signals are clocked into the 82731 with the rising edge of RCLK. Thus the attributes are valid for a specific part of the screen independent of how many characters are displayed within this part.

The 82731 supports two field attributes:

- field reverse video RRVV, and
- clock synchronization CSYN.

Row Reverse Video (RRVV)

RRVV control signal is clocked into the 82731 at the rising edge of RCLK. It immediately affects the display by the polarity of the video eoutput in both the character field and the border of the display. It is an active high signal.

Clock Synchronization (CSYN)

CSYN is a field attribute signal, because it defines the active character field in addition to its function of synchronizing CCLK and RCLK.

CSYN must be inactive (low) during the display of characters. At the first rising edge of RCLK after CSYN is deactivated (low), character data is latched into the 82731, beginning the display of the active character field (see Figure 11). At the next rising edge of RCLK after CSYN is activated (i.e. at the end of the character field), the video output is forced to zero or, if the RRVV control signal is active, to a high level. The currently transmitted character will be truncated at this location. At the same time, CCLK will be forced to match RCLK starting with the next rising edge of RCLK (see Figure 11). While CSYN is active all character attribute and data signals are ignored and only the field reverse video signal (RRVV) affects the video output.

Before the deactivation of CSYN, the data and attribute pipeline has to be filled by the CRT controller with the information of the first character.

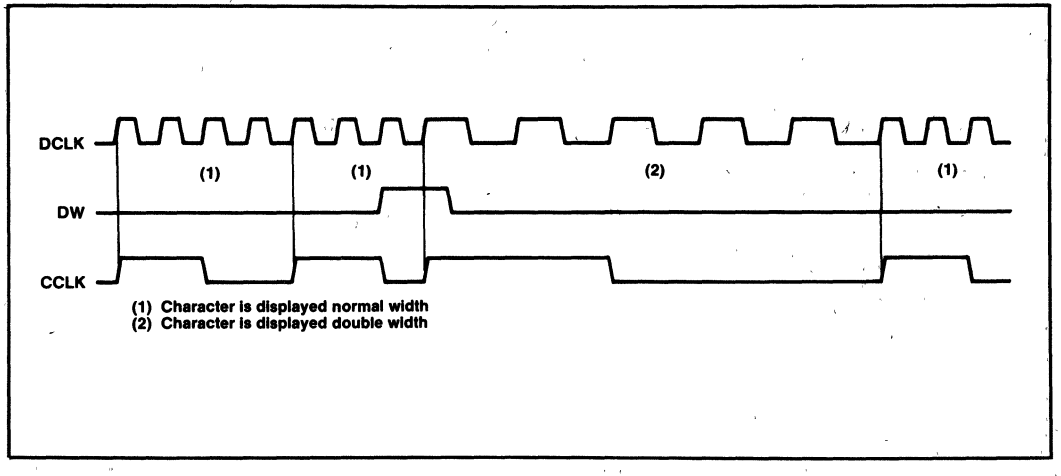


Figure 9. Function of DW on DCLK and CCLK

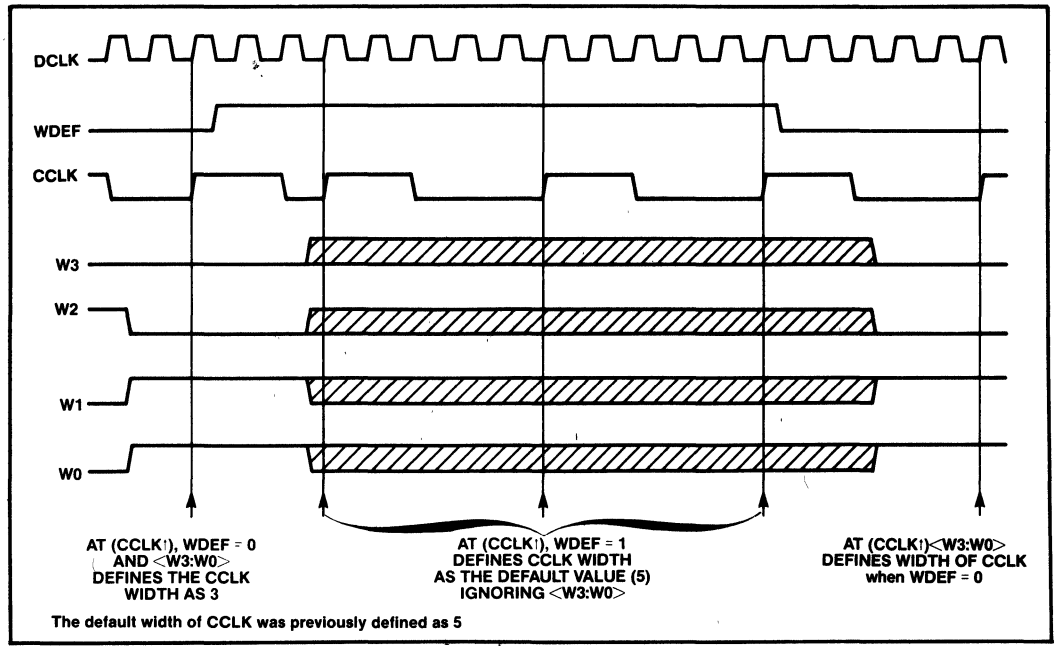


Figure 10. Function of WDEF

Tabulator Function

The 82731 supports tabulator functions by providing the CHOLD (character clock inhibit) input.

CCLK Inhibit ($\overline{\text{CHOLD}}$)

When the $\overline{\text{CHOLD}}$ signal is activated (low) it inhibits CCLK and thus freezes the information pipeline between CRT-controller and 82731 until the next tabulator location is reached. $\overline{\text{CHOLD}}$ Has to be activated simultaneously with the display of the TAB-character. If the TAB-character doesn't consist of all zeros, it must be blanked by activating CBLANK.

The width of the TAB-character can be determined by W0-W3 or by activating WDEF.

The $\overline{\text{CHOLD}}$ signal is provided by the 82730 and it is assumed to be triggered with the rising edge of CCLK (Figure 12). With the same edge of CCLK, the TAB-character will be latched into the 82731. Thus the TAB-character will be displayed completely and the CCLK will be inhibited until reaching the specified tabulator location, which is defined by $\overline{\text{CHOLD}}$ inactive (high) at the rising edge of RCLK.

In the timing diagrams it is assumed that $\overline{\text{CHOLD}}$ is deactivated by the falling edge of RCLK. Figure 12 shows the normal case where the display of the TAB-character is finished before deactivation of $\overline{\text{CHOLD}}$. The gap between the TAB- and the following character is normally blanked. In this scheme the TAB-character will be handled by the 82731 like each other character (attributes operate normally).

In case of $\overline{\text{CHOLD}}$ active width less than the TAB-character width the TAB-character will be also displayed completely. However, we have to distinguish three different cases:

1) TAB-character is terminated before reaching TAB-location. The next character will be displayed as described before. In the gap the video output is normally blanked.
2) TAB-character is finished exactly at the TAB-location. The next character will be displayed immediately without delay.
3) TAB-character is not terminated when reaching the TAB-location (see Figure 13). The following character will be displayed subsequently after the display of the TAB-character (i.e. the start of the following character is not at the TAB-location).

If the $\overline{\text{CHOLD}}$ signal is not deactivated the video output will be continuously blanked. In the gap between the end of the TAB-character and the TAB-location all character attribute signals will have no effect on the video output signal. If the RRVV control signal is active the video output signal is inverted.

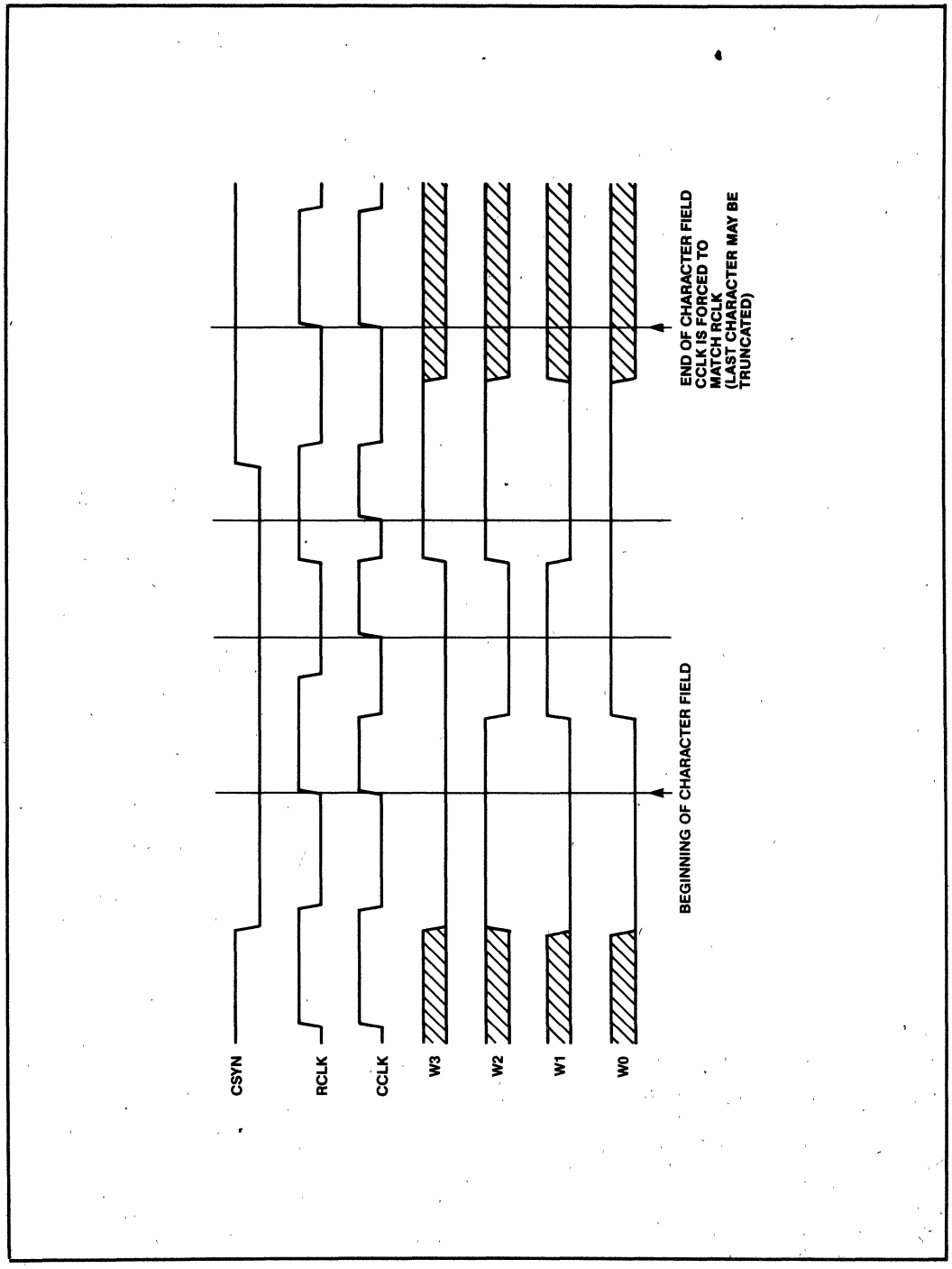
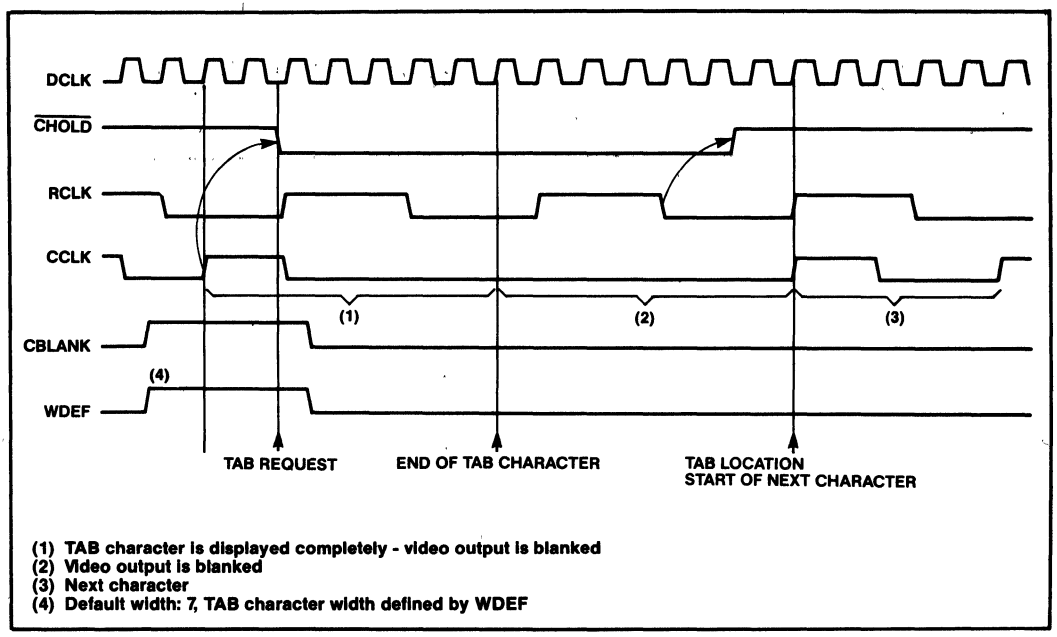
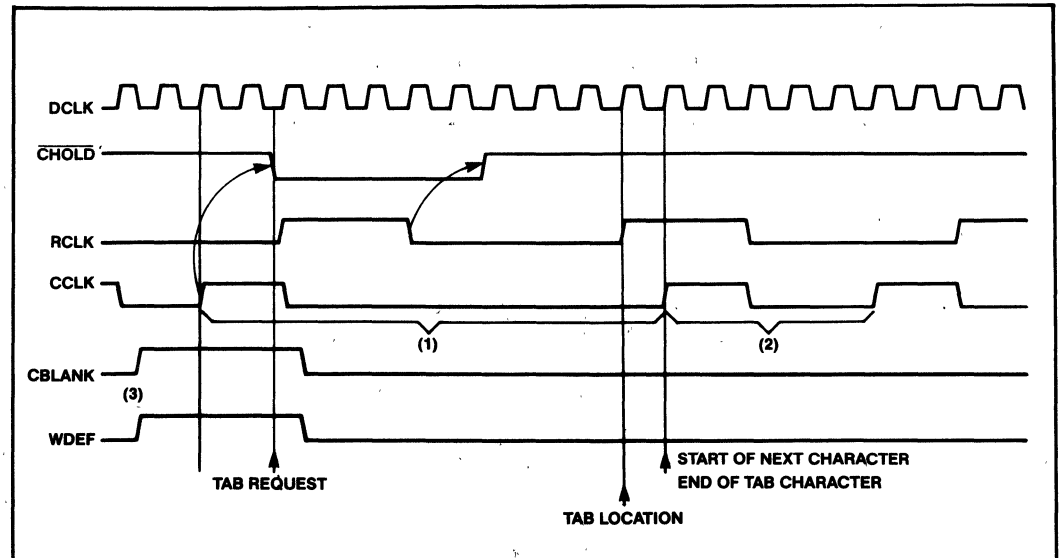


Figure 11. Function of CSYN



(1) TAB character is displayed completely - video output is blanked
(2) Video output is blanked
(3) Next character
(4) Default width: 7, TAB character width defined by WDEF

Figure 12. Function of CHOLD (Normal Case)



NOTE:
(1) TAB character is displayed completely - video output is blanked
(2) Next character is displayed (not on TAB location)
(4) Default width = 11.

Figure 13. Function of CHOLD with CHOLD Width Less than Character Width (Case 3)

Video Output

The video output provides an ECL-oriented signal (see Figure 14) and is matched to drive a 50 Ohm coax cable (see Figure 15). In case of external attribute processing the external logic can be ECL- or STTL-compatible.

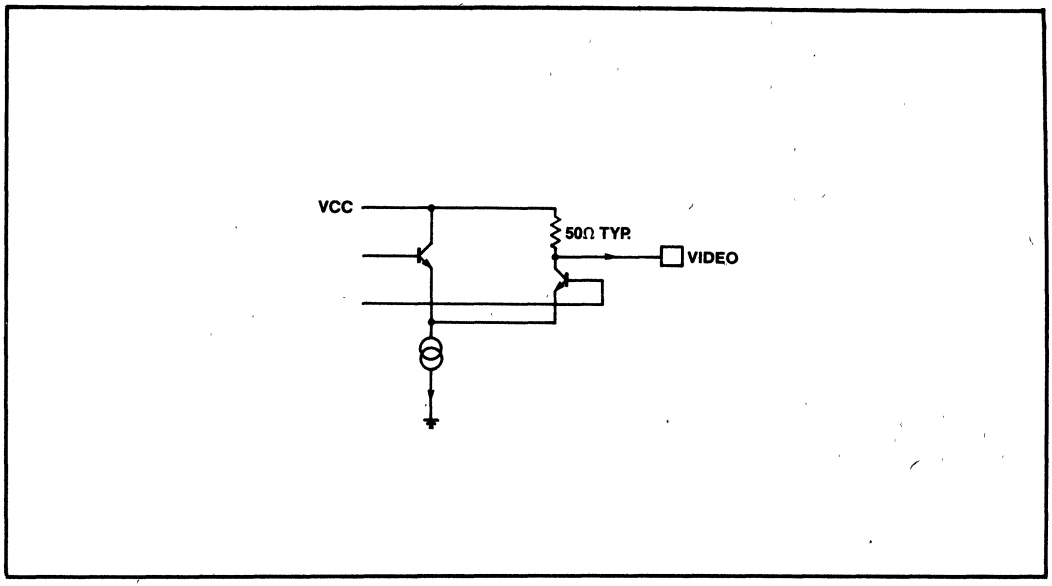


Figure 14. Video Output Stage

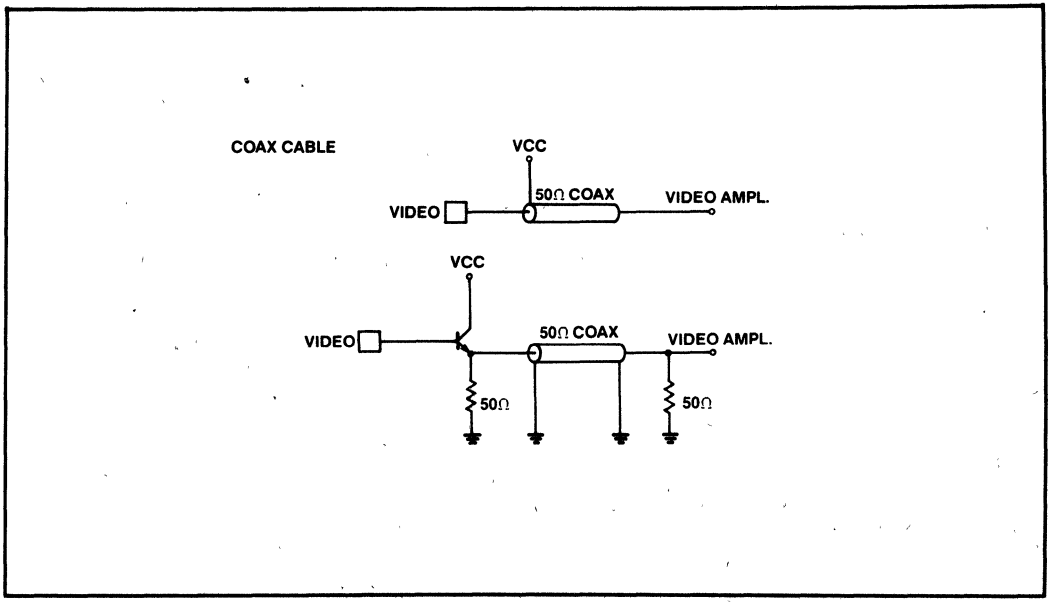


Figure 15. A Video Output Load

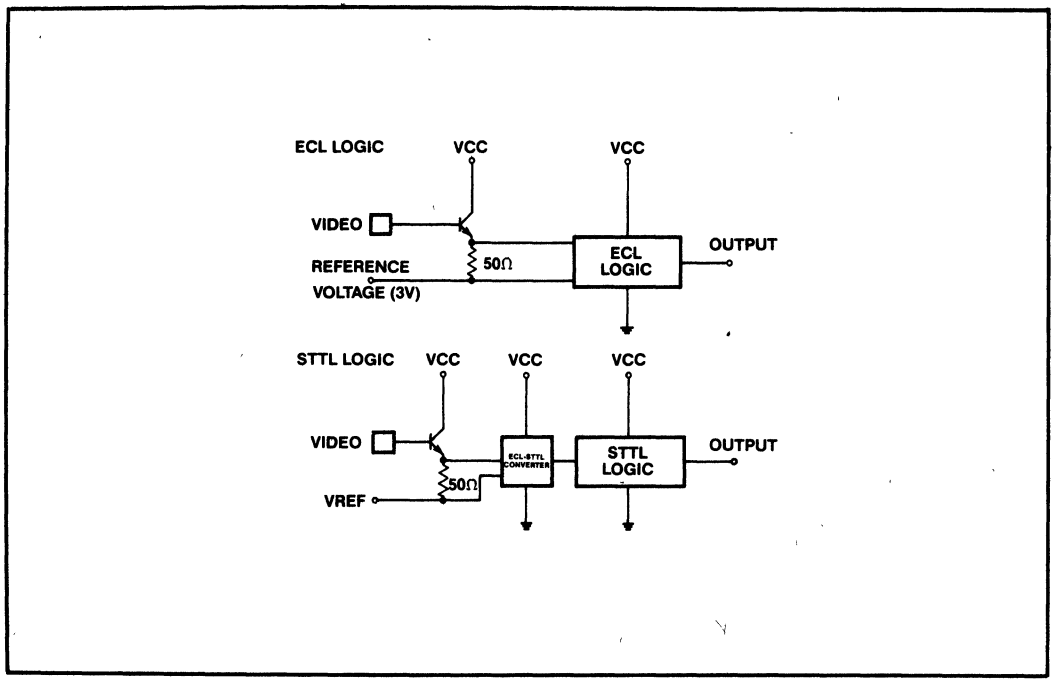


Figure 15.B Proposed Converter for Video Output to TTL Level Output

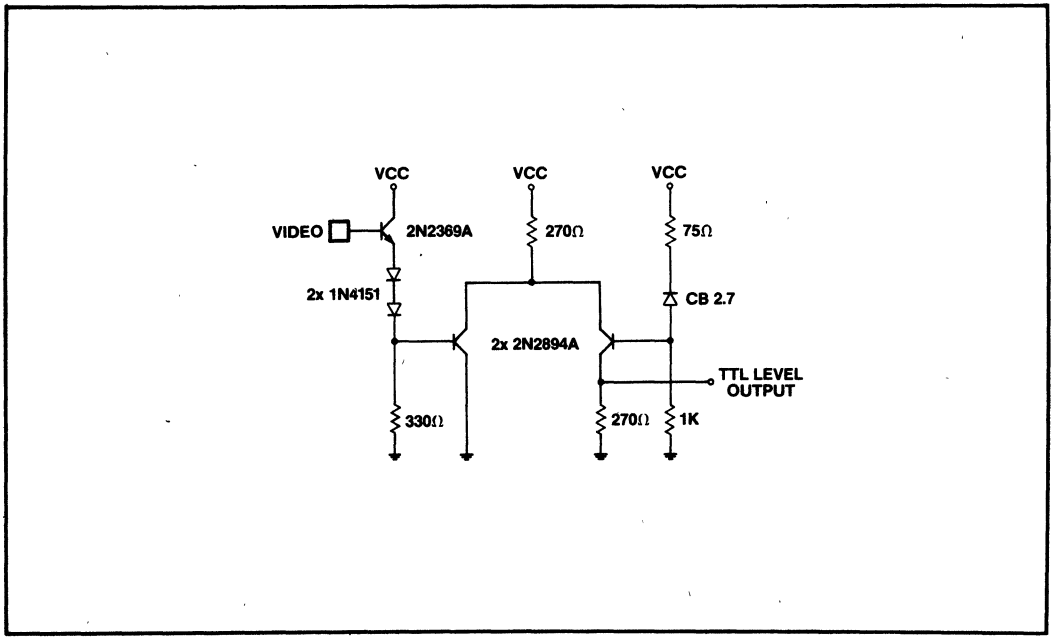


Figure 16. TTL-Level-Output Test Load

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +125°C
All Output and Supply Voltages ..... −0.5V to +6V
All Input Voltages ..... −0.6V to +5.5V
Power Dissipation ..... 1.75 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($V_{CC} = 5V \pm 10\%$, $T_A = 0°C$ to $70°C$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_C	Input Clamp Voltage		−1	V	$I_C = -5$ mA
I_F	Forward Input Current		−0.7	mA	$V_F = 0.5V$
I_R	Reverse Input Current		50	μA	$V_R = V_{CC}$
V_{OL}	Output Low Voltage				
	CCLK		0.5	V	$I_{OL} = 8$ mA
	RCLK		0.5	V	$I_{OL} = 4$ mA
	VIDEO	$V_{CC} - 1.2V$	$V_{CC} - 0.6V$	—	$I_{OL} = 0$
V_{OH}	Output High Voltage				
	CCLK, RCLK	2.4		V	$I_{OH} = -400$ μA
	VIDEO	$V_{CC} - 0.2V$	V_{CC}	—	$I_{OH} = 0$
V_{IL}	Input Low Voltage		0.8	V	
V_{IH}	Input High Voltage	2.0		V	
I_{CC}	Power Supply Current		300	mA	
Z_O	Output Impedance VIDEO	40	70	Ω	
C_{IN}	Input Capacitance		15	pF	$f_C = 1$ MHz

210925-003

A.C. CHARACTERISTICS

T_A = 0 to 70°C; V_{CC} = 5V ± 10%. All timings measured at 1.5V unless otherwise noted.

Symbol	Parameter	Limit Values				Unit	Test Conditions
		82731		82731-2			
		Min.	Max.	Min.	Max.		
T_{DHDH}	DCLK cycle period	20	125	12.5	125	ns	-
T_{CHCH}	CCLK cycle period	3	18	3	18	T_{DHDH}	Fig. 17
T_{CLCH}	CCLK low time	T_{DHDH} − 10	16 T_{DHDH}	T_{DHDH} − 10	16 T_{DHDH}	ns	
T_{CHCL}	CCLK high time	2 T_{DHDH} − 5	-	2 T_{DHDH} − 5	-		
T_{RHRH}	RCLK cycle period	6	21	6	21	T_{DHDH}	
T_{RLRH}	RCLK low time	3 T_{DHDH} − 10	18 T_{DHDH}	3 T_{DHDH} − 10	18 T_{DHDH}	ns	
T_{RHRL}	RCLK high time	3 T_{DHDH} − 5	-	3 T_{DHDH} − 5	-		
T_{DRCH}	Data and attribute input set up time	25		20			-
T_{CHDX}	Data and attribute input hold time	0		0			
T_{HLTE}	$\overline{CHOLD}$ active before end of TAB-character	25		20			
T_{HLHH}	$\overline{CHOLD}$ pulse width						
T_{HHRH}	$\overline{CHOLD}$ inactive set up before rising edge of RCLK						
T_{HLRH}	$\overline{CHOLD}$ inactive hold time after rising edge of RCLK	0		0			
T_{CHVV}	Video output valid after rising edge of CCLK	-	10	-	6	ns	Video output measured at V_{CC} − 0.4V
T_{OLOH}	TTL-output rise time				10		Fig. 17
T_{OHOL}	TTL-output fall time						
T_{VLVH}	Video output rise time		5		3		Fig. 18
T_{VHVL}	Video output fall time						

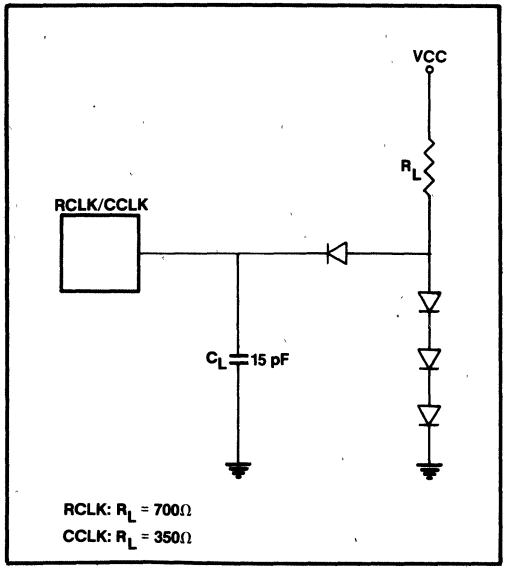


Figure 17. TTL-Level-Output Test Load

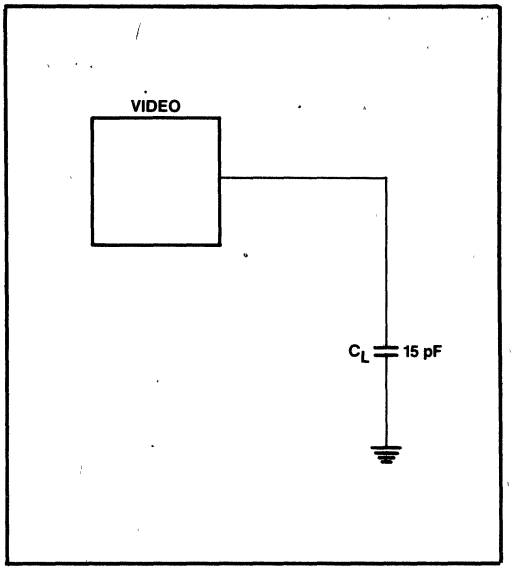


Figure 18. ECL-Level-Output Test

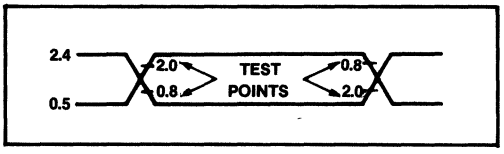


Figure 19. TTL-Level-Output Load Circuit

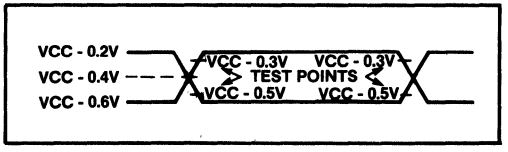


Figure 20. ECL-Level-Output Load Circuit

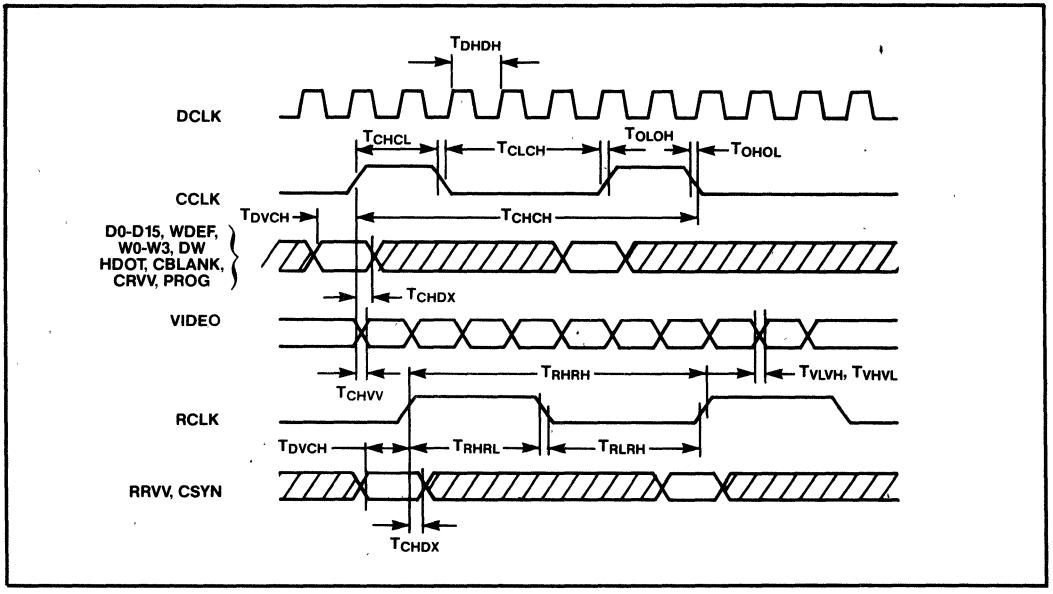


Figure 21. Basic Timing

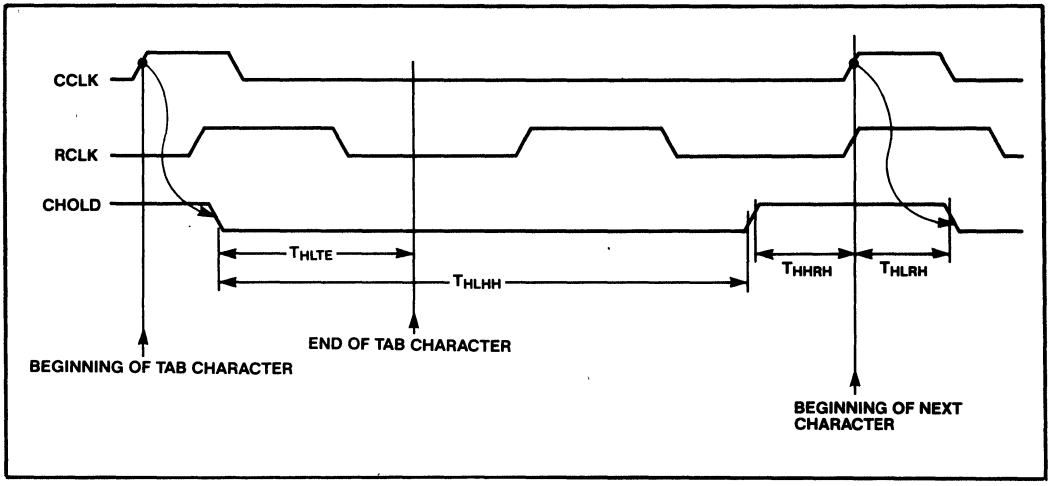


Figure 22. Timing on CHOLD

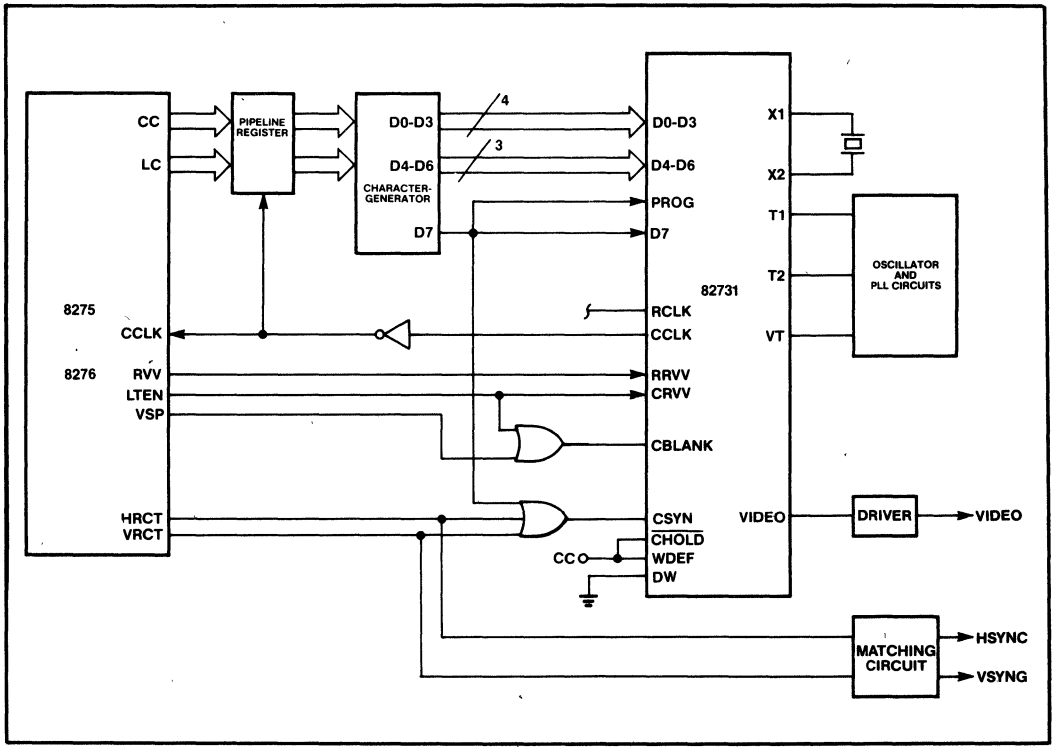


Figure 23. Example Interface to 8275

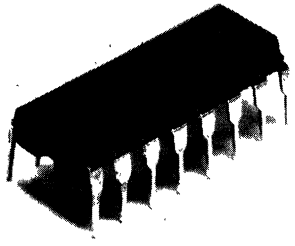
Packaging 9

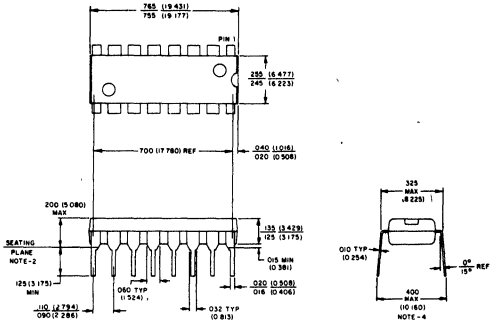
NOTES:

1 All packages drawings not to scale
2 All packages seating plane defined by 0415 to .0430 PCB holes.
3 Type P packages only Package length does not include end flash burr Burr is .005 nominal, can be 010 max at one end
4 All package drawings end view dimensions are to outside of leads.

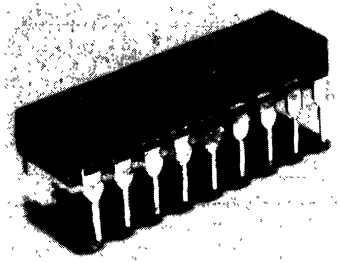
PLASTIC DUAL IN-LINE PACKAGE TYPE P

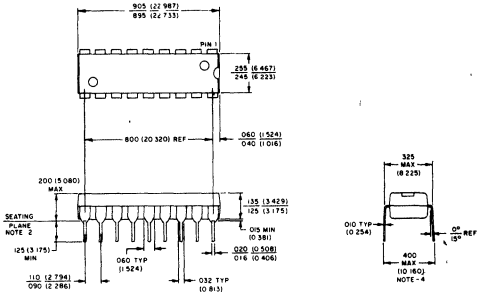
16-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



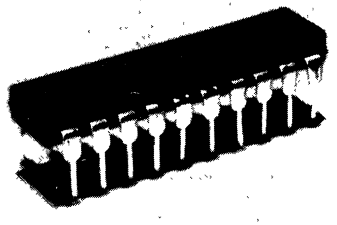


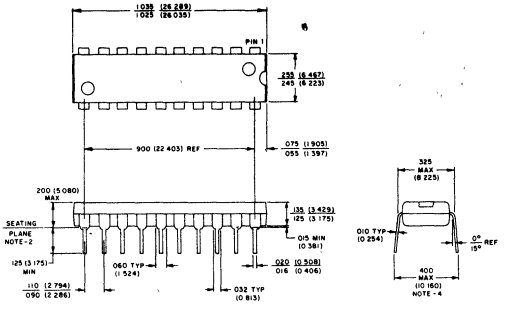
18-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



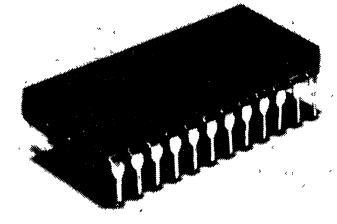


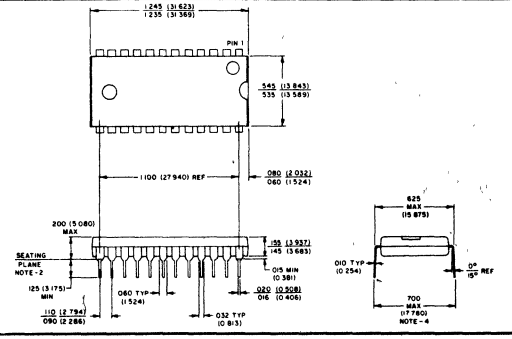
20-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P





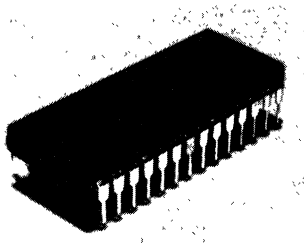
24-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P

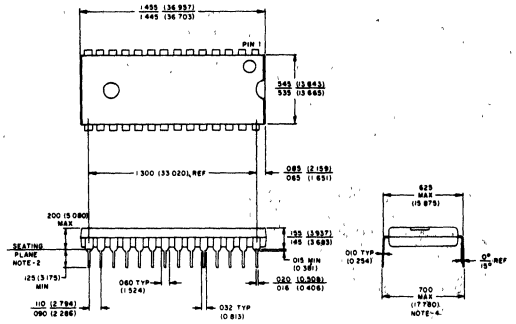




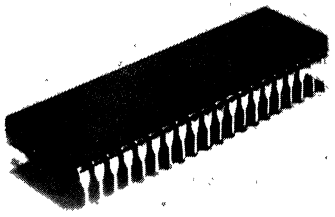
PLASTIC DUAL IN-LINE PACKAGE TYPE P

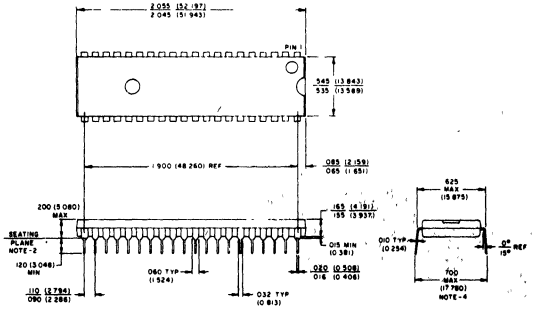
28-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



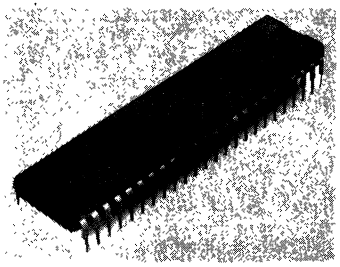


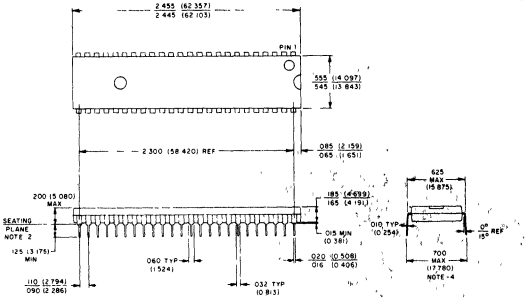
40-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P





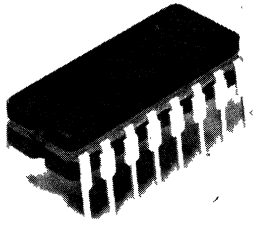
48-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P

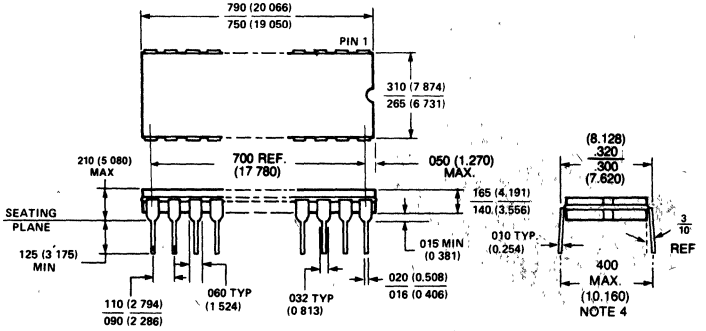




CERAMIC DUAL IN-LINE PACKAGE TYPE D

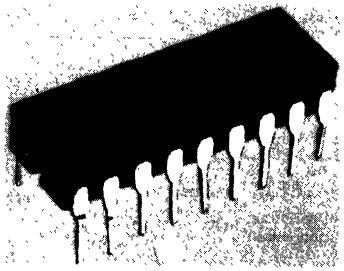
16-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D

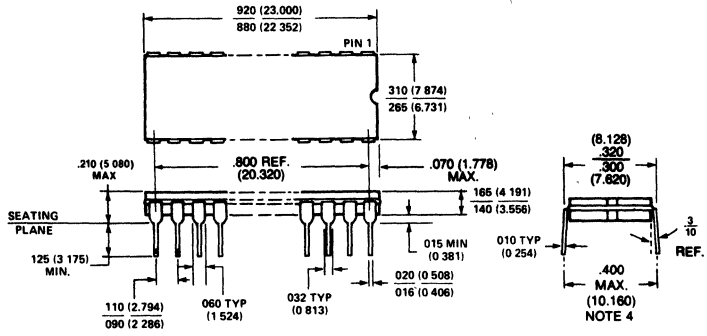




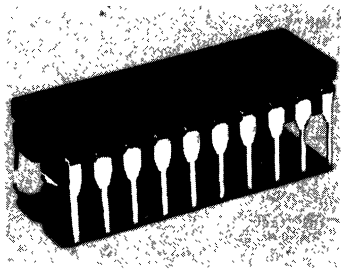
CERAMIC DUAL IN-LINE PACKAGE TYPE D

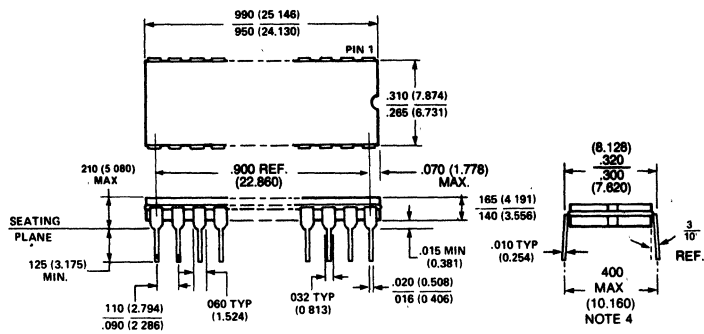
18-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



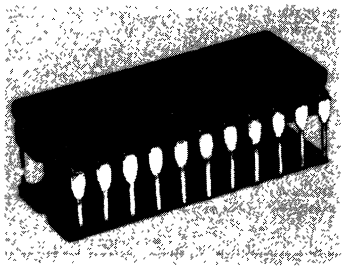


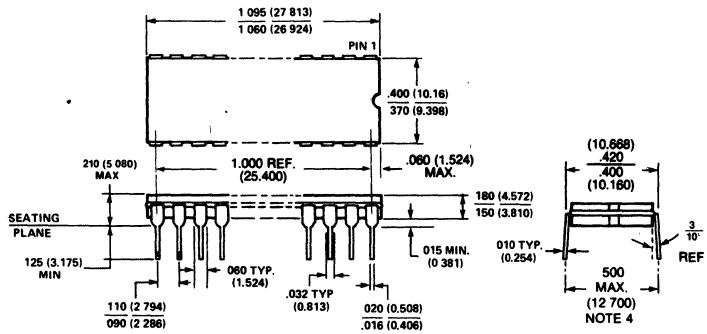
20-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



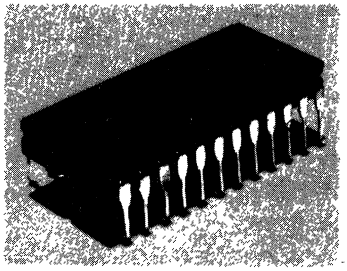


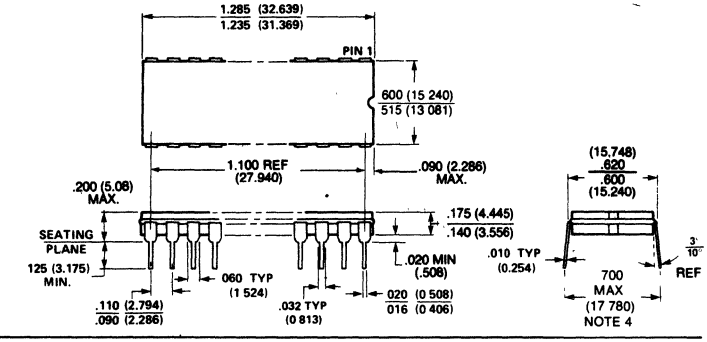
22-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



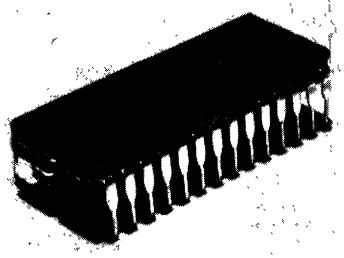


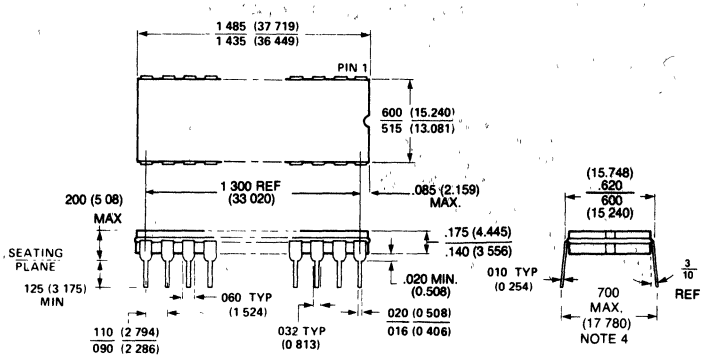
24-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



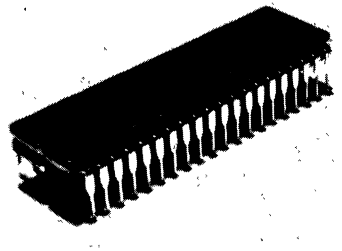


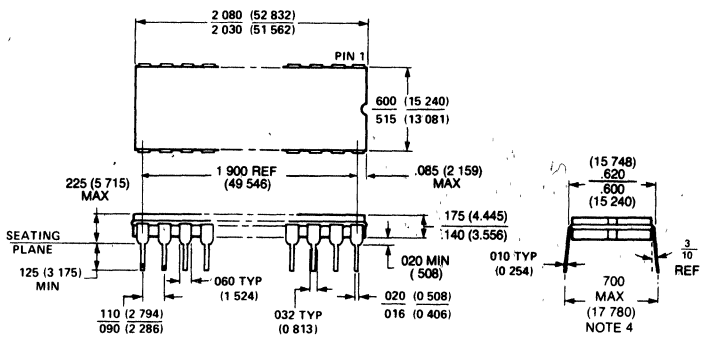
28-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



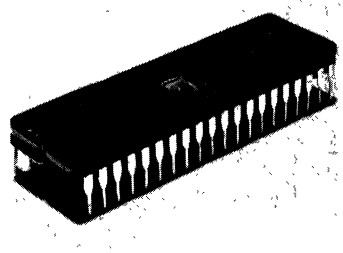


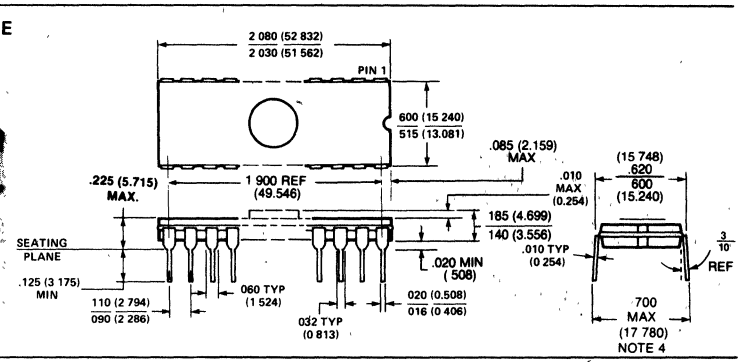
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D





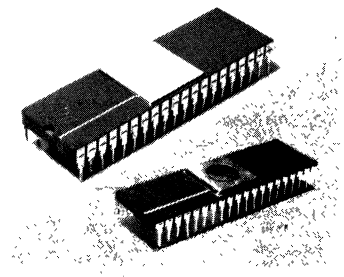
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D

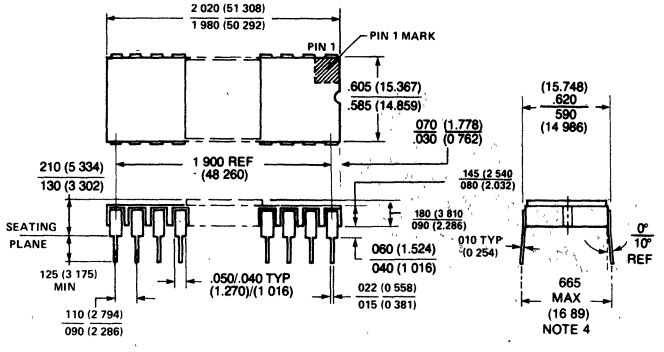




CERAMIC DUAL IN-LINE PACKAGE TYPE C

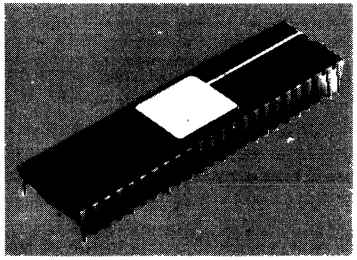
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE C

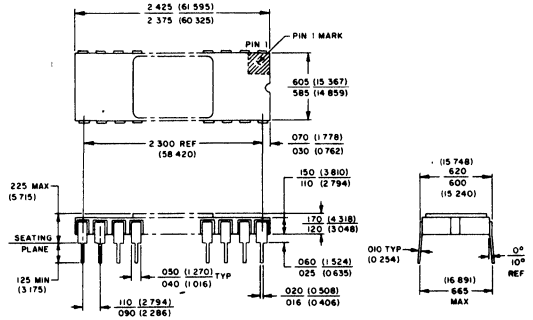




CERAMIC DUAL IN-LINE PACKAGE TYPE C

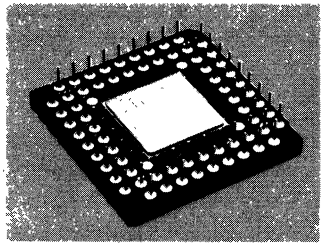
48-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE C

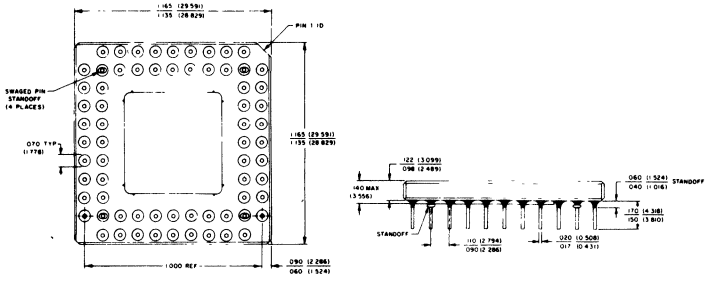




CERAMIC PIN GRID ARRAY PACKAGE TYPE CG

68-LEAD CERAMIC PIN GRID ARRAY PACKAGE TYPE CG





Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Intel International (U.K.) Ltd.
Piper's Way
Swindon, SN3 1RJ
Wiltshire, England

Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

Printed in U.S.A./C-634/120K/RRD/CG-MK
Microprocessor/Peripheral