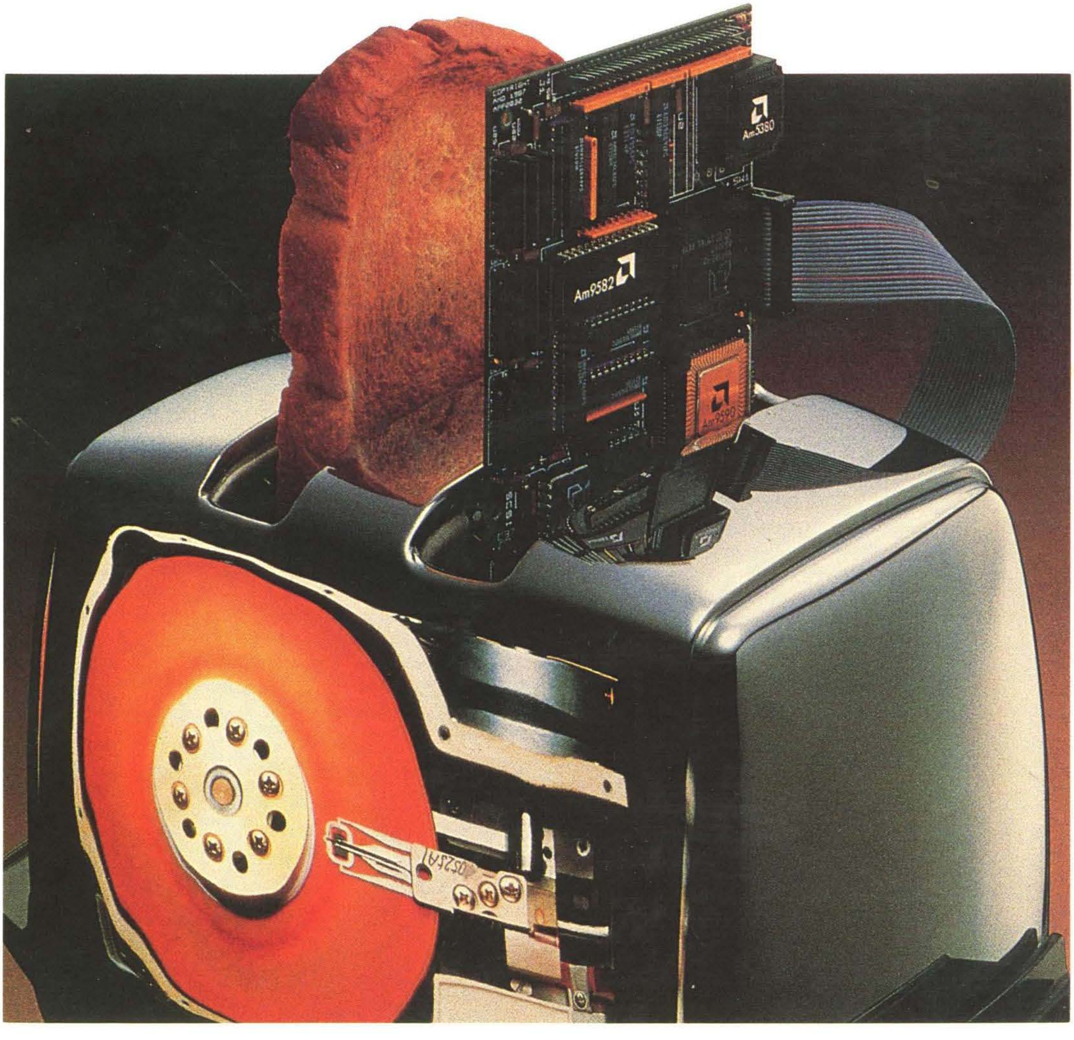
Hard Disk Controller

Technical Manual

Advanced
Micro
Devices



Advanced Micro Devices

Hard Disk Controller Am9580A/90 Technical Manual

Advanced Micro Devices reserves the right to make changes in its products without notice in order to improve design or performance characteristics.

This technical manual neither states nor implies any warranty of any kind, including but not limited to implied warranties of merchantability or fitness for a particular application. AMD assumes no responsibility for the use of any circuitry other than the circuitry embodied in an AMD product.

This information in this publication is believed to be accurate in all respects at the time of publication, but is subject to change without notice. AMD assumes no responsibility for any errors or omissions, and disclaims responsibility for any consequences resulting from the use of the information included herein. Additionally, AMD assumes no responsibility for the functioning of undescribed features or parameters.

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088-3000
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

TABLE OF CONTENTS

CHAPTER I

CHAPTER 2

CHAPTER 3

CHAPTER 4

APPENDIX

Contributors to the Am9580A/90 Technical Manual are Jochen Polster, Joe Brcich, Patrick Soheili, David Stoenner, Sherman Lee, and T. Kaneko.

Edited by Sue Louie Thilking.

CHAPTER 1

1.0 INTRODUCTION

Disk Controller Products

The magnetic disk drive market today is split into two major groups: Rigid disks and Flexible disks. Usually, the rigid, or hard disk is implemented in a disk drive device as a non-removable medium; whereas the flexible, or floppy disk is the common removable medium. Historically there are several different physical disk sizes available.

The hard-disk drive market started off with 14" diameter disks. This format is mostly used in mainframe applications. Common to all hard-disk drives is that several platters are mounted on one single spindle. Both surfaces of each platter have a separate magnetic read/write head. All heads are moved together to a certain cylinder (Figure 1-1). Other disk sizes are 8", 5.25", and 3.5" in diameter. The data storage capacities of these drives vary widely from 1-2 GByte for 14" down to 20 MByte for 3.5" disk drives.

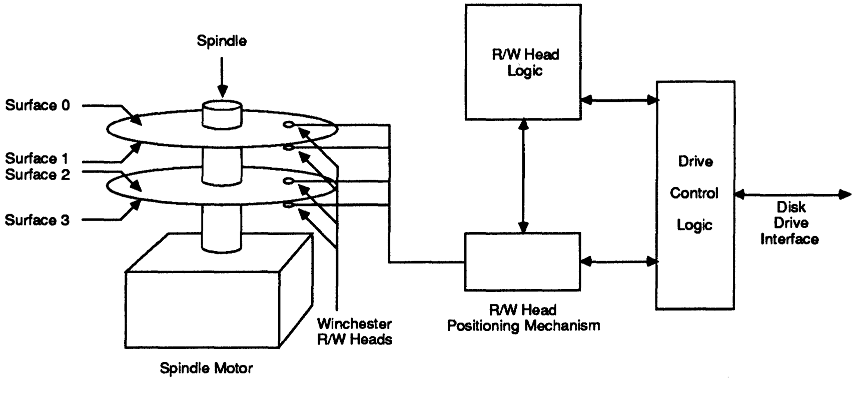
Floppy disks are also available in 8", 5.25" and 3.5". Normally, both surfaces of the disk are used to store data. Storage capacities range from 360 KByte to 2 MByte.

How Data Is Stored onto a Magnetic Disk

In order to store data onto a disk, the parallel data stream (8- or 16-bit) of the processor's bus must be serialized. The resulting NRZ (Non-Return to Zero) serial data stream is not suitable for magnetic recording. Magnetic flux transitions are required to magnetize the disk material in one out of two different directions. Only those direction changes can be detected during reading. Therefore, different methods are used to encode the NRZ data into a format suitable for the disk.

The encoding schemes use mixed data and clock pulses so that there is a fixed-time base (the clock) when reading. The oldest and simplest one is FM encoding. This scheme only looks at one bit at a time and encodes it into a maximum of two flux transition per data bit (Figure 1-2). FM encoding adds a clock pulse for each data bit. MFM encoding, on the other hand, only inserts clock pulses in strings of 0s; therefore, the data density is doubled compared with FM encoding (Figure 1-3).

Recently, a new encoding scheme that further increases the data density has become more popular. The RLL (Run Length Limited) coding



9480A 1-1

Figure 1-1 Simplified Block Diagram of a Winchester Disk Drive

rules allow many different encoding schemes by modifying their parameters. For example, FM and MFM coding schemes are special forms of RLL code. A RLL code that is widely used is the 2,7 Code. The two numbers indicate the minimum (2) and maximum (7) number of 0s in between 1s (Figure 1-4). The 2,7 Code increases the data density on the disk by 50%, compared to MFM.

Interfaces

Historically, two disk interfaces have emerged as a standard for most of today's hard-disk drives. The SMD (Storage Module Device) interface is popular in the high end (14" and 8") market. For 5.25" and 3.5" disks, the ST506/412 interface is the de facto standard.

The ST506/412 (Figures 1-5a,b) interfaces define all the control and data signals connecting up to four disk drives. The data transfer is defined as using MFM encoding scheme. The data rate for ST506/412 is fixed at 5 Mbit/s. The disk drive usually does not embed higher functions; seeks to various tracks, for example, are done by sending an appropriate number of step pulses to the drive. Recently, there are some approaches towards improving the performance of this standard; the data rate is being increased (up to 10 Mbit/s) and a different coding scheme is used (RLL).

A new standard has been introduced recently for hard-disk drive interfaces. This standard, called ESDI (Enhanced Small Device Interface) (Figures 1-6 a, b), allows for a more flexible and faster interface between the hard-disk controller and drive. It is specifically designed to interface new, high-capacity hard-disk drives (5.25") better than the older standard (ST506).

There are two major differences between ST506/412 and ESDI:

The data transfer mode between controller and hard-disk drive is no longer MFM but NRZ. Thus, the Disk Data Separator (DDS) is implemented on the drive and the data encoding and decoding are handled by the ESDI drive. This leaves the choice of an appropriate encoding scheme to the hard-disk drive manufacturer. The typical choice for ESDI drives is some form of Run Length Limited (RLL) encoding scheme. Another advantage of implementing the DDS on the disk drive is the very close coupling between DDS logic and drive electronic. This will increase the reliability because cables and additional line drivers are no longer required. The data rates are now determined by the disk drive (Data Clock is generated by the drive).

The ESDI interface defines a serial command/status link between controller and disk drive. Commands such as SEEK and RESTORE allow a much faster seek timing than with step pulses in ST506 (e.g., to a step from track 0 to track 1000, the ST506 standard requires 1000 step pulses), the ESDI standard, however, needs only one 17-bit command word. This also frees the controller from the need to know drive characteristics such as step-width, etc.

The command/status interface also adds functionality. The controller can request status information from the disk drive to get detailed information on error conditions. Another feature of ESDI is the ability to request information about the drive configuration from the disk drive. Until now the host computer needed to know the kind of drive to which it is connected. This was usually realized in the form of drive parameter tables containing information about the number of heads, cylinders, etc., of a particular drive. With the new ESDI feature, these tables are not needed. The CPU can request all necessary parameters directly from the disk drive and initialize the controller accordingly.

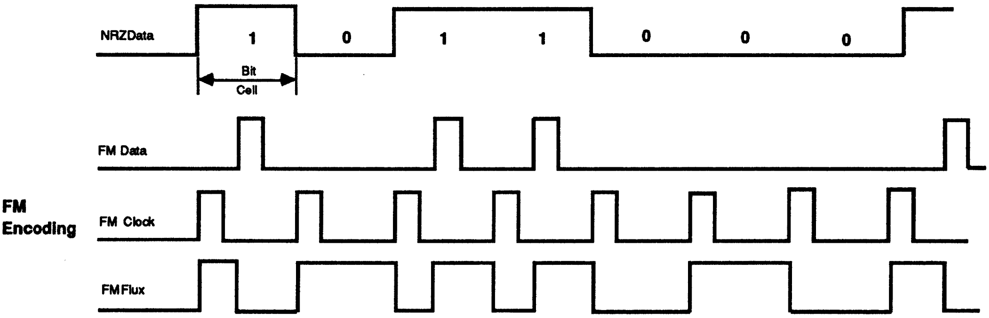
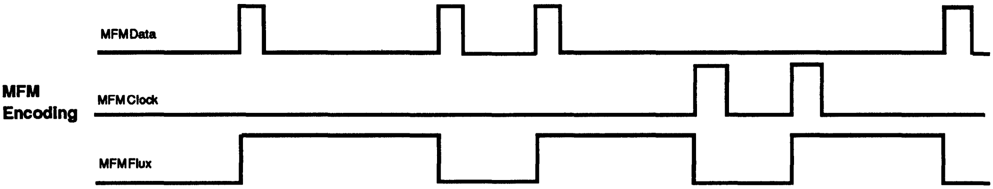


Figure 1-2 FM Encoding

9480A 1-2



FM Encoding Rules

1. Insert clock pulse for each cell
2. Insert data pulse when NRZ data is "1"

MFM Encoding

1. Insert clock pulse only if NRZ data = 0 for 2 bits
2. Insert data pulse when NRZ data is "1"

9480A 1-3

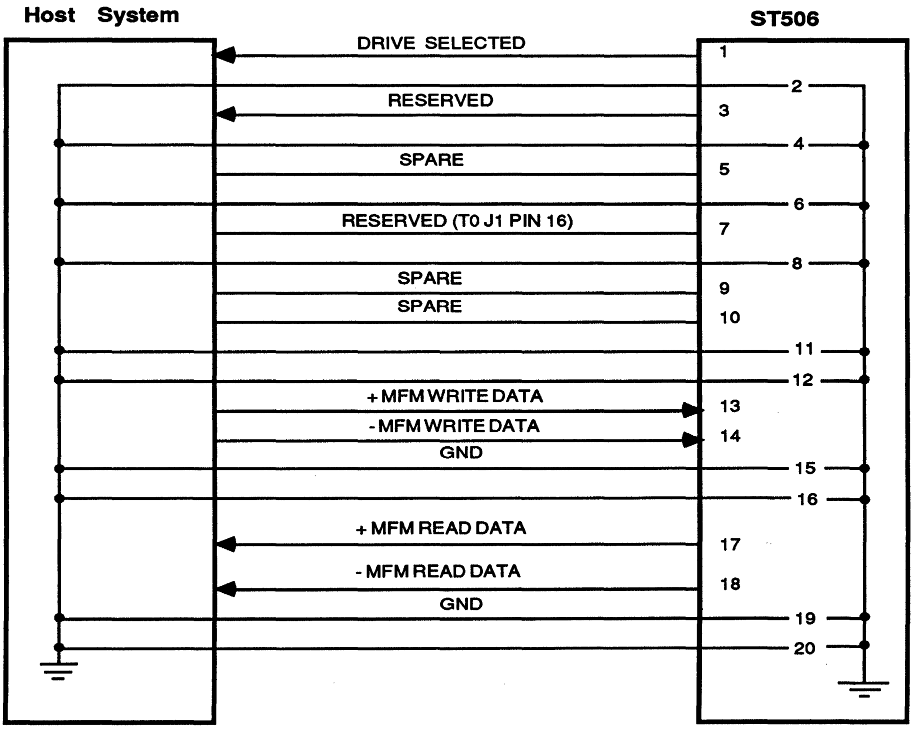
Figure 1-3 MFM Encoding

The RLL 2,7 coding rules are listed below.

NRZ	2,7	decoded data length
10	100	2
11	0100	2
000	100100	3
010	001000	3
011	000100	3
0010	00001000	4
0011	00100100	4

9480A 1-4

Figure 1-4 RLL 2,7 Coding Rules



9480A 1-5a

Figure 1-5a Data Signals

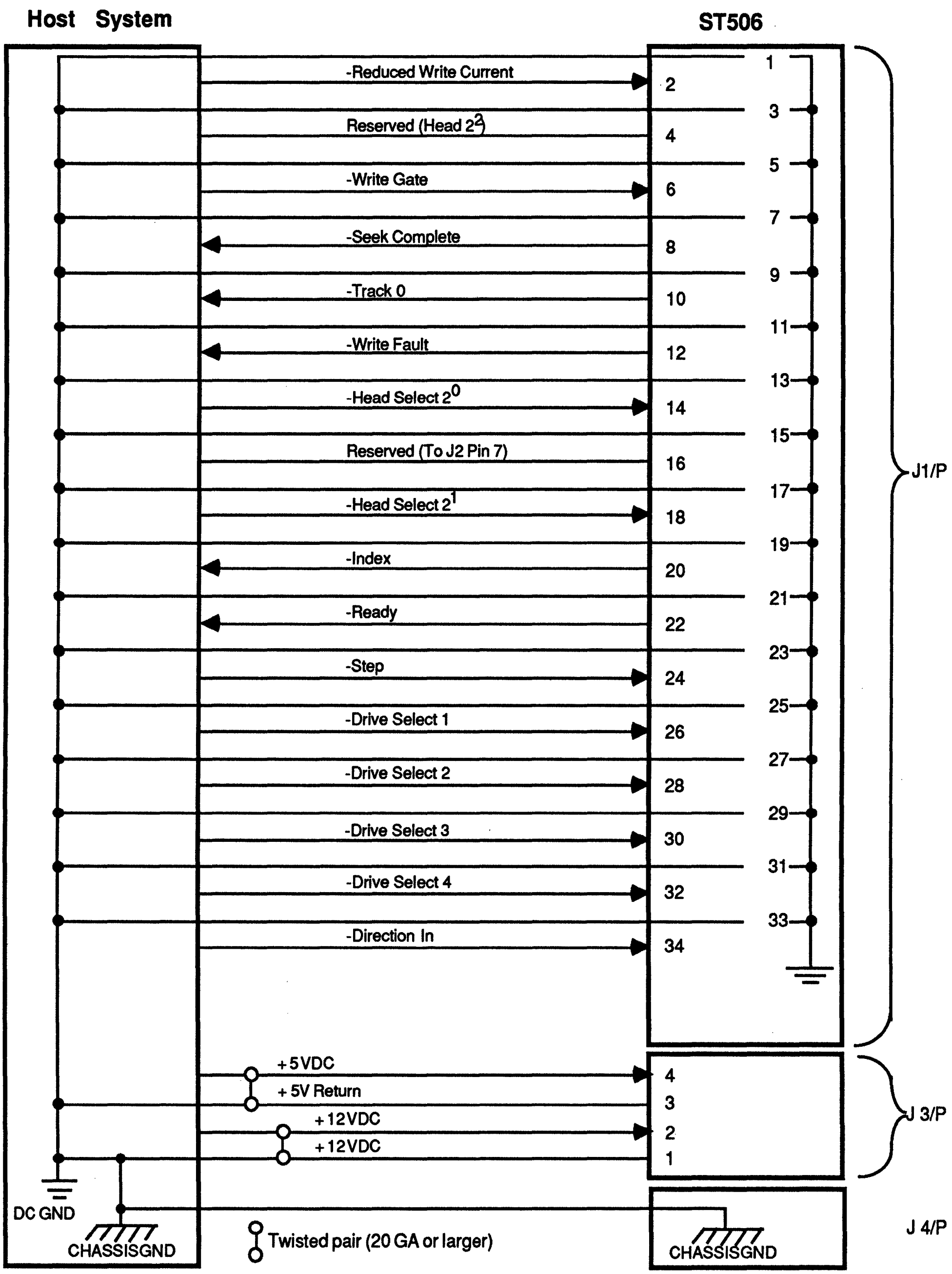


Figure 1-5b ST506 Drive Interface J1

9480A 1-5B

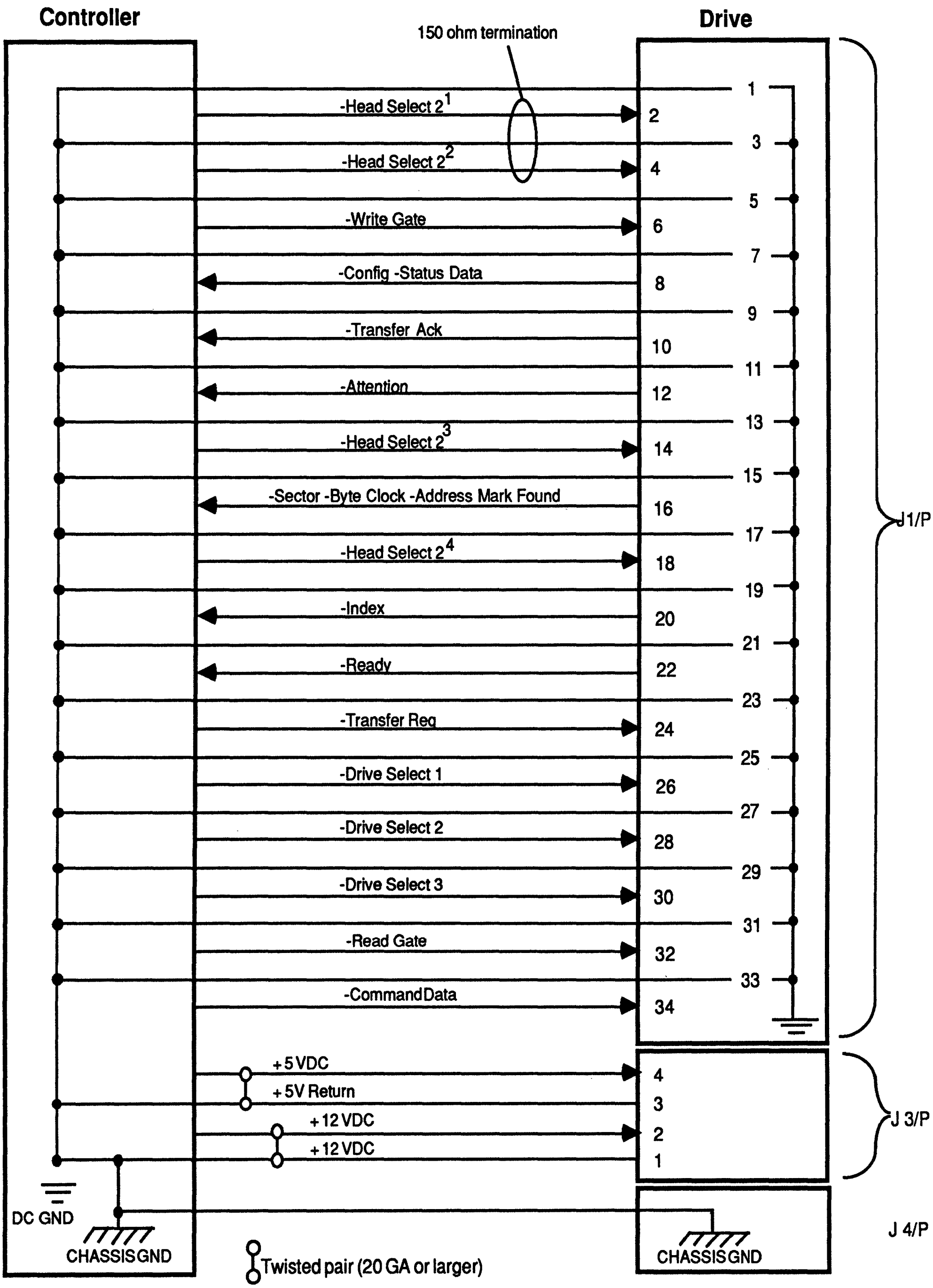
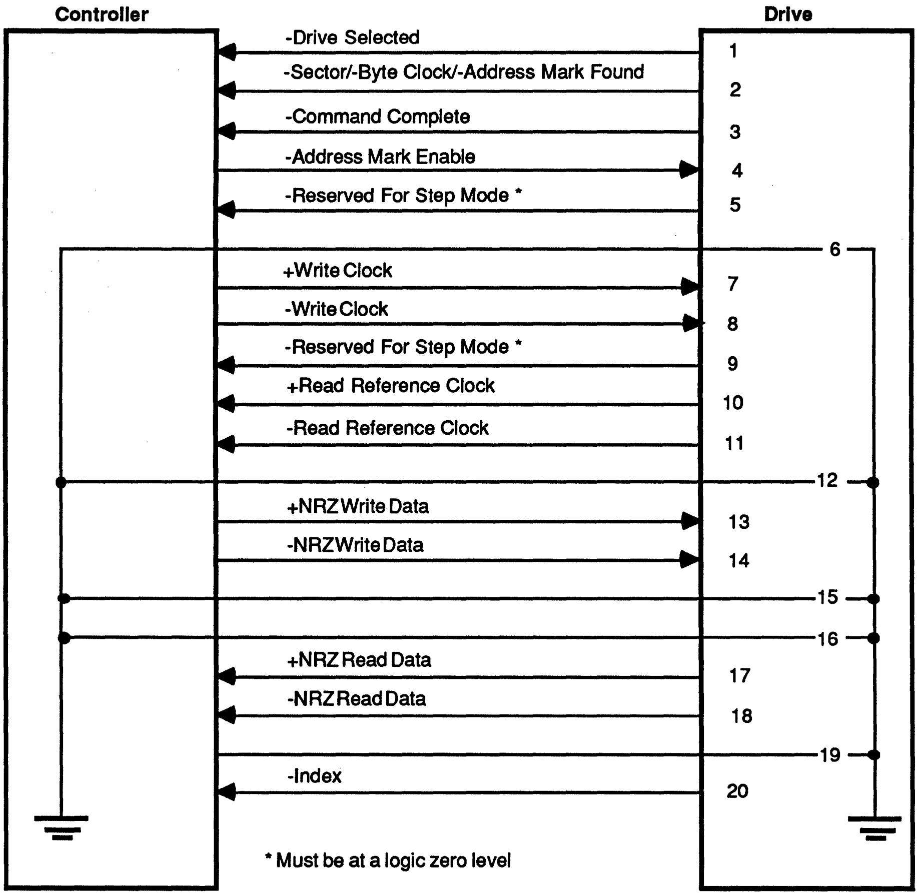


Figure 1-6a Enhanced Small Device Interface

9480A 1-6A



9480A 1-6B

Figure 1-6b Enhanced Small Device Interface

Building Blocks of a Disk Controller

A Disk Controller can typically be divided into five logical blocks (Figure 1-7): The Serializer/Deserializer, the Data Formatter, the Error-Detection-and-Correction Logic (EDAC), the Drive Control Logic, and Buffer Logic.

The Serializer/Deserializer converts the parallel data stream from the host CPU into a serial data stream suitable for the disk drive.

The Data Formatter partitions these data into sectors and adds all the necessary control information to each data block. This control information allows the later retrieval of an individual sector from the disk.

The EDAC generates certain control bytes when writing information onto the disk that are used to verify the data when they are being reread.

The fourth function a disk controller has to perform is the control of the disk drive functions, such as stepping to the desired track and selecting the read/write head. In addition, ST506 and floppy-disk controllers need to perform NRZ/MFM data encoding and decoding on board.

The problems that occur when implementing these different functional blocks are mainly complexity and large speed differences. The control functions of a disk drive, for example, are rather slow operations, whereas the disk data interface has data rates of up to 20 Mbit/s or higher. Also, it is desirable to fully decouple host CPU and disk data interface.

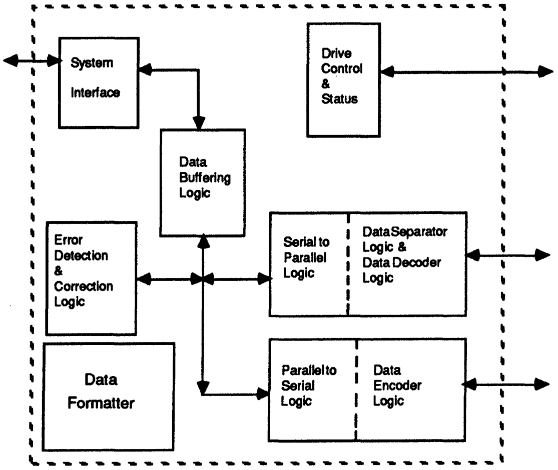
Today, these functions are usually implemented on board-level disk controllers using separate LSI elements and local CPUs or microcontrollers. Because there are considerable differences between the individual disk interfaces (ST506/412, ESDI, SMD and Floppy Disk), most boards are able to handle only one of those.

1.1 GENERAL INFORMATION

The Am9580A/90 Hard Disk Controller (HDC) is the single-chip solution to the problems encountered in designing Data Formatters and Disk System Controllers. Together with its companion part, the Am9582 Disk Data Separator (DDS), the Am9580A/90 provide all the functions which, until now, have been found only on sophisticated board-level products.

The HDC controller supports rigid and flexible disk drives and their respective data formats. The Am9580A/90 can control up to four drives, allowing any mix of rigid and flexible drives. The characteristics of each drive are independently user-programmable.

A sophisticated on-chip DMA Controller fetches commands, writes status information, fetches data to be written on disk and writes data that has been read from disk. The DMA operation is programmable to adjust the bus occupancy, data bus width (8-bit or 16-bit), and Wait State insertion. Two sector buffers allow zero-sector interleaving to access data on physically adjacent sectors, improving both file access time and system throughput. Sector sizes of 128, 256 and 512 bytes are programmable.



9480A 1-7

Figure 1-7 Disk Controller Block Diagram

The Am9580A/90 insure data integrity by selecting one of two methods: either by selecting an error detecting code (CRC-CCITT), or one of two error correcting codes (Single- or Double-Burst Reed-Solomon). Additionally, the HDCs provide handshake signals to control external ECC circuitry to implement any user-definable ECC algorithm.

The disk controllers provide signals that are necessary to control external Encode/Decode and Address Mark circuitry, e.g., the Am9582. By partitioning the disk control system this way, future developments in the field of data encoding (e.g., RLL codes) will be able to take advantage of the HDC's advanced data formatting and control capabilities.

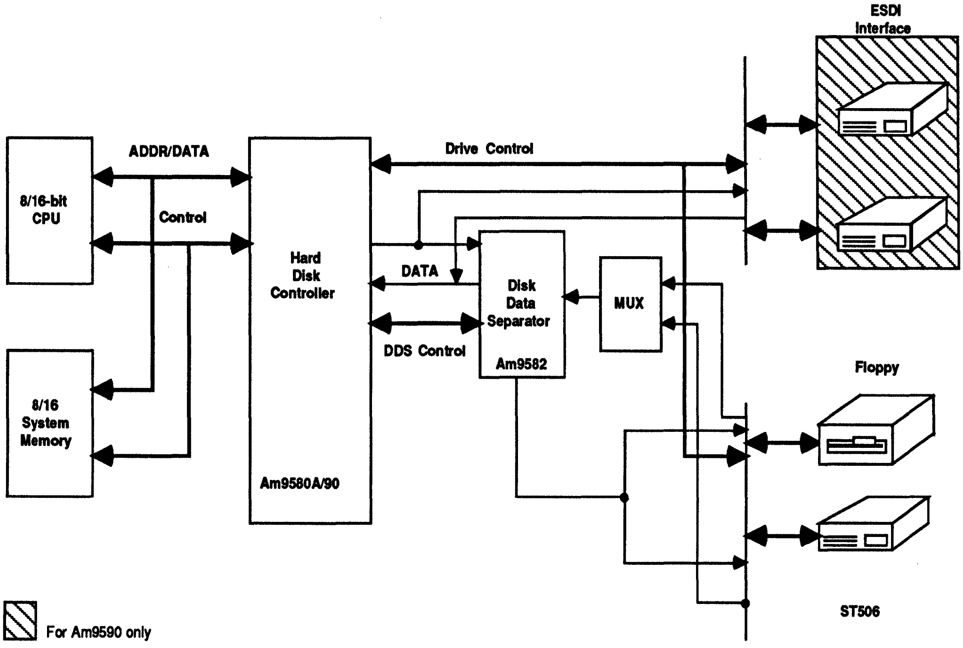
The Am9580A and the Am9590 both handle the ST506/412 and standard double-density floppy-disk data format. They also provide all the control signals required by those interfaces. Track format and control interface timing are independently switchable, which keeps the disk interface adaptable to other standards.

The Am9590 also directly supports the ESDI hard- and soft-sectored disk interface. The Am9590 is a superset of the Am9580A. It is pin-compatible and the software is fully backward-compatible.

The Am9580A/90 provides a comprehensive, high-level command set for multi-sector disk I/O, marginal data recovery, diagnostics, and error recovery. Commands may be linked together to be executed sequentially by the disk controller without any host intervention. This linked-list command structure also simplifies command insertion, deletion, or rearrangement.

How to Read this Technical Manual

This manual describes both the Am9580A and the Am9590. The two devices have most of the functions in common. Therefore, if not mentioned otherwise, all the descriptions in this manual are valid for both disk controllers.



9480A 1-8

Figure 1-8 Disk Controller System

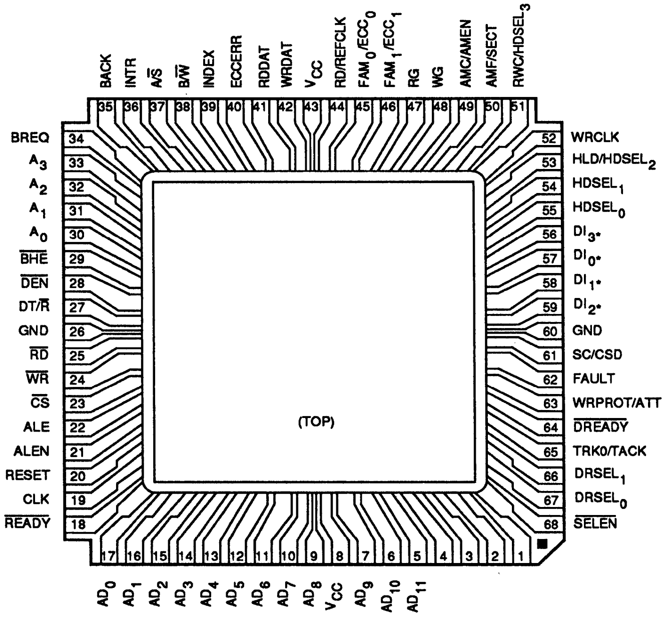
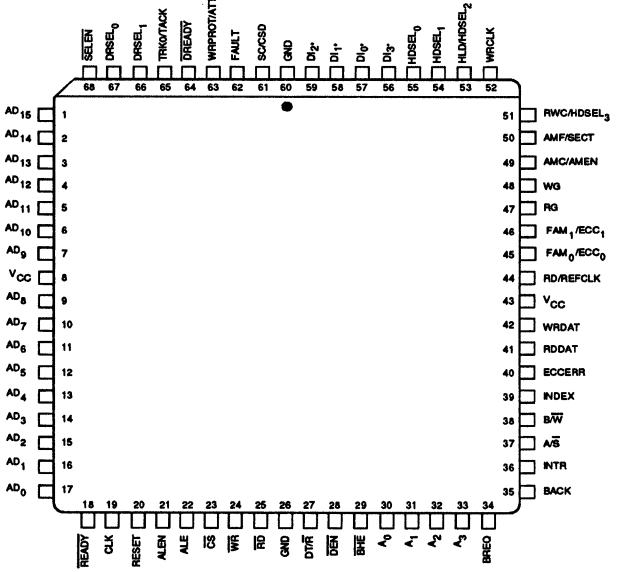


Figure 1-9a LCC

PLCC



*Refer to Pin Description section for options.

Figure 1-9b PLCC

LOGIC SYMBOL

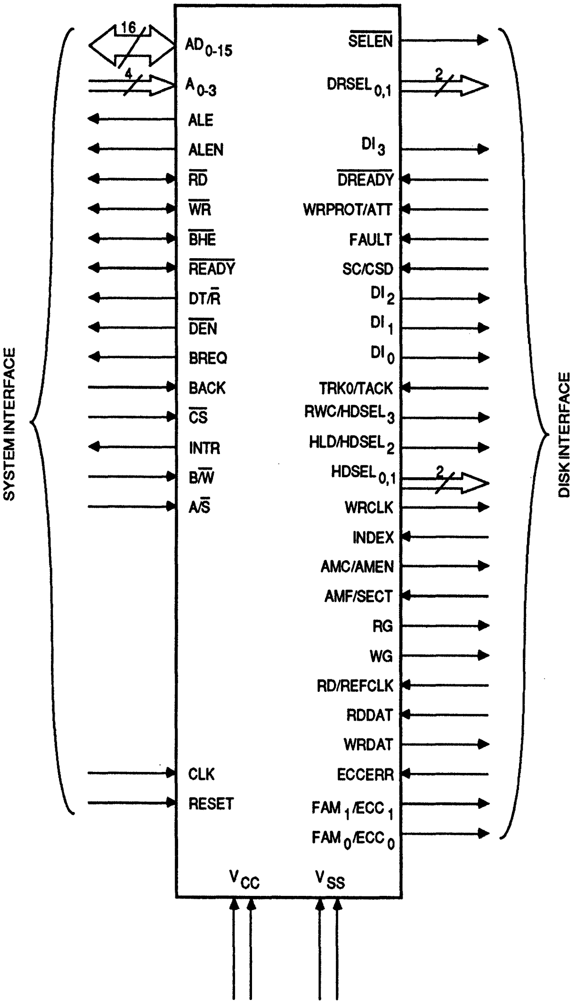


Figure 1-9c

1.2 DATA SHEET INFORMATION

Interface Signals

V_{CC1}, V_{CC2} — +5 V Power Supply (both lines must be connected)

GND_1, GND_2 — Ground (both lines must be connected)

System Interface Lines

CLK System Clock (Input)

CLK is a TTL-compatible clock input to time DMA transfers and disk control operations (e.g., Seeks). The system clock drives all except the Disk Data Section of the HDC .

$\overline{\text{RD}}$ $\overline{\text{Read}}$ (Input/Output, Active LOW)

A LOW on this line indicates that the CPU or HDC is performing an I/O or memory read cycle. When the HDC is in Slave Mode, this is an input signal used by the CPU to read the internal registers of the HDC (slave access). When the HDC is the bus master, this signal is an HDC output to access data from memory.

In Slave Mode, the transfer control signals, $\overline{\text{RD}}$ and $\overline{\text{WR}}$, must not be active simultaneously, but may be asynchronous to the clock. In Master Mode, the HDC drives this line synchronously by using a 4-clock-cycle transfer.

$\overline{\text{WR}}$ $\overline{\text{Write}}$ (Input/Output, Active LOW)

A LOW indicates that the CPU or HDC is performing an I/O or memory write cycle. When the HDC is in Slave Mode, it is an input signal used by the CPU to load the internal registers of the HDC. When the HDC is the bus master, this signal is a HDC output to write data to the system memory. In Slave Mode, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ must not be active simultaneously.

$\overline{\text{BHE}}$ $\overline{\text{Byte High Enable}}$ (Input/Output, Active LOW)

$\overline{\text{BHE}}$ enables data on the most significant byte of the data bus (AD_{15}–AD_8). The data bus is allocated as follows:

$\overline{\text{BHE}}$	$\overline{\text{AO}}$	Data Lines	Type
0	0	AD_{15}–AD_0	Word Transfer
0	1	AD_{15}–AD_8	Byte Transfer on upper byte
1	0	AD_7–AD_0	Byte Transfer on lower byte
1	1	—	none (reserved)

When the HDC is the bus master, this pin is an output. When the HDC is the bus slave, it is an input and must be stable for the entire cycle. BHE is ignored when the HDC is strapped to a byte interface.

$\overline{\text{READY}}$ $\overline{\text{Ready}}$ (Input/Output, Active LOW, Open Drain)

When the HDC is in control of the system bus, this is an input to allow slow memories and peripheral devices to extend the bus cycle. When the HDC is in Slave Mode, this is an output indicating that the HDC is ready to complete the current bus transfer. The CPU $\overline{\text{READY/WAIT}}$ input must be connected to the $\overline{\text{READY}}$ output of the Am9590 (in Slave Mode), because slave cycle length varies between 1 and 16 system clocks.

AD<15:0> Address/Data Bus (Input/Output, Active HIGH, Three-state)

The Address/Data Bus is a time-multiplexed, bidirectional, three-state, 16-bit bus used for all system transactions. A HIGH represents a "1" on the bus and a LOW represents a "0". AD_0 is the least significant bit. The presence of an address is indicated by either ALE or ALEN. When ALE is HIGH, the bus contains lower address bits (A_0–A_{15}). When ALEN is HIGH, the bus contains upper address bits (A_{16}–A_{31}). The 32-bit address allows the HDC to directly access a linear (non-segmented) memory address space of up to 4 Gbytes.

The presence of data is indicated by the $\overline{\text{RD}}$, $\overline{\text{WR}}$, and $\overline{\text{READY}}$ signals. The HDC drives data out onto the AD-bus (lines are configured as outputs) when $\overline{\text{RD}}$ is asserted in Slave Mode, or when $\overline{\text{WR}}$ is asserted in Master Mode. The

HDC reads data in from the AD-bus (lines are configured as inputs) when $\overline{\text{WR}}$ is asserted in Slave Mode, or when $\overline{\text{RD}}$ is asserted in Master Mode.

Mode	$\overline{\text{RD}}$	$\overline{\text{WR}}$	AD_{15}–AD_0
Slave	L	H	Output
Slave	H	L	Input
Master	L	H	Input
Master	H	L	Output

A_0 Address Line 0 (Input, Active HIGH)

When the HDC is in Slave Mode, a HIGH on this address line selects the upper byte of internal registers, a LOW selects the lower byte of internal registers. For word accesses, this line must be LOW. A_0–A_3 must be valid through the Read or Write cycle. In Master Mode, this line is ignored.

A_1–A_3 Address Lines 1-3 (Inputs, Active HIGH)

When the HDC is in Slave Mode, these lines select one of the internal registers (see Figure 1-16). In Master Mode, these lines are ignored.

ALE Address Latch Enable (Output, Active HIGH)

ALE latches the lower address word (A_0–A_{15}) onto the external address latch. This output is never floating.

ALEN Upper Address Latch Enable (Output, Active HIGH)

ALEN latches the upper address word (A_6–A_{31}) onto the external address latch. This signal is active whenever the upper address is to be updated. The upper address is not updated at the beginning of each DMA burst; therefore, the upper address latch must not be shared with the CPU or other DMA devices. This output is never floating. (See also "System Interface" Section)

DT/$\overline{\text{R}}$ Data Transmit/$\overline{\text{Receive}}$ (Output, Three-State)

In Master Mode, this output indicates the direction of data flow. A HIGH indicates that the data is being transmitted from the HDC to memory (master write cycle). A LOW indicates that the data is being transferred from memory to the HDC (master read cycle). This output is floating when the HDC is not in control of the system bus.

$\overline{\text{DEN}}$ $\overline{\text{Data}}$ Enable (Output, Active LOW, Three-State)

When the HDC is bus master, a LOW on this output enables an external data bus transceiver (DT/$\overline{\text{R}}$ specifies the direction). $\overline{\text{DEN}}$ is active when data is driven onto the Address/Data bus (master write cycle), or the bus is three-stated when receiving the data (Master Read cycle). This output is three-stated when the HDC is not in control of the system bus.

BREQ Bus Request (Output, Active HIGH)

The HDC activates BREQ to request control of the system bus. BREQ timing is specified by the DMA Burst Length and DMA Dwell Time parameters in the Mode Register.

BACK Bus Acknowledge (Input, Active HIGH)

BACK acknowledges the HDC bus request, indicating that the CPU has relinquished the system bus to the HDC. Since BACK is internally synchronized, transitions on BACK do not have to be synchronous with the system clock (CLK). BACK may be removed, at any time, to make the HDC release the bus (bus preemption). If the HDC DMA is preempted by removing BACK, HDC completes the current bus transaction and releases BREQ for the programmed dwell time so that external devices may gain system bus mastership. BACK must be active for at least one clock

INTR Interrupt Request (Output, Active HIGH)

INTR is activated when the HDC requires CPU service. Interrupt Request is reset whenever the upper byte (Status Byte) of the Status/Command Register (SCR) is accessed. The HDC asserts INTR after a hardware or software reset to indicate that the internal reset process has been completed. This interrupt cannot be disabled. Further interrupts are issued whenever the HDC enters the IDLE state (e.g., terminating a command chain). These interrupts may be disabled by resetting the Interrupt Mask in the Mode Register. The INTR output is never floating.

$\overline{\text{CS}}$ $\overline{\text{Chip Select}}$ (Input, Active LOW)

The host processor activates $\overline{\text{CS}}$ to enable a Slave Mode access to read or write HDC internal registers. $\overline{\text{CS}}$ may be asynchronous to the system clock (CLK). This pin is ignored when the HDC is in Master Mode.

B/$\overline{\text{W}}$ Byte/$\overline{\text{Word}}$ Strap (Input)

This pin selects either a byte (8-bit) (B/$\overline{\text{W}}$ HIGH) or word (16-bit) (B/$\overline{\text{W}}$ LOW) interface. When a byte interface is selected, only AD_0–AD_7 are used for data transfers, making all operations byte operations. When word interface is selected, AD_{15}–AD_0 are used for data transfers. The HDC always uses a 32-bit address. This input may be altered only while the HDC is in IDLE state.

HIGH = byte interface
LOW = word interface

A/$\overline{\text{S}}$ Asynchronous/$\overline{\text{Synchronous}}$ (Input)

This input selects whether the $\overline{\text{READY}}$ input is synchronous or asynchronous to the system clock (CLK). When A/$\overline{\text{S}}$ is HIGH, the HDC internally synchronizes the $\overline{\text{READY}}$ input. When A/S is LOW, $\overline{\text{READY}}$ must be synchronized externally. This input may only change state while the HDC is in IDLE state. Normally it is tied to +5V or GND.

RESET Reset (Input, Active HIGH)

When RESET is active, all outputs lines are inactive, all three-state outputs are floating, and all inputs other than RESET are ignored. With the falling edge of RESET, the chip enters the initialization procedure. A RESET pulse is required after power-up. Upon completion of initialization, an interrupt request will be issued and INTR will go HIGH. If the user attempts to read from or write to the HDC prior to completion of initialization, the $\overline{\text{READY}}$ output will remain inactive until initialization has been completed; this causes the CPU to wait. After an initial hardware reset, a software reset (loading the Command Status field of the Status/Command Register with RESET) is equivalent to a hardware reset (pulse on the RESET input).

Power-up reset must be active after V_{CC} has been stable for a certain period (see AC specification). This can be achieved by a long reset pulse generated by a RC circuit during power-up, or by a short pulse after power-up. The HDC must capture the Reset input being HIGH for two rising edges of the system clock (2 plus system clocks pulse width).

Disk Interface Lines

Some of the pins described in this section have different functions depending on the interface type selected. The cross reference list page1-18 shows the relationship between interface and pin functions.

$\overline{\text{SELEN}}$ $\overline{\text{Select Enable}}$ (Output, Active LOW)

$\overline{\text{SELEN}}$ = LOW enables the drive specified by $\text{DRSEL}_{0,1}$. When $\overline{\text{SELEN}}$ = HIGH, no drive is selected. The disk drive must respond to $\overline{\text{SELEN}}$ LOW by bringing $\overline{\text{DREADY}}$ LOW. See $\overline{\text{DREADY}}$ and MON descriptions below.

$\text{DRSEL}_{0,1}$ (Outputs, Active HIGH)

$\text{DRSEL}_{0,1}$ designates which of the four possible drives is expected to respond to the assertion of $\overline{\text{SELEN}}$.

DRSEL_1	DRSEL_0	Drive Selected
0	0	Drive 0
0	1	Drive 1
1	0	Drive 2
1	1	Drive 3

DRSEL_0 and DRSEL_1 are the two least-significant bits of the drive number specified in the IOPB.

D_{I3} Disk Interface Control 3 (Output, Active HIGH)

(PCEN) Precompensation Enable—This output indicates whether the data write encoder should initiate precompensation on the encoded disk-write data pulse stream. PCEN will be valid for at least four system clocks prior to any disk-write operation (WG LOW), and will remain valid for at least four system clocks after the disk-write operation (WG HIGH). PCEN is asserted if the current track number is greater than, or equal to, the Precompensation Track Parameter specified in the Drive Parameter Block for the selected drive. No other internal processing takes place.

HIGH—Precompensation enabled
LOW—Precompensation disabled

PCEN should be connected to PCEN/S($\overline{\text{D}}$) of the Am9582, even if precompensation is not used. When $\overline{\text{SELEN}}$ is asserted, this output is pulsed LOW to select Double-Density Floppy Mode.

(TCLK) Track # Clock—In Restricted Seek Mode and Special Mode, this output provides the shift clock for the serial track information provided on TDATA. See the following description on DI2.

$\overline{\text{DREADY}}$ $\overline{\text{Drive Ready}}$ (Input, Active LOW)

Drive Ready indicates that the currently selected drive is ready to Read, Write, or Seek. This signal should be connected to the $\overline{\text{DRSEL}}$ected signal of the drive. It must become LOW within 100,000 system clock after $\overline{\text{SELEN}}$ is asserted by the HDC (see Motor-on Description for Floppy Mode). Once asserted by the selected drive, any negation of this line causes the current IOPB to be aborted. $\overline{\text{DREADY}}$ is ignored while $\overline{\text{SELEN}}$ is HIGH. $\overline{\text{DREADY}}$ must be deasserted before reaccessing the drive.

FAULT Fault (Input, Active HIGH)

For ST506, floppy, and SMD operation, this indicates a fault in the selection of the current drive, or a fault within the selected drive. For normal operations, FAULT must be inactive (LOW) as long as $\overline{\text{DREADY}}$ is active (LOW). FAULT is considered valid after $\overline{\text{DREADY}}$ is asserted. If it is asserted by the selected drive, the HDC will immediately abort the current IOPB and deselects the drive. This signal should be connected to the $\overline{\text{READY}}$ pin of the drive.

9590

For ESDI interface, this pin has to be inactive during a read, write, or seek operation. If it is asserted during a read, write, or seek, the HDC will immediately abort the current IOPB and deselects the drive. It is disregarded during a serial communication.

WRPROT/ATT Write Protect/Attention (Input, Active HIGH)

(WRPROT) Write Protected—Before the HDC attempts to write data to the currently selected drive, the HDC samples WRPROT to determine whether the drive is write protected (WRPROT HIGH). If it is, the current IOPB is immediately aborted. Typically, in Hard-disk Mode, this input should be tied LOW (inactive), and is considered valid after $\overline{\text{DREADY}}$ is asserted. It is ignored during "Read-Only"-type commands. When $\overline{\text{SELEN}}$ is inactive (HIGH), this input is also ignored.

9590

(ATT) Attention—For ESDI interface, this line has to be valid after $\overline{\text{DREADY}}$ becomes active. With this signal, an ESDI drive indicates to the controller that status information can be read from the drive (usually this is an error condition). If Attention becomes active when the HDC attempts a read or write operation, the HDC will abort immediately and deselects the drive.

SC/CSD Seek Complete/Configuration Status Data (Input, Active HIGH)

(SC) Seek Complete—The drive asserts SC to indicate to the HDC that the head is loaded (only for Floppy Mode) and the drive is ready for another Seek operation. This line is sampled and verified HIGH before starting any seek operation.

(CSD) Configuration Status Data—If the ESDI interface (Serial Mode) is selected, this line is the serial data input for configuration/status information from the drive.

D_{I2} Disk Interface Control 2 (Output, Active HIGH)

STEP—The HDC pulses the STEP line to move the head from one track to the next. The width and spacing of pulses are programmable, allowing an easy upgrade path to higher performance drives. The disk drive should initiate the head movement with the rising edge of STEP. SC (Seek Complete) must go inactive after the HDC has issued the first step-pulse.

9590

(TRQ) Transfer Request—For ESDI interface, the HDC uses this pin to request a data transfer (one bit at a time) to or from the drive. If data are transferred to the drive, the pin is activated when a command bit is present on the Command/Data line. It is deasserted after the ESDI disk drive responds with TACK (Transfer Acknowledge). If data are received from the drive, TRQ indicates that the HDC is ready to receive a bit from the drive. Again, TRQ is deasserted with the reception of TACK.

(TDATA) Track Data—In Restricted Seek Mode, and Special Mode, TDATA provides the current track number (16-bit) each time the track number needs to be updated. A shift clock is available on the D_{I3} output.

DI_1 Disk Interface Control 1 (Output, Active HIGH)

(DIRIN) Direction In—DIRIN indicates the direction the head should move on STEP pulses. When HIGH, the head should move towards higher track numbers (in or towards the disk spindle). When LOW, it should move towards lower track numbers (Out). DIRIN will be asserted at least four clock cycles before any seek pulses are issued. It remains stable during the entire stepping operation until at least four clock cycles after the last step-pulse.

9590

Command Data—For the ESDI interface, the Am9590 uses this pin to send serial ESDI command words, plus parity, to the disk drive.

(HDSEL$_5$) Head Select 5—For the Special interface, this is the most-significant bit of the head number.

DI_0 Disk Interface Control 0

(Input/Output, Active HIGH)

(RTZ) Return To Zero—In Hard-disk Mode, a pulse on the RTZ output should recalibrate the head to Track 0. The HDC may also recalibrate the drive by issuing STEP pulses until Track 0 is reached (TRK0 becomes active). The RTZ pulse has the same width as the STEP pulse. The drive should assert SC (Seek Complete) as an indication of the completion of the requested recalibration. If the drive asserts SC (LOW), and TRK0 is LOW, the Am9590 will assume that recalibration has failed. In this case, the HDC continues to recalibrate the drive by issuing STEP pulses until Track 0 is reached (TRK0 becomes HIGH).

(MON) Motor On–In Floppy Mode, this output provides Motor-On signal for the floppy-disk drive. Whenever a floppy-disk drive is selected and MON is asserted HIGH, this turns on the spindle motor of the selected drive(s). The HDC waits for a programmed period before attempting any read or write access to the drive (see Drive Parameter Block description).

9590

Command Complete—In ESDI Mode, this input indicates to the HDC whether the drive has completed a command or if a new command may be issued. Command Complete goes inactive upon the reception of the first Command Data bit. It stays inactive until the command has been executed. Command Complete is also monitored after a head change during disk-data transfers. This allows the drive to have any head-settle time that is required.

(HDSEL$_4$) Head Select 4—For Special interface this is the second-most significant bit of the head number.

TRK0/TACK Track0/Transfer Acknowledge (Input, Active HIGH)

(TRK0) Track 0—The selected drive must assert TRK0 whenever the head is positioned over the outer-most track (Track 0). This is the only hardware indicator that the head is positioned over a specific track. This input is sampled only when the HDC is performing a drive restore operation. Here, the HDC provides single STEP-pulses (DIRIN LOW), waits for SC to go inactive (LOW), returns to active (HIGH), and then samples TRK0. Whenever TRK0 is asserted, the HDC assumes that the heads have restored to Track 0.

9590

(TACK) **Transfer Acknowledge**—For the ESDI interface, this pin, with TRQ (Transfer Request), handles the asynchronous handshake for the serial command transfer between Am9590 and ESDI hard-disk drive.

RWC/HDSEL$_3$ Reduced Write Current/ Head Select 3 (Output, Active HIGH)

(RWC) Reduced Write Current—RWC indicates that the head is over an inner track where the write current should be reduced. RWC is activated whenever the current track number is greater than, or equal to, the RWC track parameter, specified in the Drive Parameter Block for each drive. No internal processing of RWC takes place in the HDC. RWC operation is similar to that of Precompensation Enable (PCEN). If RWC is used to control the write current, the write current should be reduced when RWC goes active (HIGH). A programmable option bit within the Drive Parameter Block configures this output.

(HESEL$_3$) Head Select 3—This pin provides Bit 3 of the head number.

HLD/HDSEL$_2$ Head Load/Head Select 2 (Output, Active HIGH)

(HLD) Head Load—For floppy drives, this pin provides the Head Load signal. SC (Seek Complete) is sampled eight clocks after the assertion of HLD. If SC is LOW, the HDC waits for it to go HIGH. If SC is HIGH, the HDC assumes that the heads are already loaded.

(HDSEL$_2$) Head Select 2—For hard-disk drives, this pin provides Bit 2 of the head number.

HDSEL<1:0 Head Select (Output, Active HIGH)

These are the two lower-order bits of the head number selected.

INDEX Index (Input, Active HIGH)

INDEX marks each revolution of the disk. INDEX should be valid as long as $\overline{\text{DREADY}}$ is asserted. The HDC uses INDEX to keep track of the number of complete disk revolutions encountered during disk I/O operations and/or to indicate the physical beginning of the track. Only the leading (rising) edge of INDEX is significant. Depending on the drive parameters programmed, the first sector might begin before INDEX has gone inactive (LOW).

AMC/AMEN Address Mark Control/ Address Mark Enable (Output, Active HIGH)

(AMC) Address Mark Control—The HDC asserts AMC in conjunction with RG or WG, to command the external data separator to generate address marks (write operation), or to search for address marks (read operation). In either operation the data separator acknowledges the completion of the requested operation by asserting AMF. In write cycles, this signal indicates that the address mark has been generated. The type and length of the address mark is completely transparent to the HDC. The data separator may, therefore, generate any address mark. In read cycles, AMF indicates that an address mark has been found.

9590

(AMEN) Address Mark Enable—For ESDI interface, this pin causes the ESDI drive to either write an address mark (in conjunction with WG) or to search for an address mark (no RG activated).

AMF/SECT Address Mark Found/Sector (Input, Active HIGH)

(AMF) Address Mark Found—In ST506, or floppy mode the data separator asserts AMF, in response to AMC, to acknowledge that an address mark has been generated (write cycle) or found (read cycle). In ESDI mode (soft-sectored) the disk drive generates AMF. The AMF signal must be asserted in the RD/REFCLK cycle after the data separator has put out the last address mark bit (write cycle), or in the RD/REFCLK cycle when the data separator provides the first data bit after the address mark (read cycle).

9590

(SECT) Sector—For hard-sectored ESDI drives this signal indicates to the Am9590 the start of a new sector.

RG Read Gate (Output, Active HIGH)

RG indicates that a disk-read operation is in progress. It commands the Phase-Locked Loop (PLL) of the data separator to lock the RD/REFCLK to the serial read data from disk. This output changes synchronously with RD/REFCLK.

WG Write Gate (Output, Active HIGH)

WG indicates that a disk-write operation is in progress. It commands the data separator to lock the RD/REFCLK to a constant frequency source (e.g., crystal oscillator) to provide a stable reference clock for the write operation. This output changes synchronously with RD/REFCLK.

RD/REFCLK Read/Reference Clock (Input)

RD/REFCLK is a TTL-compatible clock input that controls the operation of the data section of the HDC. This clock samples the read data (read clock) and strobes out write data (reference clock). It is assumed to be valid 16 system clocks (CLK) after drive selection acknowledge (DREADY) is received, and must remain valid until $\overline{\text{SELEN}}$ is deasserted. While valid, this clock should be free from any glitches (the specified clock HIGH and LOW widths must not be violated).

WRCLK Write Clock (Output)

This is the reference clock output for ESDI write operations. It is inverted from, and synchronous to, the reference clock input (RD/REFCLK). WRCLK must not be connected to the Am9582 write clock input.

RDDAT Read Data (Input, Active HIGH)

RDDAT is the NRZ (Non-Return to Zero) disk data input. The HDC samples RDDAT with the rising edges of RD/REFCLK.

WRDAT Write Data (Output,Active HIGH)

WRDAT is the NRZ disk data output. Transitions occur on the rising edge of RD/REFCLK.

ECCERR External ECC Error (Input, Active HIGH)

When using the external ECC option, this input is asserted when the external ECC logic finds an error. During a read operation, the Am9590 samples ECCERR at the end of Postamble 2 to determine if an error has been detected by the external ECC logic. This input should be always grounded except for a data field read operation.

FAM$_0$ /ECC$_0$ Floppy Address Mark 0/ External ECC Control 0 (Output, Active HIGH)

FAM$_1$ /ECC$_1$ Floppy Address Mark 1/ External ECC Control 1 (Output, Active HIGH)

When using the external ECC option, the outputs ECC$_0$ and ECC$_1$, in conjunction, provide status control for the external ECC logic. These dual-function lines either control external ECC (hard-disk format, external ECC enabled) or indicate the type of address mark to be used (double-density floppy format, AMC HIGH). The four states are encoded as follows:

ECC Controls ECC$_1$,ECC$_0$	State	Comment
00$_B$	Idle	No operation in the external ECC.
01$_B$	Reset	External ECC should reset and prepares itself for an operation.
11$_B$	Generate	Whether reading or writing, the external ECC should strobe data in and generates a check-sum.
10$_B$	Check	When reading, this state indicates that the ECC should accept the check-sum from the disk. When writing, it should gate the check-sum to the disk.

These states always proceed in the Gray code progression shown above, i.e., 00-01-11-10-00, which can be decoded without glitches. The nominal state of these lines is 00 (Idle).

In Double-Density Floppy Mode, the Floppy Address Mark outputs (FAM 0, FAM 1), in conjunction, tell the data separator to generate an Index Address Mark (IXAM) rather than a normal address mark. The two states that can be encoded are as follows:

00 = Index Address Mark (IXAM)
10 = Data Field or Header Address Mark (DAM, IDAM)
In ST506 and ESDI, these will be 10.

DRIVE INTERFACE PIN CROSS REFERENCES

				9590	
Pin Name	**Pin #**	**Floppy**	**ST506**	**ESDI (Serial)**	**SMD**
$\overline{\text{SELEN}}$	68	$\overline{\text{SELEN}}$	$\overline{\text{SELEN}}$	$\overline{\text{SELEN}}$	$\overline{\text{SELEN}}$
DRSEL$_1$	66	DRSEL$_1$	DRSEL$_1$	DRSEL$_1$	DRSE$_{L1}$
DRSEL$_0$	67	DRSEL$_0$	DRSEL$_0$	DRSEL$_0$	DRSEL$_0$
$\overline{\text{DREADY}}$	64	$\overline{\text{DREADY}}$	$\overline{\text{DREADY}}$	$\overline{\text{DREADY}}$	$\overline{\text{DREADY}}$
WRPROT/ATT	63	WRPROT	WRPROT	ATT	WRPROT
FAULT	62	FAULT	FAULT	FAULT	FAULT
SC/CSD	61	SC	SC	CSD	SC
TRK0/TACK	65	TRK0	TRK0	TACK	———
D_{I3}	56	PCEN	PCEN	———	TCLK
D_{I2}	59	STEP	STEP	TRQ	TDATA
D_{I1}	58	DIRIN	DIRIN	CD	HDSEL$_5$
D_{I0}	57	MON	RTZ	CC	HDSEL$_4$
RWC/HDSEL$_3$	51	RWC	RWC/HDSEL$_3$	HDSEL$_3$	HDSEL$_3$
HLD/HDSEL$_2$	53	HLD	HDSEL$_2$	HDSEL$_2$	HDSEL$_2$
HDSEL$_1$	54	———	HDSEL$_1$	HDSEL$_1$	HDSEL$_1$
HDSEL$_0$	55	SIDE	HDSEL$_0$	HDSEL$_0$	HDSEL$_0$
INDEX	39	INDEX	INDEX	INDEX	INDEX
AMC/AMEN	49	AMC	AMC	AMEN	AMC/AMEN
AMF/SECT	50	AMF	AMF	AMF/SECT	AMF/SECT
RG	47	RG	RG	RG	RG
WG	48	WG	WG	WG	WG
RD/REFCLK	44	RD/REFCLK	RD/REFCLK	RD/REFCLK	RD/REFCLK
WRCLK	52	———	———	WRCLK	WRCLK
WRDAT	42	WRDAT	WRDAT	WRDAT	WRDAT
RDDAT	41	RDDAT	RDDAT	RDDAT	RDDAT
ECCERR	40	———	ECCERR	ECCERR	ECCERR
FAM$_1$/ECC$_1$	46	FAM$_1$	FAM$_1$/ECC$_1$	ECC$_1$	ECC$_1$
FAM$_0$/ECC$_0$	45	FAM$_0$	ECC$_0$	ECC$_0$	ECC$_0$

1.3 FUNCTIONAL DESCRIPTION

The HDC supports two interfaces as shown in the block diagram. The system interface (see Figure 1-8) communicates with the host CPU and system memory. The disk interface controls the data separator and the disk drives.

System Interface

The HDC is designed for easy interfacing to most 8-bit or 16-bit, multiplexed or demultiplexed, synchronous or asynchronous, microprocessor buses. A strap pin programs the system interface for either byte (8-bit) or word (16-bit) mode. In Slave Mode the host CPU can access the internal registers of the HDC. In Master Mode, the on-chip DMA controller controls the system bus.

DMA Controller

The on-chip DMA controller provides the HDC with the ability to execute complex disk I/O operations without host CPU intervention. The DMA controller scans the command chain stored in system memory, updates the Status Result Area when errors occur, and transfers the data between the internal sector buffers and system memory. Data may be stored in non-contiguous memory, for example, to support linked-list data storage in word processing systems.

The DMA controller generates 32-bit linear addresses to directly access up to 4 GBytes of system memory. For multiple bus-master systems, DMA transfers can be throttled to dedicate only a certain portion of the system bus bandwidth to the HDC. The Mode Register (Figure 1-17) specifies DMA burst length and dwell. DMA bursts can be preempted by removing Bus Acknowledge (BACK). The HDC can insert a programmable number of software Wait States into DMA bus cycles. Additionally, hardware Wait States can be added via the $\overline{\text{READY}}$ input. The HDC updates the upper address word (A_{16}–A_{31}) when there is a carry out of the lower 16 address bits.

User Registers

The Mode Register defines the operation of the DMA controller. The Status/Command Register controls the basic operation of the HDC itself. The Next Block Pointer (NBP) Register links to the first Input/Output Parameter Block (IOPB) of the command chain. The Status Result Pointer Register and the Status Result Length Register define the Status Result Area where the HDC stores the status for each IOPB.

Main Sequencer

The Main Sequencer translates the high-level system commands into the control signals for the various independent functional sections of the HDC. This 16-bit processor relieves the system CPU of complex data manipulations.

Drive Parameter RAMs

The Drive Parameter RAMs store the specification parameters for drives that adapt the HDC to any combination of disk recording schemes. The contents can be altered anytime with a single IOPB. Once loaded these parameters allow disk commands to be independent of the drive type or track format. For example, the write command is the same whether it is for a double-density floppy-disk drive or a Winchester hard-disk drive.

Error Checking

The HDC features two powerful Reed-Solomon error correcting codes, as well as the industry standard error detection code, CRC-CCITT. It also supports a user-definable, external error correction scheme. Along with programmable retry and correction attempt policy, the HDC allows maximum control of data integrity.

Sector Buffers

The HDC transfers data to or from disk without adding time constraints on the system bus bandwidth. The two internal sector buffers (Figure 1-10) can be filled or emptied at any speed w/o interfering with data transfers between sector buffers and the disk. The two internal sector buffers are toggled for zero-sector interleave disk data operations.

While one sector buffer is filled with data from disk, the other buffer is emptied by the DMA controller. Physically, contiguous sectors on a track can thus be read or written on the fly (during one revolution of the disk).

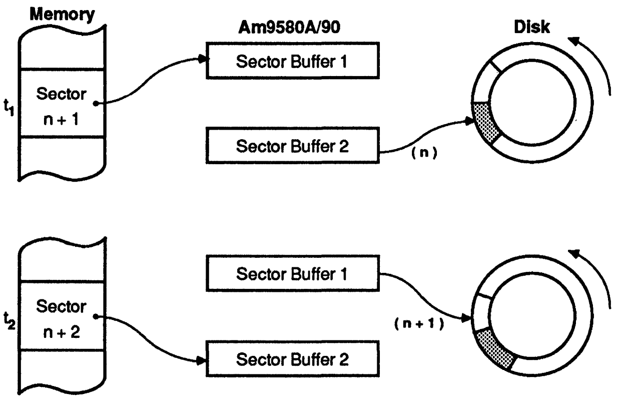
Disk Control Interface

9580A

The Am9580A has a disk control interface which provides all the control lines to directly handle the ST506/412 and double-density floppy-disk interface. Other drive interface standards can be supported using external circuitry. The controller can support any combination of floppy-disk and hard-disk drives. The drive parameters can be individually specified in one out of four drive parameter blocks.

9590

The Am9590 has a disk control interface which provides all the control lines to directly handle the ESDI, ST506/412 and double density floppy-disk interface. Other drive interface standards can be supported using external circuitry. By using the ST506/412 and Floppy options, the Am9590 issues step pulses to position the heads to the desired cylinder. Step width and dwell times, as well as head settling times, are programmable. If the ESDI interface is selected, the HDC will automatically generate SEEK and RESTORE commands to the ESDI drive using the serial communication link (Figure 1-11). Other ESDI commands like REQUEST STATUS, REQUEST CONFIGURATION, can be issued by the CPU using the ESDI CHANNEL IOPB of the Am9590. The controller can support any combination of floppy-disk, ESDI and ST506 hard-disk drives. The drive parameters can be individually specified in one out of four drive parameter blocks.



9480A 1-10

Figure 1-10 Dual On-Chip Sector Buffers

The HDC can perform implied and overlapped seeks. When the implied seek option is selected, the HDC automatically causes the appropriate seeks when issuing a read or write command. If this option is disabled, the seek operation has to be performed externally.

When the overlapped seek option is selected, several drives can seek in parallel, thus minimizing the seek overhead in multiple disk-drive systems. After the HDC has issued a seek command to one drive, and while this drive performs the seek, the HDC scans subsequent IOPBs for commands requiring seeks on other drives. If the HDC finds such commands, it issues seek commands to these drives to make them seek in parallel. After the first drive has finished the seek operation, the HDC resumes execution of the command chain.

Disk Data Interface

9580A

The Disk Data Interface of the Am9580A handles the serial data input and output to the disk drive. It controls the Address Mark handshake with the data separator as well as the optional external ECC logic. Operating asynchronously to the other blocks of the device, the Disk Data Interface is driven by the Read/Reference Clock (RD/REFCLK) generated by the data separator. The Disk Data Interface converts the data stored in the sector buffer into a serial bit-stream for the disk or it deserializes the incoming bit-stream to be loaded into one of the sector buffers. Non-data-information such as the header (sector ID field), pads, gaps, preambles, and postambles, is conditioned according to the parameters stored in the Drive Parameter RAMs to meet the defined recording standard.

9590

The Disk Data Interface of the Am9590 handles the serial data input and output to the disk drive. It controls the Address Mark handshake for soft-sectored ESDI drives as well as hard-sectored drives. In ST506/412 and Double-Density Floppy Mode it also controls the data separator. Operating asynchronously to the other blocks of the device, the Disk Data Interface is driven by the Read/Reference Clock (RD/REFCLK). This clock is either driven by the disk drive (ESDI) or by the data separator (ST506, Floppy Disk). In ESDI mode the device provides a synchronous Write Clock output. The Disk Data Interface converts the data stored in the sector buffer into a serial bit-stream for the disk or it deserializes the incoming bit-stream to be loaded into one of the sector buffers. Non-data information such as the header (sector ID field), pads, gaps, preambles, and postambles, is conditioned according to the parameters stored in the Drive Parameter RAMs to meet the defined recording standard.

IOPB Command Structure

The HDC features a high-level data and command structure (Figure 1-12). The basic unit of this command structure is the Input/Output Parameter Block (IOPB). The host CPU creates IOPBs in system memory to pass control and status information to the HDC. The HDC fetches these IOPBs by using the on-chip DMA controller. Each IOPB specifies one disk command and contains all parameters needed to execute it (Figure 1-13). To start execution of an IOPB, the host CPU loads the address of the IOPB into the Next Block Pointer Register and writes the "Start Chain" command by programming the Status/Command Register. After the IOPB is executed, the HDC reports the status

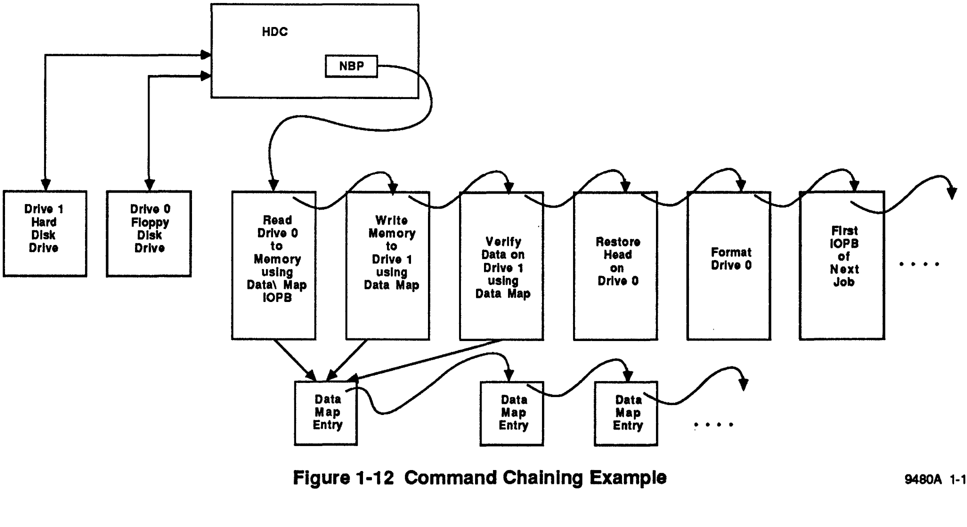


Figure 1-12 Command Chaining Example

9480A 1-12

information and waits for further instructions. The host CPU can examine the Status/Command Register for information about the command termination. The CPU can also get status from the Status Result Block in memory if an error occurs.

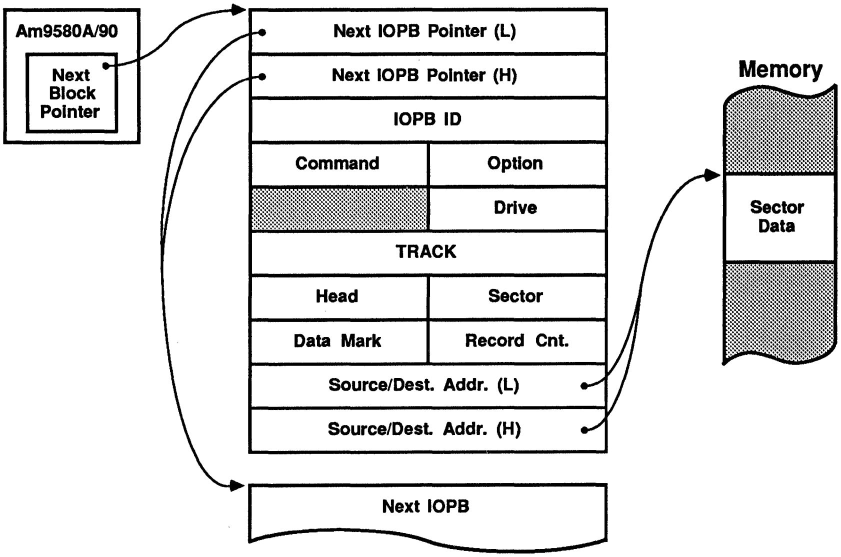
As an option, IOPBs may be put together in a linked-list format which the HDC can interpret sequentially. With this structure, a complex list of disk commands can be set up and then executed by the HDC without CPU intervention. The CPU is then totally free from any disk control processing. For example, the host CPU might set up a list of commands for the HDC to copy an entire floppy disk to a hard disk and verify that the data has been copied correctly. Upon verification, the HDC reformats the floppy disk, all without host CPU intervention.

An IOPB command chain is basically a queue of jobs waiting for HDC execution. This offers a predefined and efficient structure for the operating system to handle its disk I/O. The ID field of the IOPB provides the linkage between a particular disk command and the user process that made the disk request. The jobs can thus be placed in the HDC job queue and then ignored by the operating system unless an error occurs. All the information required to retrace an error is provided by the HDC Status Result Block.

Since the HDC manages the disk job queue, it can look ahead in the queue to overlap several time-consuming operations. Head movement (seeking) can require a major portion of the disk access. Since the HDC controls up to four drives, it can perform an IOPB operation on one drive, while it is executing seeks for future IOPBs on the other drives. This eliminates the seek-time overhead when those subsequent IOPBs are finally executed (Figure 1-14).

Data Mapping

Sector data to be transferred to or from the disk may be stored in non-contiguous system memory using the data mapping option (Figure 1-15). Definable portions of a disk file can be written or read from separate areas of memory on a byte-by-byte basis. The Data Map defines the linked-list data structure. The Data Map option is processed by the HDC, while the disk is in operation, so that data maps can be handled without affecting data transfer rate. Virtual memory systems can employ this feature to arrange memory pages directly with the HDC and eliminate the time-consuming task of moving data blocks to the appropriate locations.



9480A 1-13

Figure 1-13 IOPB Address Sequence

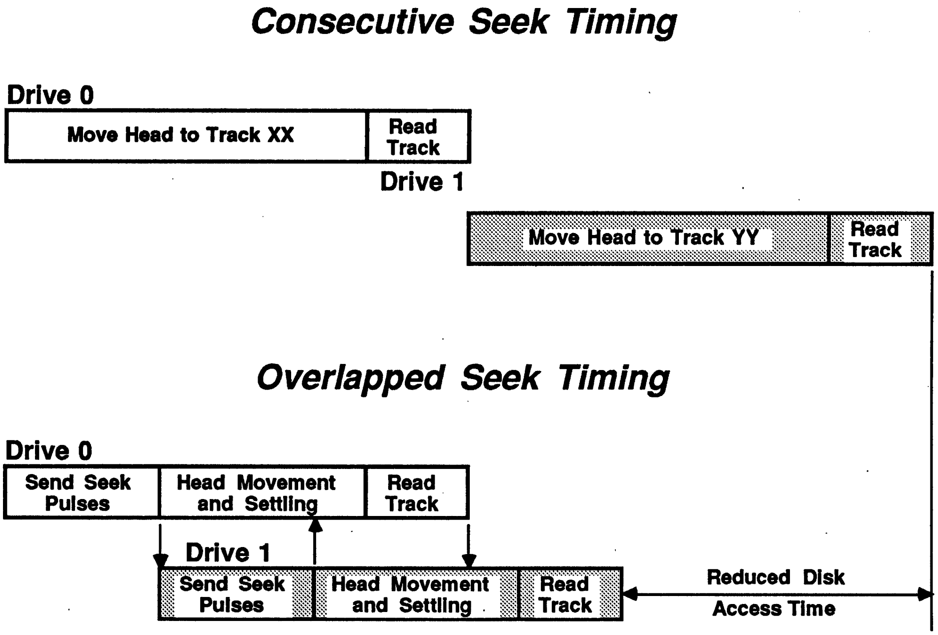
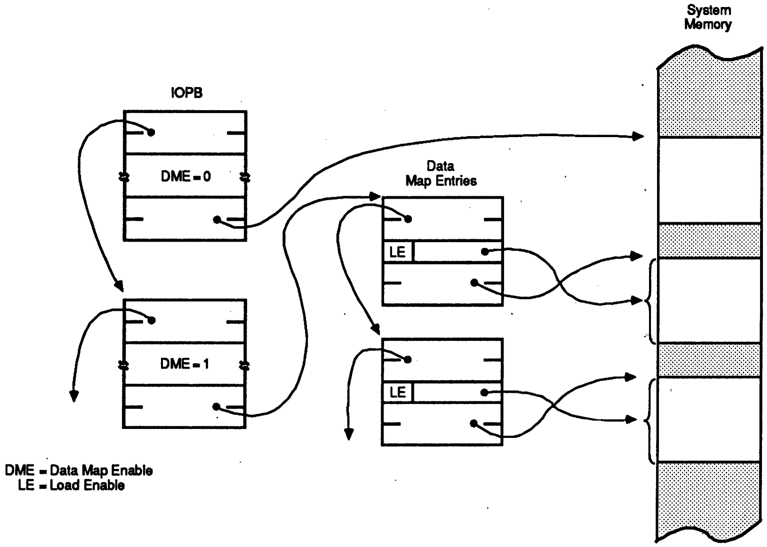


Figure 1-14 Seek Modes

9480A 1-14



9480A 1-15

Figure 1-15 Data Mapping

Status Result Blocks

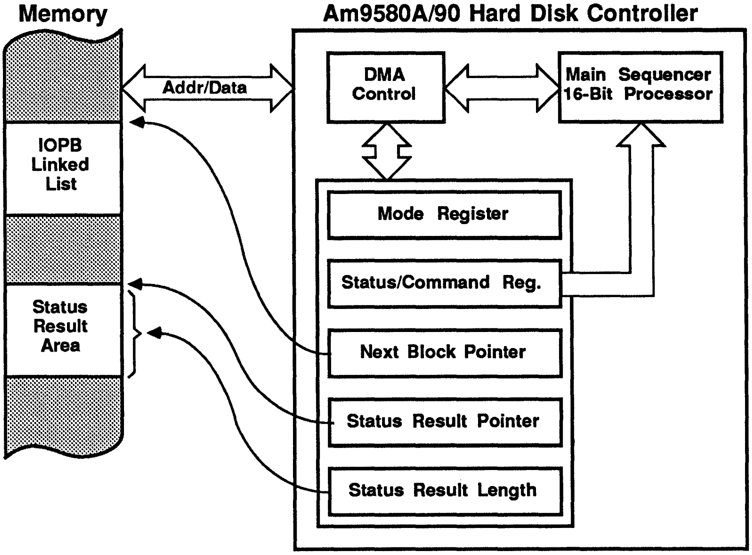
When the HDC finds that an IOPB has caused an error, it writes a Status Result Block (SRB). Errors might be caused by invalid command codes, disk read and write errors, and seek or memory time-outs. Since the SRB contains the ID number for the IOPB which caused the error, the operating system can determine which disk I/O job caused the error and report this to the user. Depending upon the type of error and what policy has been selected, the HDC may continue with the IOPB chain automatically, or wait for the host processor to tell it whether to start over or continue. The SRBs contain all the specific information required to find the exact location of the error and to make recovery as complete as possible.

Registers

When the IOPB command chain has been set up, the Next Block Pointer Register of the HDC is set to point to the first IOPB in the chain. Writing a Start Chain command to the Status/Command Register causes the HDC to copy the first IOPB into its internal memory. The Next Block Pointer always point to the current command in the IOPB chain. Status/Command Register also reports HDC error and status codes (such as memory time-outs, IOPB option results, and other information related to the internal operation of the HDC).

The Status Result Pointer points to an area of contiguous memory (Status Result Area) reserved for Status Result Blocks (SRBs). The length of this memory block is defined by the Status Result Length Register, which specifies the number of SRBs (each SRB is 10 bytes). The error handling scheme of the operating system can manipulate this as needed to coordinate disk use.

The Mode Register controls how much shareof the system bus bandwidth is allocated to the HDC, and how much is left to meet other system requirements.



9480A 1-9

Figure 1-16 Software Interface

Register Description

Four registers control the operation of the HDC. These registers can directly be accessed by the host CPU in byte or word mode. The addresses are shown in Figure 1-16a.

Mode Register

Bit assignments of this 16-bit read/write register are shown in Figure 1-17. This register may be read/written at any time. A hardware or software reset clears all bits in this register. During normal operation, the HDC only reads this register.

$\overline{CS}$	A_3	A_2	A_1	Register Accessed
L	L	L	L	Status/Command Register
L	L	L	H	Mode Register
L	L	H	L	Next Block Pointer (low word)
L	L	H	H	Next Block Pointer (high word)
L	H	L	L	Status Result Pointer (low word)
L	H	L	H	Status Result Pointer (high Word)
L	H	H	L	Status result Length
L	H	H	H	Reserved
H	X	X	X	No Register Accessed

9480A 1-16

Figure 1-16a Register Address

Burst Programming

The upper half of this register controls the bus occupancy of the HDC. The DMA Burst Length specifies the maximum number of bytes the HDC is allowed to transfer per DMA burst. After the burst is completed the HDC stays off the bus (BREQ stays inactive) for a number of programmable clock cycles (DMA Dwell Time). After this time interval has expired the HDC may request the bus again. The maximum DMA burst length is equal to the programmed sector size (maximum 512 bytes). If the burst length is set to one byte, the HDC will only transfer bytes, independent of whether it is programmed for a byte or word interface. The programmed Dwell time is the minimum time. Burst length is specified in powers of two: 1,2,4,8,...512. Dwell time is specified in multiples of 16 clock pulses.

DMA
Dwell Time: 16 • N (where N is 4-bit DMA Dwell Time)

DMA
Burst Length: 2^N (where N is 4-bit DMA Burst Length)

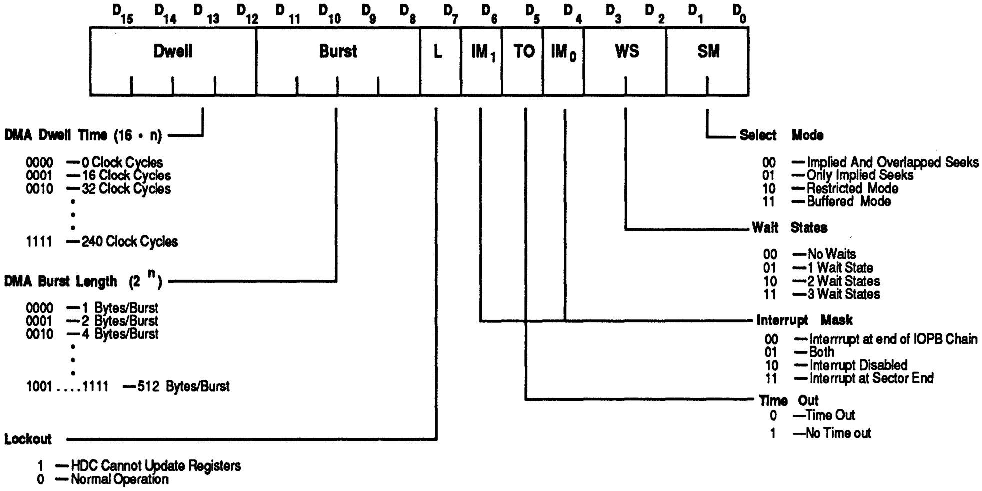


Figure 1-17 Mode Register

9480A 1-17

Wait States

The Wait State field determines the number of software Wait States to be inserted into the normal four clock cycle bus transfer. First, the HDC inserts the programmed number of Wait States, then waits until the $\overline{\text{READY}}$ input is activated (hardware Wait States). If no acknowledgement is received within 512 clock cycles the memory cycle is aborted and a memory time-out error is generated by setting the Controller Fault bit. The Controller Fault Type indicates whether the time-out occurred when fetching an IOPB ("IOPB Time-Out"), when transferring a SRB ("SRB Time-Out"), or when transferring data ("Data Time-Out"). On a data time-out, the HDC also generates an "Data Time-Out" SRB, which indicates exactly where the time-out occurred. All time-outs are resumable; however, the HDC reexecutes the entire block memory transfer, e.g., the entire IOPB or SRB, or the data block (max. size: one sector) with the start address and length indicated in the "Data Time-Out" SRB.

Lockout

Some registers require multiple CPU accesses to be completely reloaded (e.g., two-word pointer registers). Lockout ensures that the HDC does not access a partially updated register. Once the Lockout bit is set, the register set is frozen. If the Lockout bit is set while the HDC is executing a command, the HDC will stop when it tries to access one of the users accessible registers (e.g., the HDC tries to access the Status Result Pointer register to issue a SRB). The latest point where the HDC stops is when it tries to access the Next Block Pointer register to load the next IOPB. The Lockout bit is set to zero on a hardware or software reset. It needs to be set only if the HDC is not IDLE when changing registers.

Interrupt Mask

The Interrupt Mask bit enables/disables the interrupt output. The HDC always interrupts the CPU after it has completed the initialization procedure, executed after Reset. This interrupt cannot be disabled. Loading the Mode Register, subsequently with the Interrupt Mask bit set High, disables all further interrupts. The CPU is expected to poll the upper-half of the Status/Command register to determine when the HDC stops. The CF-bit is set when the HDC stops. If the Interrupt Mask bit is reset Low, the HDC will interrupt the CPU every time it enters the "Idle" state (see Status/Command Register description). The Interrupt Mask bit is set to zero on a hardware or software reset (Interrupts are enabled).

Seek Modes

The Seek Mode bits select the disk interface mode: Implied and Overlapped Seeks, Only Implied Seeks, Restricted Mode, or Buffered Mode.

Implied and Overlapped Seeks

In this mode the HDC uses all disk control inputs and outputs. *Implied Seeks* are seek operations which are automatically performed if a track change is required to execute the requested command. This means that commands do not need to be preceded by the SEEK IOPB. *Overlapped Seeks* are seek operations where the HDC scans subsequent IOPBs to identify IOPBs requiring seeks. The HDC will initiate these seek operations in advance, to seek multiple drives in parallel, thereby improving the overall system performance. These overlapped seeks are performed wherever possible.

The HDC scans the IOPB chain until it finds an IOPB with the "Wait" option bit set, until all four drives are busy seeking, or until it reaches the end of the IOPB chain. If not all four drives are actively used, the HDC will not necessarily find an IOPB for a particular drive, and therefore, will scan to the end of the IOPB chain. To limit the bus overhead, which is involved every time the HDC scans the IOPB chain, the user should set the "WAIT" option bit when operating with long IOPB chains. If the HDC encounters an IOPB with the "WAIT" option bit set, it will terminate executing the IOPB chain at this IOPB, and generates a CFT "WAIT STOP". If overlapped seeks are enabled, the HDC will re-scan the IOPB chain when it starts a new IOPB.

Only Implied Seeks

In this mode the HDC uses all disk interface inputs and outputs; however, it does not perform overlapped seeks.

Restricted Mode

The HDC controls all disk interface inputs and outputs except STEP, DIRIN, and RTZ. Neither implied nor overlapped seeks are performed. All head positionings are done external to the HDC. The disk status lines are still being monitored. The HDC waits for SC to be asserted (similar to above modes).

It is only in restricted mode that the HDC strobes out the current track number serially on the STEP output (see Figure 1-52). The STEP output becomes the serial data output port. The most significant bit of the 16-bit track number is put out first. The PCEN output provides the shift clock. STEP should be sampled with the rising edge of PCEN, which is pulsed 16 times. The HDC provides a new track number while $\overline{\text{SELEN}}$ is active or inactive. To flag the external logic that the track number has updated, the HDC generates a falling edge on $\overline{\text{SELEN}}$. The HDC then samples SC and waits until the heads are positioned on the new track. After SC is asserted, the HDC performs the disk data access (read or write operation).

Buffered Mode

In Buffered Mode, the HDC is only a data buffer or data formatter. It does not perform any drive selection, head positioning, head selection or drive status control. No operations on the drive control bus will be performed. However, the HDC still monitors the FAULT input to allow abortion of the current disk data transfer. In this mode, the HDC controls the data transfer (RDDAT, WRDAT, RD/REFCLK, WRCLK, RG and WG), the data separator and ECC operation (AMC, AMF, INDEX, FAM$_1$/ECC$_1$, FAM$_0$/ECC$_0$ and ECCERR).

Time Out Mode

If the Time Out bit in the mode register is reset, the HDC will perform a Bus Time Out whenever the system bus does not activate the $\overline{\text{READY}}$ line within 512 system clocks after the HDC starts a memory transfer. If this bit is set, no Time Out will be performed. The device will wait for $\overline{\text{READY}}$ indefinitely.

The usual choice for this mode should be to turn the Time Out function on. Switching this function off can cause the HDC to hang up waiting for $\overline{\text{READY}}$ to become active. This mode should only be used if the answering device (e.g., a SCSI interface) is known to be slow.

The Mode Register may be altered at any time and any changes will take effect on the next DMA transaction. (Note: A DMA transaction may consist of several burst and dwell sequences.)

STATUS/COMMAND REGISTER

The Status/Command Register (Figure 1-18) is the main control register of the HDC; it may be accessed anytime. The lower byte contains the 2-bit command field which controls the basic operation of the HDC. With this control field, the IOPB chain operation can be started, resumed, or stopped. The upper byte of this register is the status half; here the HDC reports when it has finished or aborted the IOPB chain. The execution of commands modifies the content of this register. A hardware or software reset initializes all bits of the Status/Command Register to zero.

The lower half of this register is readable and writable. The upper half is only readable. A write access to the upper half is allowed, but it will not alter the status. To ensure compatibility to future revisions, all reserved or "read only" fields should be set to zero when loading the register. The state of reserved bits when reading the register is not defined.

Command

When the command field is written by the host CPU, it causes the HDC to enter the programmed state. By reading this field, the host CPU can determine the current state of the HDC. (Note that there is a certain latency (up to 20 system clocks) in updating the command field: If the CPU writes a new command to the Status/Command Register, it might not be able to immediately read the new value back. However, eventually the HDC will update this field, which acknowledges that the HDC has accepted the new command. The user must be aware of this behavior of the command field, especially when performing register diagnostics.

D_1	D_0	RD	WR	Command	Status
0	0	L	H		Idle
1	X	L	H		Busy
0	1	L	H		(HDC does not allow access)
0	0	H	L	Forced Idle	
0	1	H	L	Software Reset	
1	0	H	L	Resume Chain	
1	1	H	L	Start Chain	

Idle

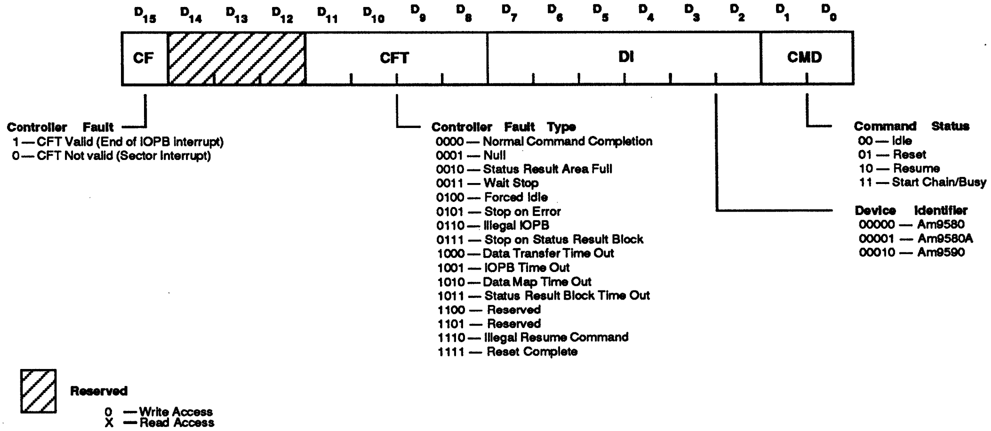
The HDC is not performing any function. The HDC sets the CF-bit (Controller Fault) when it enters the Idle state. At this time, the CFT field (Controller Fault Type) gives the result of the action performed; any command (Reset, Start Chain, or Resume Chain) can be written to this command field while the HDC is idle. A Forced Idle command aborts the IOPB chain currently being executed. The device will finish the current IOPB and stops further executions of the command chain. A Resume Chain command allows the operation to continue.

Reset

The software reset is executed immediately. On completion of a software reset, the HDC enters the Idle state. Any access of a register while in the Reset state holds READY inactive until reset is completed, which causes the CPU to wait. When the initialization procedure is completed, the HDC enters the Idle State. CF is set and CFT indicates "Reset Complete". A software reset is equivalent to a hardware reset.

Resume Chain

The HDC resumes execution of the IOPB chain where it was interrupted, if the HDC was forced to abort due to certain conditions in an IOPB and the error was not fatal (i.e., unrecoverable). It allows the HDC to continue, cleanly, in the middle of a command which was interrupted by disk errors, user options, or interface time-outs. The HDC does not reexecute the beginning of the IOPB. Issuing a Start Chain command instead of a Resume Chain command forces execution of the interrupted IOPB from the beginning. If a Resume Chain command is issued illegally (without a preceding IOPB chain abort), the HDC will abort the command, enter the Idle state and CFT will indicate an "Illegal Resume Command". If the HDC receives an Idle command while being in the Resume Chain (Busy) state, the HDC cleanly aborts the command chain. The forced Idle will take affect after the current IOPB has been completed. The HDC completes the current command and then enters the Idle State.



9480A 1-18

Figure 1-18 Status/Command Register

Start Chain

The Start Chain command is the normal "Go" command of the HDC. It starts executing the IOPB chain pointed to by the Next Block Pointer. If the HDC receives the Idle command while being in the Start Chain (Busy) state, the HDC completes the current IOPB and then enters the Idle state.

Busy

This status indicates that the HDC is currently executing an IOPB chain. The Next Block Pointer points to the IOPB which is currently being executed (not affected by overlapped seek operation—IOPB look-ahead). A Reset command will force the HDC to abort the command chain immediately and to transit into the Idle state.

Controller Fault

When this bit is set, the Controller Fault Type field is valid. The CF-bit is set by the HDC when entering the Idle state and reset when the CPU reads the upper half of the Status/Command Register. When CF is set and Interrupt is enabled (Interrupt Mask is reset) the HDC asserts the Interrupt output (INTR). This bit can be polled to determine a command completion.

Controller Fault Type

The Controller Fault Type indicates the status of the HDC upon entering the Idle state. The HDC updates this field only when it enters this state. It is valid while the CF flag is set and the HDC is idle. It is not valid while the CF flag is reset (zero).

The following is a list of Controller Fault Types:

Normal Complete—The IOPB command chain was completed successfully from the last Start Chain or Resume Chain command, however, an individual IOPB in the command chain might have terminated with a fatal error. The user must examine the SRBs generated.

Null NBP—A resume or start chain command was given with a Next Block Pointer equals to zero.

SRA Full—The Status Result Area (SRA) allocated for SRBs has been filled; the Status Result Length register has decremented to zero. The Status Result Pointer register and the Status Result Length register should be updated and the chain execution may be resumed.

Wait Stop—The HDC stopped execution of the IOPB chain after it has finished an IOPB with the W(ait) option bit set.

Forced Idle—The execution was terminated at the end of the current IOPB due to an Idle command which was issued while the HDC was executing the current IOPB.

Stop on Error—Non-recoverable disk error(s) in conjunction with the SE-bit set caused the HDC to terminate chain execution.

Illegal IOPB—The HDC attempted to execute an illegal IOPB (e.g., an IOPB with an undefined command code).

SRB Error—An Status Result Block (SRB) was written when executing an IOPB with the Stop on SRB (SSRB) option set.

Data Time Out—Memory Time Out when transferring (reading/writing) data (such as sector data, Data Map Entries, sector map, drive parameter blocks). The SRB issued ("Data Time-Out") lists the parameter for the data block during which the time-out occurred. If the operation is resumed, the HDC will retransmit the entire data block.

IOPB Time Out—Memory Time Out when loading the current IOPB, or an IOPB Read during Seek Look-Ahead. If the operation is resumed, the HDC will re-read the entire IOPB.

DM Time Out—Memory Time Out while reading Data Map Entries. If the operation is resumed, the HDC will reread the entire Data Map Entry.

SRB Time Out—Memory Time Out when writing a Status Result Block (SRB). If the operation is resumed, the HDC will rewrite the entire SRB.

Illegal Resume—An attempt was made to resume a command which is not resumable.

Reset Complete—Hardware or software reset has been completed.

Any Read operation from the status registers clears a pending Interrupt (Interrupt pin is reset).

The following three registers define where the HDC is to fetch commands and where to store status feedback.

NEXT BLOCK POINTER REGISTER

The Next Block Pointer Register is a 32-bit register which points to the IOPB currently being executed, or that will be executed on a Start Chain or Resume Chain command. The HDC updates it upon IOPB completion pointing to the next IOPB in the chain, or to zero if it has reached the end of the chain. This pointer does not reflect any IOPB look-a-heads while the HDC performs overlapped seeks. If the IOPB chain execution is interrupted (e.g., Wait Stop, see options bits of IOPB), then this register points to the IOPB which caused the interruption. If the IOPB chain terminates without interruptions, this register is zero.

When the HDC is given a Start Chain or Resume Chain command, the pointer loaded into this register must link to the first IOPB in the command chain. It can be assigned any 32-bit value except "0", because "0" indicates the end of the command chain. No IOPB chain can start from address "0" in system memory. This register may only be written while the HDC is in the "Idle" State.

STATUS RESULT POINTER REGISTER

The 32-bit Status Result Pointer register points to the system memory location where the next Status Result Block can be written. It is updated after adding a new Status Result Block to the Status Result Area. It points to the location in the Status Result Area where the next SRB should be loaded. Initially, this register may be set to any 32-bit value including "0". This register may only be written while the HDC is in the "Idle" State.

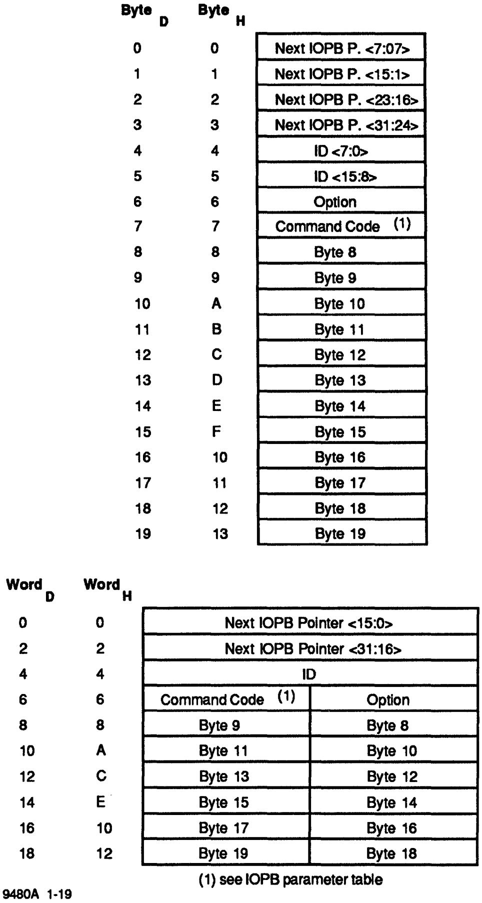
STATUS RESULT LENGTH REGISTER

The 16-bit Status Result Length register defines the size of the Status Result Area in terms of number of Status Result Blocks. Each SRB is 5 words or 10 bytes. This register thus allocates a block of 10*N bytes for the Status Result Area where N is the 16-bit value loaded into this register. The HDC updates the Status Result Pointer and Status Result Length after a SRB has been loaded into the Status Result Area. For each SRB written this register is decremented by one. When this register is decremented to zero, the HDC will abort the IOPB chain, set the CF flag in the Status/Command Register, and the CFT field will indicate a Status Result Area overflow ("SRA Overflow"). Because this register is post-decremented, 2^{16} (65536) blocks are allocated if this register is initially set to zero. This register may only be written while the HDC is in the "Idle" State.

COMMAND DESCRIPTION

All operations of the HDC result from commands which are set up in system memory in IOPBs (I/O Parameter Blocks). The HDC starts interpreting the command list after receiving the "Resume Chain" or "Start Chain" command from the host CPU (see Status/Command Register description). Errors and warnings on command execution are reported by adding Status Result Blocks (SRB) to the Status Result Area. Figure 1-32 cross references the error types (SRBs) and commands (IOPBs).

Each IOPB consists of ten 16-bit words which reside in system memory (Figure 1-19). The HDC always fetches all 10 IOPB words, sequentially. It does not skip "Don't Care" fields. For Seek-Look-Ahead fetches only the first six words are read.



Byte_D	Byte_H	
0	0	Next IOPB P. <7:07>
1	1	Next IOPB P. <15:1>
2	2	Next IOPB P. <23:16>
3	3	Next IOPB P. <31:24>
4	4	ID <7:0>
5	5	ID <15:8>
6	6	Option
7	7	Command Code (1)
8	8	Byte 8
9	9	Byte 9
10	A	Byte 10
11	B	Byte 11
12	C	Byte 12
13	D	Byte 13
14	E	Byte 14
15	F	Byte 15
16	10	Byte 16
17	11	Byte 17
18	12	Byte 18
19	13	Byte 19

Word_D	Word_H		
0	0	Next IOPB Pointer <15:0>	
2	2	Next IOPB Pointer <31:16>	
4	4	ID	
6	6	Command Code (1)	Option
8	8	Byte 9	Byte 8
10	A	Byte 11	Byte 10
12	C	Byte 13	Byte 12
14	E	Byte 15	Byte 14
16	10	Byte 17	Byte 16
18	12	Byte 19	Byte 18

(1) see IOPB parameter table

9480A 1-19

Figure 1-19 IOPB Structure

After one IOPB has been loaded completely, the HDC starts execution. The first two words form a 32-bit linear address (Next IOPB Pointer), that links to the next IOPB in the chain. Set this pointer to zero to terminate the chain (the last or the only IOPB in the chain).

The third word is called the "IOPB Identification" field (ID). This 16-bit integer number is supplied by the user and should be unique to this IOPB. If the HDC issues SRBs for this command, it copies the identification number onto the ID field of the SRB, to identify which SRBs belong to which IOPBs within a command chain. An IOPB may generate any number of SRBs.

The fourth word is broken into two distinct fields: the IOPB command code and the IOPB options byte. The IOPB command code is an 8-bit value which defines the type of IOPB. It is valid for values from 0 to 10_H. All other values are illegal. If the HDC encounters an IOPB with an illegal command code, it terminates execution of the IOPB chain. The Controller Fault bit will be set and the Controller Fault Type indicates "Illegal IOPB". The IOPB option byte contains control bits which select between various, conditional IOPB execution policies (see section "Option Bits").

In the fifth word, only the lower byte is defined consistently for all IOPBs. This byte specifies the number of the disk drive to be accessed. The full byte specifies a value between 0 and 3, to select one of four disk drives. The values 4 to 255 are illegal and should not be used. The remaining byte of this word and all subsequent words carry command specific parameters (see Figure 1-20).

OPTION BITS

One byte in each IOPB contains a set of options applicable to the particular command. Any combination of options may be set.

W—WAIT

0=Disabled: After execution of current IOPB, the HDC continues with the next IOPB.

1=Enabled: After execution of current IOPB the HDC terminates IOPB chain execution. It flags the termination by a "Wait Stop" Controller Fault (see Status/Command Register description). Seek-look-ahead operations will not be performed past any IOPB with this option bit set. A stop on this option is resumable.

		Options								Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
Command	Code	7	6	5	4	3	2	1	0	9	8	11	10	13	12	15	14	17	16	19	18
Read	0C	W	SE	SSRB	0	DME	DM	*	RD	Flag	Drive	Track		Head	Sector	Data Mark	Record Count	Destination <15 ••• 0>		Destination <31 ••• 16>	
Write	0D	W	SE	SSRB	FW	DME	DM	*	RD	Flag	Drive	Track		Head	Sector	Data Mark	Record Count	Source <15 ••• 0>		Source <31 ••• 16>	
Verify	0F	W	SE	SSRB	0	DME	DM	0	RD	Flag	Drive	Track		Head	Sector	Data Mark	Record Count	Source <15 ••• 0>		Source <31 ••• 16>	
Format	07	W	SE	SSRB	0	DME	0	0	0	00	Drive	Track		Head	Pattern	Track Count		Map Pointer <15 ••• 0>		Map Pointer <31 ••• 16>	
Relocate Track	0B	W	SE	SSRB	0	0	0	0	0	00	Drive	Track		Head	00	Alternate Track		Alternate Head	00	0000	
Load Driver Para Block	00	W	SE	SSRB	0	0	0	0	0	00	Drive	0000		00	00	00	00	Source <15 ••• 0>		Source <31 ••• 16>	
Dump Drive Para Block	03	W	SE	SSRB	0	0	0	0	0	0	Drive	0000		00	00	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Read Physical Sector	0A	W	SE	SSRB	0	TV	DM	0	0	00	Drive	Track		Head	Physical Sector	Data Mark	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Read ID	09	W	SE	SSRB	0	AS	0	0	0	00	Drive	Track		Head	Physical Sector	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Load Buffer	01	W	SE	SSRB	0	DME	0	0	TB	00	Drive	0000		00	00	00	00	Source <15 ••• 0>		Source <31 ••• 16>	
Dump Buffer	02	W	SE	SSRB	0	DME	0	0	TB	00	Drive	0000		00	00	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Load Syndrome	04	W	SE	SSRB	0	0	0	0	0	00	Drive	0000		00	00	00	00	Source <15 ••• 0>		Source <31 ••• 16>	
Dump Syndrome	05	W	SE	SSRB	0	0	0	0	0	00	Drive	0000		00	00	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Correct Buffer	06	W	SE	SSRB	0	LD	0	0	0	00	Drive	0000		00	00	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Seek	0E	W	SE	SSRB	0	TV	0	0	0	00	Drive	Track		Head	00	00	00	0000		0000	
Restore	08	W	SE	SSRB	0	TV	0	0	HP	00	Drive	0000		00	00	00	00	0000		0000	
ESDI Channel	10	W	SE	SSRB	0	0	0	SR	WCC	00	Drive	Command		00	00	00	00	Destination <15 ••• 0>		Destination <31 ••• 16>	
Move Data	01	W	SE	SSRB	0	DME	DM	0	TB	00	Drive	# of Set		Destination <15 ••• 0>		Destination < 31 ••• 16 >		Source <15 ••• 0>		Source < 31 ••• 16 >	

* R/WL: 0 = Normal Read/ Write
1 = Read Long/Write Long

Figure 1-20 IOPB Parameters

9480A 1-20

SE—STOP ON ERROR

0=Disabled: After execution of current IOPB the HDC continues with next IOPB.

1=Enabled: The HDC stops the execution of the current IOPB chain if the current IOPB terminates with a fatal error (e.g., Fatal Seek Error, Record Not Found, or Drive Selection Fault). A stop on this option is not resumable.

SSRB—STOP ON STATUS RESULT BLOCK

0=Disabled: After execution of current IOPB the HDC continues with next IOPB.

1=Enabled: The HDC stops the execution of the current IOPB chain if the current IOPB causes a Status Result Block (of any kind) to be written. SSRB is a superset of SE. This means, the HDC will also stop for all cases it would stop if SE is set. So, the stop is only resumable if the SRB was caused by a non- fatal error. The HDC terminates the command chain immediately after issuing the SRB. This may be in the middle of executing the current IOPB.

DME—DATA MAP ENABLE

0=Data Mapping disabled: The Source/ Destination pointer links directly to the data block in system memory. This block is addressed in a linear manner.

1=Data Mapping enabled: The Source/ Destination Pointer links to the first Data Map Entry. The data block is pointed to by the pointer contained in the Data Map Entry. The data may be stored in non-contiguous memory.

DM—DATA MARK

0=Disabled: The HDC uses the default data mark which is $F8_H$ for Hard-disk ST506 Mode, FE_H for ESDI, and FB_H for Floppy-disk Mode. The FORMAT command writes the default data mark. The HDC cannot ignore data marks.

1=Enabled: Any type Data Mark may be used. The Data Mark is specified in the data mark field of the IOPB. This function is available only for normal disk I/O commands (read, write, and verify) and Read Physical Sector.

TV—TRACK VERIFY

0=Disabled: The HDC does not attempt to read a sector ID field to verify whether the heads are positioned correctly.

1=Enabled: The HDC attempts to read a sector ID field to verify that the heads are positioned correctly. If it detects that the heads are mispositioned, it issues a Seek Error SRB. However, it does not restore and reseek the heads automatically. The normal disk I/O commands perform an implied Track Verify. These commands always verify the track before any read or write access to the drive is attempted.

TB—TOGGLE BUFFER

0=Disabled

1=The HDC toggles to the other sector buffer after the current sector buffer has been completely loaded or dumped.

RD—RELOCATE DISABLED

0=If this Bit is reset and the Auto Relocate Bit in the drive parameter block is set, the HDC will automatically handle relocated tracks

1=If this bit is set, auto-relocating is switched off. Therefore, the HDC does not check for a relocated track when starting to read or write. This feature allows higher performance by decreasing the sector skew at the beginning of each track to one.

FW—FLOPPY WAIT

0=Floppy Wait disabled

1=After writing the physically last sector to a track, the HDC wait for 1 ms (assuring 10 MHz clock) before it attempts any seek operation.

AS—ARBITRARY SECTOR

0=Disabled

1=Enabled (see Read ID command description)

LD—LOCATOR DUMP

0=Disabled

1=Enabled (see Correct Buffer command description)

HP—HEAD PARK

0=The Am9590 will try to RESTORE the heads starting at the current track.

1=The Am9590 will step in 5 tracks before attempting to restore. Therefore, even if the heads were parked at a track number lower than 0, the RESTORE command will work properly. The method to move the head to the park position is drive dependent.

9590

WCC—WAIT for COMMAND COMPLETE

0=When sending an ESDI command (ESDI channel IOPB) to the drive the HDC will not wait for COMMAND COMPLETE to go active before executing the next command

1=The Am9590 will wait for COMMAND COMPLETE after sending an ESDI command (ESDI channel IOPB) before it executes the next IOPB.

SR—STATUS RETURNED

0=If an ESDI command is being sent with this bit reset, the HDC will not expect status information to be returned by the disk drive.

1=The Am9590 will wait for status information to be returned by the disk drive. The status information will be transferred into the memory location indicated by the Destination Pointer of the IOPB.

TYPICAL COMBINATIONS

Unconditional Stop

If WAIT is enabled (SE and SSRB are disabled), the HDC will stop execution of the current IOPB chain after the current IOPB has been terminated with or without errors (fatal or non-fatal). The Controller Fault bit in the Status/Command Register is set and the Controller Fault Type indicates a "Wait Stop". The chain execution may be resumed by issuing the Resume Command to the Status/Command Register.

Conditional Stop on Non-Fatal or Fatal Errors

If Wait is disabled and SSRB is enabled (SE may be enabled or disabled), the HDC will stop execution of the current IOPB chain only if the current IOPB has been terminated with fatal or non- fatal errors (one or more SRBs have been generated). The Controller Fault bit in the Status/Command Register is set and the Controller Fault Type indicates a "SRB Error". The chain execution may be resumed by issuing the Resume command to the Status/Command Register.

Conditional Stop on Fatal Errors

If Wait and SSRB are disabled, and SE is enabled, the HDC will stop execution of the current IOPB chain only if the current IOPB has been terminated with a fatal error (e.g., Fatal Seek Error, Record Not Found, or Restore Fault). The Controller Fault bit in the Status/Command Register is set and the Controller Fault Type indicates a "Stop On Error". The chain execution may be resumed by issuing the Resume command to the Status/Command Register.

If one of the above "Conditional Stop" options is selected, but the HDC does not encounter errors (no SRBs are written), it will only stop after executing an IOPB which has a "Next IOPB Pointer" set to zero (first four bytes of IOPB are 00_H). In this case the Controller Fault bit is set and the Controller Fault Type indicates a "Normal Command Completion".

Unconditional Continue

If none of the options (W, SE, or SSRB) is enabled, then the HDC will continue the execution of the IOPB chain independent of whether the current IOPB terminates with or without errors (fatal or non-fatal). The HDC terminates the execution of the IOPB chain after it has executed an IOPB with the "Next IOPB Pointer" set to zero (first four bytes of the IOPB are 00_H). In this case the Controller Fault bit is set and the Controller Fault Type indicates a "Normal Command Completion".

A "Normal Command Completion" does not indicate that all IOPBs have been executed successfully; it only indicates that the IOPB chain execution was not interrupted. The HDC might still have encountered fatal errors which caused it to terminate individual commands. The user must still examine the SRBs. If no SRBs have been generated, then all IOPBs have been executed successfully (without errors). If only

SRBs for non-fatal errors have been generated, that means the HDC has encountered errors and has recovered. The SRBs list keeps a log of how the HDC could recover from the non-fatal error.

NORMAL DISK I/O COMMANDS

The HDC supports three normal disk I/O commands. The multi-sector operation is performed on the drive designated by DRIVE in the IOPB and starting at the desired TRACK, HEAD and SECTOR. The sector numbering may start at 0 or 1 (see Drive Parameter Block description). The commands use the virtual identification of the sectors, i.e., the logical sector number rather than the physical location on the track. RECORD COUNT defines the number of sectors to be transferred. The General Select Byte in the Drive Parameter Block specifies the track/head overflow policy. Optionally, track/head overflow may be disabled, or the track number or the head number may be incremented on sector overflow.

9590

For the ESDI track format the FLAG byte specifies the ID Flag used in the sector header field (Figure 1-36). The Am9590 will compare this parameter to the ID Flag actually read from the desired sector header. If there is a mismatch, the HDC will issue a Flag Mismatch SRB.

SOURCE/DESTINATION ADDRESS is the starting address of the data block in system memory (DME LOW) or the address of the first Data Map Entry (DME HIGH). These commands verify the the head position before attempting the data transfer and hence verify seeks implicitly. If the head position verification fails (Seek Error SRB), the HDC automatically initiates a RESTORE command and a new Seek to the specified track. If this seek fails again, the HDC will abort (Fatal Seek Error SRB). The Data Mark option allows the DATA MARK parameter to be used instead of the default values $F8_H$ (Hard-disk Mode) or FB_H (Floppy-disk Mode).

READ

READ performs a multi-sector data transfer from disk to system memory.

WRITE

WRITE performs a multi-sector data transfer from system memory to disk.

VERIFY

VERIFY compares multi-sector data on disk with data stored in system memory and reports any mismatches (Data Non-Verify SRB).

INITIALIZATION COMMANDS

FORMAT TRACK

The HDC formats the number of tracks specified by TRACK COUNT starting at HEAD and TRACK. The head and track numbers are incremented according to the General Select Byte in the Drive Parameter Block. The sectors are numbered as per order given on the sector map, which is sequentially loaded from system memory starting at MAP POINTER. The HDC does not verify whether the sector map is complete or consistent. The sector map may start at any sector number. However, to support automatic track/head overflow on multi-sector commands, the sector numbers must be in the 0 to N-1, or 1 to N range (see drive Parameter Block description, Start Sector Option bit in Data Select Byte). All numbers in the selected range must be present in the first N bytes of the system memory, starting at the MAP Pointer; however, the requirement is arbitrary. Any sector-interleaving factor is supported by arranging the sector map appropriately. The number of sectors/track is specified in the Drive Parameter Block of this DRIVE. The data field of each sector is filled with PATTERN. The data default mark is $F8_H$ (Hard-disk Mode) or FB_H (Floppy-disk Mode). Since Read, Write, and Verify can specify data marks other than the default, the Format command does not need to write data marks other than the default.

9590

The Am9590 allows the formatting of a track with the ESDI/SMD track format. The ESDI header contains a FLAG byte which can be individually programmed for each sector. A FLAG byte list which is appended to the sector map list in memory allows to specify these bytes during the format procedure. The Am9590 requires each sector map entry to have an accompanying FLAG byte entry (Figure 1-21).

The HDC can format the track with spare sectors (Figure 1-22) or can mask out bad sectors automatically. Here, the spare or bad sector is allocated a large sector number (e.g., FF_H). After the Format Track has been performed, the Drive Parameter Block should be updated to reflect the new sector count, which is the number of sectors formatted minus the number of spare or bad sectors. When performing subsequent multi-record Normal Disk I/O Commands, the HDC will automatically skip any sectors with large sector numbers ("FF").

A good way to improve the performance of a disk drive is to skew the sectors from track to track. Whenever the HDC switches heads or cylinders, executing a multi-sector command that crosses a track boundary, it needs to verify if the newly selected track is relocated or not. Therefore, the device reads the first available header and makes a decision based on the information retrieved. This causes the first sector on the new track to pass without being read or written to. In a zero interleave system, the result is that the HDC has to wait an entire revolution before it can access the next desired sector.

Skewing the sector start from track to track by one or two sectors allows a continues read or write operation of the HDC. An example for a sector skew of two sectors for a disk drive with four heads and 17 sectors/track is shown below

```
HEAD 0: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
HEAD 1: 0F 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
HEAD 2: 0D 0E 0F 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
HEAD 3: 0B 0C 0D 0E 0F 10 00 01 02 03 04 05 06 07 08 09 0A
```

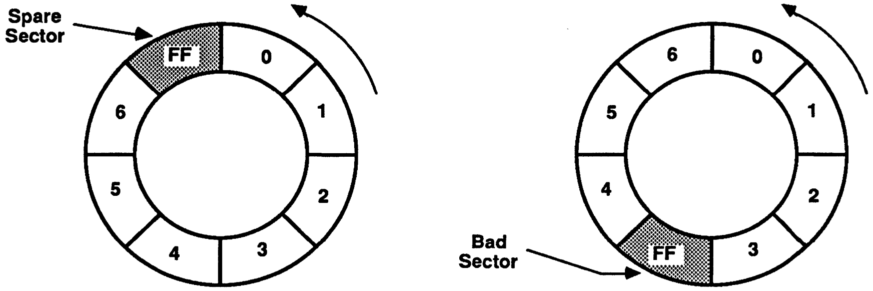
The list for a 32 sector ESDI format should look as follows assuming a FLAG-BYTE of 55_H for all sectors

```
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f,
55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55,
55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55
```

Figure 1-21

9480A 1-21

Bad Sector Relocation



Formatting with Spare Sector **Bad Sector Relocation**

FORMAT:

(1) Each Track has a spare sector. This sector typically is the last sector of the track.
So, the sector map for a track which has no bad sectors is as follows: 00,01,02,03,04,05,06

(2) If the tracks contain a bad sector, the map is changed appropriately: 00,01,02,03,FF,04,05,06

READ/WRITE/VERIFY :

The Drive Parameter Block specifies 7 sectors/track. The HDC will perform normal zero-interleaved disk access. If it encounters sector "F", it will just skip it.

Figure 1-22

9480A 1-22

RELOCATE TRACK

This command will relocate the track/head specified by TRACK and HEAD by reformatting this track/head. The data mark of the ID field is changed from the default FE$_H$ to FD$_H$. The sector number is always 0. The data mark of the data field is F8$_H$ (default value). For subsequent Normal Disk I/O commands the HDC automatically vectors to the specified altemate track/head (ALTERNATE TRACK and ALTERNATE HEAD) if the Auto Vector Enable bit in the Drive Parameter Block is set. The HDC flags the encountering of a relocated track by issuing a "Relocate Track" SRB.

A relocation vector is written to all data fields of a relocated (bad) track. The relocation vector consists of the new track number (ALTERNATE TRACK) and the new head number (ALTERNATE HEAD) (i.e., the track's relocated location). Also, the address data mark of the sector ID fields is changed from FE$_H$ to FD$_H$ to mark that this track has been relocated. Thus, when the HDC encounters three consecutive sector headers with a FD$_H$ data mark, it assumes the track has been relocated. The data mark of the data field is F8$_H$ (Hard-disk Mode) or FB$_H$ (Floppy-disk Mode). The logical sector number is set to zero for all sector ID fields. When executing this command the HDC does not reformat or write to the alternate track.

LOAD DRIVE PARAMETER BLOCK

This command loads a Drive Parameter Block (DPB) for the selected DRIVE into the internal Drive Parameter RAM (see Drive Parameter Block description). This command must be performed once for each drive connected to the HDC. The Drive Parameter Block may be reloaded at any time within the IOPB chain to dynamically alter drive parameters. However, to ensure correct disk read or write operation, the current values defining the sector format (such as DELAY, GAP, PAD, PREAMBLE, and POSTAMBLE length) should be the same as when this track was formatted. If these values are altered, the HDC might not be able to find sector ID fields or data fields. This causes the HDC to issue SRBs such as Data Sync Fault or Index Error.

DUMP DRIVE PARAMETER BLOCK

The Drive Parameter Block for the selected DRIVE is transferred from the internal Drive Parameter RAM to system memory pointed to by DESTINATION ADDRESS. This command may be used for power-on diagnostics.

MARGINAL DATA RECOVERY COMMANDS

The marginal data recovery commands allow the user to retrieve data from the disk, in case significant portions of the track (such as the sector ID field) are damaged so that normal disk I/O command cannot retrieve the data. The user must realize that these commands can lead to false results if not used properly.

READ PHYSICAL SECTOR

This command allows the user to recover a marginal data field which is unrecoverable by normal disk error recovery procedures. First it seeks to the desired track. If the TRACK VERIFY option is enabled, the HDC reads a number of ID fields until it can verify whether it is on the right track or not (it examines the first sector ID field without CRC error). Beginning with the next index mark, the HDC starts counting the PHYSICAL SECTOR number of IDs to locate the desired sector. If PHYSICAL SECTOR is set to "1", it searches for the first sector.

Using the ST506 or double-density Floppy Disk Format, the HDC approximates the location of the next sector ID field or data field based on the sector format parameters specified by the Drive Parameter Block,. If it does not find an address mark in the approximated window, the HDC assumes that this particular field is damaged to an extent where the data separator cannot even find an address mark. The HDC then approximates the position of the next address mark (sector ID field or data field) and tries to find this address mark.

9590

For the ESDI interface the Am9590 counts either address marks (soft-sectoring) or sector pulses (hard-sectoring). Since those occur only once per sector at the beginning of the header, the HDC will always try to read the header. However, if the header information is damaged, the device will calculate the approximate start of the data field and try to recover it.

This sophisticated algorithm allows the HDC to find a data field even if the corresponding sector ID field is damaged, or missing, or if other sector ID fields and/or data fields are damaged or missing. However, this algorithm only operates successfully if the sector format parameters reflect the sector format exactly. When retrieving the data field, the HDC disregards the sector ID field and no retries are performed.

Regardless of any CRC/ECC errors, the HDC will dump the data field read into the system memory starting at Destination Address. In case of CRC/ECC errors, which is indicated by the appropriate SRB, the data field should be examined and a user-supported data recovery procedure should be initiated.

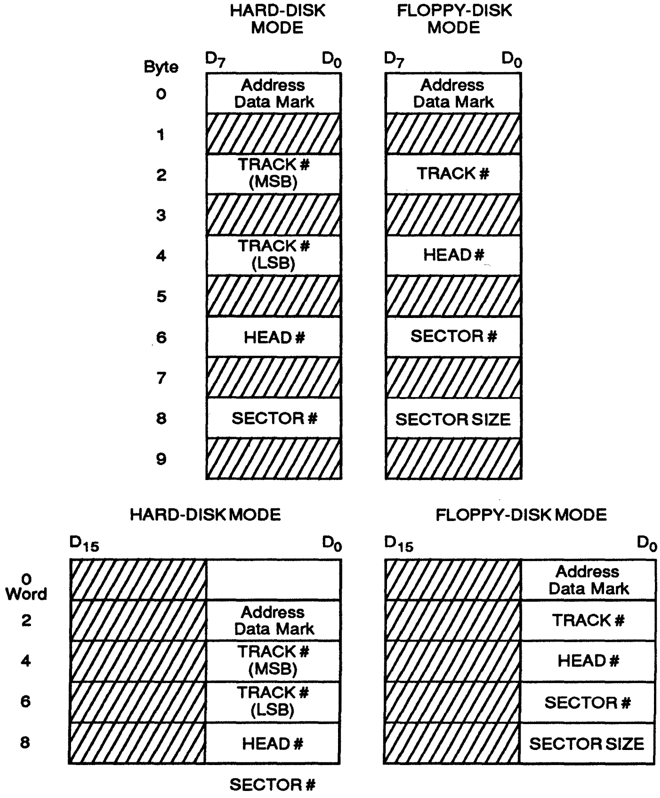
For extensive disk diagnostics, this command may be used to examine the information on the disk which follows the address mark of the sector ID field. By altering the sector format parameters appropriately, the HDC can be tricked into reading a sector ID field instead of a data field. Here, the HDC will most likely report a CRC/ECC error. In this case, the memory contains 128 to 512 bytes of data read sequentially from the disk starting with the address mark of the sector ID field.

READ ID

READ ID attempts to recover the header ID information of a marginal sector. If the ARBITRARY SECTOR option is enabled, then an arbitrary sector ID field (the first valid one the HDC encounters) including the address data mark (FD_H or FE_H) is transferred to the system memory starting at DESTINATION ADDRESS (see Figure 1-23). If the ARBITRARY SECTOR option is disabled, the sector ID field of the absolute sector specified by PHYSICAL SECTOR is transferred to DESTINATION ADDRESS. The PHYSICAL SECTOR number can be any number between 1 and N. If it is set to "1," the first sector ID field is read. If this command is successfully executed, it updates the HDC's track position. If the HDC finds the specified sector ID, but with a CRC error, it will monitor this by issuing an "ID CRC Error" SRB. In this case the ID field can only be (partially) recovered by using the "trick" described in the "Read Physical Sector" command description.

LOAD BUFFER

The data pointed to by the SOURCE ADDRESS is transferred to the internal sector buffer. The number of bytes transferred is determined by the sector size for the selected DRIVE (see Drive



9480A 1-23

Figure 1-23

Parameter Block description). This command may be used for device diagnostics and/or marginal data recovery.

DUMP BUFFER

The data in the internal sector buffer is transferred to system memory starting at DESTINATION ADDRESS. The number of bytes transferred is specified by the sector size for the selected DRIVE (see Drive Parameter Block description). This command may be used for device diagnostics and/or marginal data recovery.

LOAD ECC SYNDROMES

This command transfers the Reed-Solomon syndrome bytes from system memory starting at SOURCE ADDRESS to the internal syndrome RAM of the HDC. This command generates an error if the Drive Parameter Block of the selected DRIVE specifies CRC or external ECC error checking. The number of bytes loaded depends on the sector size and type of Reed-Solomon mode (6..15 bytes) (see Disk Data Protection Section). This command may be used for device diagnostics and/or marginal data recovery.

DUMP ECC SYNDROMES

This command transfers the Reed-Solomon syndrome bytes from the internal syndrome RAM to system memory starting at DESTINATION ADDRESS. This command generates an error if the Drive Parameter Block of the selected DRIVE specifies CRC or external ECC error checking. The number of bytes transferred depends on the sector size and type of Reed-Solomon mode (6..15 bytes). This command may be used for device diagnostics and/or marginal data recovery.

CORRECT BUFFER

This command uses the contents of the Reed-Solomon Syndrome RAM and attempts to correct the sector data in the internal sector buffer. The Error Detection and Correction Policy Field specifies whether to use Single- or Double-Burst Reed-Solomon code. If CRC or External ECC is selected, this command generates an error ("ECC Not Selected" SRB). If the LOCATOR DUMP option is enabled, then, in addition to correcting the data, the locations and values of the errors will be written sequentially to system memory, starting at DESTINATION ADDRESS. The Locators are stored sequentially starting with the Locator Group for interleave 0, then the Locator Group for interleave 1, etc.

Each Locator Group contains one (Single-Burst Reed-Solomon) (Figure 1-24) or two (Double-Burst Reed-Solomon) Locators (Figure 1-25). A Locator is defined as follows:

Buffer Address (2 bytes)
Error Pattern (2 bytes)

Only the lower byte of the Error Pattern is relevant. The upper byte is set to zero. The result of the correction can be interpreted in several different ways:

If the Buffer Address is larger than the sector size and the error pattern is non-zero, the error found for this locator is not correctable.

If the Buffer Address and the Error Pattern are both zero, no error is detected for this locator.

If the Buffer Address is larger than the sector size and the Error Pattern is zero, the error occurred in the check bytes themselves.

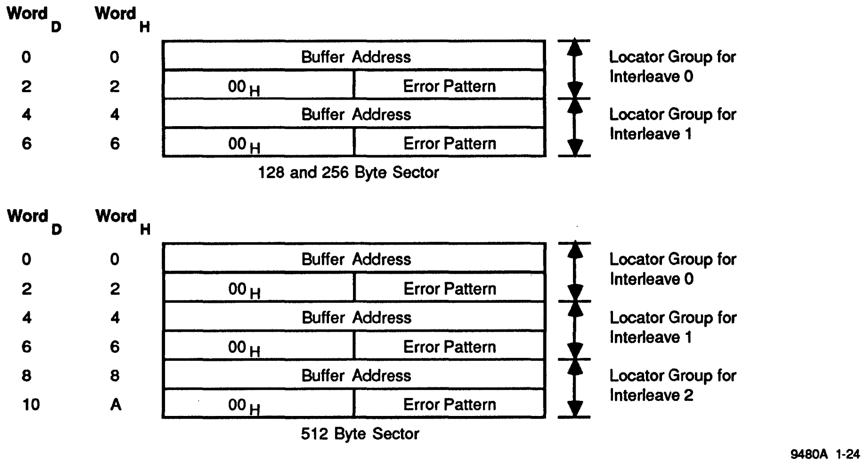
HEAD MOVEMENT COMMANDS

SEEK

The HEAD of the selected DRIVE is moved to the desired TRACK. If TRACK VERIFY is selected, the HDC reads the first encountered header to verify that it is on the right track. If the track numbers do not match, the HDC reports an error ("Seek Error" SRB). The HDC does not attempt an automatic restore. In ST506 Mode (Implied or Overlapped Seek Mode), the HDC issues the first step pulse, waits for SC to go inactive (LOW), issues the remaining step pulses, and then waits for SC to go active (HIGH). In Restricted Mode, the HDC provides the actual track number by shifting out a 16-bit value on STEP. PCEN provides the shift clock (see Figure 1-53). In Floppy Mode SC is expected to be always active.

9590

In ESDI mode the Am9590 will issue a SEEK command to the appropriate track using the serial command/status interface to the disk drive. Before sending the command, the COMMAND COMPLETE line must be active. It is expected to go inactive after sending the first command bit. In Overlapped Seek mode, the HDC will not wait for this line to go active again if it can start a seek on another drive (independent of the drive type).



Word D	Word H			
0	0	Buffer Address		Locator Group for Interleave 0
2	2	00 H	Error Pattern	
4	4	Buffer Address		Locator Group for Interleave 1
6	6	00 H	Error Pattern	

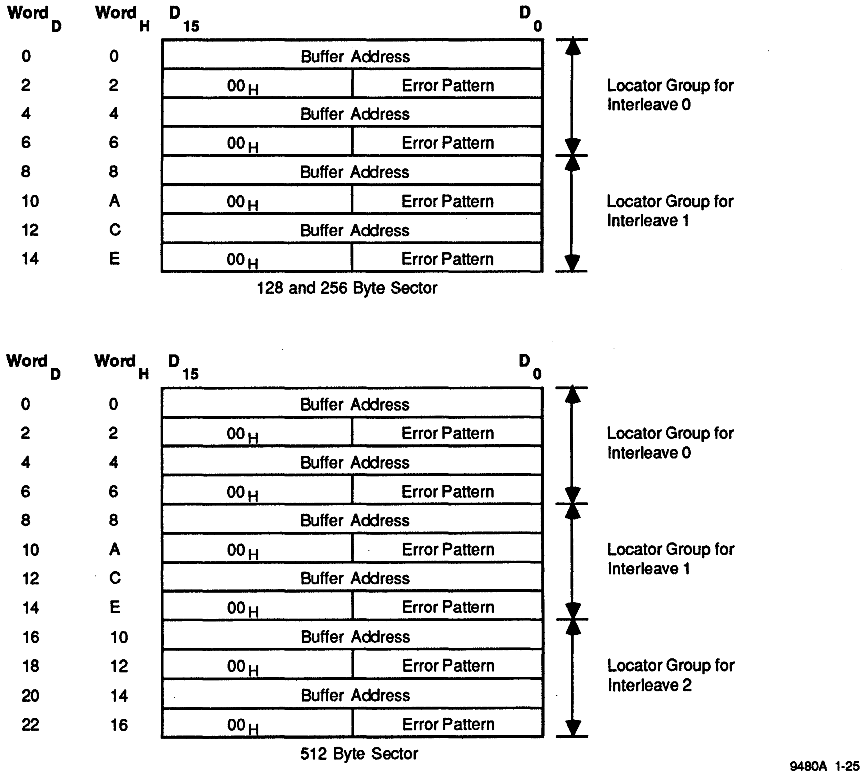
128 and 256 Byte Sector

Word D	Word H			
0	0	Buffer Address		Locator Group for Interleave 0
2	2	00 H	Error Pattern	
4	4	Buffer Address		Locator Group for Interleave 1
6	6	00 H	Error Pattern	
8	8	Buffer Address		Locator Group for Interleave 2
10	A	00 H	Error Pattern	

512 Byte Sector

9480A 1-24

Figure 1-24 Locator Dump for Correct Buffer Command (Single-Burst Reed-Solomon)



Word D	Word H	D15	D0	
0	0	Buffer Address		Locator Group for Interleave 0
2	2	00 H	Error Pattern	
4	4	Buffer Address		
6	6	00 H	Error Pattern	
8	8	Buffer Address		Locator Group for Interleave 1
10	A	00 H	Error Pattern	
12	C	Buffer Address		
14	E	00 H	Error Pattern	

128 and 256 Byte Sector

Word D	Word H	D15	D0	
0	0	Buffer Address		Locator Group for Interleave 0
2	2	00 H	Error Pattern	
4	4	Buffer Address		
6	6	00 H	Error Pattern	
8	8	Buffer Address		Locator Group for Interleave 1
10	A	00 H	Error Pattern	
12	C	Buffer Address		
14	E	00 H	Error Pattern	
16	10	Buffer Address		Locator Group for Interleave 2
18	12	00 H	Error Pattern	
20	14	Buffer Address		
22	16	00 H	Error Pattern	

512 Byte Sector

9480A 1-25

Figure 1-25 Locator Dump for Correct Buffer Command (Double-Burst Reed-Solomon)

RESTORE

The RESTORE command moves the heads of the selected DRIVE back to track 0. This command synchronizes the HDC and the drives after power-up, or recovers the system from seek errors. The HDC supports two restore options for the ST506 and Floppy interface. It can restore drives by issuing step-out pulses until the drive acknowledges reaching track 0 by asserting TRK0. The maximum number of step pulses issued is twice the Tracks/Surface parameter of the corresponding Drive Parameter Block, or 2 16, whichever is less. If the drive does not return a TRK0 after the maximum number of step pulses has been issued, the HDC reports an error ("Restore Fault" SRB). In Hard-disk Mode, The HDC waits, after each step pulse, for SC to be deasserted and then reasserted. It then checks the TRK0 input to verify whether it has reached track 0. Since the HDC waits for SC in-between step pulses, the restore operation is typically executed considerably slower than a seek to track 0 operation. In Floppy-disk Mode, SC has to be HIGH.

As an option, drives with build-in restore logic may be restored by a pulse on the RTZ (Return To Zero) line see also Pin Description). The RTZ pulse has the same timing as the STEP pulse. If TRACK VERIFY is selected, the HDC will also verify a successful restore by reading one sector ID field.

An option bit in the RESTORE IOPB allows the slight modifying of the restore sequence. If this bit is set, the HDC steps in five tracks before starting to restore the heads to track 0. This allows to operate disk drives with a "head parking zone" on a track number smaller then 0.

9590

If the ESDI interface is selected, the Am9590 will issue a RESTORE command to the disk drive. Before sending the command, the COMMAND COMPLETE line must be active. It is expected to go inactive after sending the first command bit and become active again after reaching track 0.

ESDI CHANNEL

This command send ESDI command words (Bytes 10 & 11) to the ESDI drive and receives status information from the drive. Status information will be dumped into the destination memory location. The command generates and checks the parity bit automatically.

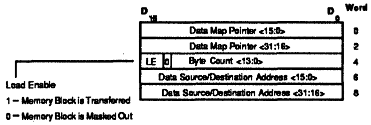
DATA MAPPING OPTIONS

The Data Mapping option lets the HDC process data to be stored in non-contiguous system memory. This option is available on five commands: Read, Write, Verify, Load Buffer, and Dump Buffer. It is enabled by setting the Data Map Enable bit (DME-bit) in the IOPB.

The last two words of the IOPB (Source/Destination Address) link to the first Data Map Entry (Figure 1-26). If this pointer is zero, then the Data Map does not exist and the HDC does not transfer data. Data Map Entries are linked together via the Data Map Pointer. The Data Map linked-list is terminated if the Data Map Pointer is set to zero. Each Data Map Entry defines a data buffer in system memory, starting at the address defined by the Data Source/Destination Address. The size of this buffer is defined by Byte Count. For Read operations, the HDC will skip a data block of the specified size, if LE = 0, (Figures 1-27...1-29). For Write operations, the HDC will write a data block of random data to the disk. When the Load Enable (LE) bit is reset to zero, the HDC masks off a data block with the size specified by Byte Count.

STATUS BLOCK REGISTER (SRB)

The host CPU reserves a Status Result Area defined by the 32-bit Status Result Pointer register and the 16-bit Status Result Length register. Whenever the HDC terminates a command and an error occurs, it adds one or more Status Result Blocks (Figures 1-31,32) to the Status Result Area. Each Status Result Block carries the ID number of the IOPB where the error occurred, to provide a unique cross-reference between IOPBs and SRBs. The SRBs generated are divided into two groups: fatal and non-fatal errors. Non-fatal errors let the HDC continue with the current command. Fatal errors cause an immediate termination of the current command. Furthermore, the IOPB options specify whether the HDC is allowed to proceed with the next command after encountering a fatal or non-fatal error with the current command. A cross reference between SRBs and IOPBs is given in Figure 1-32. The following status codes appear in Byte #3 of the SRBs.


D_{15}
D_0
Word
Data Map Pointer <15:0>
0
Data Map Pointer <31:16>
2
LE
0
Byte Count <13:0>
4
Data Source/Destination Address <15:0>
6
Data Source/Destination Address <31:16>
8
Load Enable
1 – Memory Block is Transferred
0 – Memory Block is Masked Out

9480A-1.26

Figure 1-26 Data Entry Map

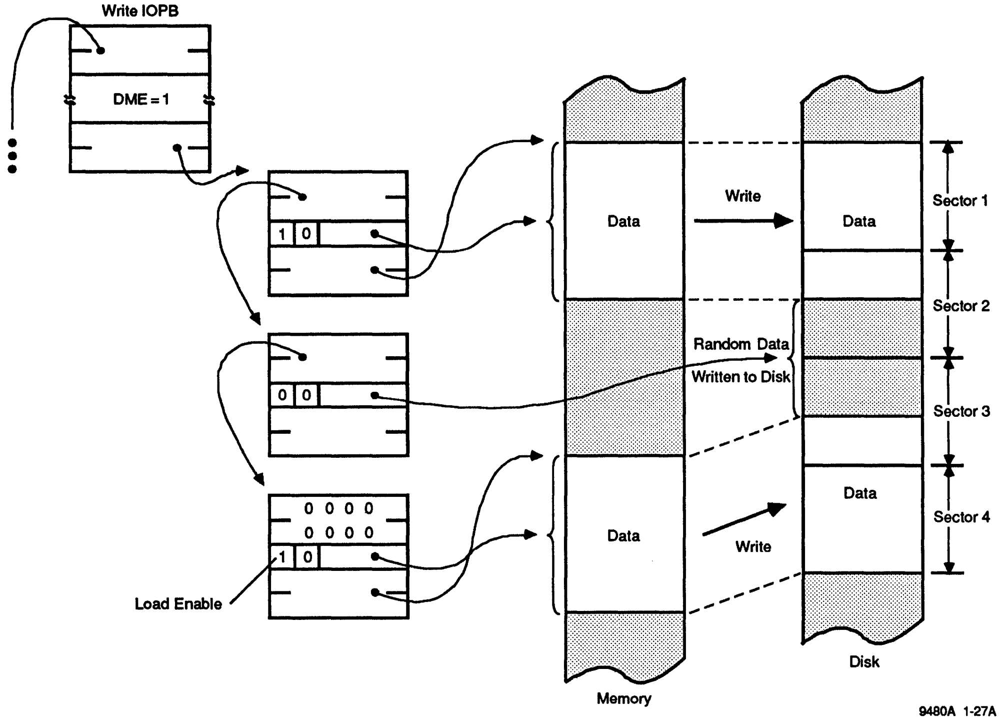


Figure 1-27 Disk Write with Data Mapping

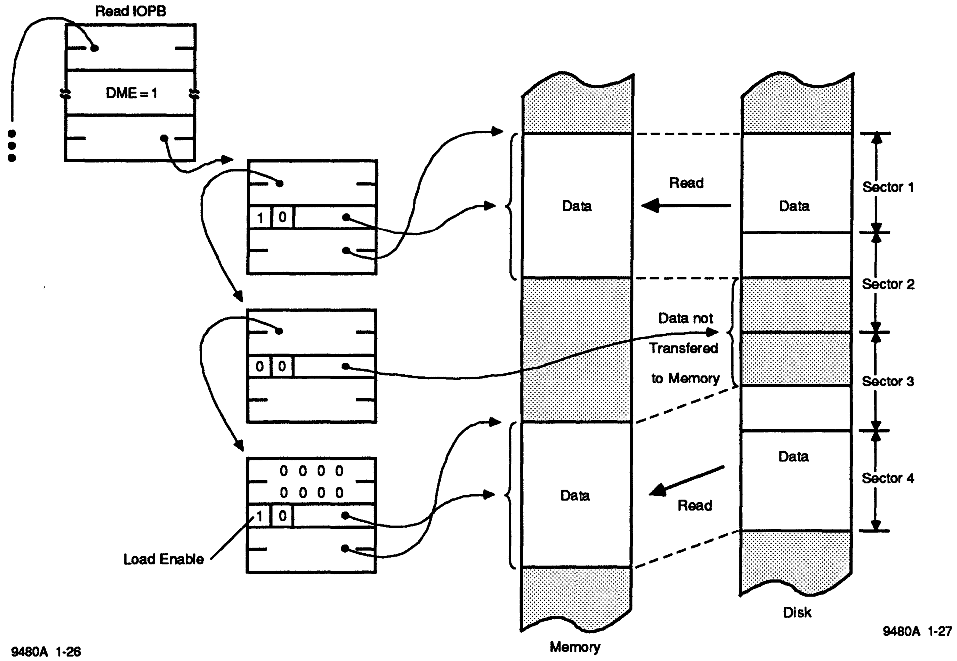
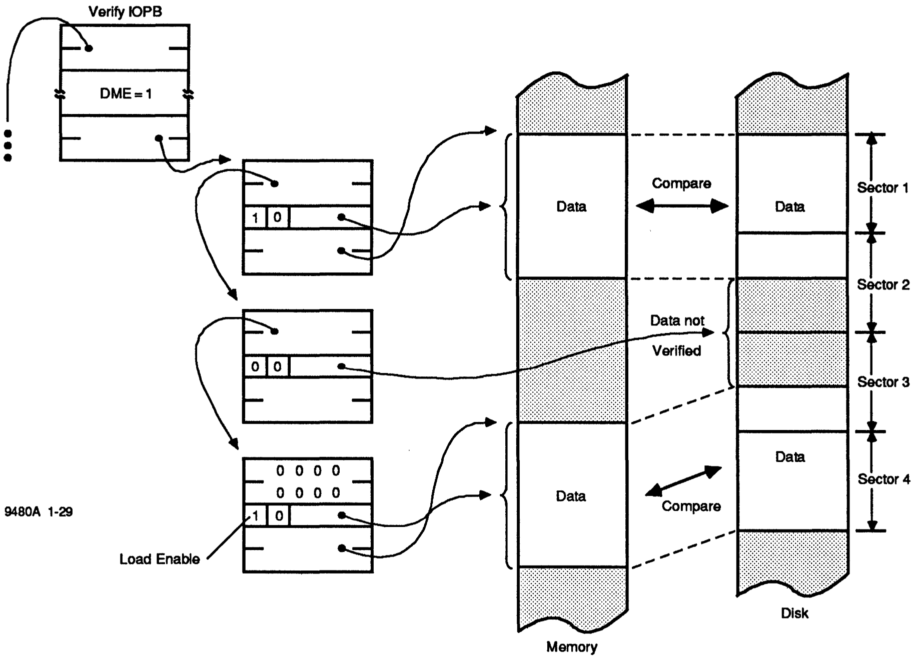


Figure 1-28 Disk Read with Data Mapping



9480A 1-29

Figure 1-29 Disk Verify with Data Mapping

Byte	D_7 … D_0
0	Byte 0
1	Byte 1
2	Byte 2
3	Byte 3
4	Byte 4
5	Byte5
6	Byte 6
7	Byte 7
8	Byte 8
9	Byte 9

Word	D_{15}	D_0
0	Byte 1	Byte 0
2	Byte 3	Byte 2
4	Byte 5	Byte 4
6	Byte 7	Byte 6
8	Byte 9	Byte 8

9480A 1-30

Figure 1-30 Layout of Status Result Blocks

STATUS RESULT BLOCK	Byte 1	Byte 0	Byte 3	Byte 2	Byte 5	Byte 4	Byte 7	Byte 6	Byte 9	Byte 8
Index Error	ID		▨	00_H	Track		▨	Head	▨	▨
No IDs Found on Track	ID		▨	01_H	Track		▨	Head	▨	▨
Seek Error	ID		▨	02_H	Current Track		Desired Track		Current Head	Desired Head
Data Recovered with ECC	ID		Retry Count	03_H	Track		▨	Head	▨	Sector
Data Sync Fault	ID		▨	04_H	Track		▨	Head	▨	Sector
Relocated Track Found	ID		▨	05_H	Current Track		Relocated Track		Current Head	Relocated Head
Relocated Track, No Vector Recovered	ID		▨	06_H	Track		▨	Head	▨	▨
Record Not Found with ID Errors	ID		▨	07_H	Track		▨	Head	▨	Sector
Fatal Seek Error	ID		▨	08_H	Current Track		Desired Track		Current Head	Desired Head
Record Not Found	ID		▨	09_H	Track		▨	Head	▨	Sector
Data Recovered with Retries	ID		Retry Count	$0A_H$	Track		▨	Head	▨	Sector
Data Non-Verify	ID		▨	$0B_H$	Track		▨	Head	▨	Sector
Data Time-Out	ID		▨	$0C_H$	Byte Count		Block Address <15:0>		Block Address <31:16>	
Multi-Record Overflow	ID		▨	$0D_H$	Track		▨	Head	▨	Sector
Data Mark Error	ID		Current Data Mark	$0E_H$	Track		▨	Head	▨	Sector
Sector Size Mismatch	ID		Actual Size	$0F_H$	Track		▨	Head	▨	Sector
ECC Not Selected	ID		▨	10_H	▨	▨	▨	▨	▨	▨
Drive Selection Fault	ID		Status	11_H	▨	▨	▨	▨	▨	▨
Fault While Seeking	ID		Status	12_H	Track		▨	▨	▨	▨
Fault While Head Select	ID		Status	13_H	▨	▨	▨	Head	▨	▨
Drive Trap Status	ID		Status	14_H	▨	▨	▨	▨	▨	▨
End of Data Map	ID		▨	15_H	▨	▨	▨	▨	▨	▨
Restore Fault	ID		Status	16_H	Track		▨	▨	▨	▨
Data Not Recovered	ID		▨	17_H	Track		▨	Head	▨	Sector
Multi Record Command Terminated	ID		▨	18_H	Track		▨	Head	▨	Sector
ID CRC Error	ID		▨	19_H	▨	▨	▨	▨	▨	▨
ESDI Channel Error	ID		Status	$1A_H$		Bit Count	▨	▨	▨	▨
ESDI ID Flag Mismatch	ID		Actual Flag	$1B_H$	Track		▨	Head	▨	Sector
Data Not Corrected	ID		▨	$1D_H$	▨	▨	▨	▨	▨	▨
Drive Deselect Fault	ID		Status	$1E_H$	▨	▨	▨	▨	▨	▨

9480A 1-31

Figure 1-31

Commands	Status Result Blocks (SRBs)																												
	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1D	1E	1A	1B
Read	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	Y	Y
Write	Y	Y	-	-	Y	Y	Y	Y	Y	-	-	Y	Y	-	Y	-	Y	Y	Y	Y	Y	Y	-	Y	-	-	Y	Y	Y
Verify	Y	-	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	Y	Y
Format	Y	-	-	-	-	-	-	-	-	-	-	Y	Y	-	-	-	Y	Y	Y	Y	-	-	-	-	-	-	Y	Y	-
Relocate Track	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	-	-	-	Y	Y	-
Load Parameter	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Dump Parameter	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Read Physical Sector	Y	Y	Y	Y	-	-	-	-	-	-	-	Y	-	-	-	-	Y	Y	Y	Y	-	-	Y	-	-	-	Y	Y	-
Read ID	-	Y	-	-	Y	-	-	-	-	-	-	Y	-	-	-	-	Y	Y	Y	Y	-	-	-	-	Y	-	Y	Y	Y
Load Buffer	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-
Dump Buffer	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-	Y	-	-	-	-	-	-	-	-
Load Syndrome	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-
Dump Syndrome	-	-	-	-	-	-	-	-	-	-	-	Y	-	-	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-
Correct Buffer	-	-	-	Y	-	-	-	-	-	-	-	Y	-	-	-	Y	-	-	-	-	-	-	-	-	-	Y	-	-	-
Seek	-	Y	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	Y	Y	Y	-	-	-	-	-	-	Y	Y	Y
Restore	-	Y	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	Y	-	Y	-	-	-	-	-	-	Y	Y	Y
ESDI Channel	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	-

Key: Y = SRB can be generated
- = SRB is never generated

Figure 1-32

Index Error (00$_H$)

While performing a data access to the disk, the disk drive has issued an INDEX pulse at an incorrect time relative to the operation performed. For a FORMAT or RELOCATE TRACK command, an INDEX pulse is received before all sectors, including the GAP after the last data field, have been formatted. READ, WRITE, and VERIFY issue an Index Error if an INDEX pulse is encountered while reading a header or while accessing a data field. READ SECTOR generates an INDEX Error if the physical sector number is larger than the number of sectors on disk.

No IDs Found on Track (01$_H$)

A data access to the disk drive was performed and the HDC could not find any valid sector ID field (header) on the specified track. This could be caused by the data separator if it never returned an Address Mark Found (AMF) (e.g., the track is not formatted).

Seek Error (02$_H$)

The HDC has read a valid sector ID field from the current track but it contains a head or track number that is different from the expected values. For the RESTORE and SEEK commands, the track verify option must be enabled to see this SRB. For READ, WRITE, and VERIFY, the HDC retries the seek operation by performing an automatic RESTORE and a subsequent SEEK. If the SEEK operation fails again, the HDC issues a Fatal Seek Error.

Data Recovered with ECC (03$_H$)

The requested sector data was recovered using Reed-Solomon error correction. The Retry Count parameter of this SRB indicates the number of repeated read retries, before the ECC unit corrects the errors found in the data field.

Data Sync Fault (04$_H$)

The HDC found the requested sector ID field successfully, but could not find the data field of the sector. Based on the drive parameters such as the length of Postambles, Pads, and Preambles, the HDC computes a window within which it must find the address mark for the requested data field. If AMF is not returned within this window, the HDC will time-out and issue a Data Sync Fault. For the READ ID command, this SRB is generated if the Arbitrary Sector option is enabled and the requested sector ID field could not be found.

Relocated Track Found (05$_H$)

During a normal read, write, or verify operation on a track, a specially flagged track, i.e., a track that has be physically relocated elsewhere on disk was found and if Auto Vector Enable bit in the Drive Parameter Block is set, the HDC automatically moves the R/W heads to the new, relocated track to continue normal, uninterrupted disk I/O.

Relocated Track, No Vector Recovered (06$_H$)

During a normal read, write, or verify operation, a specially flagged track, i.e., a track that has been physically relocated elsewhere on the disk, was found, but the HDC could not recover a good data field on the track (same condition as Data Sync Error). Since the relocation vector is contained in the data fields on the specially flagged track, no relocation vector was found and the HDC could not successfully relocate the R/W heads.

Record Not Found with ID Error(s) (07$_H$)

The sector requested (logical sector number) could not be found on the desired track after at least two complete revolutions of the disk (three INDEX pulses). However, some (one or more) ID fields found were damaged and could not be successfully checked.

Fatal Seek Error (08$_H$)

After encountering a Seek Error during a normal track access operation, and after performing a restore and reseek, the first valid sector ID field still indicates that the current track/head differs from the desired track/head.

Record Not Found (09$_H$)

The sector requested (logical sector number) could not be found on the desired track, after at least two complete revolutions of the disk (three INDEX pulses). However, no sector ID errors were encountered. This means that the R/W heads are positioned correctly (track and head values have been verified), but the desired sector number is not located on this track. Possibly, this sector was not part of the sector map when formatting this track.

Data Recovered with Retries ($0A_H$)

The indicated sector was recovered using repeated read attempts. The number of attempted reads required to recover the sector is indicated by the retry count parameter of this SRB.

Data Non-Verify ($0B_H$)

The sector read from the disk does not match when compared to the data block in memory. Only the VERIFY command issues this SRB.

Data Time Out ($0C_H$)

A memory time-out occurred while the on-chip DMA controller attempted to access data in the system memory. Here, everything except IOPBs and SRBs is considered to be data. The Block Address parameter in the SRB indicates the starting location of this data block transfer. Byte Count shows the size of the data block the HDC attempted to transfer. The time-out occurred anywhere within this data block. After the memory error is corrected, the transfer may be resumed by issuing a RESUME command. The HDC then retransmits the entire data block during the time-out.

Multi-Record Overflow ($0D_H$)

A multi-record command has been terminated due to a head/track overflow. This SRB is only issued if the Multi-Record Policy (MRP) option, which is selected by the corresponding Drive Parameter Block, does not allow automatic track changes.

Data Mark Error ($0E_H$)

The data mark of the indicated sector did not match the selected data mark. If the Data Mark option in the option field of the IOPB is disabled the selected data mark is $F8_H$ (Hard-disk Mode) or FB_H (Floppy Mode). If this option is enabled the data mark is specified in the data mark field of the IOPB.

Sector Size Mismatch ($0F_H$)

The sector size value read from the sector ID field does not match the sector size specified in the Drive Parameter Block. This SRB is only issued in Floppy-disk Mode.

00_H–128 byte sector
01_H – 256 byte sector
02_H – 512 byte sector

ECC Not Selected (10_H)

This SRB is generated if Reed-Solomon error correction is not selected and a Correct Buffer or Load command is not attempted (see Drive Parameter Block description).

Drive Selection Fault (11_H)

A fault occurred on the drive control interface when the HDC attempts to select the designated disk drive. The drive status field shows the status of the drive control inputs at the time the fault occurred:

D_4 = $\overline{\text{DREADY}}$—The disk drive does not acknowledge the drive selection within 1 ms (at 10 MHz system clock) or if DREADY is deasserted.

D_3 = **SC**—The HDC does not time-out on SC. This fault is generated if the disk drive deasserts SC at a time when it should stay active (e.g., after the head is positioned over the requested track and SC has been asserted).

D_2 = **FAULT**—The disk drive asserted Fault.

D_1 = **WRPROT**—The HDC monitors the WRPROT input when it attempts to write the first time to a (new) track. This input must be valid in Floppy- and Hard-disk Modes.

D_0 = **TRK0**—This bit indicates the latched state of the TRK0 input. The HDC does not monitor TRK0 for correctness. It does not detect if TRK0 is asserted at the wrong time or if TRK0 is not asserted when the head is positioned on track 0. The HDC checks TRK0 only when it performs a restore operation.

Fault While Seeking (12_H)

This occurs during a seek operation if the drive control interface flags a fault condition and the HDC terminates the Seek operation. The Drive Status field indicates which line (FAULT, $\overline{\text{DREADY}}$, SC) caused the error.

Fault While Head Select (13_H)

This fault occurs if, while changing the head address, the drive control interface indicates a fault condition. The Drive Status field indicates which line (FAULT, $\overline{\text{DREADY}}$, SC) caused the error.

Drive Status Trap (14_H)

This fault occurs if, during a normal disk I/O operation, the disk control interface indicates a fault condition (fault input asserted). A normal disk I/O operation is a read (RG active) or a write (WG active) operation. Here, the HDC does not attempt to change heads or tracks.

End Of Data Map (15_H)

The data map provided by the IOPB was insufficient to account for the proper transfer of all the requested disk data.

Restore Fault (16_H)

The HDC could not reset the disk drive to track 0. A "Fault While Seeking" type of error condition occurred during a Restore command or the HDC tried to restore after a Seek error. However, after issuing twice the number of step pulses programmed in the Tracks/Surface parameter in the Drive Parameter Block or 2 10 step pulses, whichever is less, the drive still did not assert TRK0.

Data Not Recovered (17_H)

The indicated sector data was not recovered by the HDC despite retries and/or ECC. For multi-record commands the HDC continues to read/write the remaining records (non-fatal error).

Multi-Record Command Terminated (18_H)

The current multi-record IOPB (read, write, or verify command) has been terminated due to an error condition at the specified track, head and sector. Record Count specifies the number of unprocessed sectors.

ID CRC Error (19_H)

The desired sector ID field for a READ ID command with Arbitrary Sector option enabled; or, any sector ID fields for other commands on the current track/head have CRC errors and could not be successfully recovered.

9590

ESDI Channel Error ($1A_H$)

This error indicates a fault condition on the serial command/status interface. The error can either be a parity error or a time out (100,000 system clocks) on the command or status line. The BIT COUNT in the SRB shows how many bits have been transferred before the error occurred.

ESDI ID Flag Mismatch ($1B_H$)

The ID Flag in the sector header does not match the ID Flag programmed in the IOPB executed.

RESERVED (IC_H)

Data Not Corrected (ID_H)

The "Correct Buffer" command could not correct the data in the buffer.

Drive Deselect Fault (IE_H)

While trying to deselect the drive, a fault condition occurred, or the drive did not deselect within 10,000 system clocks of deasserting $\overline{\text{SELEN}}$.

RESERVED (IF_H)

DISK DATA I/O

Sector Formats

Data is stored on the disk in sectors. Each sector consists of two fundamental parts: the header or sector ID field and the data field. The sizes of all pads, gaps, preambles, postambles and data fields are programmable in the Drive Parameter Block. The Double-Density Floppy-Disk Format also has an Index field at the beginning of each track.

9590

Figures 1-35 and 1-36 show the sector layout for ESDI hard- and soft-sector format.

Header

The header of the ST506 and Double-Density Floppy Format contains the Address Mark, the track number, the head number, and sector number (Figure 1-33). Two trailing CRC check bytes protect the header. The beginning of the header is marked by an ID Address Mark (IDAM). The Double-Density Floppy Format and the ST506 Hard-disk Format use a two-part address mark. The first part of the address mark is a unique clock/data pattern written and detected by the data separator. Since the unique clock/data pattern is written by the disk data separator, its length and layout are transparent to the HDC (see also section on "Disk Data Protection", "CRC/CCITT"). The second part, which is processed by the HDC, specifies that this is a normal (FE$_H$ or a relocated (FD$_H$) track.

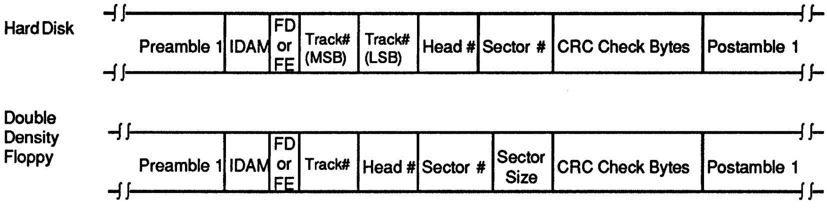
9590

The header of the ESDI hard sector format (Figure 1-35) consists of a PLO synchronization field followed by a byte synchronization pattern. The following bytes of the header indicate the track, head and sector number as well as a flag status byte. Two CRC check bytes protect the header information.

The start of a header looks different for the ESDI soft sectored format (Figure 1-36). It starts with a three byte address mark. This address mark is a unique clock/data pattern written and detected by the control logic implemented on the disk drive. The address mark is followed by the PLO SYNC FIELD. The rest of the header is similar to the hard sector format.

Data Field

The ST506 format starts the data field with an address mark similar to the header's address mark (Figure 1-34). The data field is protected by one of three different user selectable EDC/ECC codes appended to the data: CRC-CCITT (error detection), Single- or Double-Burst Reed-Solomon (error detection and correction). For external ECC processing this code field is ignored. The data field has a user-programmable length of 128, 256, or 512 bytes. For external ECC, the number of ECC bytes attached may vary from 1 to 256 bytes depending on the ECC option, the data field size, and the drive type.



Hard Disk

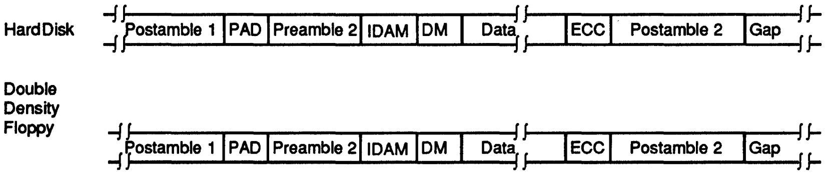
Preamble 1	IDAM	FD or FE	Track# (MSB)	Track# (LSB)	Head #	Sector #	CRC Check Bytes	Postamble 1

Double Density Floppy

Preamble 1	IDAM	FD or FE	Track#	Head #	Sector #	Sector Size	CRC Check Bytes	Postamble 1

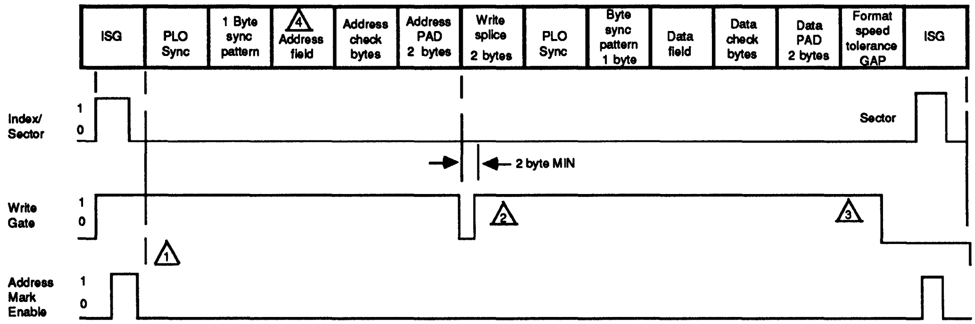
9480A 1-33

Figure 1-33 Sector Header Formats



9480A 1-34

Figure 1-34 Sector Data Fields Format



△1 Trailing edge of address mark enable signifies the strat of hearder PLO sync field. Drive will not write an address mark on the disk media

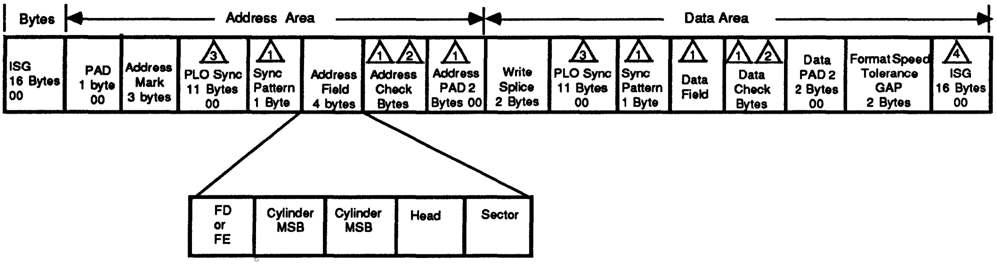
△2 Transition required only if the disk is read after a format and prior to a data field write update

△3 Controller must reinitialize timing with each sector pulse (need not deactivate write gate)

△4 Format speed tolerance gap is required if reference clock is not tied to rotational speed

9480A 1-35

Figure 1-35 Hard Sector Formats and Address Mark, Write Gate Timing



△1 These areas are examples only and may be structured to suit individual customer requirements

△2 The number of check bytes are user defined

△3 PLO sync field is ≥ 11 bytes

△4 Format speed tolerance gap is required if reference clock is not tied to rotational speed

Figure 1-36 Soft Sector Format (Step Mode)

9480A 1-36

The Hard-disk Format has a two-byte address mark consisting of the same unique pattern as the header address mark and a user-programmable Data Mark. For the Double-Density Floppy Format, the address mark has three bytes of the unique pattern and one Data Mark.

9590

The ESDI format does not have any address mark or sector pulse leading the data field. A PLO synchronization field separates the header from the data field. This SYNC field is filled with 0s. The HDC determines the start of a data field by waiting for the first logical "1" in the data stream indicating the first bit of the BYTE SYNC pattern.

Data Separator and Disk Interface Signals

The data lines (RDDAT and WRDAT) transfer the data to and from disk. The transfer is controlled by five signals (INDEX, AMC, AMF, RG, and WG). Three ECC lines interface with external ECC hardware to allow user-definable error detection/correction.

9590

The Am9590 uses the AMF input also as the Address Mark Found input for the soft-sectored ESDI interface and the SECT pulse of the hard-sectored ESDI interface

Header Search Mechanism

When the HDC attempts to read a header or to access a data field, it asserts Address Mark Control (AMC). In Floppy-disk Mode, FAM_1/ECC_1 differentiates between the index address mark and the other address marks.

If the ST506 or Double-Density Floppy Format is selected, the HDC asserts Read Gate (RG) and waits for the data separator to acknowledge. The data separator should assert Address Mark Found (AMF) when it detects an address mark. The rising edge of AMF indicates that valid data will be on the RDDAT input, starting on the next rising edge of RD/REFCLK (see Figures 31 and 32). For Double-Density Floppy and ST506 Hard-disk Formats, the HDC checks the first eight bits (the data mark) for a FD_H or FE_H. If the check fails, then it assumes it did not really find an ID address mark and deactivates AMC and RG. After AMF becomes inactive, AMC is reasserted, followed by RG, and the whole procedure is repeated until either a sector ID field is found, or it has been determined that no sector ID field can be found.

9590

If the ESDI or SMD soft-sectored track format is selected the Am9590 asserts AMEN (Address Mark Enable) and waits for the disk drive to activate AMF (Address Mark Found). After the disk drive finds an address mark, the HDC will wait for the PAD and PLO synchronization field to pass. The first logical "1" indicates the start of the Byte Sync pattern. The HDC will then read this pattern and compare it. The header and data byte sync patterns are FE_H by default. The data Byte Sync pattern can be programmed to a different pattern. If the header Byte Sync pattern is different from FE_H the Am9590 will assume a relocated track and read the information in the data field as a relocation vector. If the programming option of the data Byte Sync pattern is used, the user must ensure that the first bit is a logical "1" ($\geq 80_H$).

In hard-sectored mode AMEN is not used during a read operation. The HDC just waits for the SECT or INDEX pulse. These signals are basi-cally treated like the AMF signal in soft-sectored mode, so that the following procedure is similar.

After finding an ID address mark, the HDC compares the next 32 bits (4 bytes) of serial data with the track, head, and sector number (and sector size for the floppy format) to determine if the right sector ID field has been found. A CRC check insures the correct header information. If the track or head numbers of the sector ID field do not match the desired track and head numbers, an error is flagged and the command is aborted. In the Floppy-disk Mode, the sector size is also checked against the sector size defined for the current drive.

After confirming the correct head position, the sector number is checked to see if this is the sector being searched. If the correct sector number is found, the search for the data address mark begins. If the correct sector number is not found, the process is repeated until either the desired sector is found or the HDC determines the sector cannot be found on the current track. If three index pulses are detected (indicating at least two full disk revolutions), the HDC aborts the command and flags an error.

Data Reads

In ST506 and Double-Density Floppy Mode the HDC uses the same handshake procedure to search for the address mark of the data field (Figure 1-37). The HDC then compares the byte following the address mark to the programmed

Data Mark for a match. If they match, the subsequent data field will be transferred. Based on the drive parameters specified in the drive parameter block, the HDC computes a window within which the address mark of the data field must fall. If the HDC does not find an address mark within this window, it aborts the command and flags an error. This procedure forces the HDC to search for the data field corresponding to the desired sector ID field.

9590

The ESDI and SMD sector formats do not have any address mark or sector pulse to indicate the beginning of a data field, Figures 1-35, 36. The Am9590 will activate RG within the PLO SYNC field leading the data field. The first logical "1" indicates the start of the Data Sync pattern. The HDC will then compare this pattern to the value programmed. If it matches, the data field will be read into one of the toggle buffers; if not, the HDC will flag a data mark error.

Following the data field the ECC check bytes are read. At the end of the data field, RG is turned off and sector read operation is completed. When external ECC is enabled, the HDC ignores the check bytes. The external ECC logic should scan the check bytes to verify the integrity of the data field.

Data Writes

While searching for the desired sector, write commands proceed similar to sector reads. After finding the desired sector ID field, the subsequent data field, including Pad, Preamble 2, Data Mark, and Postamble 2 are overwritten (Figure 1-38). The HDC activates Write Gate (WG) at the beginning of the pad. Write Data on the WRDAT output is valid starting with the next rising edge of the clock (RD/REFCLK). The HDC then writes the new Pad and Preamble 2 fields. Upon completion of the Preamble 2 field, AMC is activated if the ST506 or Double-Density Floppy Format is selected. The data separator asserts AMF while it writes the last Address Mark bit. The HDC resumes data output with the next clock and writes the Data Mark byte (either the programmed pattern or the default values F8$_H$ for Hard-disk or FB$_H$ for Floppy-disk Mode. WG is turned off immediately after the Postamble 2 field is written. This completes the write sector sequence.

9590

Using the ESDI data format, the header search mechanism is similar to read operation. If a valid header has been found, WG is switched on at the beginning of the WRITE SPLICE. The subsequent information, including WRITE SPLICE, PLO SYNC and BYTE SYNC pattern are overwritten. WG is switched off after writing the DATA PAD 2 field.

Format Track

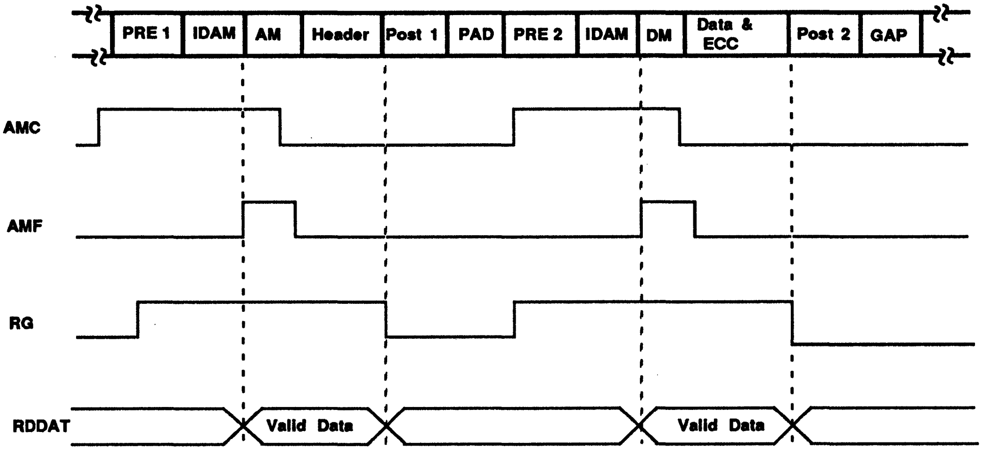
Format Track always formats entire tracks. Only this command and "Relocate Track" will write the sector header. Beginning with the rising edge of the INDEX pulse, the HDC asserts WG and outputs the pattern for the delay field (Figure 1-41). For floppy formats, an Index Address Mark (IAM) field follows the Delay field, and the HDC starts writing sectors.

For writing the headers, the HDC uses the track, head and sector size information supplied by the drive parameter block. A sector map in system memory supplies the logical sector number sequence (Figure 1-42).

The first byte of the sector map is written in the sector number field of the first physical sector on track. The second byte is written in the second physical sector. This sequence continues until the required number of sectors have been formatted. For multiple track format commands, the HDC uses the same sector map repetitively. The Map Pointer of the Format IOPB points to the beginning of this sector map. Skewing sectors are implemented by changing the Sector Map for each track individually.

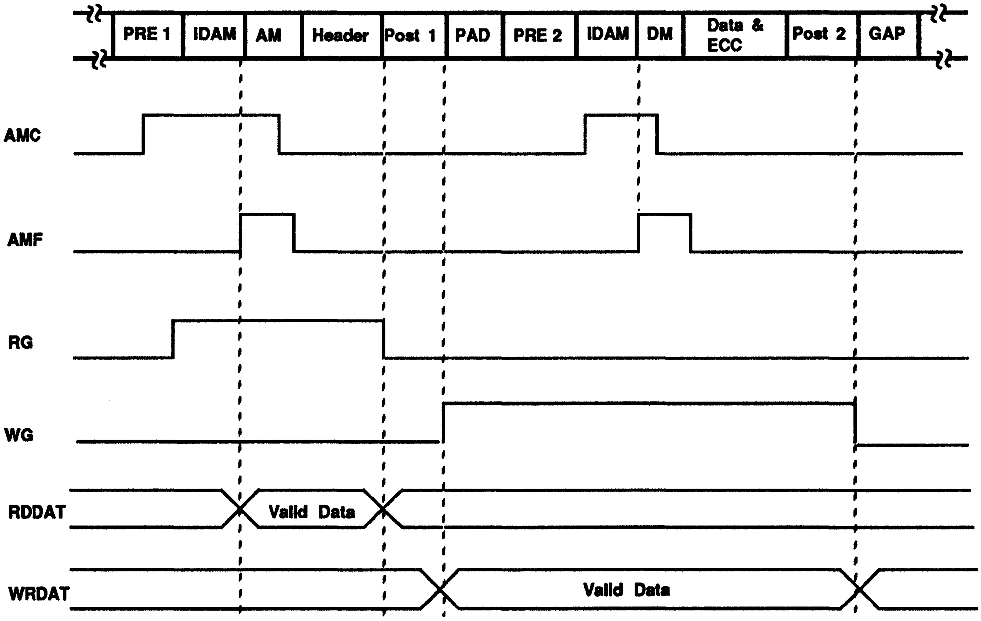
For writing address/data marks in ST506 and Double-Density Floppy Format, the HDC proceeds as described in the "Data Field Write" section. The data field is initialized with the user-supplied pattern byte that is specified in the Format IOPB. The gap between the end of the last sector (end of the Intersector Gap - ISG) and the rising edge of the index pulse is filled by the gap pattern. It is different from the Intersector Gap that has a user-definable length. The patterns for all fields are shown in the following table:

Field	Pattern
Delay	4E$_H$
Preamble 1	00$_H$
Postamble 1	4E$_H$
Pad	00$_H$
Preamble 2	00$_H$
Postamble 2	4E$_H$
Gap	4E$_H$



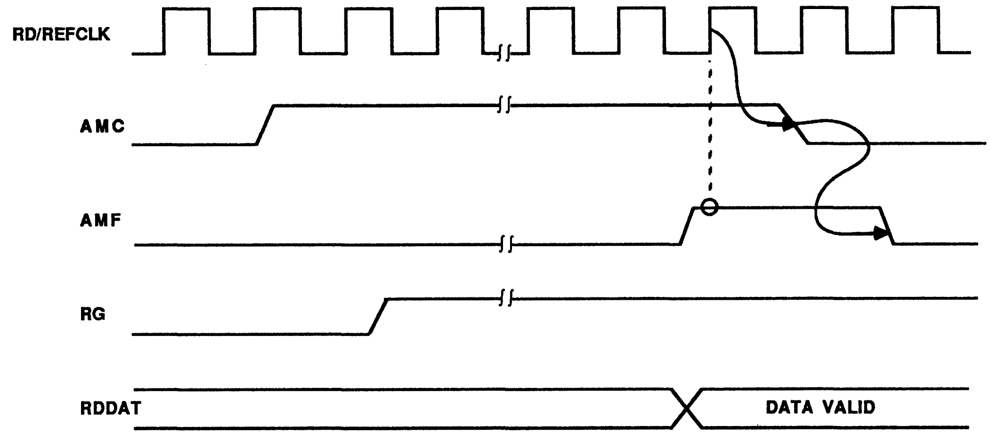
9480A 1-37

Figure 1-37 Read Sector Control Sequence (ST506)



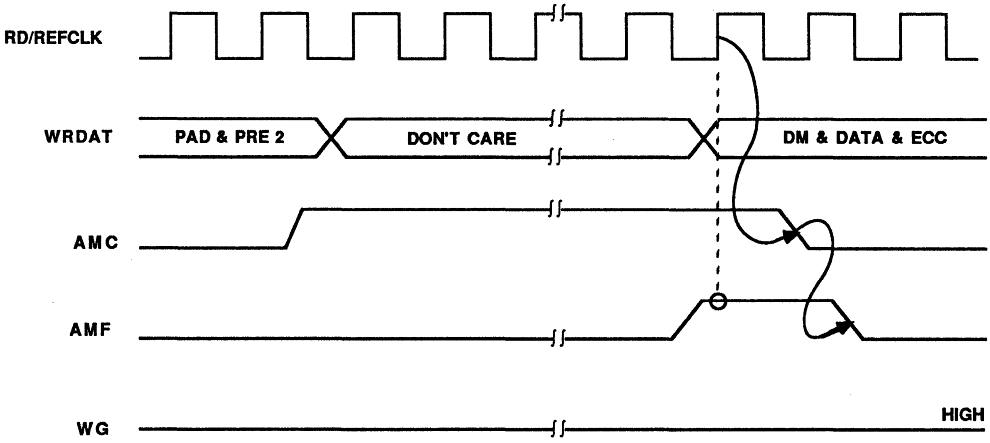
9480A 1-38

Figure 1-38 Write Sector Control Sequence (ST506)



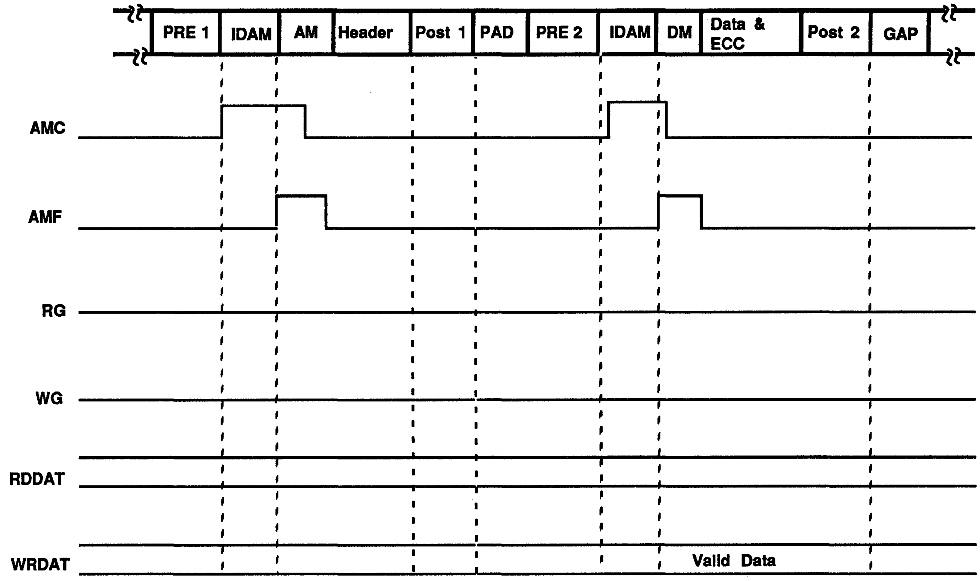
9480A 1-39

Figure 1-39 Address Mark Control/Address Mark Found
(Read Data, ST506)



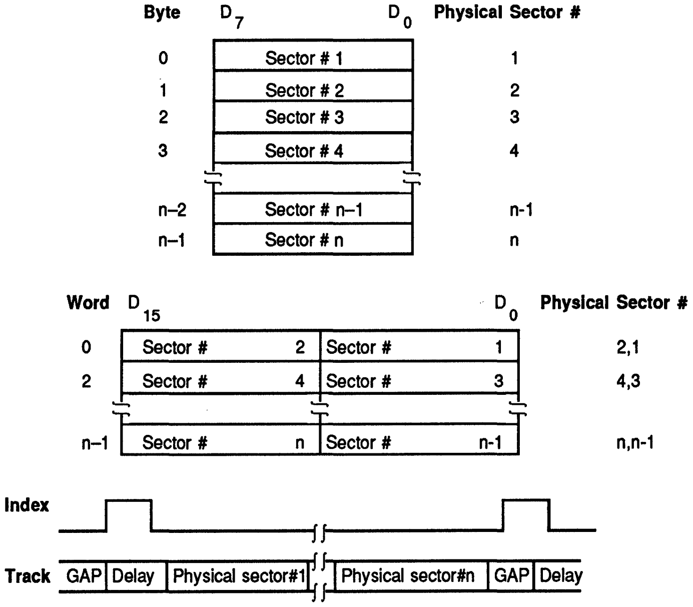
9480A 1-40

Figure 1-40 Address Mark Control/Address Mark Found Handshake
(Write Data, ST506)



9480A 1-41

Figure 1-41 Format Sector Control Sequence



Byte	D7 D0	Physical Sector #
0	Sector # 1	1
1	Sector # 2	2
2	Sector # 3	3
3	Sector # 4	4
n–2	Sector # n–1	n-1
n–1	Sector # n	n

Word	D15		D0	Physical Sector #
0	Sector # 2	Sector # 1		2,1
2	Sector # 4	Sector # 3		4,3
n–1	Sector # n	Sector # n-1		n,n-1

9480A 1-42

Figure 1-42 Sector Map for Format Commands

Formatting an ESDI/SMD type track the Am9590 uses patterns according to the following table:

Field	Pattern
Delay	$4E_H$
PLO SYNC 1	00_H
ADDRESS PAD	00_H
WRITE SPLICE	00_H
DATA PAD	00_H
ISG	$4E_H$

For the hard-sectored track format the HDC starts writing right after the INDEX pulse. It will format the first sector according to the drive parameter block. If the sum of all fields programmed is shorter than the sector defined by the disk drive, the HDC will fill the gap with the ISG pattern. If the sum of all fields programmed is longer than the sector defined by the disk drive, the HDC will fill flag a format error and abort the command.

The format procedure for the soft-sectored ESDI/SMD track format is similar to the procedure in ST506, except that there is no data address mark written. Table 1-1 shows how the track format parameters for ST506 (DD Floppy) and ESDI (SMD) relate to each other.

Table 1-1 Am9590 Parameter Cross Reference Chart (ST506/ESDI)

HDC Drive Parameters	ESDI (Soft Sector)	ESDI (Hard Sector)
Delay	ISG (After Index Pulse Only)	ISG (After All Index & Sector Pulses)
Preamble 1	ID PLO Sync	ID PLO Sync
Postamble 1	Address Pad	Address Pad
Pad	Write Splice	Write Splice
Preamble 2	Data PLO Sync	Data PLO Sync
ECC	ECC	ECC
Postamble 2	Data Pad	Data Pad
Gap	ISG (For All Sectors Except Index)	ISG (Before All Index & Sector Pulses)

DRIVE PARAMETER PROGRAMMING

The HDC contains one set of Drive Parameter Registers for each drive. The system can only indirectly access this register set by the following commands: Load Drive Parameter Block and Dump Drive Parameter Block. The block (Figure 1-44) is set up in contiguous system memory. The numeric byte values specified in the Drive Parameter Block should not be set to zero.

General Select Byte (Figures 1-44, 45)

The General Select Byte has five distinct fields: Auto Vector Enable, Return to Zero Enable, Error Detection and Correction Type, Multi-Record Policy, and Track Format.

Auto Vector Enable (AVE)—This bit selects whether the HDC is allowed to automatically seek to the new track if the current track has been relocated. If the bit is set, the HDC will issue the SRB "Relocated Track" and continues automatically seeking to the new track. If the bit is reset, the HDC issues a "Relocated Track" SRB and terminates the current IOPB.

Return To Zero Enable (RTZE)—If RTZE is reset, a restore operation will be performed by issuing Step pulses until TRK0 is asserted (see Restore command description). If this bit is set, the HDC just pulses the RTZ output once to restore the heads. The RTZ option is only available in Hard-disk Mode.

Error Detection and Correction Type (EDCT)—This field selects the type of error detection and correction code (EDC) that is used to protect the data in each sector. Four options are available: CRC/CCITT, Single-Burst Reed-Solomon, Double-Burst Reed-Solomon, and External ECC (see Disk Data Protection).

Multi-Record Policy (MRP)—This field determines how the HDC will respond when it must select a new track to continue a multi-record IOPB (Format, Read, Write, and Verify). A new track is selected if the last processed sector is the last sector as indicated by the the Sector/Track Parameter (see also Start Sector Option in Data Select Byte). When this occurs, and the multi-record command still has to process more sectors, the HDC selects a new track according to one of the three methods described in the following:

BYTE$_D$	BYTE$_H$	D_7 … D_0
0	0	General Select Byte
1	1	Data Select Byte
2	2	Tracks/Surface <7:0>
3	3	Tracks/Surface <15:8>
4	4	HEADS/DRIVE
5	5	Sectors/Track
6	6	RWC Track <7:0>
7	7	RWC Track <15:8>
8	8	Seek Dwell <7:0>
9	9	Seek Dwell <15:8>
10	A	STEP WIDTH
11	B	HEAD SETTLE
12	C	PRECOMPENSATION TRACK <7:0>
13	D	PRECOMPENSATION TRACK <15:8>
14	E	Retry Policy Byte
15	F	Motor-On Delay
16	10	Delay Length
17	11	Preamble 1 Length
18	12	Postamble 1 Length
19	13	Pad Length
20	14	Preamble 2 Length
21	15	ECC Length
22	16	Postamble 2 Length
23	17	GAP Length

WORD$_D$	WORD$_H$	D_{15} … D_8	D_7 … D_0
0	0	Data Select Byte	General Select Byte
2	2	Tracks/Surface	
4	4	Sector/Track	HEADS/DRIVE
6	6	RWC Track	
8	8	Seek Dwell	
10	A	HEAD SETTLE	STEP WIDTH
12	C	PRECOMPENSATION TRACK	
14	E	Motor-On Delay	Retry Policy Byte
16	10	Preamble 1 Length	Delay Length
18	12	Pad Length	Postamble 1 Length
20	14	ECC Length	Preamble 2 Length
22	16	GAP Length	Postamble 2 Length

9480A 1-44

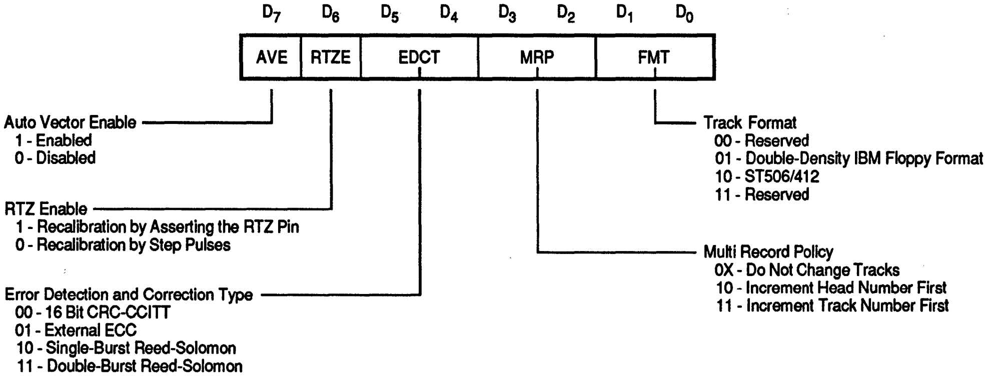
Figure 1-44 Drive Parameter Block

D_3	D_2	Mode
0	X	The HDC will not change tracks. If a change is required, it will abort the current IOPB and issue a "Multi-Record Overflow" SRB.
1	0	The HDC increments the head number first. If the head number overflows, the HDC resets the head number to zero and then increments the track number and steps one track further. If the track number overflows, the HDC resets it to zero.
1	1	The HDC increments the track number first. If the track number overflows, it resets the track number to zero and it increments the head number. If the head number overflows the HDC resets it to zero.

Tracks/Surface and Heads/Drive specify the maximum values. On track change, the HDC increments these values and compares them against the maximum values specified (Tracks/Surface and Heads/Drive). If the current and maximum value are equal, an overflow is detected and the HDC reacts as described above.

When the HDC executes a multi-record command, initially it does not check the specified starting values for head and track with the values allowed by the Drive Parameter Block. However, on the first track change, these values will automatically be set to zero. This means, if the track number requested is larger than the maximum value specified in the Drive Parameter Block, the HDC will reset the track number to zero on the first track change.

Whenever the HDC starts accessing a new track after a track overflow, it starts with Sector 0 or 1, depending on the option selected in the Data Select byte.


D7 D6 D5 D4 D3 D2 D1 D0
AVE
RTZE
EDCT
MRP
FMT
Auto Vector Enable
1 - Enabled
0 - Disabled
RTZ Enable
1 - Recalibration by Asserting the RTZ Pin
0 - Recalibration by Step Pulses
Error Detection and Correction Type
00 - 16 Bit CRC-CCITT
01 - External ECC
10 - Single-Burst Reed-Solomon
11 - Double-Burst Reed-Solomon
Track Format
00 - Reserved
01 - Double-Density IBM Floppy Format
10 - ST506/412
11 - Reserved
Multi Record Policy
0X - Do Not Change Tracks
10 - Increment Head Number First
11 - Increment Track Number First

9480A 1-44

Figure 1-44 General Select Byte (Am9580A)

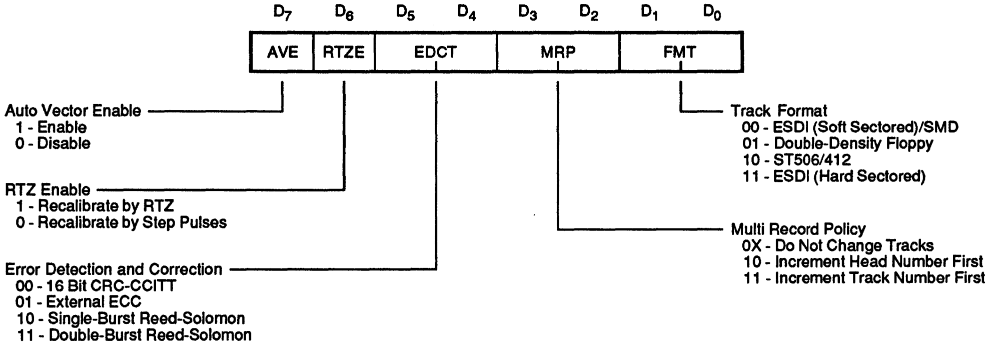

D7 D6 D5 D4 D3 D2 D1 D0
AVE
RTZE
EDCT
MRP
FMT
Auto Vector Enable
1 - Enable
0 - Disable
RTZ Enable
1 - Recalibrate by RTZ
0 - Recalibrate by Step Pulses
Error Detection and Correction
00 - 16 Bit CRC-CCITT
01 - External ECC
10 - Single-Burst Reed-Solomon
11 - Double-Burst Reed-Solomon
Track Format
00 - ESDI (Soft Sectored)/SMD
01 - Double-Density Floppy
10 - ST506/412
11 - ESDI (Hard Sectored)
Multi Record Policy
0X - Do Not Change Tracks
10 - Increment Head Number First
11 - Increment Track Number First

Figure 1-45 General Select Byte (Am9590) 9480A 1-45

9580A

Track Format (FMT)

This 2-bit field indicates the track format to be used; two formats are available. The IBM-compatible Double Density Floppy (DDF) Soft Sector format should only be used in conjunction with Floppy-disk Mode (see Disk Control Interface Type specified in the General Select Byte). The ST506/412 Soft Sector format should only be used in conjunction with Hard-disk Mode.

D_1	D_0	Track Format Type
0	0	(reserved)
0	1	IBM-compatible DDF Soft-Sector (DCIT 1,0 = 01 B)
1	0	ST506/412 Soft-Sector (DCIT 1,0 = 10 B)
1	1	(reserved)

9590

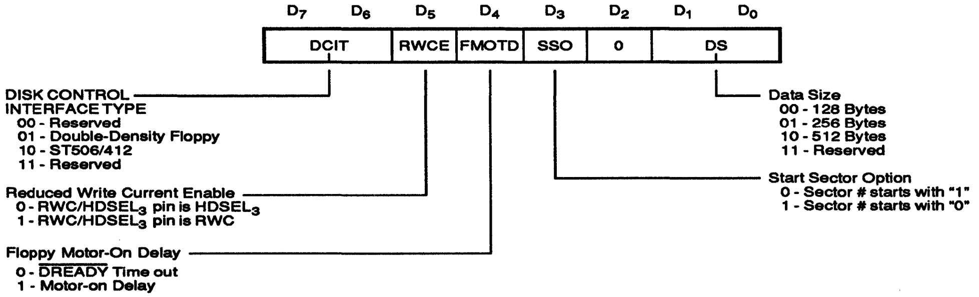
Track Format (FMT)

This 2-bit field indicates the track format to be used; four formats are available. The IBM-compatible Double-Density Floppy (DDF) Soft Sector format should only be used in conjunction with Floppy-disk Mode (see Disk Control Interface Type specified in the General Select Byte). The ST506/412 Soft-Sector format should only be used in conjunction with Hard-disk Mode. The SMD as well as the ESDI sector formats can be used with either the hard- or the soft-sectored mode.

D_1	D_0	Track Format Type
0	0	SMD
0	1	IBM-compatible DDF Soft-Sector (DCIT 1,0 = 01 B)
1	0	ST506/412 Soft-Sector (DCIT 1,0 = 10 B)
1	1	ESDI

Data Select Byte (Figure 1-46a/46b)

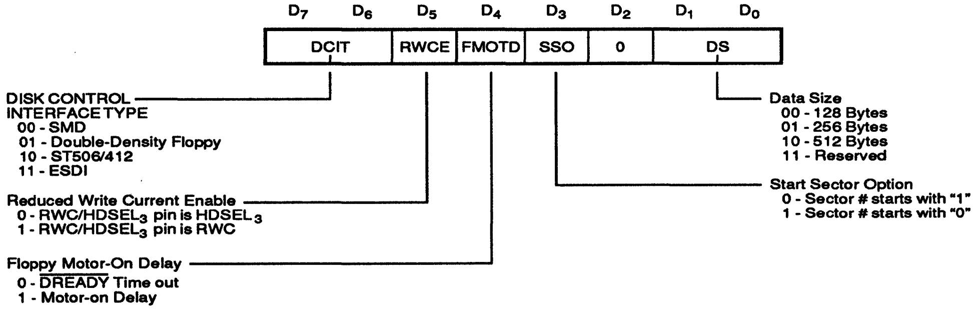
The Data Select Byte contains four distinct fields. They determine sector size, the RWC (Reduced Write Current) option, and the Floppy Motor-On Delay option.



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
DCIT
RWCE
FMOTD
SSO
0
DS
DISK CONTROL INTERFACE TYPE
00 - Reserved
01 - Double-Density Floppy
10 - ST506/412
11 - Reserved
Reduced Write Current Enable
0 - RWC/HDSEL$_3$ pin is HDSEL$_3$
1 - RWC/HDSEL$_3$ pin is RWC
Floppy Motor-On Delay
0 - $\overline{\text{DREADY}}$ Time out
1 - Motor-on Delay
Data Size
00 - 128 Bytes
01 - 256 Bytes
10 - 512 Bytes
11 - Reserved
Start Sector Option
0 - Sector # starts with "1"
1 - Sector # starts with "0"

9480A 1-46a

Figure 1-46a Data Select Byte (Am9580A)



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
DCIT
RWCE
FMOTD
SSO
0
DS
DISK CONTROL INTERFACE TYPE
00 - SMD
01 - Double-Density Floppy
10 - ST506/412
11 - ESDI
Reduced Write Current Enable
0 - RWC/HDSEL$_3$ pin is HDSEL$_3$
1 - RWC/HDSEL$_3$ pin is RWC
Floppy Motor-On Delay
0 - $\overline{\text{DREADY}}$ Time out
1 - Motor-on Delay
Data Size
00 - 128 Bytes
01 - 256 Bytes
10 - 512 Bytes
11 - Reserved
Start Sector Option
0 - Sector # starts with "1"
1 - Sector # starts with "0"

9480A 1-46b

Figure 1-46b Data Select Byte (Am9590)

9580A

Disk Control Interface (DCIT)

This 2-bit field selects the disk control interface. Two modes are supported:

D_7	D_6	Disk Control Interface
0	0	(reserved)
0	1	Floppy-Disk Mode
1	0	ST506/412 Hard-Disk Mode
1	1	(reserved)

9590

Disk Control Interface (DCIT)

This 2-bit field selects the disk control interface. Four modes are supported:

D_7	D_6	Disk Control Interface
0	0	ESDI/SMD Soft-Sector
0	1	Floppy-Disk Mode
1	0	ST506/412 Hard-Disk Mode
1	1	ESDI/SMD Hard-Sector

Reduced Write Current Enable (RWCE)

The HDC multiplexes the Reduced Write Current (RWC) option with the fourth Head Select line ($HDSEL_3$). If RWCE is set, this output provides RWC. If RWCE is reset, this output provides the fourth Head Select line.

Floppy Motor-On Delay (FMOTD)

FMOTD-bit set: The HDC expects $\overline{\text{DREADY}}$ to be asserted within 10,000 clocks after the drive is selected ($\overline{\text{SELEN}}$ LOW). After the HDC received $\overline{\text{DREADY}}$ it waits up to the time delay (50,000 x N clocks) specified by the MON Delay parameter before attempting any seek, read, or write operation. This gives the motor time to bring the disk up to speed.

FMOTD-bit reset: When a drive is selected, the HDC asserts MON, and waits up to the delay (50,000 x N clocks) defined by the MON Delay parameter in the Drive Parameter Block before $\overline{\text{DREADY}}$ is timed out. If $\overline{\text{DREADY}}$ is asserted within this time interval, the HDC proceeds. If $\overline{\text{DREADY}}$ is not asserted within this interval, the HDC generates a Drive Selection Fault SRB.

Start Sector Option (SSO)

SSO bit set: On a head/track overflow for a multi-sector read/write command, the HDC will start searching for Sector "0" on the new track, therefore, sectors should be numbered from 0 to N–1.

SSO bit reset: On a head/track overflow for a multi-sector read/write command, the HDC starts searching for Sector "1" on the new track, therefore, sectors should be numbered from 1 to N.

Data Size (DS)

This two-bit field specifies the size of the data field in each sector on the disk (sector size). The field is encoded as follows:

D_1	D_0	Sector Size
0	0	128 Bytes/Sector
0	1	256 Bytes/Sector
1	0	512 Bytes/Sector
1	1	(reserved)

Retry Policy Byte (Figure 1-47)

The Retry Policy Byte contains five distinct fields that allows the user to choose between different error recovery procedures using ECC and/or repetitive rereads.

Pre ECC Enable (PRE)— If this bit is set, the HDC first uses ECC to recover the data before a reread attempt. If the ECC fails, data is reread. If the reread data still contain errors, another ECC is used before yet another re-read.

ECC Enable (ECC)—If this bit is set, the HDC uses ECC to recover the data after a reread. If ECC fails, the HDC will reread again.

Post ECC Enable (POST)—If this bit is set, the HDC rereads the data up to a specified number of times. ECC is only used if the final reread still contains errors.

The allowed combinations of these lists are:

PRE	ECC	POST
X	–	X
X	X	–
X	–	–
–	X	–
–	–	X

Retry Enable (RE)—If this bit is set, retries are enabled. The maximum number of retry attempts is specified by the RC (Retry Count) field. If this bit is reset, the HDC will not attempt any retry.

Retry Count (RC)—This 4-bit value specifies the number of retries allowed if retries are enabled. If this field is set to zero, the HDC will perform up to 16 retry attempts.

TRACK FORMAT PARAMETERS

The following parameters specify the individual length of the various fields found on the track. The value programmed indicates the field size in bytes (eight bits). The allowed range is from 1 to 255 unless otherwise specified.

Delay Length

After the HDC has detected the INDEX pulse, it waits for the specified Delay Length, until it starts searching for an Address Mark.

9590
For the ESDI format this parameter defines the length of the ISG after each sector.

Preamble 1 Length

Preamble 1 is the synchronization field for the PLL of the data separator for the header field. The length is a function of the lock-up performance of the data separator (see data separator specification). The HDC accepts a length of 2 to 255 bytes.

9590
This parameter represents the header PAD and PLO SYNC field for the ESDI/SMD track format.

Postamble 1 Length

Postamble 1 gives the Disk Controller time to process the sector ID field it has read (CRC check, and parameter comparison). The HDC accepts a field length of 2 to 255 bytes. For the highest efficiency this value should be small, but compatibility to standard track formats may require larger values.

9590
This parameter represents the header ADDRESS PAD field for the ESDI/SMD track format.

Pad Length

This is the read/write splice area between the sector ID field and the data field. It is designed to allow for glitches while turning on the disk write circuitry. This value ranges from 1 to 255.

9590
This parameter represents the WRITE SPLICE for the ESDI/SMD track format.

Preamble 2

Preamble 2 (like Preamble 1) is the synchronization field for the PLL of the data separator to allow it to lock on to the data frequency of the data field. The length is a function of the lock-up

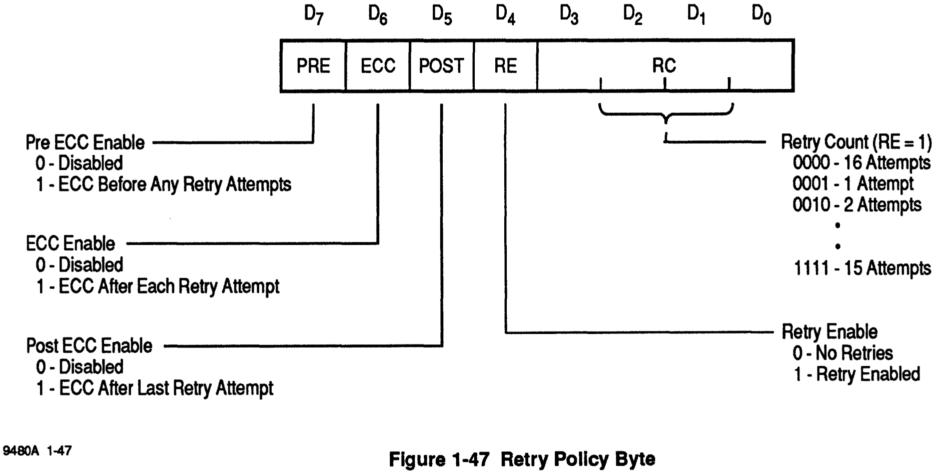


Figure 1-47 Retry Policy Byte

performance of the data separator (see data separator specification). The HDC accepts a length of 2 to 255 bytes.

9590

This parameter represents the data PLO SYNC field for the ESDI/SMD track format.

Postamble 2

Postamble 2 gives the Disk Controller time to process the sector data field it has read (CRC check, error detection with ECC, and data mark comparison). The HDC accepts a field length of 2 to 255 bytes. For the highest efficiency, this value should be small, but compatibility to standard track formats may require larger values.

9590

This parameter represents the DATA PAD field for the ESDI/SMD track format.

Gap Length

The Gap is the inter-sector gap to separate a data field and the following sector ID field. It accounts for motor speed variations in the disk-drive (see drive specification published by disk drive manufacturer).

9590

This parameter represents the SPEED TOLERANCE GAP for ESDI/SMD.

ECC Length

When external ECC is enabled, this parameter specifies the number of check bytes appended to the data field. This parameter is invalid when the floppy format is selected. The Dump Drive Parameter Block command transfers the actual length of the ECC field. This is not necessarily the loaded value. For CRC/CCITT this parameter is automatically set to 2. For Reed Solomon this parameter is either 6, 9, 10, or 15.

DRIVE PARAMETERS

Heads/Drive

This parameter specifies the number of moving read/write heads. All values from 1 to 255 are allowed. For the ST506 and Double-Density Floppy Format, the HDC has four Head Select lines so that it can only address 16 heads. Internally, it processes the full 8-bit number. The head number written into the header field ranges from 0 to N–1.

9590

The SMD interface option of the Am9590 directly addresses 64 heads. Two new Head Select lines (DI_0 and DI_1) are activated for this interface.

Tracks/Surface

This 16-bit parameter specifies the number of tracks per surface. All values from 0 to $FFFF_H$ (65535) are allowed. If it is set to zero, the HDC addresses 65536 tracks. If the value is set to N, the track addresses range from 0 to N-1. The outermost track is track 0.

Sectors/Track

This parameter specifies the number of sectors per track. All values from 1 to 255 are allowed. If the value is set to N and Start Sector Option (SSO) bit in Data Select Byte is set, the logical sector addresses range from 0 to N-1. If the value is set to N and Start Sector Option (SSO) bit in Data Select Byte is reset, the logical sector addresses range from 1 to N. When performing multi-sector commands, the HDC overflows the track after reaching N-1 (SSO is set) or N (SSO is reset) according to the multi record policy selected. For single-sector commands the logical sector number may be larger than N as long as there is a sector located on the track that carries this specific sector number (see sector map of Format command).

RWC Track

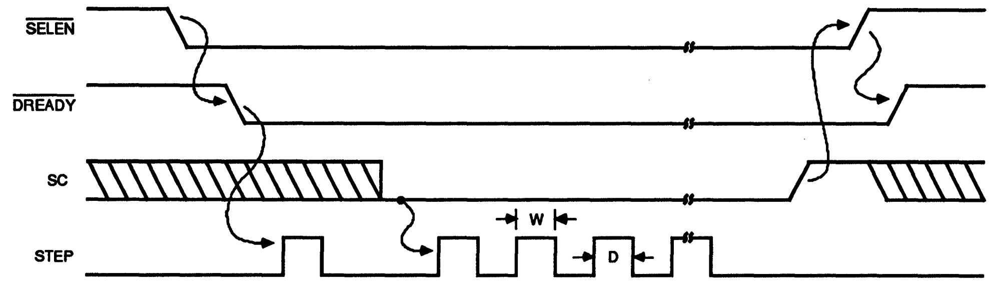
This 16-bit field determines on which tracks the Reduced Write Current output will be asserted. If the current track value is greater or equal to the RWC Track, then RWC is HIGH.

Precompensation Track

This 16-bit field determines for which tracks the Precompensation Enable (PCEN) output will be asserted. If the current track value is greater or equal to the Precompensation track, then PCEN is HIGH.

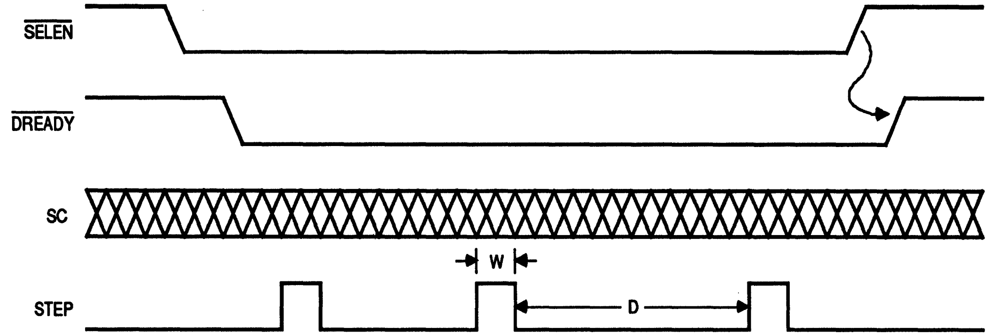
Step Width

This parameter determines the width of the STEP and RTZ pulse in system clocks (Fig. 1-48–50). If this parameter's set to one & mode is ST506, then



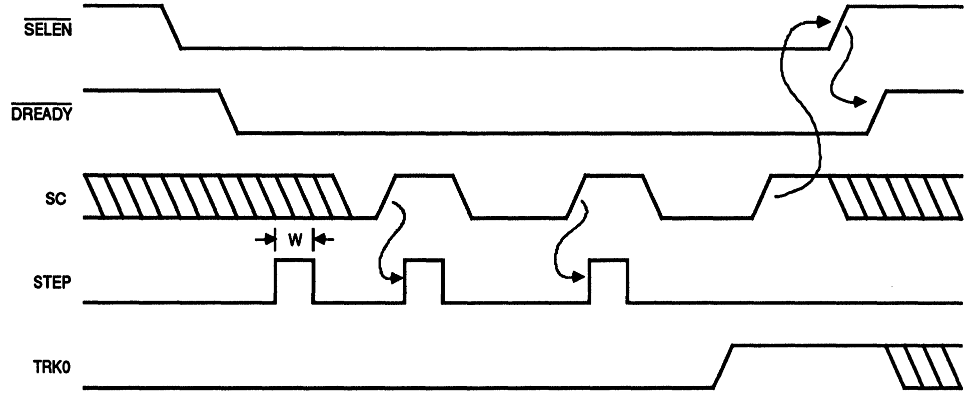
9480A 1-48

Figure 1-48 Seek Timing (Hard Disk Mode)



9480A 1-49

Figure 1-49 Seek or Restore Timing (Floppy Disk Mode)



9480A 1-50

Figure 1-50 Restore Timing (Hard Disk Mode)

the step pulse width and Dwell time length is nine system clocks. This overrides any value in Seek Dwell parameter. If the value ranges from 2 to 255, the step pulse width can be calculated via the following formula (the first step width and dwell time is slightly larger than specified):

5 + (8 • N) System Clocks

where N is the 8-bit integer value of Step width. N can be one in floppy mode. If N= 0, it is equivalent to 256.

Seek Dwell

This 16-bit value determines the Dwell time (the period from the falling edge of Step to the next rising edge of Step or the period between two step pulses). If this parameter is set to one, the seek Dwell time (after the first pulse) will be nine system clocks. If this parameter is set to zero, the value becomes 65536. The formula for the Dwell time is as follows:

21 + (8 • N) System Clocks

where N is the 16-bit integer value of Seek Dwell.

Head Settle

This parameter specifies a time period to allow the head of the selected drive to settle after the head select lines have changed. The formula is as follows:

0 + (8 • N) System Clocks

where N is the 8-bit integer value of Head Settle. This value should not be set to zero.

** For floppy drives, this formula must be multiplied by 256.*

Motor-On Delay

For Hard-disk Mode, this byte is reserved. For Floppy-disk Mode, this parameter specifies Floppy Motor-on Delay (see FMOTD-bit of Data Select Byte). The formula is as follows:

(50,000 • N) System Clocks

where N is the 8-bit Motor-On Delay value. This value should not be set to zero. At 10 MHz, the

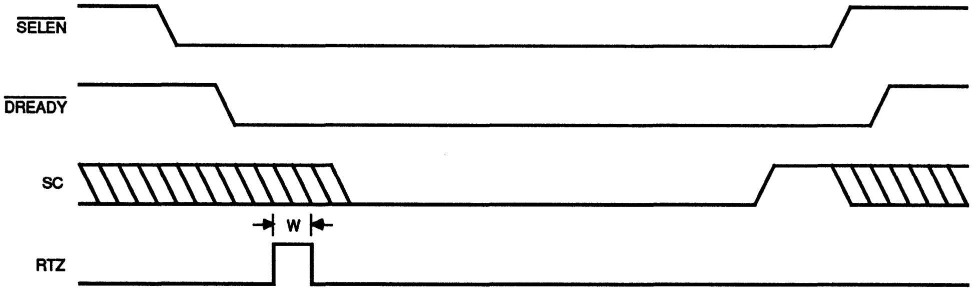


Figure 1-51 Restore Timing (Option In Hard Disk Mode)

9480A 1-51

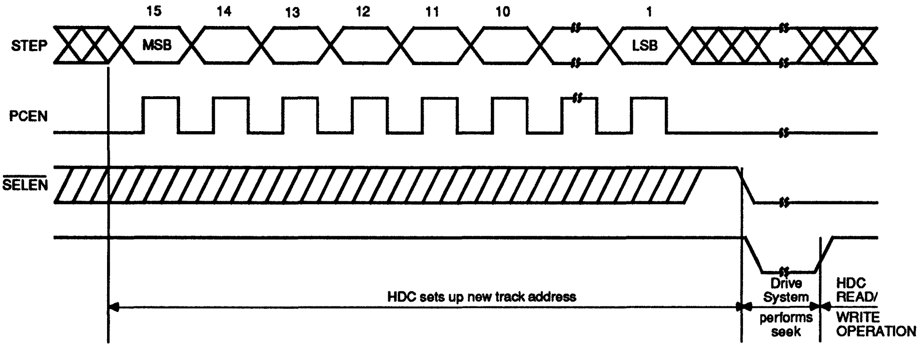

15
14
13
12
11
10
1
STEP
MSB
LSB
PCEN
SELEN
HDC sets up new track address
Drive System performs seek
HDC READ/ WRITE OPERATION

Figure 1-52 Serial Track Address In Restricted Mode

9480A 1-52

Motor-on Delay may be specified in increments of 5 ms.

SECTOR INTERLEAVING

The HDC supports any sector interleaving factor because the Format command numbers the sector according to the sector map. Sample sector maps are given in the following table (for 16 sectors/ track):

Zero-Sector Interleaving:	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
Interleaving Factor 1:	0,8,1,9,2,A,3,B,4,C,5,D,6,E,7,F
Interleaving Factor 2:	0,B,6,1,C,7,2,D,8,3,E,9,4,F,A,5
Interleaving Factor 3:	0,4,8,C,1,5,9,D,2,6,A,E,3,7,B,F

The following formula gives the minimum length of the GAP and Preamble 1 to allow the HDC to Read/Write consecutive sectors (zero-sector interleaving).

$P = (3 + 6 \cdot D/S)$ bytes

P = Delay
D = Disk Clock Freq.
S = System Clock Freq.

P specifies the delay from the end of Postamble 2 to the time the HDC activates RG, and AMC is asserted one byte time later (eight RD/REFCLK cycles). Note that the Data Separator requires a certain Sync Field (Preamble 1) length to synchronize correctly to the data read off the disk (see Data Separator specification).

The following table lists the sample values of "P" for various HDC clock combinations:

	Data Rate			
System Clock	**5 MHz**	**10 MHz**	**15 MHz**	
6 MHz	8	13	18	Bytes
8 MHz	6.75	10.5	14.25*	Bytes
10 MHz	6	9	12	Bytes

** These numbers must be rounded up to the nearest integers.*

DISK DATA PROTECTION

The HDC employs one Error Detection Code (EDC) and two Error Correcting Codes (ECC) schemes to ensure data integrity when transferring data to and from disk. Furthermore, it actively supports off-chip data protection schemes (External ECC). One of these four options must be selected during any disk data transfer. Also, the same code must be used for both read and write operations. The four modes are as follows:

- 16-bit CRC-CCITT (error detection only)
- Single-Burst Reed-Solomon (single-burst correction)
- Double-Burst Reed-Solomon (single- and double-burst correction)
- External ECC

The error checking scheme for each drive is defined by the EDCT field in the General Select Byte of the Drive Parameter Block. Typically, one EDC/ECC code is used for the entire disk. However, it is possible to select different EDC/ECC codes for individual disk platters, tracks, or even sectors, provided that the additional overhead for the dynamic reload of the Drive Parameter Block is taken into account. However, this is not a recommended procedure.

CRC-CCITT is mandatory for the protection of the sector ID field, but the data may be protected by any of the four codes mentioned above.

CRC-CCITT

The CRC-CCITT code is a cyclic based error detecting/non-correcting code. It is the standard code used by floppy-disk systems in the industry today. The HDC is completely compatible with the industry standard. The generator polynomial is:

$$X^{16} + X^{12} + X^{5} + 1$$

The guaranteed capabilities of the CRC code are listed below:

- Detects all odd number bit errors
- Detects all single-burst errors of 16-bit or less
- Detects all single-, double-, and triple-bit errors

The CRC bytes are preset to 1s (IBM standard). The CRC check bytes cover the entire sector ID field or data field, the data mark, and the address mark. The processing of the address mark is completely off-loaded from the HDC (see AMC/AMF handshake). Therefore, the HDC is unaware of the specific address mark that the data separator generates and detects. To ensure compatibility to IBM's Double-Density Floppy-disk Format, the CRC check bytes are preset for a three byte $A1_H$ address mark. (Initially, the CRC bytes are $FFFF_H$. After shifting three bytes $A1_H$, the CRC bytes become $CDB4_H$. So, the HDC effectively presets its CRC logic for floppy- and hard-disk modes to $CDB4_H$ and the CRC bytes do not cover the address mark. The HDC does not require explicit CRC coverage for the address mark because the data separator will ensure the correctness of the address mark by asserting AMF).

For Hard-disk Mode, the CRC bytes are also preset for the three byte $A1_H$ address mark. Since hard disks are not used as an interchangeable media, there are no restrictions.

Error Correction Codes

The HDC supports two error correction codes, Single-Burst Reed-Solomon and Double Burst Reed-Solomon. A single burst of errors (Figure 1-53) is defined as any number of bit errors (contiguous or non-contiguous) where the distance between the first and the last bit error does not exceed the burst length given in the table below. The code protects the check bytes as well as the data including the data mark.

Unlike many other error detection/correction codes, such as the CRC/CCITT or the IBM 3330 56-bit Fire code, the total number of check bytes employed by the Reed-Solomon code varies depending on factors such as interleave size and the record (data field) size. Since the record size is variable (128, 256, and 512 bytes) and more errors are likely to occur when the transfer to or from the disk is longer, the amount of protection by the on-chip Reed-Solomon logic is actually increased for longer data fields.

Both Reed-Solomon codes use a basic protect field size and then interleave these fields. Every Nth byte of a record belongs to the same protect field. For example, if the number of interleaves is 3, then bytes 0, 3, 6, 9,...are protected by the first protect field; bytes 1, 4, 7, 10,...are protected by the second protect field; and bytes 2, 5, 8, 11,...are protected by the third protect field. However, this scheme does not alter or rearrange the data record in any way. Also, the term "interleave" used is not related in any way to the term "Sector Interleaving".

SINGLE-BURST REED-SOLOMON

This code detects single-, double- and some triple-burst errors and corrects single-burst errors. The size of the Reed-Solomon code varies relative to the data field it protects. The basic protect field size for Single-Burst Reed-Solomon is three bytes. The number of interleaves is two (128 or 256 byte sectors) or three (512 byte sectors). So, between six and nine check bytes are generated. The guaranteed performance of this code is shown below (see also Figures 1-54, 55, and 56):

Sector Size (# of bytes)	Detection Single-Burst	Capability Double-Burst	Correction Single-Burst	Capability Double-Burst	# of Check Bytes
128	33	9	9	0	6
256	33	9	9	0	6
512	57	17	17	0	9

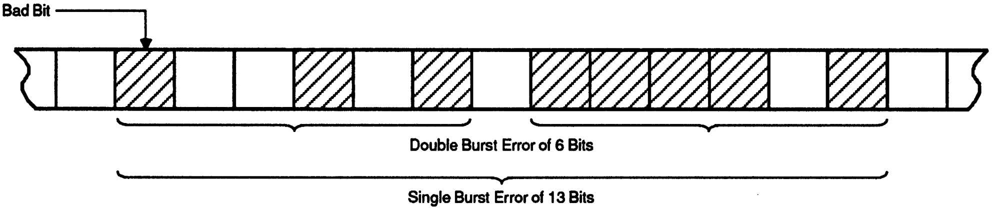
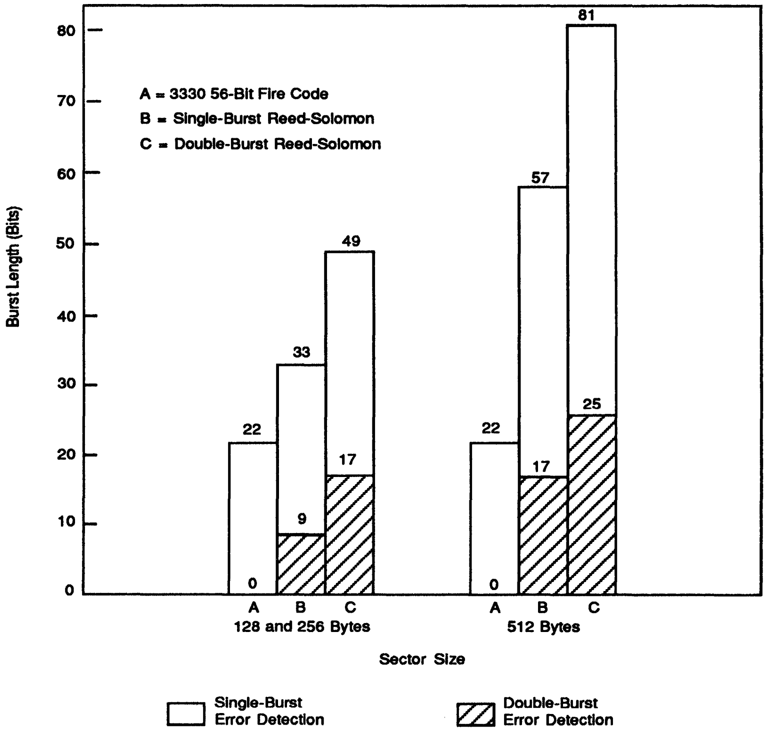
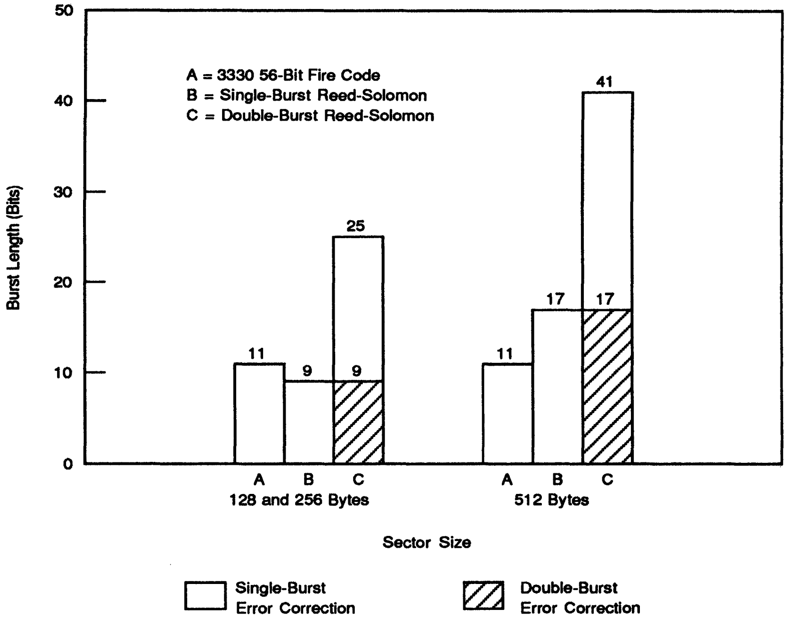


Figure 1-53 Burst Errors



9480A 1-54

Figure 1-54 Burst Error Detection Comparison



9480A 1-55

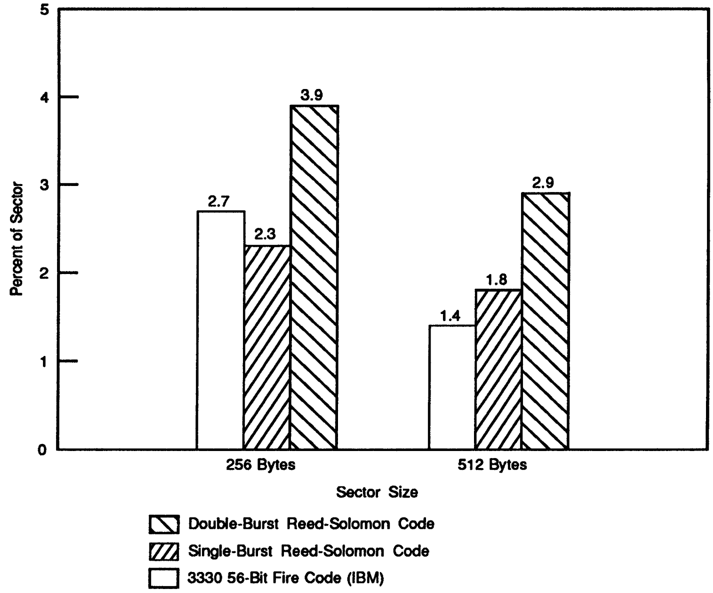
Figure 1-55 Burst Error Correction Comparison

As the table shows, in a sector of 256 bytes, any single-burst errors with a length of up to 33 bits will be detected. Note that two single-bit errors separated by more than 32 bits will count as a double-burst error. Alternatively, any two random single-bursts (double-burst) of up to 9 bits each will be detected also. This code can correct single-burst errors of up to 9 bits and cannot correct any double-burst errors. The table presents the guaranteed capabilities under worst case conditions. Under certain circumstances, the code is capable of detecting longer bursts or even triple bursts.

DOUBLE-BURST REED-SOLOMON

Double-Burst Reed-Solomon is an enhanced version of the Single-Burst Reed-Solomon. This code can detect and correct single- and double-burst errors. The size of the protect field is five bytes. So, 10 and 15 check bytes are generated (two or three interleaves). The table below lists the guaranteed capabilities of this code under worst case conditions:

Sector Size (# of bytes)	Detection Single-Burst	Capability Double-Burst	Correction Single-Burst	Capability Double-Burst	# of Check Bytes
128	49	17	25	9	10
256	49	17	25	9	10
512	81	25	41	17	15



9480A 1-56

Figure 1-56 ECC Overhead Comparison

EXTERNAL ECC

The fourth option available to protect the disk data is External ECC. This option disables all internal error detection/correction mechanisms on the data field. The sector ID field is still protected by the internal CRC/CCITT. The HDC provides the appropriate command/status protocol to simplify the connection of external ECC logic (Figures 1-57 and 58). The controlling state machine is contained inside the HDC.

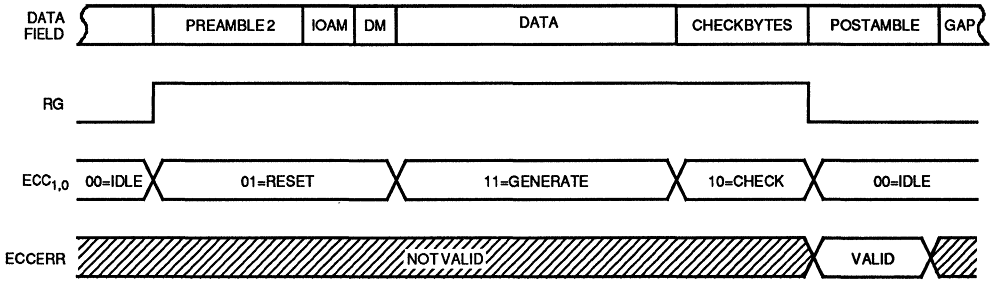
Three lines ($ECC_{1,0}$, and ECCERR) implement this control/status interface. $ECC_{1,0}$ present the status of the HDC to allow the external ECC to run synchronously. The status is coded in Gray code; only one bit changes when going from one state to the next.

In the IDLE (00B) state no data field of a sector is written or read. The external ECC should be inactive.

RESET (01B) should reset the external ECC to prepare itself for a ECC process. The ECC syndrome bits should be preset to the appropriate value. At the end of the data mark, while reading or writing the last bit, the status lines change into the GENERATE (10B) state.

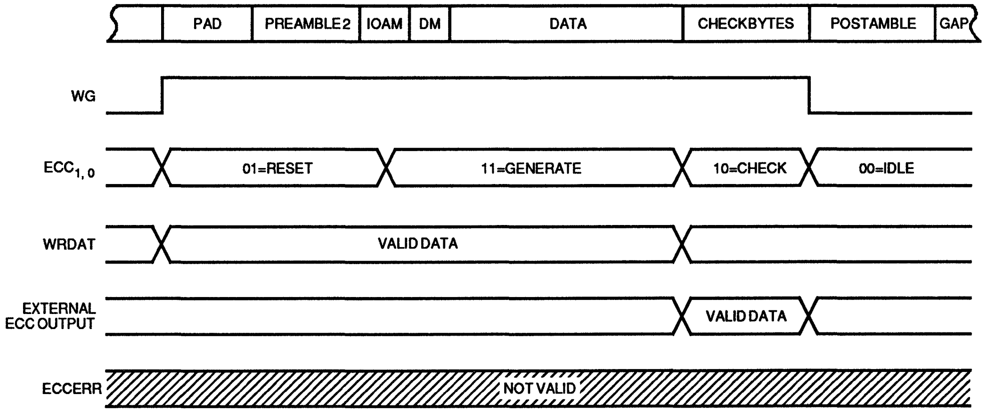
On the next rising edge of the RD/REFCLK the external ECC must be prepared to receive valid data on either the RDDAT or WTDAT lines (depending on whether RG or WG is asserted). The external ECC should generate the check bytes. When reading or writing the last bit of the data field, the lines change to the CHECK state.

CHECK (11B) enables the external ECC either to multiplex the check bytes on WRDAT (WG active) or to compare the generated check bytes with the bytes read from RDDAT (RG active). With the external ECC, a programmable number of check bytes can be added to the data field of a sector (1-256 bytes).



9480A 1-57

Figure 1-57 External ECC Handshake (Read Data)



9480A 1-58

Figure 1-58 External ECC Handshake (Write Data)

On completion of the check byte field, at the last bit of the last check byte, the status lines change back to the IDLE state. On the next rising edge of the RD/REF CLK, the IDLE state will be in effect. During the last byte of the Postamble 2 field, following the check bytes, the ECCERR pin will be sampled by the HDC for an error signal from the external ECC. If the ECCERR pin is not asserted, then the HDC assumes the data is valid. If the ECCERR line is asserted, then the HDC assumes an error has occurred in the data field.

When operating the HDC in zero-sector interleave mode together with the external ECC option, the track format parameters must be chosen carefully, to allow the HDC to transfer the contiguous sectors. In the read case, the results of the data read are not available until the end of the Postamble 2. Due to this extra waiting period, the HDC must prepare itself for the next disk data operation within the gap. However, the gap is typically configured to compensate any speed variations. So, under worst case conditions, this field can have a length of zero. In order to allow the HDC to perform zero-sector interleaving, the gap length (intersector gap) should be extended. The number of extra bytes, beyond the amount normally allocated for speed tolerances, is a function of the system clock and the read/reference clock (disk clock). This value can be determined from the formula or the table given in the "Sector Interleaving" section (value "P").

SYSTEM INTERFACE

For both Slave Mode and Master Mode the system bus interface can be programmed for byte ($B/\overline{W}$ HIGH) or word ($B/\overline{W}$ LOW) transfers.

Slave Mode

In Slave Mode, the host CPU can access the five internal registers of the HDC. A_0–A_3 indicate the address of the internal registers. In Byte Mode, the High byte is accessed if A_0 is HIGH and the Low byte is accessed if A_0 is LOW. In Byte Mode, $\overline{BHE}$ is ignored and the HDC asserts $\overline{READY}$ to indicate that it is ready to complete the access.

Master Mode

The HDC is in Master Mode when it controls the system bus. To request the mastership on the system bus, the HDC asserts Bus Request (BREQ). The bus is granted to the HDC when Bus Acknowledge (BACK) is active. The HDC keeps BREQ asserted until it releases the bus, after finishing a DMA burst of programmable length or when the burst is preempted by removing BACK.

A byte transfer occurs in Word Mode when only one byte remains to be transferred or when the system address is odd. The throttling of DMA transfers on the system bus is controlled by the Mode Register. The HDC inserts a programmable number of software Wait States into the DMA bus cycle. Additionally, it inserts hardware Wait States until the memory asserts $\overline{READY}$.

Upper Address Latch Enable (ALEN) may latch the upper address word (A_{16} thru A_{31}) in an external address latch. The upper address is only updated if a change demands this update, or if the internal DMA controller starts a new block transfer. If the DMA burst length is set to the maximum (sector size) then the HDC updates the upper address at the beginning of every DMA burst. The upper address latch may be shared between multiple DMA devices. If the DMA burst length is less than the sector size, the HDC splits data block transfers into several bursts. The upper address is only updated for the first burst, and in this case, the upper address latch should not be shared between multiple DMA devices.

Interrupts

The HDC interrupts the host CPU when it has completed the initialization procedure executed after a hardware or software reset or when it has completed a command chain (CF-bit is set). An initialization interrupt cannot be disabled. The interrupt on command chain completion can be enabled or disabled by the Interrupt Mask bit in the Mode Register. The interrupt is reset when the upper half of the Status/Command Register is accessed.

DISK CONTROL INTERFACE

The Disk Control Interface selects drives and heads, and controls the head positioning. It is programmable to provide a floppy-disk type of interface or to conform with the ST506/412 drive interface standard.

9590

For the Am9590, the disk control interface can be also programmed to accommodate the ESDI standard. With some external logic SMD can be easily implemented.

PAUSE

Drive Selection

Drives are deselected when $\overline{\text{SELEN}}$ is HIGH. The two bits DRSEL$_{1,0}$ select one of up to four drives. DRSEL$_{1,0}$ are valid when $\overline{\text{SELEN}}$ is LOW. $\overline{\text{SELEN}}$ remains LOW as long as the drive is selected. The selected drive must acknowledge the selection by activating $\overline{\text{DREADY}}$. If $\overline{\text{SELEN}}$ is not acknowledged within 2 16 clocks, then the HDC assumes that the selected drive is not present and generates a time-out error ("Drive Selection Fault" SRB). If FAULT is asserted after activating $\overline{\text{DREADY}}$, then the HDC de-selects the drive and generates a fault error ("Drive Selection Fault" SRB).

For floppy-disk drives, the Motor-On Delay parameter determines when the HDC will time out on $\overline{\text{DREADY}}$. This gives sufficient time to the drive motor to come up to speed.

Head Positioning

The ST506 and Double-Density Floppy-Disk format performs Seek operations via the lines STEP, DIRIN, SC, RTZ, and TRK0. Normal Seeks pulse the STEP line to move the head to the desired track. Restore may pulse the STEP line or RTZ line to move the head to Track 0. DIRIN specifies the direction in which the head should move on Seek pulses. In Hard-disk Mode, the HDC pulses the STEP output once, waits for SC to go inactive (LOW), issues the remaining STEP pulses at the specified rate (See Step Width and Seek Dwell), and finally waits for SC to go active (HIGH), which indicates that the drive has completed the Seek operation.

SC is asserted by the drive to indicate that the head has moved to the desired track. Once the drive has acknowledged the completion of a seek by activating SC, it must keep SC active as long as it is selected, or until it receives another Seek command; otherwise, the HDC issues a drive fault error. When executing the Restore or Seek commands, the drive must acknowledge the first STEP pulse or the RTZ pulse (SC pulsed LOW) within 2 16 clocks or the HDC will generate a Seek error.

9590

The ESDI interface uses the serial command interface to transmit Seek and Restore commands. Before sending a command to the drive, the Am9590 waits for the COMMAND COMPLETE line to become active. It then transfers the 16-bit command word (Table 1-1). A TRANSFER REQUEST—TRANSFER ACKNOWLEDGE handshake for each bit makes this transfer independent of drive and controller speed. After sending the command, the HDC will wait for the drive to execute the Seek command (implied seek) or search the IOPB chain for a seek operation on another drive. COMMAND COMPLETE indicates that the heads are positioned over the desired tracks. The HDC will then select the particular head and wait for the time programmed as HEAD SETTLE TIME. The data transfer will be started after COMMAND COMPLETE becomes active.

Write Protect

The Write Protect line is sampled just prior to execution of a WRITE or FORMAT command. If the line is HIGH, the command is aborted. This input is sampled for both hard-disk and floppy-disk modes. For normal operation in Hard-disk Mode, this input should be LOW.

CHAPTER 2

2.0 SOFTWARE

Software Description

Three different software packages have been written for the Am9580A/90. The IOPB development program is designed to introduce the user to the Am9580A functions. An IBM/PC-DOS driver is used to give a typical example for an application program. It is written in "C" and allows most programs written for the IBM/PC to run. The third software example involves a BIOS driver that implements full compatibility to all PC programs. This software package is written in assembler language.

All software packages (source and object codes) are available from AMD and run on the Am9580A/90 disk controller board for the IBM/PC-AT.

2.1 The IOPBGENerator

The 'IOPBGENerator' is a software tool that assists the user in developing Input Output Parameter Blocks (IOPBs) commands for the Am9580A Hard Disk Controller. It allows the user to create and modify commands for the Am9580A on a high-level basis. No knowledge of the actual "layout" of the different commands and parameters is required. A series of menus guides the user through the various programming steps and in the assembly of commands and drive specifications.

Although IOPBGEN is a high-level programming tool, the user is assumed to have some knowledge about disk drives and disk controller operations, specifically the Am9580A. IOPBGEN familiarizes the user with the Am9580A by eliminating difficult bit manipulations.

How to Use IOPBGEN

The software is written for the IBM Disk Operating System (DOS). It is invoked by the command "IOPBGEN". After initializing the controller board, the main menu will show up on the screen. To select single functions from the menu, the cursor keys move a highlighted rectangle to the desired function. Upon entering the "CR" key, the selected function will be executed. This function can either be a command or a sub-menu indicated by a description appearing below the menu line when it is highlighted. The "ESC" key recalls the next higher menu level.

Commands such as EDIT generate a form on the screen which can be filled or modified. The cursor key allows skipping from item to item on these forms. Items either must be filled with decimal or hexadecimal values, or they can be chosen from a number of options which can be scrolled by using "+" and "–" keys. All form modifications are done on the screen, as long as the "CR" key has not been entered. "ESC" leaves the form without saving the modifications.

It is possible to store information on or retrieve information from disk. The "FILE" command stores and retrieves IOPB chains including drive parameter blocks. The "MEMORY" sub-menu provides two functions to save and load portions of the system memory onto the disk.

Example for an Am9580A/90 Test Program

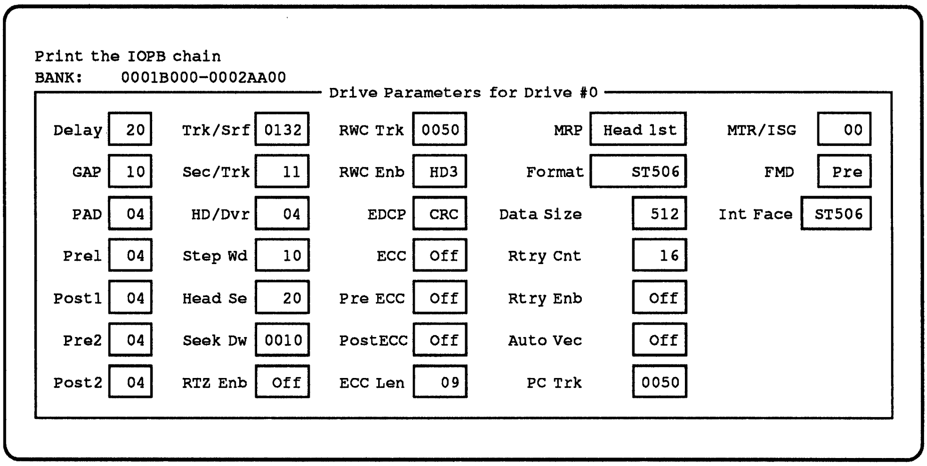
The following example shows the functionality of the IOPBGENerator software. It connects a 10 Mbyte hard-disk drive to the Am9580A/90 and runs a simple test program which includes Format, Read and Write commands.

First, the drive parameter block for the specific hard-disk drive has to be set up. Starting from the main menu of IOPBGEN, the "IOPB", followed by the "DRIVES" function will generate a form for the drive parameter block on screen. Assuming that the physical drive number will be "0", Figure 2-1 shows a typical layout for a hard-disk drive with ST506 interface. The parameters used in this example may vary for different drives.

A "CR" saves this form and shows an empty form for Drive #1. "ESC" leaves the form and returns to the "IOPB" menu.

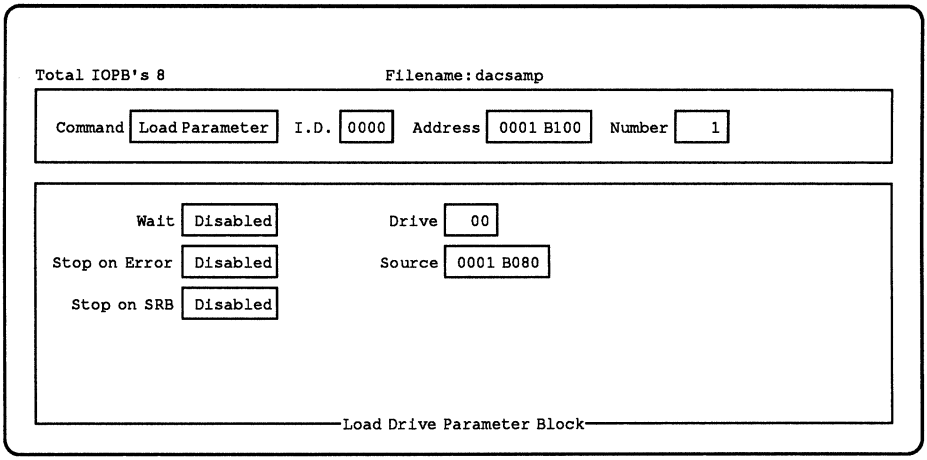
The next step is to generate some IOPBs (the actual Am9580A commands) to see if the chosen drive parameters are correct. The "EDIT" command calls an empty form for the first IOPB. After filling one of the IOPB forms, the "CR" key saves the contents and calls another empty form. All IOPBs are automatically linked together as long as no insert (Function Key F1) or delete (Function Key F2) has been invoked. In this case, the IOPBs need to be ordered with the "RELOCATE" command. The PG UP and PG DN keys scroll through the chain of IOPBs. Using the "+" and "–" keys, an appropriate command for the hard-disk controller may be chosen. Each command will show its own individual set of possible parameters.

The Load Parameter function (Figure 2-2) allows the selection of WAIT, STOP ON ERROR, and STOP ON SRB options. This function will display a drive number and the source address of the drive parameter block which may be modified by the user.



```
Print the IOPB chain
BANK:     0001B000-0002AA00
                                  Drive Parameters for Drive #0

  Delay  20    Trk/Srf 0132   RWC Trk  0050        MRP  Head 1st     MTR/ISG    00
    GAP  10    Sec/Trk   11   RWC Enb   HD3     Format     ST506         FMD   Pre
    PAD  04     HD/Dvr   04      EDCP   CRC  Data Size       512    Int Face ST506
   Prel  04    Step Wd   10       ECC   Off   Rtry Cnt        16
  Post1  04    Head Se   20   Pre ECC   Off   Rtry Enb       Off
   Pre2  04    Seek Dw 0010   PostECC   Off   Auto Vec       Off
  Post2  04    RTZ Enb  Off   ECC Len    09     PC Trk      0050
```

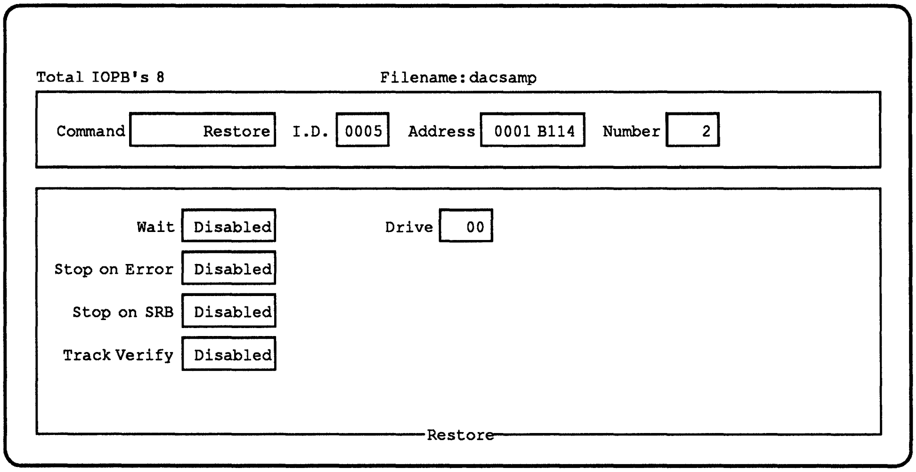
Figure 2-1



```
Total IOPB's 8                       Filename:dacsamp

  Command Load Parameter  I.D. 0000  Address 0001 B100  Number   1

           Wait  Disabled          Drive   00
   Stop on Error  Disabled         Source  0001 B080
     Stop on SRB  Disabled

                          Load Drive Parameter Block
```

Figure 2-2

The next seven IOPBs (Figures 2-3 to 2-9) show the rest of the small test-program which will format two different tracks, write two different blocks of data to the disk, and read them back.



Total IOPB's 8 Filename:dacsamp

Command Restore I.D. 0005 Address 0001 B114 Number 2

Wait Disabled

Stop on Error Disabled

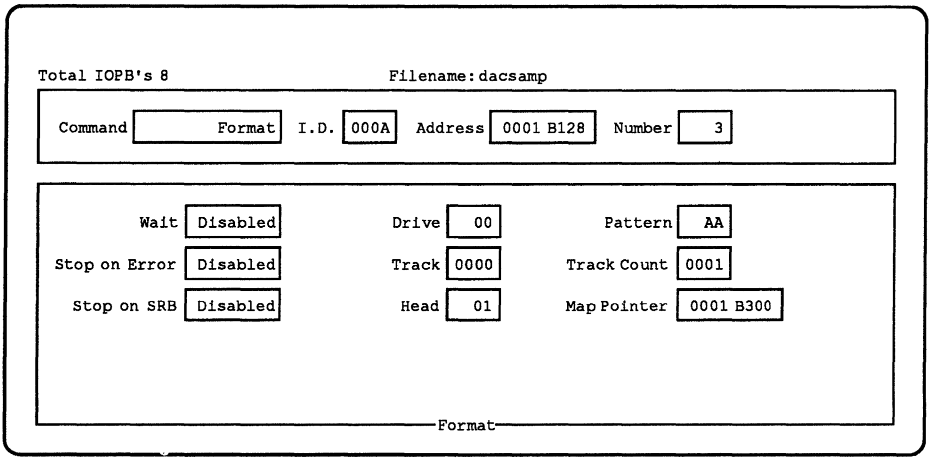
Stop on SRB Disabled

Track Verify Disabled

Drive 00

Restore

Figure 2-3



Total IOPB's 8 Filename:dacsamp

Command Format I.D. 000A Address 0001 B128 Number 3

Wait Disabled

Stop on Error Disabled

Stop on SRB Disabled

Drive 00

Track 0000

Head 01

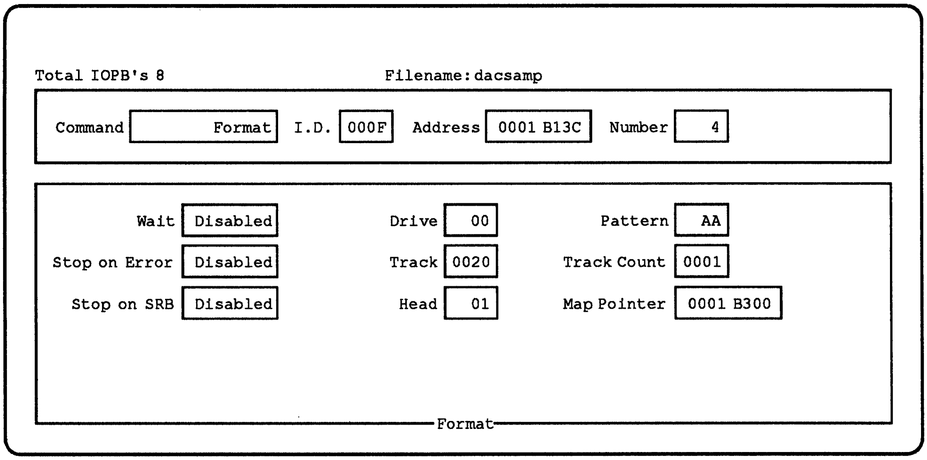
Pattern AA

Track Count 0001

Map Pointer 0001 B300

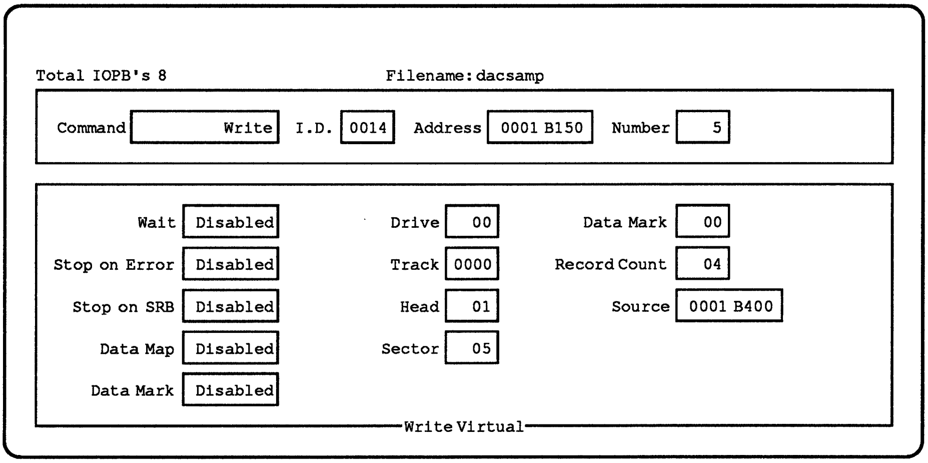
Format

Figure 2-4



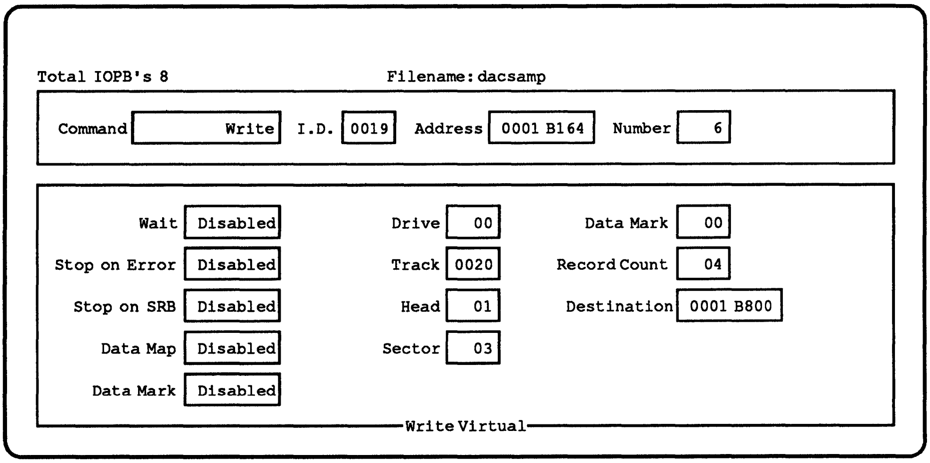
```
Total IOPB's 8                    Filename:dacsamp

Command        Format  I.D. 000F  Address 0001 B13C  Number  4

         Wait  Disabled       Drive    00        Pattern   AA
Stop on Error  Disabled       Track  0020    Track Count 0001
  Stop on SRB  Disabled        Head    01    Map Pointer 0001 B300

                              Format
```

Figure 2-5



```
Total IOPB's 8                    Filename:dacsamp

Command         Write  I.D. 0014  Address 0001 B150  Number  5

         Wait  Disabled       Drive    00      Data Mark   00
Stop on Error  Disabled       Track  0000   Record Count   04
  Stop on SRB  Disabled        Head    01         Source 0001 B400
     Data Map  Disabled      Sector    05
    Data Mark  Disabled

                           Write Virtual
```

Figure 2-6



Total IOPB's 8 Filename: dacsamp

Command Write I.D. 0019 Address 0001 B164 Number 6

Wait Disabled Drive 00 Data Mark 00

Stop on Error Disabled Track 0020 Record Count 04

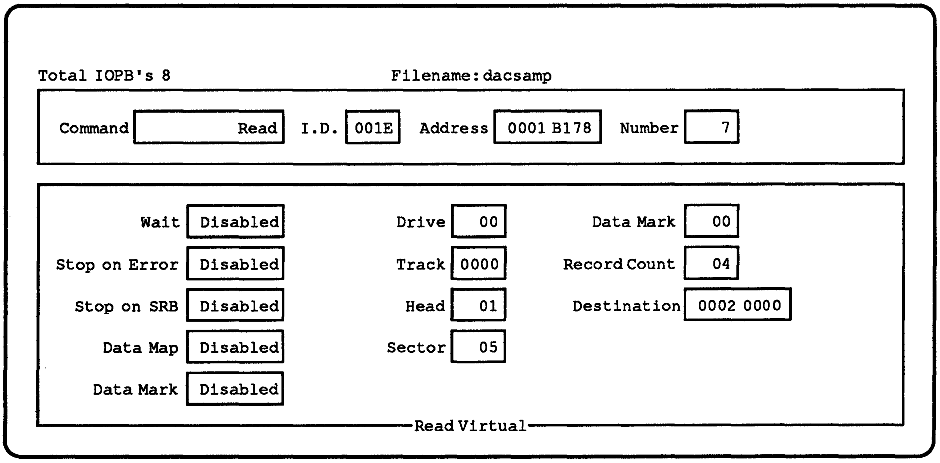
Stop on SRB Disabled Head 01 Destination 0001 B800

Data Map Disabled Sector 03

Data Mark Disabled

Write Virtual

Figure 2-7



Total IOPB's 8 Filename: dacsamp

Command Read I.D. 001E Address 0001 B178 Number 7

Wait Disabled Drive 00 Data Mark 00

Stop on Error Disabled Track 0000 Record Count 04

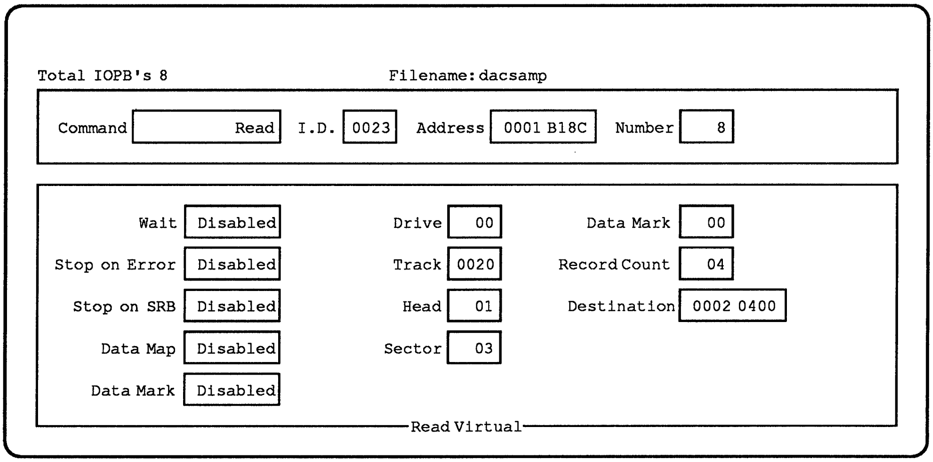
Stop on SRB Disabled Head 01 Destination 0002 0000

Data Map Disabled Sector 05

Data Mark Disabled

Read Virtual

Figure 2-8



```
Total IOPB's 8                      Filename:dacsamp

 Command [          Read] I.D. [0023]  Address [0001 B18C]  Number [  8]

           Wait [Disabled]          Drive [  00]        Data Mark [ 00]
  Stop on Error [Disabled]          Track [0020]     Record Count [ 04]
    Stop on SRB [Disabled]           Head [  01]      Destination [0002 0400]
       Data Map [Disabled]         Sector [  03]
      Data Mark [Disabled]
                                   Read Virtual
```

Figure 2-9

The FORMAT command used in this test program uses a mapping-list located at memory location 1A00H. This list can be generated by using the "MEMORY" – "MODIFY" command.

The main menu provides an "EXECUTE" command to execute the assembled hard-disk controller program. This command calls a submenu that allows the program to be dumped to the system memory and executed with the "GO" – "ALL REG" command. Upon completion, a status line shows if the program was correctly executed. The status line is an interpretation of the status register of the Am9580A. In case of an error, Status Result Blocks (SRBs) are written to memory. These SRBs can be interpreted by using the "SRBINT" command; it shows the user which error has occurred and for which IOPB.

2.2 The DOS Driver

The Disk Operating System allows special hardware drivers that are loaded during power up. These drivers can be used by all application programs that call DOS to do file transfers. Since it is uncommon for application programs to directly call the BIOS routines, a DOS driver will work in almost all cases.

The advantage of a DOS driver over a BIOS driver is readability. It is possible to write DOS drivers in a high-level language such as "C". DOS drivers are also software loadable, whereas BIOS drivers must be in system PROM.

The DOS driver written for the Am9580A/90 supports up to two hard-disk drives. When loaded, the driver will automatically locate the new drives above the device identifiers (e.g., A, B, C) already used by the system. In the standard configuration, the DOS driver automatically assumes a 10 MByte disk drive with four heads and 306 cylinders. This can be easily changed in a table in the source code.

In order to start the DOS driver during power up, the hard-disk drive must be physically formatted; a separate program (drvrtest) can be used. The program also allows the hardware to be exercised and the basic software routines to be tested.

2.3 The BIOS Driver

The BIOS driver allows all IBM/PC programs to run on a hard-disk drive using the Am9580A controller board. This software is written in assembly language and supports ST506 and ESDI disk drives. Therefore, the original list of disk drives used by the IBM-PC/AT has been modified. The entry FH is now reserved for ESDI disk drives. Only one entry for all ESDI drives is required since the BIOS Driver configures the Am9590 automatically according to the drive specifications. This is done by requesting all the necessary information from the ESDI drive during initialization.

A BIOS driver usually resides in system ROM. Since it is not desirable to exchange the system software for the HDC emulation board, there is a loadable version of the BIOS driver available. In the loadable version of the driver software, the drive type entry cannot be located in the CMOS configuration RAM of the IBM-PC/AT. A "setup" procedure allows a configuration entry that emulates the boot-up diskette. This program writes a setup-configuration file onto the diskette from where it is running. The same diskette must be used when installing the BIOS driver in RAM.

The procedure will ask for the number of disk drives (1 or 2) and the type of disk drive. All entries are similar to the original IBM/AT entries. The reserved entry FH is used to indicate an ESDI drive.

Please note that the maximum usable drive capacity is 33 MByte independent of the actual drive capacity. This is due to a DOS limitation and can be avoided using special drive partitioning software.

The BIOS driver itself must be started after system boot up. The procedure "hdcinstl.exe" installs the driver and reboots the system. The BIOS driver is then installed and the hard-disk drive(s) will get the next available drive identifier(s).

Running the BIOS driver for the first time

If the hard-disk drive used is not formatted at all or was formatted with a different controller, the BIOS driver cannot be installed. In order for BIOS to recognize a hard-disk drive, it must be physically formatted. In this case, the program "IOPBGEN" can be used to initially format the disk drive making it possible to install the BIOS driver.

In any case, it is necessary to format the drive with the same parameters that the BIOS driver uses. A physicall format program as provided by IBM (Service Manual) can, therefore, substitute the "IOPBGEN" program for ST506 drives. Such a program also allows the flagging of defects on the drive media.

After physically formatting the drive, the DOS procedure "fdisk" partitions the drive into different sections (see DOS manual) and initializes the file allocation table. After a new warm reset (CTRL-ALT-DEL), the DOS function, "format", logically formats the disk drive and makes it available for BIOS.

TYPE	CYLINDERS	HEADS	WRITE PRE-COMP	LANDING ZONE
1	306	4	128	305
2	615	4	300	615
3	615	6	300	615
4	940	8	512	940
5	940	6	512	940
6	615	4	no	615
7	462	8	256	511
8	733	5	no	733
9	900	15	no8	901
10	820	3	no	820
11	855	5	no	855
12	855	7	no	855
13	306	8	128	319
14	733	7	no	733
15	RESERVED—SET TO ZEROS			

Table 2-1

CHAPTER 3

3.0 APPLICATIONS

3.1 Am9580A/90 Bus Interfaces

The Am9580A/90 Hard Disk Controller (HDC) is an intelligent peripheral controller. Its universal bus interface allows easy interfacing to various bus types:

- 8-bit and 16-bit data bus width
- up to 32-bit address space
- multiplexed/demultiplexed address/data buses
- synchronous/asynchronous buses

A large variety of systems can take advantage of its advanced features. This applies to minicomputer systems as well as microcomputer systems based on popular CPUs, such as the 8051, Z80, iAPX family of microprocessors (8088, 8086, 80188, 80186, 80286), 680XX, and 320XX microprocessors.

In this chapter, the first section discusses the general considerations in interfacing with the HDC. It is followed by a number of specific implementations.

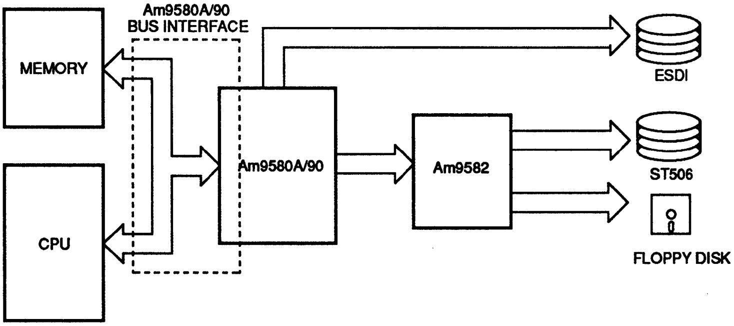
3.1.1 General System Bus Application Hints

The following lists the specific functions that need to be taken into consideration when using the HDC.

Reset

After power-up the HDC requires a positive pulse on the RESET input to start the initialization procedure. The minimum width of the reset pulse is 2 clock cycles. Operating at 10 MHz bus clock, the HDC takes approximately 200 ms to execute this initialization procedure. After initialization, the HDC issues an interrupt request. This interrupt cannot be masked off by programming the HDC. It is the responsibility of the system to either respond to or disregard the interrupt. The interrupt request stays active until the CPU accesses the Status/Command Register.

If the CPU attempts to access the HDC registers prior to the completion of reset, the $\overline{\text{READY}}$ line remains inactive until reset is completed; this causes the CPU to wait.



9480A 3-1

Figure 3-1 Am9580A Bus Interface

Interrupts

The HDC interrupts the CPU upon entering the IDLE state (see Status/Command Register description). Interrupts except hardware reset interrupt may be disabled by setting the Interrupt Mask bit (IM-bit) in the Mode Register. However, the hardware reset interrupt cannot be disabled this way. The interrupt request line then either connects to the non-vectored interrupt input of the CPU or to the input of an interrupt controller such as the 8259A or Am9519A to support vectored interrupts. The 80188 and 80186 have on-chip interrupt controllers.

The interrupt controller must also resolve the interrupt priority in systems with multiple interrupt sources. Systems employing Active Low interrupt request bus, driven by open collector gates, can be supported by interfacing an inverter with open collector output to the INTR output (Active High, totem-pole output).

Byte/Word Strap

The Byte/Word (B/$\overline{\text{W}}$) strap pin selects either a byte (8-bit) or word (16-bit) interface. Typically, this pin will be strapped to either Ground (word mode) or Vcc (byte mode). However, this pin may also be dynamically controlled. For instance, the HDC may interface to an 8-bit CPU in slave mode, but to 16-bit memory in master mode; in which case the pin would be driven to Low (V_{IL}) in slave mode and High (V_{IH}) in master mode.

$\overline{\text{READY}}$

Ready is a bidirectional signal. When the CPU accesses the HDC in slave mode, this signal indicates that the current transfer cycle may be terminated. It is essential that this output is connected to the Wait or Ready input of the CPU because the access time is variable. It can vary from 2 to 16 HDC bus clock cycles (timing parameter 106) because the slave access is internally synchronized to the operation of the microsequencer.

Note: $\overline{\text{READY}}$ has the opposite polarity from that of the ready input of iAPX µprocessors. Ready must always be connected to the Ready/Wait input of the CPU.

In master mode this input allows the bus cycle to be extended for slow memories or peripheral devices. However, in master mode it may be tied Low permanently either to insert no wait states or to control wait state insertion by programming the Mode Register. The later allows automatic insertion of up to three wait states in each HDC bus cycle.

Slave Transfers

After reset, the CPU initializes the internal registers of the HDC to activate command processing. The register set is kept small (only five registers) to support a very flexible and transparent, memory based communication between the CPU and HDC.

The CPU initiates a Slave transfer cycle by asserting chip select ($\overline{\text{CS}}$). The slave address bus (A_0–A_3), in combination with Byte High Enable ($\overline{\text{BHE}}$), selects the appropriate register. In byte mode, $\overline{\text{BHE}}$ is disregarded and A_0 specifies which half of the 16-bit register is to be accessed. In word mode, $\overline{\text{BHE}}$ and A_0 indicate whether the CPU is performing a word transfer (both are Low), or a byte transfer to the high byte ($\overline{\text{BHE}}$ Low, A_0 High) or Low byte ($\overline{\text{BHE}}$ High, A_0 Low).

Note: iAPX microprocessors do not provide a latched BHE.

Upper Address Latch

The upper 16-bit of the 32-bit linear address is multiplexed on the 16-bit address/data bus. The HDC updates the upper address when starting the execution of a new IOPB, and on a demand basis while executing the IOPB; this minimizes the overhead of upper address updates.

Each upper address cycle consists of four clock cycles where the upper address is strobed out during the first clock cycle (T1). The Upper Address Strobe (ALEN) indicates that the upper address is stable. The address may be latched by a standard transparent latch such as the Am29841 (10-bit latch, ALEN connected to LE) or a 74LS373 (8-bit latch, ALEN connected to G).

Note: The upper address latch must not be shared between devices. The HDC must have its own upper address latch.

However, the lower address latch may be shared between devices because the lower address is updated at the start of each bus cycle. A common address latch enable can be generated by ORing the various address latch enables and/or inverted address strobes.

Clock Requirements

The system clock input is TTL compatible, therefore, the HDC does not require special clock drivers. At its highest frequency the HDC demands a 50% duty cycle. At a lower frequency the duty cycle is irrelevant if the minimum width of both clock High and Low are satisfied. The system clock must not stop.

Memory Organization of Various CPUs

The Am9580A has a memory interface that adapts to the memory organization of CPUs such as 8086, 80186, etc. However, it is possible to connect the Am9580A to other CPUs, such as a 68000, without problems despite the difference in byte organizations of the two processors.

For both types of CPU, word boundaries are on even addresses, the low-order byte is transferred on the lower part of the data bus and the high order byte on the upper lines. Therefore, there is no difference between the two types of CPU when only transferring data words starting on an even address. However, when accessing data bytes, there is a difference between memory organizations, as shown in Figure 3-2.

In the following example the CPU is assumed to be a 68000 processor; the Am9580A is the device with the Intel memory interface. Assuming the 68000 has written a byte string into the system memory, the Am9580A will try to read these data starting with an even address. It expects the low-order byte on the lower half of the data bus corresponding to this address, and the system memory will put the high-order byte on the upper data bus. As a result, the disk controller will fetch invalid data for each byte access. The same problem occurs for write accesses.

There are several solutions. The most common way is to use only data word transfers. In most cases this will be sufficient because only this mode can the HDC take full advantage of the higher speed of a 16-bit bus. As shown in Figure 3-3, even when using word transfers, bytes will be written in a reversed order to the disk media; the HDC always puts out the low-order byte first.

There are two ways to implement a data byte transfer when required. Figure 3-4 shows an application that swaps data bytes by using two transceivers. This approach works for all applications using only byte mode. The Am9580A, however, accesses command words as 16-bit words and not as bytes. As a result command words are swapped and must be written by the CPU into memory in a reverse order. This can be easily done by software.

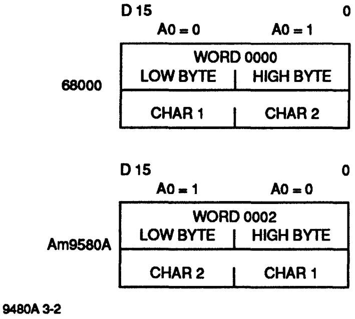


Figure 3-2 Motorola-Intel Byte Order

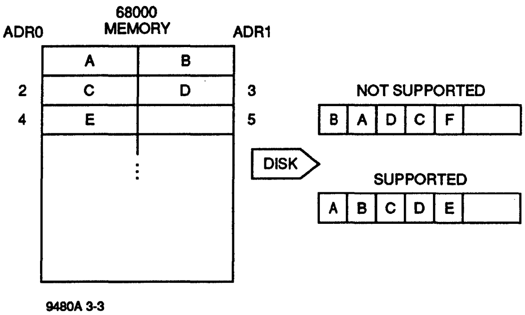
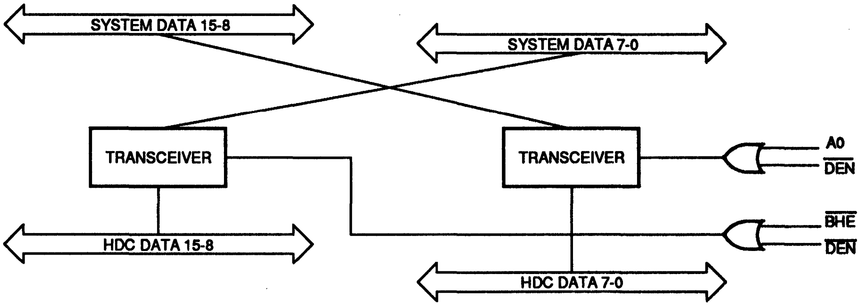


Figure 3-3 Word Transfers

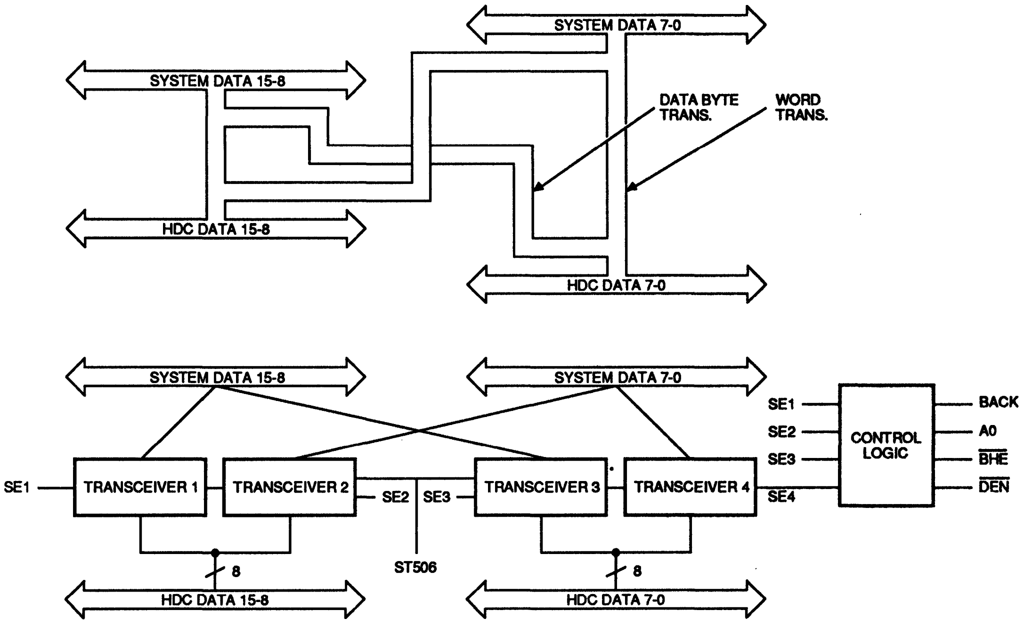
Figure 3-5 shows a fully adapted interface. Four transceivers allow differentiation between word and byte accesses and provide two data paths. One of them transfers the data straight to the Am9580A (word transfers), the other one swaps the data bytes (byte transfers). Control logic enables the different drivers according to the state of the signals $\overline{BHE}$ and A0, as shown in the following diagram. AMD has a new part, the Am29C923, that can accomplish all the combinations shown in Figure 3-5.

$\overline{BHE}$	A0	Size
L	L	word
L	H	upper byte
H	L	lower byte
H	H	reserved



9480A 3-4

Figure 3-4 Swapping Data Bytes



9480A 3-5

Figure 3-5 Swapping Data Bytes

3.2 BUS INTERFACES

3.2.1 VME Bus Interface

Meeting DC Requirements For VME

The VME specification puts some rather stringent DC requirements on bus signals. AMD makes a family of bus interface parts that satisfy the requirement of most of the VME bus signals. The following is a summary of devices that could be used as bus drivers/receivers.

The Bus Grant and the Interrupt Acknowledge daisy-chain signals need not be buffered because their drive requirement is moderate. Other signals, such as AS and DS, are buffered by a 74S244.

As a bus slave, the HDC card must receive an address from the VME bus and decodes it to generate a Chip Select to the Am9580A. AMD's Am29809 and Am29806 equal-to-comparators are ideal for this purpose. In slave mode, the AmPAL22V10 receives all necessary control signals from the VME bus to initiate the required bus cycle at the Am9580A. To complete the cycle, $\overline{\text{DTACK}}$ is generated by buffering the Am9580A $\overline{\text{READY}}$ signal through the address decoder, and $\overline{\text{ACK}}$ is output to meet the drive requirements of $\overline{\text{DTACK}}$ on VME. The Am2947 provides data transceiver functions necessary to drive the VME bus or the Am9580A. The Am29863 or Am29C982/983 can also be used.

Two signals must go through a 74LS38 open-collector AND gate to provide the drive capability and proper output type, Bus Request ($\overline{\text{VBRX}}$), and Bus Busy ($\overline{\text{VBBSY}}$) out of the PAL device. The interrupt out of the Am9580A ($\overline{\text{INT}}$) should be connected to a VME compatible interrupt controller.

VME Address Modifier Lines

These lines, as specified by VME, allow the master presently on the bus to pass additional information to the slave it is addressing. Some of the address modifier codes are defined by VME to perform certain functions, while others are user-definable. These address modifier codes allow such advanced features as system partitioning (i.e., privileged or non-privileged accesses), memory mapping, or sequential access cycles that allow the accessing of several locations without providing a separate address each time. This design shows the address modifier lines used that only 15 additional address lines need to be decoded. This allows accesses such as "short supervisory I/O access". The user can easily modify this example according to the requirements.

Handling Bus Exceptions

The VME specification provides for a bus time-out module that can detect system errors or problems. This module monitors DTACK, Bus Error (BERR), and also the rising edge on either Data Strobe. If DTACK was not asserted, the time-out module will drive DTACK and BERR low. The transfer is then completed and the CPU is informed of a problem.

The Am9580A can also use a portion of this feature to abort a DMA in progress if a bus exception is detected. The PAL device moniotrs the Bus Error signal ($\overline{\text{BERR}}$). If the Am9580A is bus master and $\overline{\text{BERR}}$ becomes active at any time, the PAL device will suppress any $\overline{\text{READY}}$ signal to the HDC. This will force the HDC to time out on a bus cycle. The CPU will be notified of this bus time-out error by an interrupt and/or a Status Result Block.

Design of the VME Bus to 9580A Interface PAL Device

In this design, a single AmPAL22V10 provides the bus arbitration handshake and most of the control signal translation necessary to allow the Am9580A HDC to perform both as master and slave on the VME bus. The arbitration handshake uses three VME signals and two Am9580A signals. To initiate the bus arbitration phase to gain bus mastership, the Am9580A asserts its bus request line (BREQ). Once the PAL device detects this signal, it generates the VME bus request ($\overline{\text{VBRX}}$) if there is currently no active VME bus grant ($\overline{\text{VBGXIN}}$). This conditioning of the VME bus request is necessary in order to prevent the Am9580A from improperly usurping the bus from a lower priority device on the bus grant daisy-chain.

After the bus request ($\overline{\text{VBRX}}$) has been issued, the Am9580A bus acknowledge signal (BACK) is generated upon receipt of the next active VME bus grant in

($\overline{\text{VBGXIN}}$). Simultaneously, the VME bus busy line ($\overline{\text{VBBSY}}$) is asserted. This indicates to the rest of the VME system that the data transfer section of the bus is in use. Also, upon receipt of the bus grant ($\overline{\text{VBGXIN}}$), the bus grant daisy-chain is severed if there is a current Am9580A bus request (BACK). This prevents passing the grant down the daisy-chain and captures the bus for the Am9580A.

The AmPAL22V10 also generates the VME address strobe and data strobes ($\overline{\text{VAS}}$, $\overline{\text{VDS0}}$, $\overline{\text{VDS1}}$) combinatorially from the Am9580A read, write, and byte high enable control lines; and the least significant address bit ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{BHE}}$, A0) when the Am9580A is in master mode. When in slave mode, the AmPAL22V10 generates the read and write strobes ($\overline{\text{RD}}$, $\overline{\text{WR}}$) from the decoded chip select ($\overline{\text{CS}}$ from the Am29806/809) and the VME bus write signal ($\overline{\text{VWRITE}}$). The AmPAL22V10 also generates the control for the Am2947 bus transceivers from the chip select, bus acknowledge and data transmit/receive signals ($\overline{\text{CS}}$, BACK, IDT/$\overline{\text{R}}$). The equations for this PAL device are shown in Figure 3-6b.

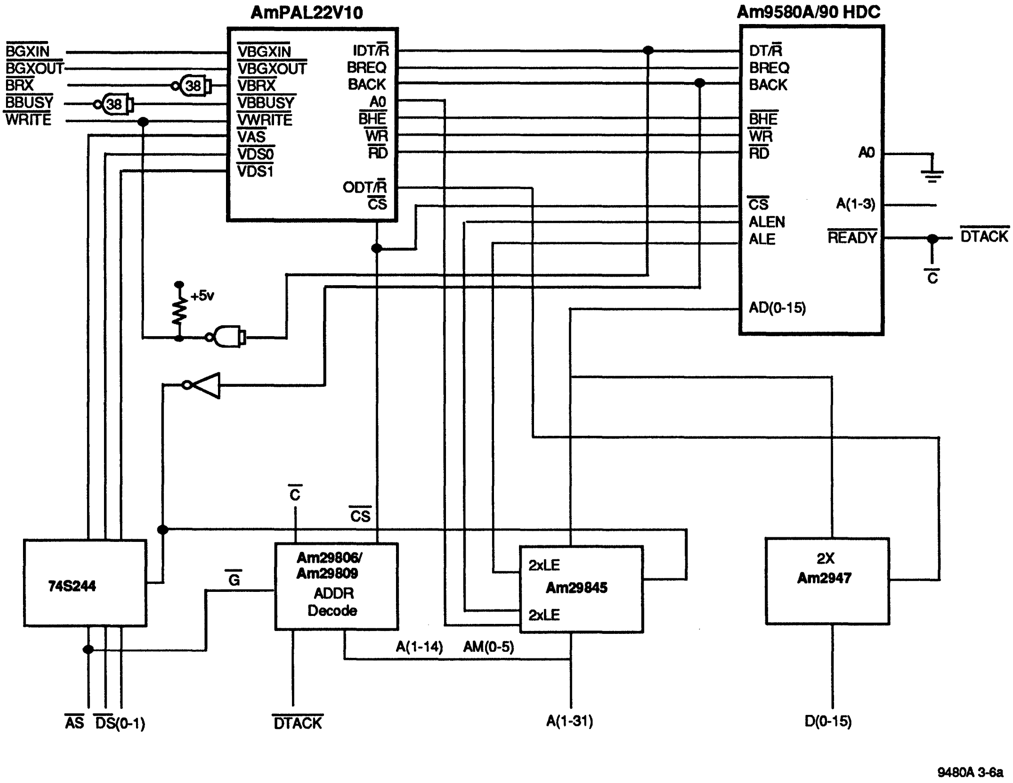


Figure 3-6a Am9580A/90 to VME Bus Interface

```
DEVICE VME_bus_to-Am9580_interface (AmPAL22V10) :

"This device performs all of the signal translation necessary to interface the Am9580A Hard
Disk Controller to the VME bus.  It also implements the handshake necessary for an option ONE
Priority Arbiter.  This file can be assembled on AMD's PLPL Assembler."

PIN

"Inputs from Am9580A"
          /cs = 1          a0 = 2          /bhe = 3
          breq = 4         idtr = 7

"Inputs from VME bus"
          /vbgxin = 5      /vwrite = 6

"Outputs to Am9580A"
          back = 14        odtr = 23

"Outputs to VME bus"
          /vas = 15        /vds0 = 16      /vdsl = 17
          /vbrx = 18       /vbbsy = 19     vdgxout = 20

"Bidirectional Am9580A signals"
          /rd = 21         /wr = 22;

BEGIN

"VME arbitration signals"
          vbrx = breq • /back • /vbgxin • /vbbsy;

vbgxout = vbgxin • /breq • /vbbsy
           + vbgxin • vbgxout • /vbbsy;

vbbsy = back;

"Bus acknowledge to Am9580A"
          back = breq • vbgxin • /vbgxout
               + breq • vbbsy • back;

"Slave mode signals"
          IF (/back) THEN ENABLE (/rd);
          rd = cs • /vwrite;

          IF (/back) THEN ENABLE (/wr);
          wr = cs • /vwrite;

          odtr = /vwrite • cs;

"Master mode signals"
          vas = (rd + wr);

          vds0 = /bhe • (rd + wr);

          vdsl = /a0 • (rd + wr);

          odtr = idtr • back;

END
```

Figure 3-6b

3.2.2 IBM PC/XT Bus Interface

Introduction

This section describes the interface of the Am9580A/82 to the IBM PC bus while maintaining compatibility with both the Xebec command set and the IBM PC Advanced Diagnostics. This interface is partitioned into three sections, starting with the control nucleus and branching out to both the PC bus and the hard drive interfaces.

Design

8051/9580A Interface

The function of the 8051 is threefold:

1. Initialize and handle the boundary and exception conditions of Am9580A operations.
2. Provide the Xebec to Am9580A command set translation.
3. Arbitrate and monitor the transfer of data between the Am9580A and the IBM PC bus.

The Am9580A operates in byte-wide and asynchronous READY mode. The 8051/ Am9580A interface is divided into two sub- sections: the slave mode interface and the master mode interface. Both modes of operation utilize the same address, data and control signals which simply are made up of 8051 I/O port control lines (A3–A0, AD07–AD00, $\overline{\text{CS}}$, BREQ, BACK, RD, WR, READY, INTR). See Figure 3-7.

Slave Mode

Slave mode operations provide the 8051 access to the sixteen byte-wide internal registers of the Am9580A. These accesses permit the 8051 to initialize and interrogate the state of the 9580A and initialize master mode operations for both parameter and data transfers. The transfer protocol and timing diagram are as follows:

1. Deassert BACK.
2. Wait for BREQ to be negated.
3. Assert $\overline{\text{CS}}$ and A3–A0.
4. Assert $\overline{\text{RD}}$ or $\overline{\text{WR}}$ and AD7–A0.
5. Wait for $\overline{\text{READY}}$ to be activated.
6. Sample AD7–AD0 if RD is asserted.
7. Deassert $\overline{\text{RD}}$ or $\overline{\text{WR}}$ and AD7–AD0.
8. Deassert $\overline{\text{CS}}$ and A3–A0.
9. Wait for BREQ to be activated.
10. Assert BACK.

Master Mode

Master mode operations are initialized by commands sent from the 8051 to the Am9580A internal registers during slave mode operations. There are essentially four types of master mode operations that are predetermined by the latched high order bits (A31 and A30) of the Am9580A master mode transfer address.

	A31 OR MODE 1	A30 OR MODE 0	TRANSFER TYPE
0	0	0	DATA READ/WRITE
1	0	1	IOPE READ/SRB WRITE
2	1	0	IBM I/O DATA READ/WRITE
3	1	1	IBM DMA DATA READ/WRITE

Table 3-1

The first two transfer types allow the 8051 to "spoon feed" IOPBs to the Am9580A and receive SRBs from the Am9580A. With the capability of loading IOPBs, the 8051 can initialize DPBs loading, disk data transfer and various diagnostic functions on the Am9580A. The second two transfer types are used to transfer data from the Am9580A to the IBM PC bus and vice versa. These last two operations will be described more in the IBM PC interface subsection.

The transfer protocol and timing diagram for master mode operations are as follows:

1. WAIT for BREQ to be activated.
2. Assert BACK and Deassert $\overline{\text{READY}}$.
3. Wait for RD or WR to be activated.
4. Sample MODE1 and MODE0.

5a. Sample AD0–AD7 if RD active and correct mode.

5b. Assert AD0–AD7 if WR active and correct mode.

6. Assert $\overline{\text{READY}}$.
7. Wait for RD or WR to be negated.
8. Deassert $\overline{\text{READY}}$.
9. Wait for BREQ to be negated.
10. Deassert BACK.

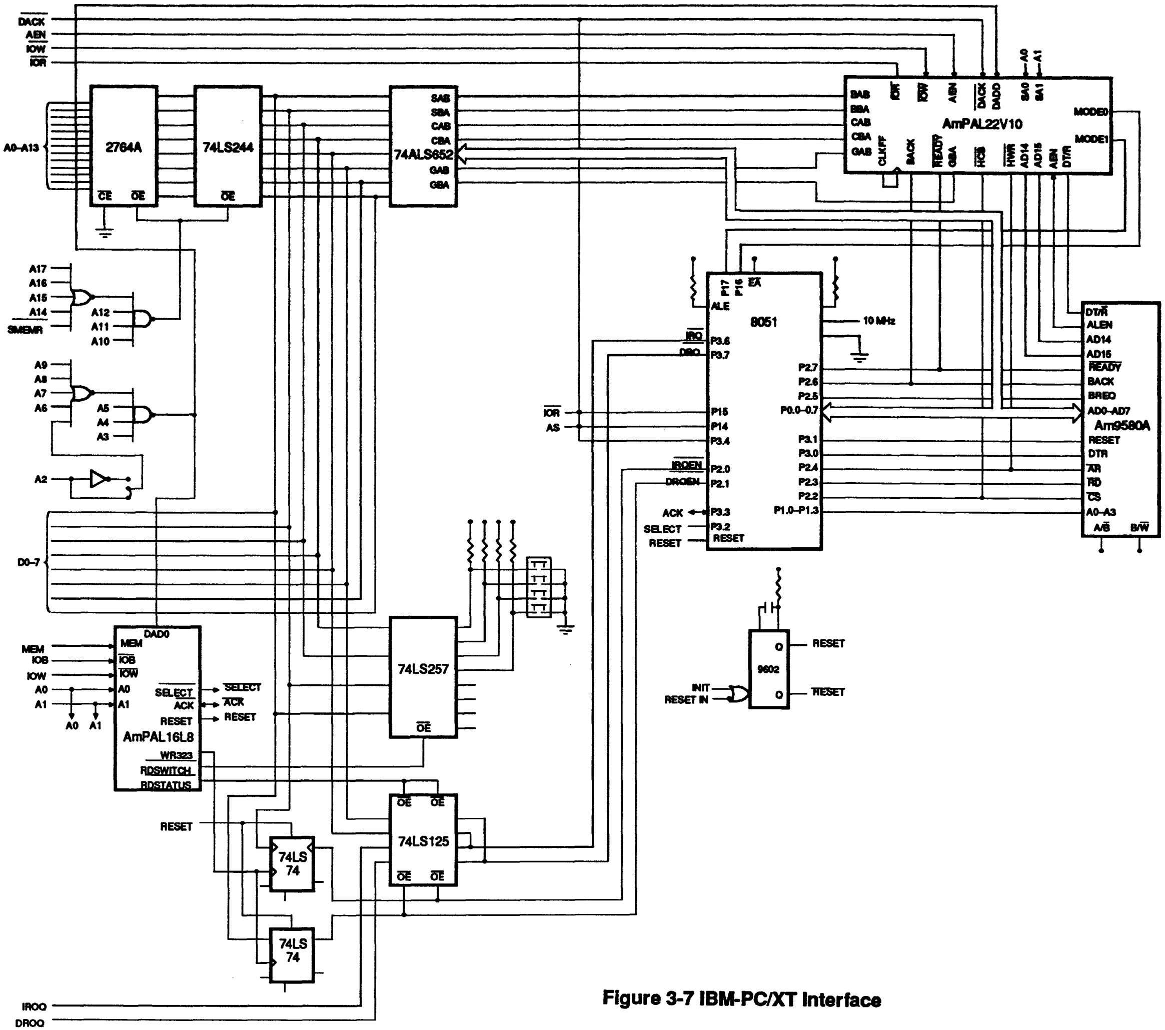


Figure 3-7 IBM-PC/XT Interface

IBM PC Interface

BIOS ROM

The BIOS ROM provides the necessary firmware for the host IBM PC to initialize, "boot" and perform data transfer and diagnostic operations to disk drive via the disk controller. The code residing in the EPROM is mapped into an 8K byte block starting at Segment 0C800H. The EPROM decode logic, consisting of the 1/2 74LS260 and 1/2 74LS13, is connected to both the EPROM's and 74LS244's $\overline{OE}$ signal. The 74LS244 buffers the EPROM's data onto the IBM PC I/O expansion bus.

I/O

The I/O peripheral interface emulates that of the IBM PC-XT disk controller. The I/O addresses are mapped into a block of four points starting at either 300H or 320H, and are comprised of 1/2 74LS260, 1/2 74LS13 and the AmPAL16L8 decode PAL. (See Figure 3-7 and Figure 3-13.) The breakdown of the I/O ports are as follows:

The I/O read ports, Status and Drive types, are comprised of the 74LS257 and the 1/2 74LS125. The source of these I/O port originates either from the 8051 or user selectable jumpers. The data in read port is located in the 74ALS652's internal A-side storage register.

I/O PORT	READ	WRITE
3X0	DATA IN	DATA OUT
3X1	STATUS	RESET
3X2	DRIVE TYPE	SELECT
3X3	NOT USED	ENABLE

Table 3-2

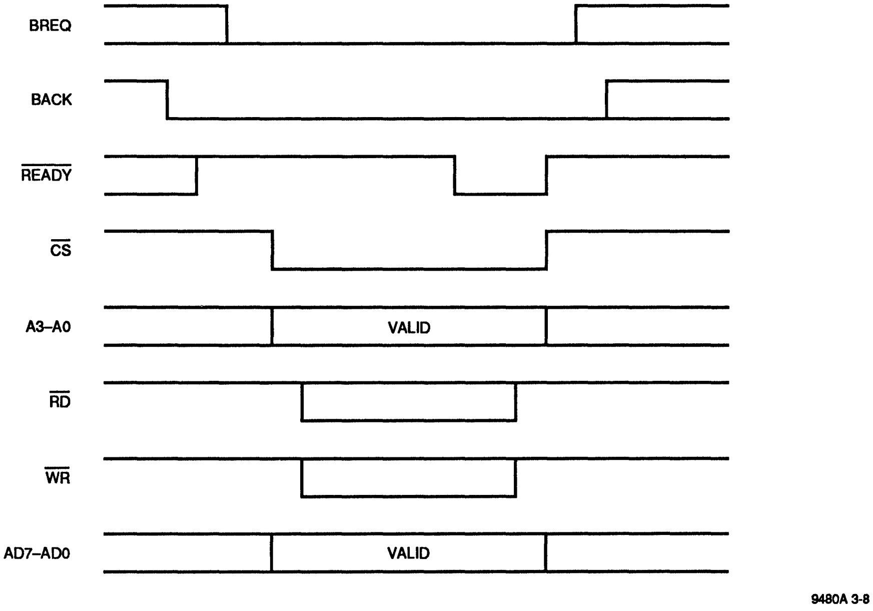


Figure 3-8 Transfer Protocol and Timing Diagram for Slave Operations

The I/O Data Out write port is also located in the 74ALS652 but is in the B-side internal storage register. A write to the reset I/O port causes a strobe to the 9602 one-shot to subsequently resetting the whole disk controller. The select I/O port generates an interrupt to the 8051 which in turn sets the Busy output. The last I/O port enables the interrupt and DMA requests onto the IBM PC expansion bus.

DMA

This subsection shows how the Am9580A, in bus master mode, transfers its data that is stored in its internal RAM directly onto the IBM PC expansion bus during an IBM DMA operation. The goals for using the following technique are:

i) To eliminate the counters and MUXs required to sequence through the RAM, and
ii) Reduce the latency of the data transfer.

Both objectives are accomplished by playing a timing trick with the Am9580A's bus interface unit, thereby, fully utilizing it on a buffer RAM and DMA counters. This in turn reduces the time required to move the data from on-chip RAM to on-board RAM to the IBM PC bus.

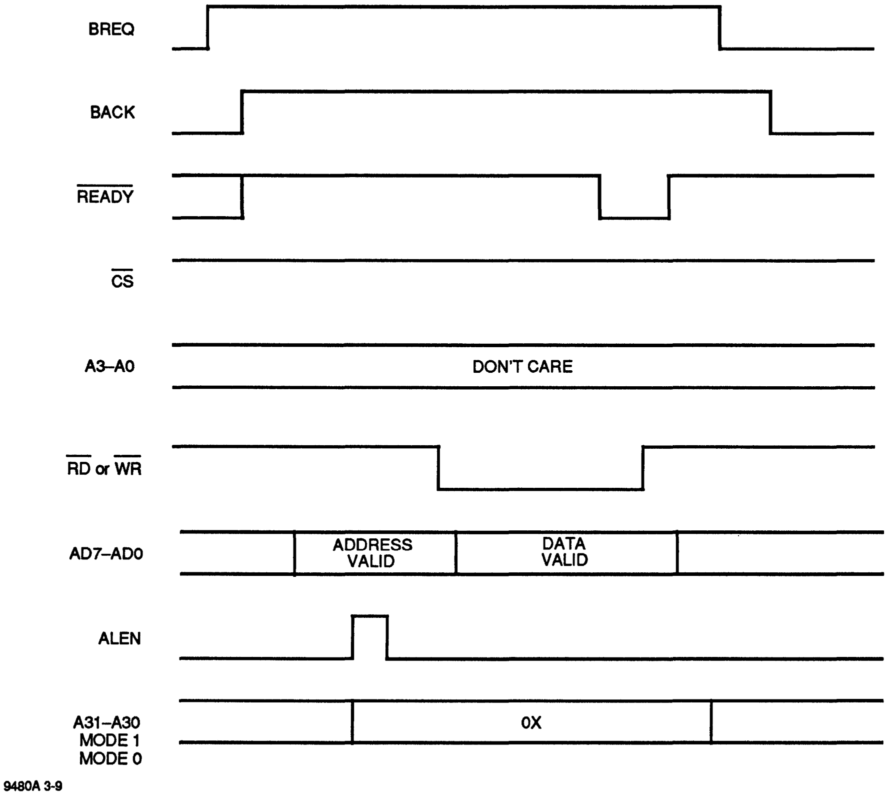


Figure 3-9 Transfer Protocol and Timing Diagram for Mode Operations

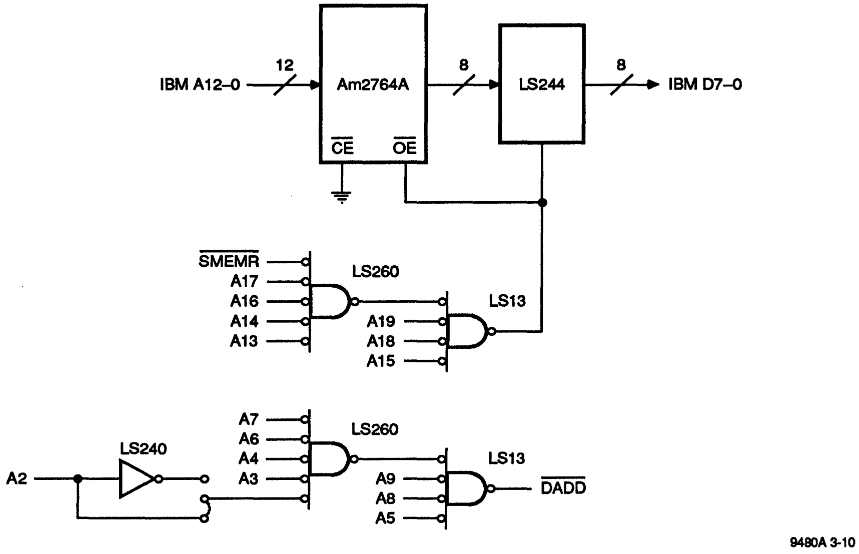


Figure 3-10

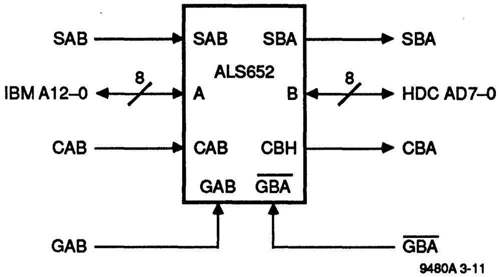


Figure 3-11

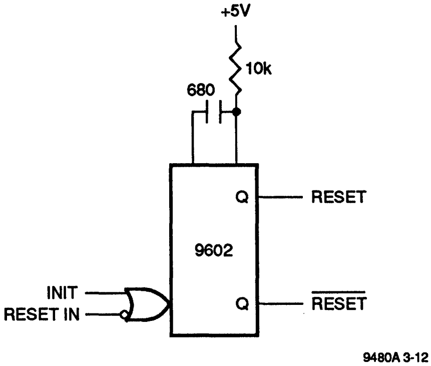
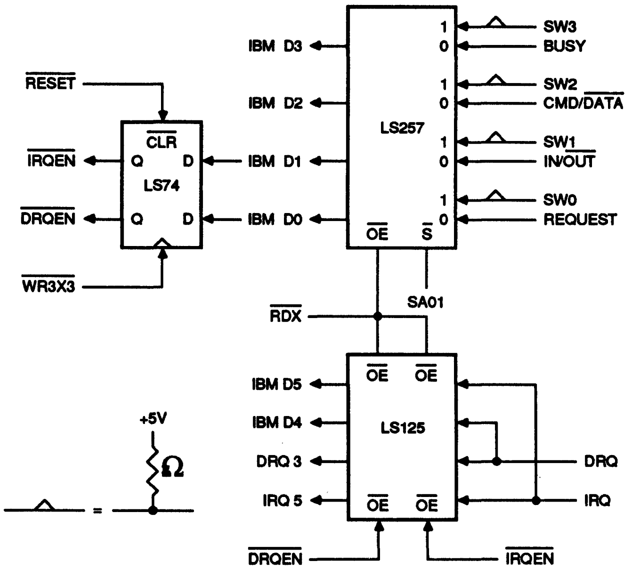


Figure 3-12

AmPAL22V10 Equation

```
/AEN     = DADD• SA01 • SA00  •/IOW   • /IOR

CAB      = /AEN • DADD • /SA01 • /SA00 • IOW

RESETIN = /AEN • DADD • /SA01 • SA00  • IOW

SELECT   = /AEN • DADD • SA01  • /SA00 • IOW

WR = X3 = /AEN • DADD • SA01  • SA00  • IOW


RDX      = /AEN • DADD • /SA01 • SA00  • IOR
         + /AEN • DADD • SA01  • /SA00• IOR

ACK      = /AEN • DADD • /SA01 • /SA00• IOW
         + /AEN • DADD • /SA01 • /SA00• IOW
```

Figure 3-13



9480A 3-14

Figure 3-14

The conceptual flow is as follows:

1. The 8051 accepts an IBM PC command for a sector transfer.
2. The 8051 spoon feeds the proper IOPB to the Am9580A.
3. The Am9580A internal state controller seeks the proper track and reads from it, if so desired.
4. The 8051 checks for errors.
5. If no errors, the 8051 signals the IBM PC to start data transfer.
6. The 8051 spoon feeds another IOPB to the Am9580A to start the 512-byte data transfer.
7. The 8051 generates the DREQ to IBM PC.
8. Transfer begins with the 8051 only monitoring for errors. The real-time transfer interface is handled by the AmPAL22V10.
9. After transfer is completed, the 8051 checks and resolves Am9580A error conditions, if any.
10. Reports to the IBM PC the complete code of the requested operation.

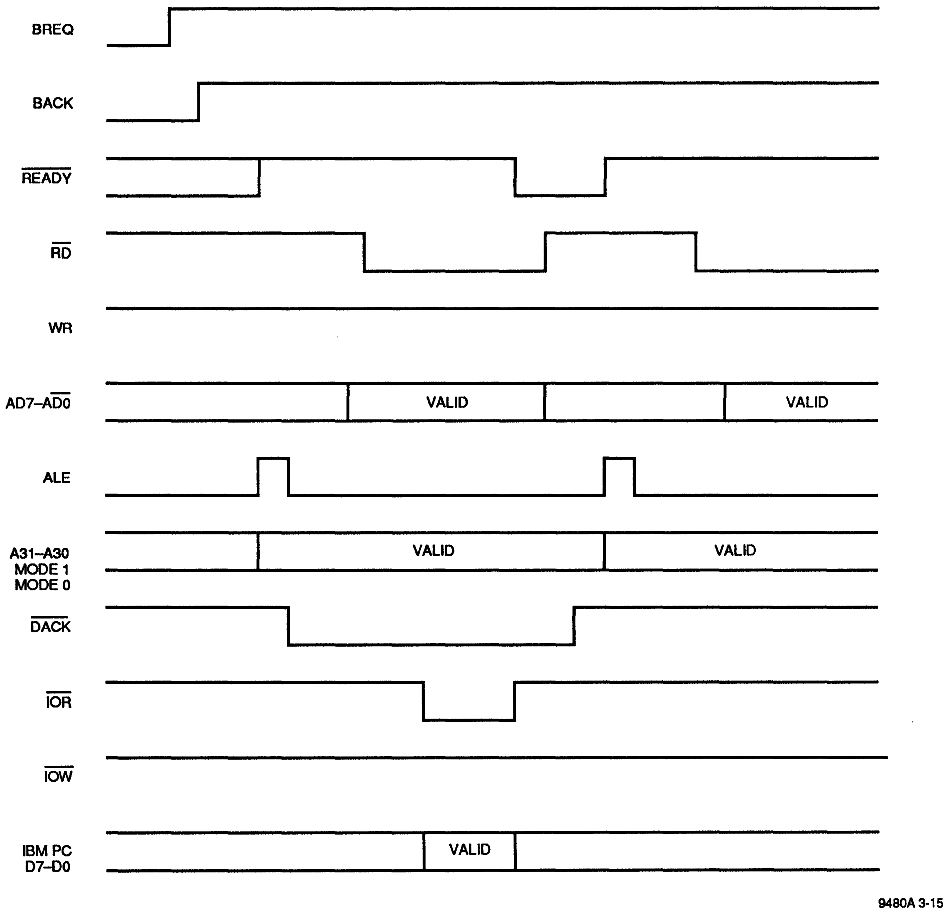
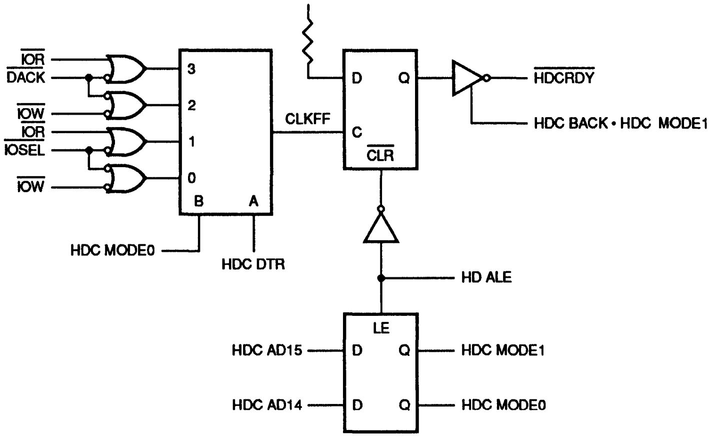


Figure 3-15 Interface Timing



9480A 3-16a

Figure 3-16a

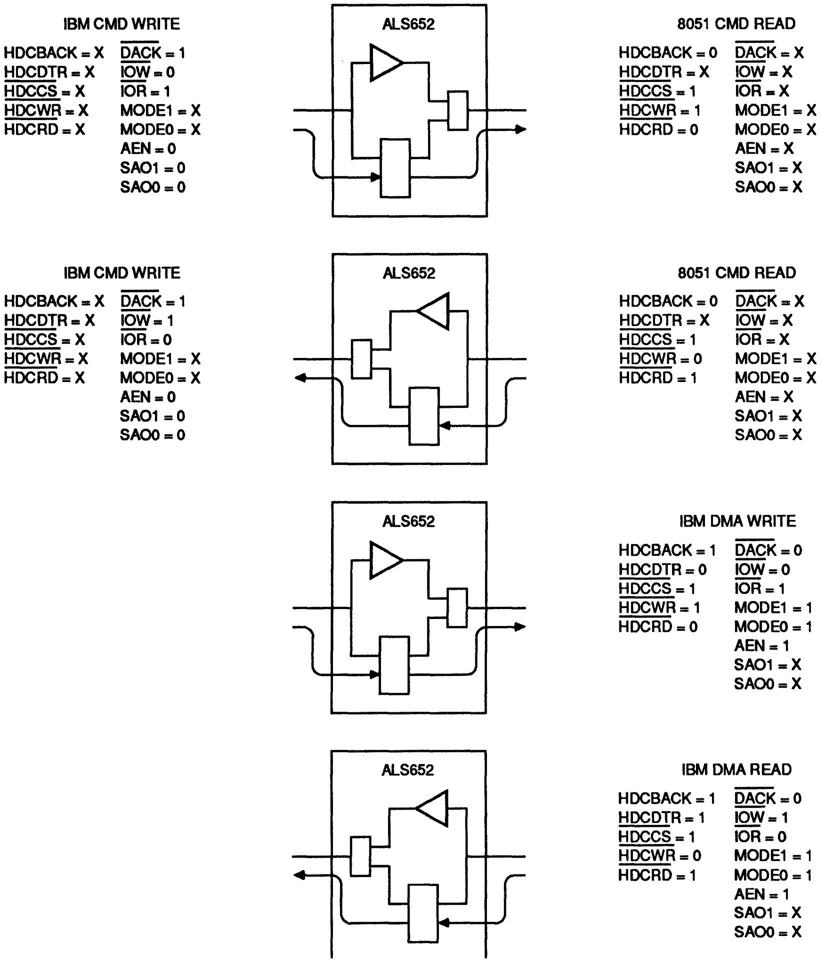
```
/AEN      DADD     SA01    SA00    /IOR    /IOW    /DACK
HDCBACK  HDCDTR  /HDCRD  /HDCWR  /HDCCS  HDCA15  HDCA14
HDCALE

CEH    = /HDCBACK · /HDCCS · HDCWR

SAB    = /HDCBACK · /HDCCS

SBA    = /DACK
       + /AEN · DADD · /SA01 · /SA00 · MODE1 · /MODE0

GBA    = HDCBACK · HDCDTR · DACK · IOR · MODE1 · MODE0
       + /AEN · DADD · /SA01 · /SA00 · IOR

GAB    = /HDCBACK · /HDCCS · HDCRD
       + HDCBACK · /HDCDTR · HDCRD · DACK · MODE1 · MODE0
       + HDCBACK · /HDCDTR · HDCRD · /AEN · DADD · /SA01 · /SA00
         MODE1 · /MODE0

MODE1 = HDCALE · HDCA15
       + /HDCALE · MODE1

MODE0 = HDCALE · HDCA14
       + /HDCALE · MODE0

CKFF   = /HDCMODE0 · /HDCDTR · IOW · /AEN · DADD · /SA01 · /SA00
       + /HDCMODE0 · HDCDTR · /IOR · /AEN · DADD · /SA01 · /SA00
       + HDCMODE0 · /HDCDTR · IOW · DACK
       + HDCMODE0 · HDCDTR · /IOR · DACK

HDCRDY := VCC
IF (HDCALE) THEN ARE SET (HDCRDY),
IF (HDCBACK · HDCMODE1) THEN ENABLE (HDCRDY):
```

Figure 3-16b

The AmPAL22V10 provides the real-time handshaking between the IBM PC and the Am9580A by monitoring both bus control signals and generating the READY signal to the Am9580A. The IBM PC bus transfers never have wait states inserted, thereby guaranteeing maximum utilization. The transfer control protocol and timing are as follows:

1. The 8051 gives the bus to Am9580A and generates DREQ.
2. The ALE output negates the $\overline{\text{READY}}$ signal.
3. The RD signal is activated holding data valid on the AD7–AD0 bus.
4. IBM PC $\overline{\text{DACK}}$ asserted.
5. IBM PC $\overline{\text{IOR}}$ asserted.
6. Data valid on IBM PC data bus.
7. Rising edge of IOR generates a $\overline{\text{READY}}$ signal to the Am9580A.
8. The Am9580A completes the bus cycle.
9. The procedure repeats from step 2.

The difference between DMA read and write cycles is that the falling edge of the $\overline{\text{IOW}}$ signal generates $\overline{\text{READY}}$ to the Am9580A in order to guarantee data hold time. The logic is quite simple (See Figure 3-16a) and is realized in the AmPAL22V10 (See Figure 3-16b). The remaining terms and outputs of the PAL are used as steering control of the 74ALS652 in order to connect the appropriate source and destination buses (See Figure 3-17).



9480A 3-17

Figure 3-17 Data Flow

ST506/412HP Interface

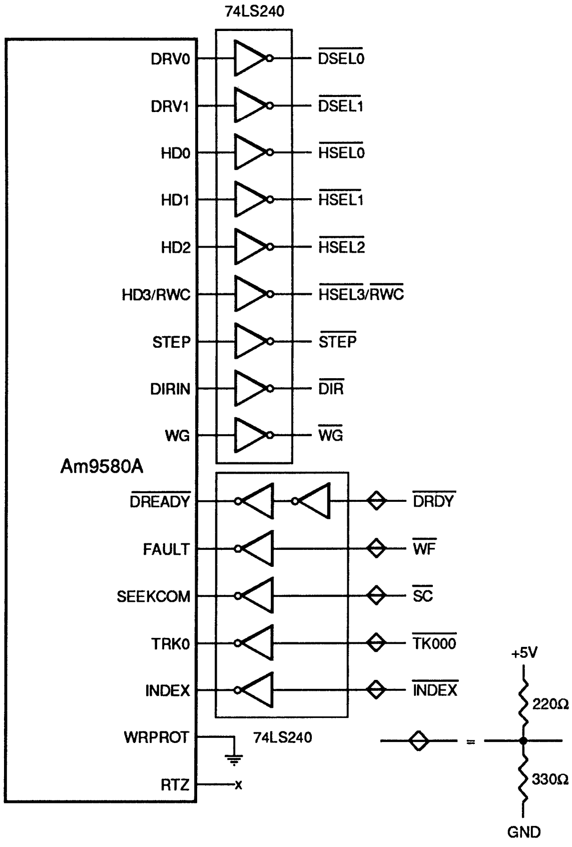
The rigid disk interface chosen for this application note is the ST506/412HP standard. This interface was selected for its popularity, familiarity and low system cost but any other back-end drive interface may be easily attached (i.e., ESDI, SMD) to the Am9580A.

Control

The ST506/412HP Control Interface is very straightforward. It consists of two 74LS240 inverting drivers, one 220/330 resistor termination package, and one 34-pin, 0.1" spacing, dual row header plug. The connections from the Am9580A to the header plug are virtually one-to-one through the inverting drivers. (See Figure 3-18.)

Data

The ST506/412HP Data Interface consists of an Am9582 DDS that provides the required precompensation and PLL/VCO recovery circuitry needed to read and write MFM data. The MFM write data is buffered by an Am26LS31 RS-422 differential driver to the two 20-pin data connectors, one for drive 0 and one for drive 1. The MFM read data is taken from the two 20-pin connectors, separately terminated with a 100 ohm resistor and transmitted to the Am9582 via an Am26LS32/33 RS-422 differential receiver. The Am26LS32/33's output enable controls are connected to the Am9580A's DRV0 and DRV1 select lines to operate in a MUX configuration. This connection eliminates another package for a MUX, but restricts the number of disk drives to two. Lastly, the 5 MHz nominal ST506 data transfer clock is selected as a divide-by-two of the Am9580A's 10 MHz system clock input. The 10 MHz nominal ST412HP data transfer clock is taken directly from the Am9580A's system clock input.


74LS240
Am9580A
DRV0
DRV1
HD0
HD1
HD2
HD3/RWC
STEP
DIRIN
WG
$\overline{\text{DSEL0}}$
$\overline{\text{DSEL1}}$
$\overline{\text{HSEL0}}$
$\overline{\text{HSEL1}}$
$\overline{\text{HSEL2}}$
$\overline{\text{HSEL3}}/\overline{\text{RWC}}$
$\overline{\text{STEP}}$
$\overline{\text{DIR}}$
$\overline{\text{WG}}$
$\overline{\text{DREADY}}$
FAULT
SEEKCOM
TRK0
INDEX
WRPROT
RTZ
$\overline{\text{DRDY}}$
$\overline{\text{WF}}$
$\overline{\text{SC}}$
$\overline{\text{TK000}}$
$\overline{\text{INDEX}}$
74LS240
+5V
220Ω
=
330Ω
GND

9480A 3-18

Figure 3-18

Summary

The concepts of connecting a bus master peripherals device to the slave-only IBM PC bus have been explored and detailed. The feasibility and simplicity of the design minimize both IC chip count and data transfer latency. The design described has not been built, but would easily fit on a 5" long IBM PC-XT height I/O card. The 8051 operation code is beyond the scope of this application note. But the hardware design does not preclude the implementation of a full Xebec command set compatible zero-interleave rigid-disk controller using the Am9580A and Am9582 Hard Disk Controller chip set.

3.3. Microprocessor Interfaces

3.3.1 8086–Am9580A/90 Interface

The Am9580A/90 is designed to be iAPX bus compatible. Consequently, interfacing the Am9580A/90 to the 8086 is straightforward. Figure 3-19 shows the basic interface; the 8086 is operated in Minimum Mode (MN/$\overline{\text{MX}}$ High). The following list summarizes the specific considerations:

- The lower address latches and the data bus transceiver may be shared between the 8086 and the Am9580A/90. However, the upper address latches must be separate.

- ALE is not three-stated by the 8086 or Am9580A/90. Therefore, the ALE is ORed to control the latch enable of the shared lower address latch.

- The slave interface of the Am9580A/90 requires demultiplexed addresses and control signals. $\overline{\text{CS}}$ is decoded from the latched address. A_{0-3} is connected to the latched address bus.

- The interrupt request is connected to an interrupt controller because the 8086 does not support non-vectored interrupts directly. Interrupt request must not be connected to the NMI input of the 8086 because the Am9580A/90 generates an interrupt which cannot be disabled immediately after reset

- $\overline{\text{READY}}$ is bidirectional. In slave mode, it must be connected to the ready input of the CPU to be able to stretch the slave access cycles.

- The 8086 and the Am9580A/90 are driven by the same clock. It is suggested that the supplied clock should be verified to satisfy the frequency specification and the High and Low width. With the 8086 this may not be the case because the 8086 demands a clock with approximately a 33% duty cycle.

3.3.2 8088–Am9580A/90 Interface

Figure 3-20 shows the 8088 to Am9580A/90 interface operating the 8088 in Minimum Mode. This interface is almost identical to the 8086 to Am9580A/90 interface outlined in Section 3.2. The only differences between the 8086 and 8088 are:

- The 8088 has an 8-bit data bus, therefore, the data bus transceiver is now 8 bits, and the Am9580A/90 is strapped to byte mode (B/$\overline{\text{W}}$ High).

- The memory/IO line has the reversed polarity. A NAND generates a memory request while the Am9580A/90 is bus master.

3.3.3 80186–Am9580A/90 Interface

Figure 3-21 shows the 80186 to Am9580A/90 interface. This interface is similar to the 8086 to Am9580A/90 interface outlined in Section 3.2. Since the 80186 incorporates control logic, such as a clock generator, chip select decoders, and an interrupt controller, it is possible to lower the interface device count. The differences are as follows:

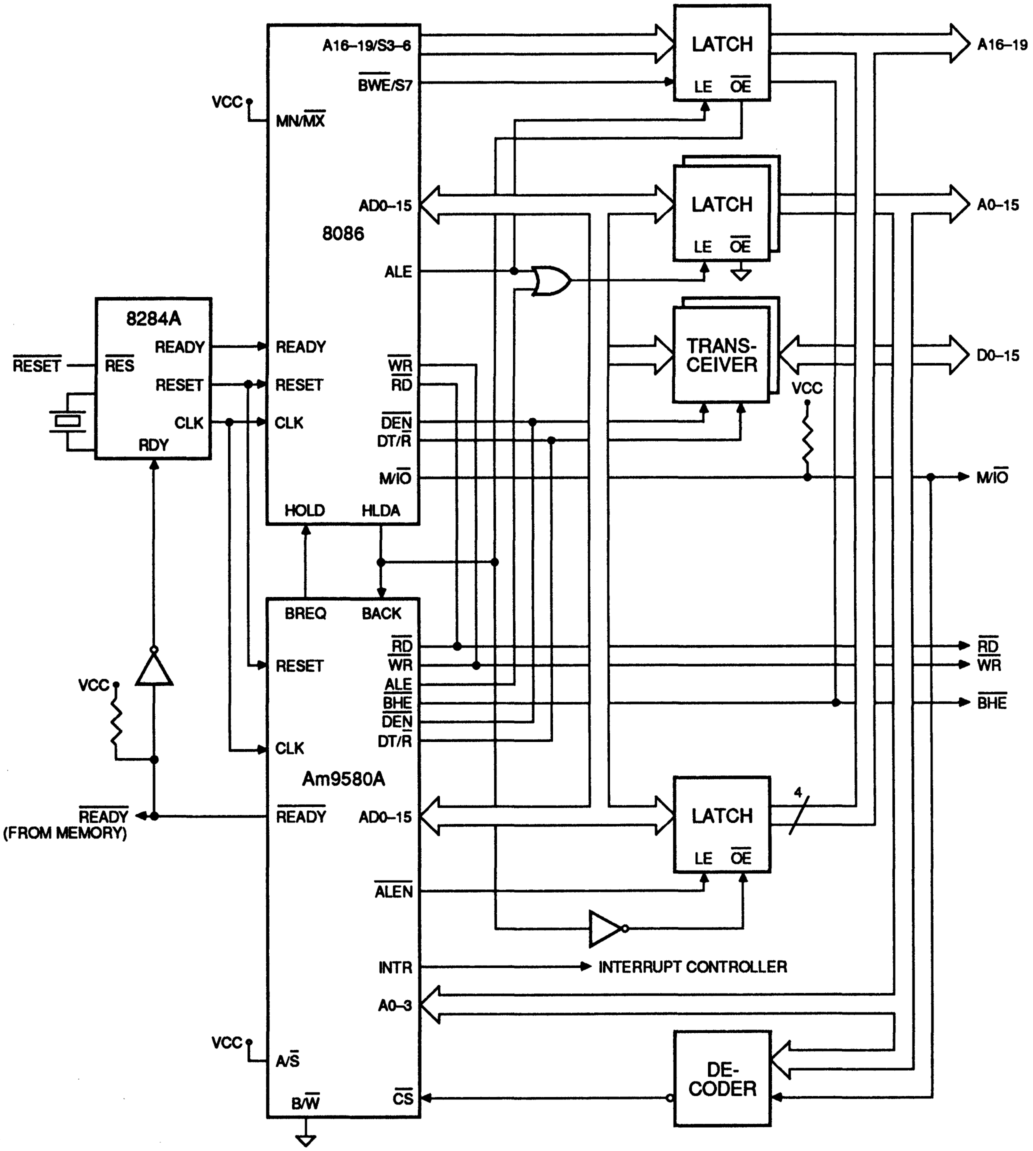
- The 80186 has on-chip memory address decoders. However, this internal logic must be complemented by external logic to cover the case when the Am9580A/90 is bus master.

- An on-chip peripheral select decoder ($\overline{\text{PCS}}$ output) provides the chip select ($\overline{\text{CS}}$) for the Am9580A/90.

- The interrupt request of the Am9580A/90 may be connected directly to one of the interrupt inputs of the 80186; the 80186 has an on-chip interrupt controller.

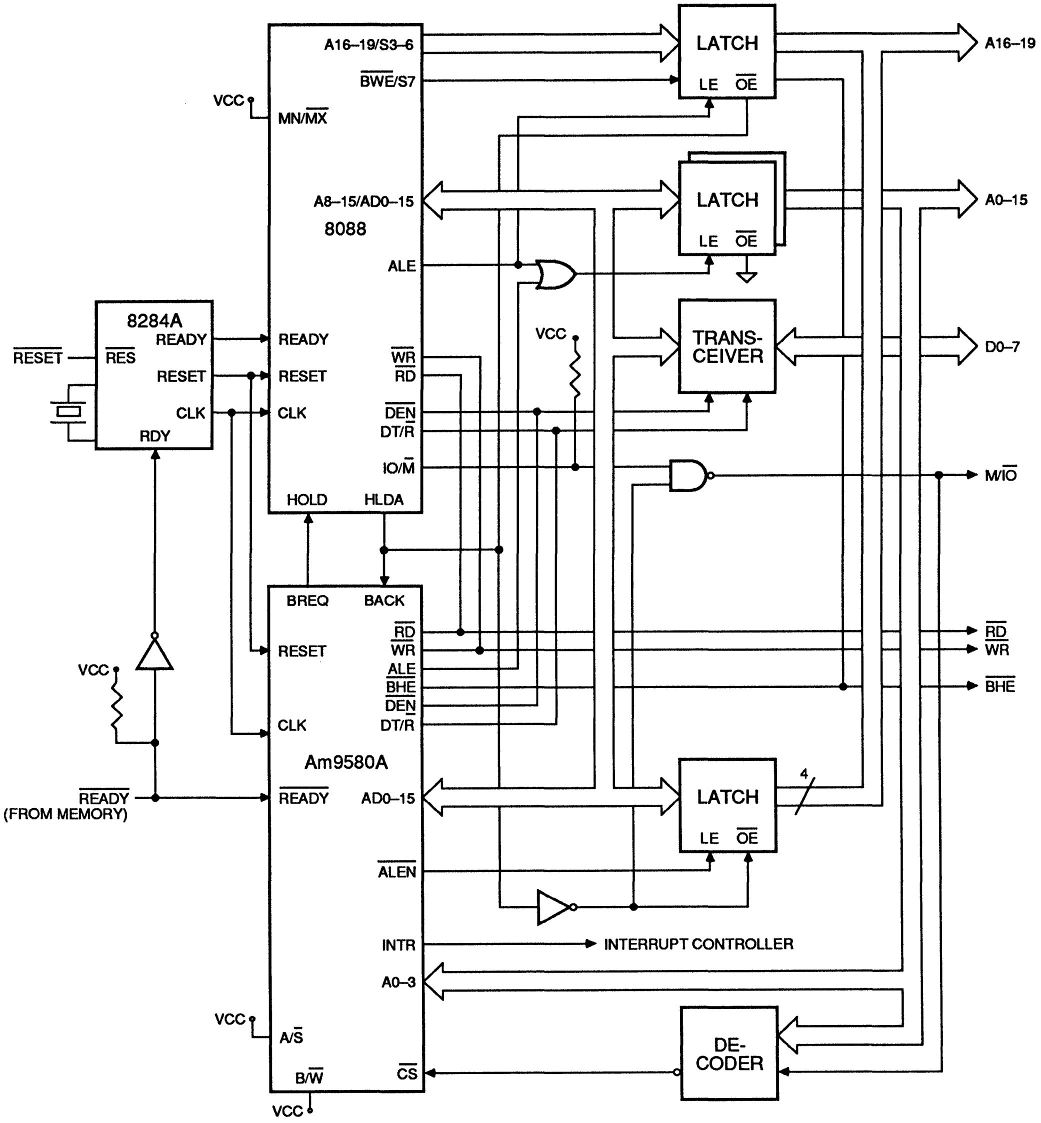
- The on-chip crystal oscillator provides a TTL compatible clock for the Am9580A/90. Since the provided clock (CLKOUT) has approximately a 50% duty cycle, the timing is relaxed when compared to the clock provided by the 8284.

- Since the 80186 has a reset-out output, it is used to reset the Am9580A/90.



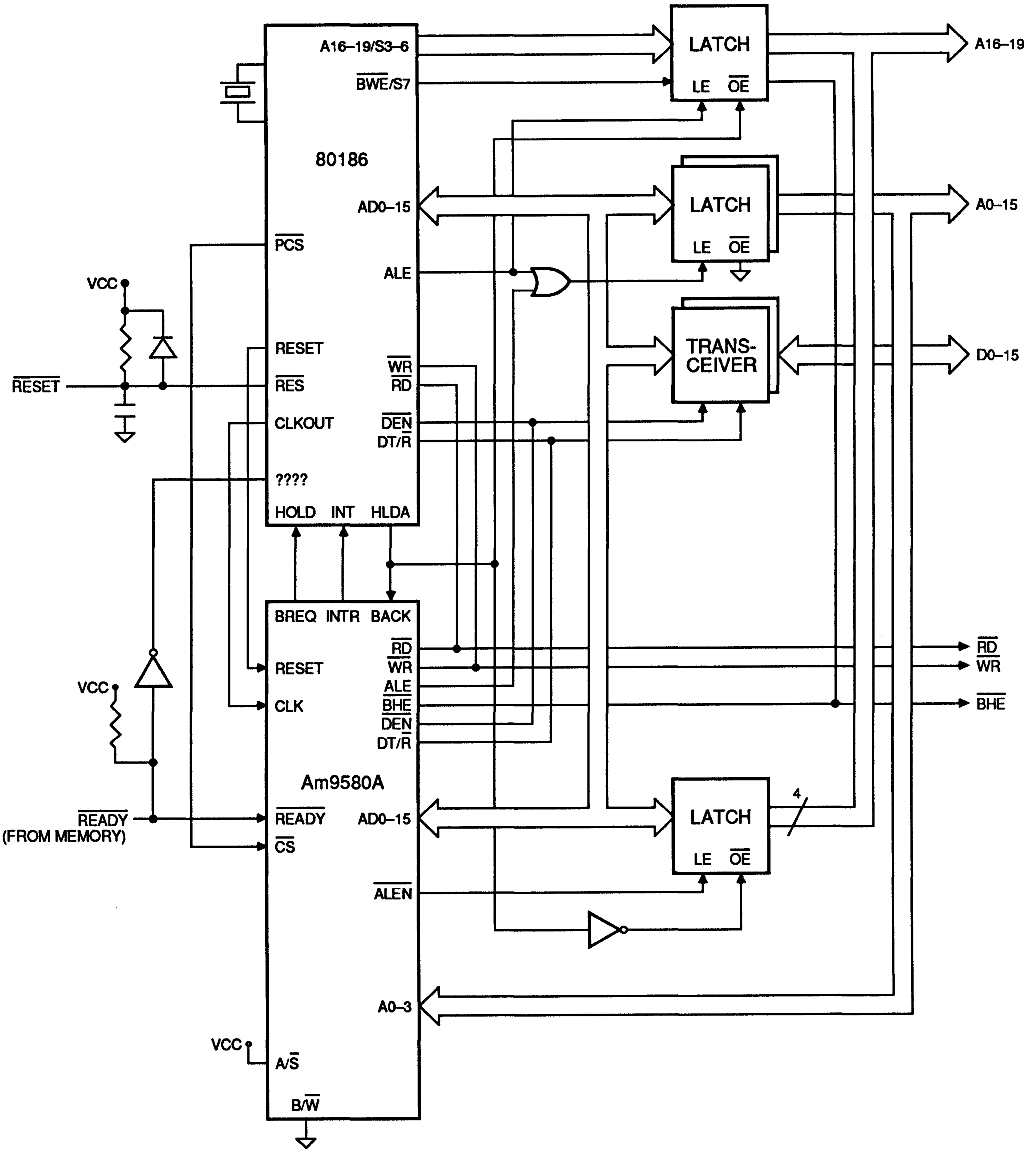
9480A 3-19

Figure 3-19 8086-Am9580A/90 Interface



9480A 3-20

Figure 3-20 8088-Am9580A/90 Interface



9480A 3-21

Figure 3-21 8086–Am9580A/90 Interface

3.3.4 80286–Am9580A/90 Interface

Figure 3-22 shows the 80286 to Am9580A/90 interface. The major difference to the previous applications is the demultiplexed address and data bus of the 80286. Three latches extend the 80286 address timing to meet the 8086 bus specifications, the other three latches are used to demultiplex the AD-bus of the Am9580A/90. Other specific considerations are listed in the following:

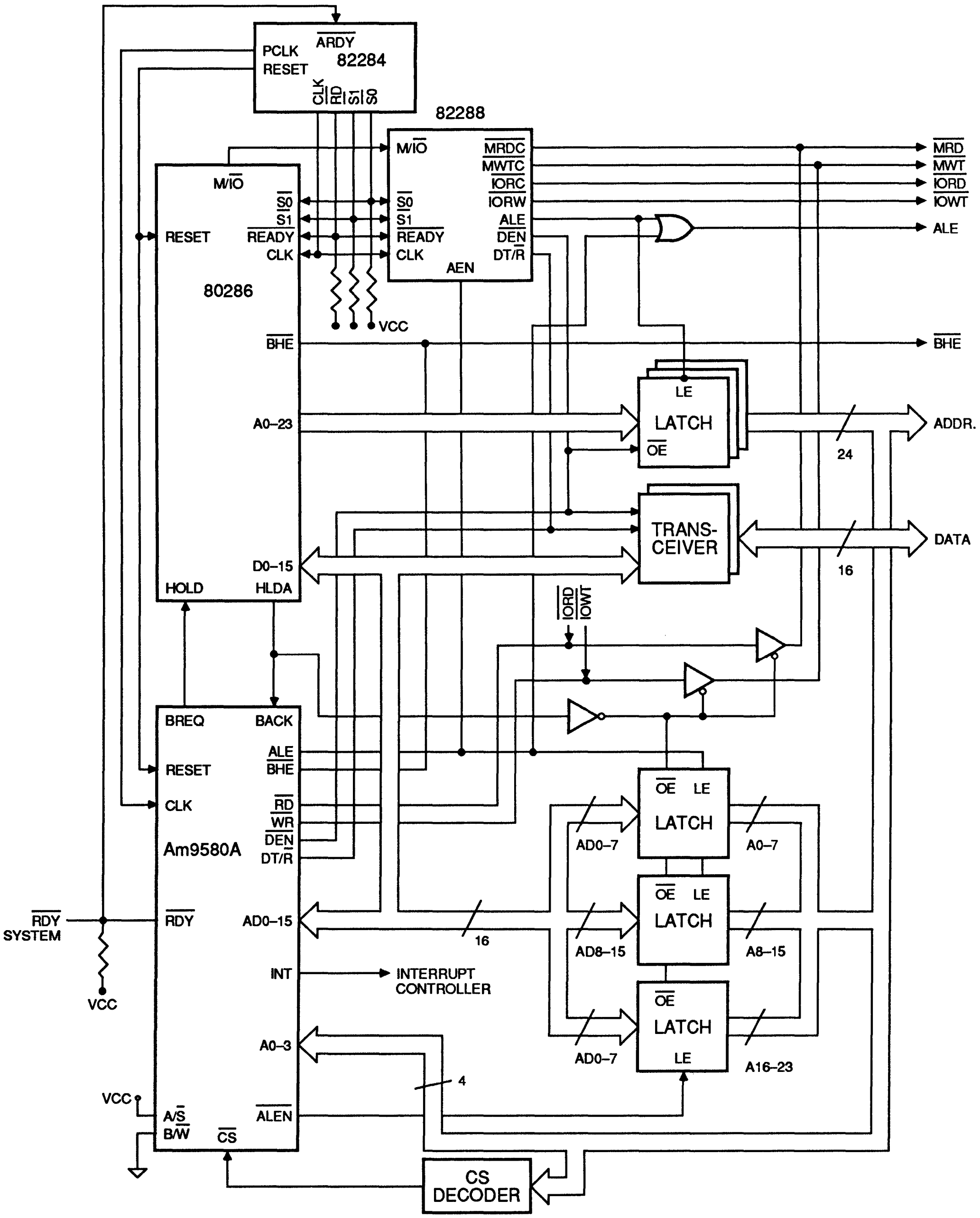
- The 82284 clock generator provides two different clocks. One of them is used to drive the 80286. The other clock (half the frequency (PCLK)) is connected to the Am9580A/90.
- The $\overline{\text{RDY}}$ line of the Am9580A/90 is a bidirectional signal which must be connected directly to the system $\overline{\text{READY}}$.
- In master mode the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals of the Am9580A/90 drive the system bus signals $\overline{\text{MRD}}$ and $\overline{\text{MWR}}$. The 80286 is disabled during this time. In slave mode, the two bus drivers for these signals are disabled; the HDC signals, $\overline{\text{RD}}$ and $\overline{\text{WR}}$, which are now inputs for the Am9580A/90, are driven by the system bus signals $\overline{\text{IORD}}$ and $\overline{\text{IOWT}}$.
- ALE is not three-stated by the Am9580A/90 nor by the 80286; therefore, both are ORed to generate the system ALE.

3.3.5 68000–Am9580A Interface

Figure 3-23 shows the 68000 to Am9580A interface. Some external control logic is necessary to adapt both devices. The 68000 has a demultiplexed data and address bus ,whereas, the Am9580A needs a multiplexed bus interface. Three latches demultiplex the Am9580A data bus. Two Am29841s store the lower 16 bits of the address and are controlled by the ALE (Address Latch Enable) signal, another Am29841 latches the upper 8 bits of the address and is controlled by the ALEN (Address Latch Enable N) signal.

In slave mode, a $\overline{\text{CS}}$ (Chip Select) signal is generated by an address decoder. The address inputs, A1 to A3 of the Am9580A, are directly connected to the 68000 addresses A_1 to A_3 and allow to access the internal registers of the disk controller. The input A0 is grounded because the slave mode uses only word accesses.

A single PAL device (AmPAL22V10) generates all the necessary control signals in slave and master modes. In slave mode—indicated by $\overline{\text{BGACK}}$ (Bus Grant Acknowledge) Inactive—the 68000 signals $\text{R}/\overline{\text{W}}$ (Read/Write), $\overline{\text{LDS}}$ (Lower Data Strobe) and $\overline{\text{UDS}}$ (Upper Data Strobe) are translated into $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals for the Am9580A. The line $\overline{\text{BHE}}$ is kept LOW in slave mode because slave accesses are always word transfers.



9480A 3-22

Figure 3-22 8086–Am9580A/90 Interface

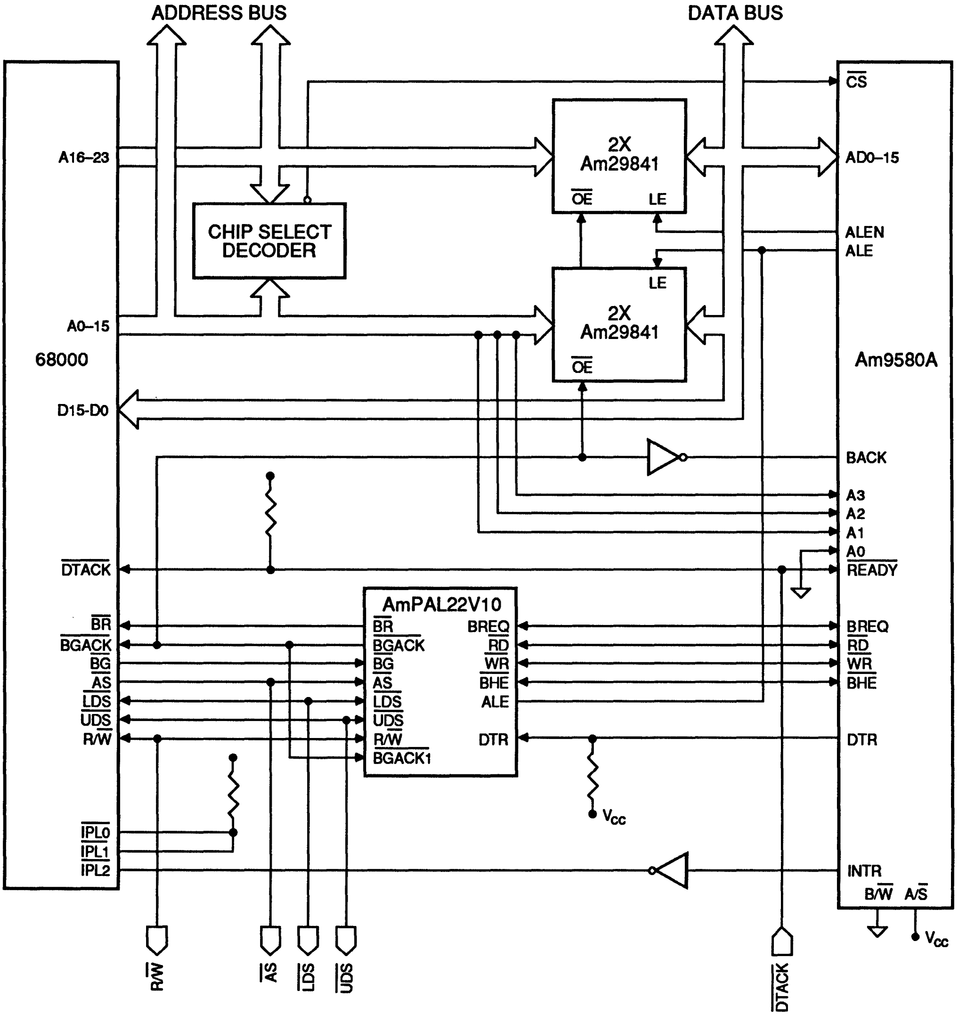


Figure 3-23 68000–Am9580A/90 Interface

The two-wire bus exchange protocol of the Am9580A (BREQ, BACK) must be translated into the three-wire protocol (BR, BG, BGACK) of the 68000 processor. The timing diagram in Figure 3-24 shows the sequence of the bus exchange. The Am9580A requests the processor bus by activating its BREQ (Bus Request) line. The control logic immediately asserts a BR (Bus Request) signal to the 68000. It then waits for the CPU to answer with the signal BG (Bus Grant) that indicates that the processor will finish the current operation with the next rising edge of the AS (Address Strobe) line. This event causes the control logic to generate a BGACK (Bus Grant Acknowledge) signal to notify the 68000 that the HDC is the new bus master. The same inverted signal also indicates to the Am9580A that the bus has been released (BACK, Bus Acknowledge). The BGACK line is fed back into the PAL device to provide the BR 20 ns of hold time required by the 68000. The Am9580A then becomes the new bus master.

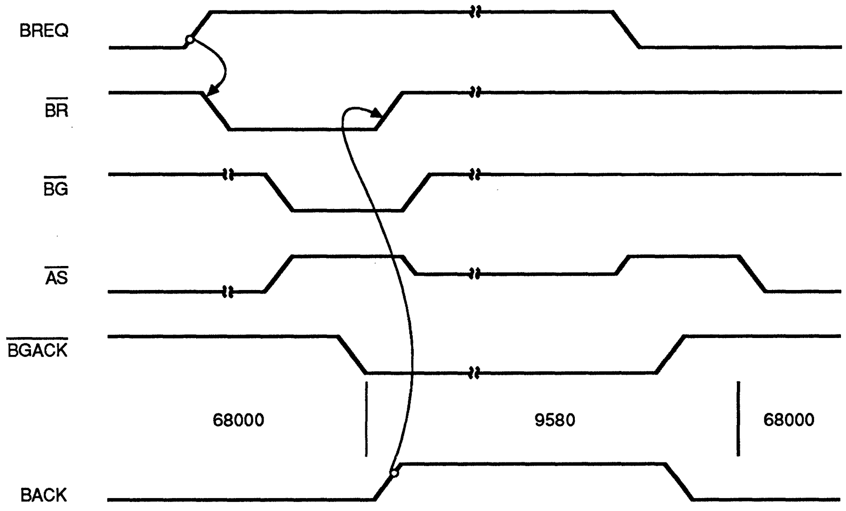
In master mode, the timing of the disk controller signal (ALE) must be adapted to the AS timing that a 68000 system requires. If the system memory disregards the AS signal, or uses it to latch the address internally, AS can be connected to ALE in master mode. If, on the other hand, the memory uses the AS line for timing purposes, such as "start memory cycle", the timing of the ALE signal must be modified. This modification prevents the AS line to go LOW before the first ALE pulse and during an upper address update (ALEN). The following timing diagram shows the relationship.

After finishing the DMA transfer, the Am9580A releases the bus by deasserting the BREQ line. This causes the PAL device to drop the BGACK line and the 68000 regains control of the system bus.

3.3.6 NS32016 Interface

The following application note shows the interface between the Am9580A/90 and the NS32016 CPU (Figure 3-25). The NS32016 processor has a multiplexed address-data bus like the HDC. Therefore, the Am9580A or Am9590 can be directly connected to the CPU bus. In order to generate the upper eight bit of the address, the HDC requires a latch, used exclusively by the Am9580A/90. This latch is controlled by ALEN and will be enabled if the HDC is in control of the system bus (BACK active).

One PAL device AmPAL16L8 generates all the necessary timing signals in slave and master modes. (See Figure 3-26.)



9480A 3-24

Figure 3-24 Bus Request Timing

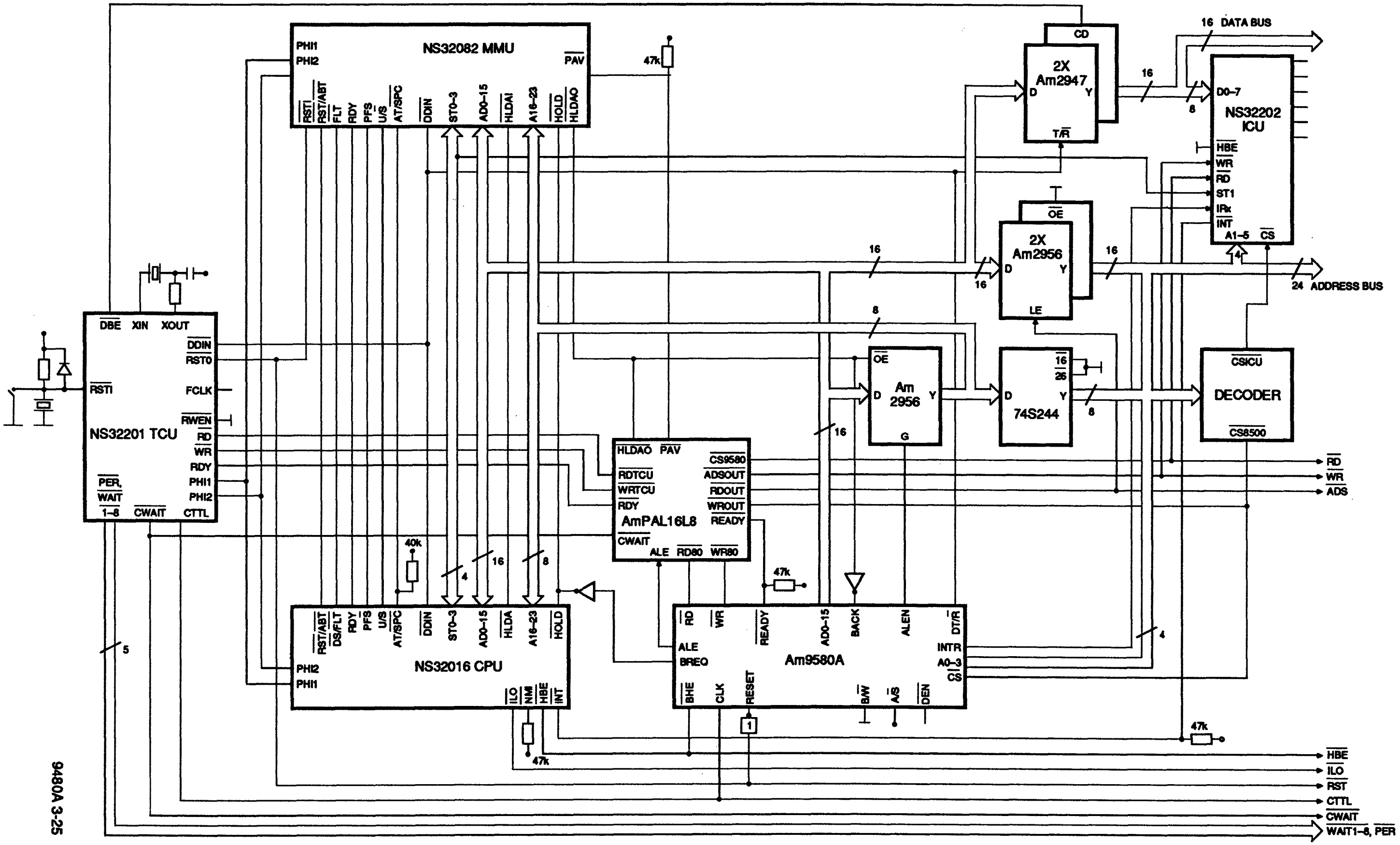


Figure 3-25 NS32016–Am9580A/90 Interface

Slave Mode

In slave mode, the CPU addresses the HDC registers using the address inputs A_{0-3}. Since the HDC interface is selected as a word interface, these accesses will be 16 bits wide. Register accesses to the HDC can have different length, depending on when the HDC presents the selected register onto the bus. In order to adapt the speed, the $\overline{\text{READY}}$ signal (output of the HDC in slave mode) is connected to the $\overline{\text{CWAIT}}$ line of the NS32201 TCU.

Master Mode

In master mode, all timings are generated by the Am9580A/90. Both the NS32016 and the HDC have a four-clock bus cycle. The timings of the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ cycles are only slightly different between the two processors. If the memory shared between the devices is running asynchronously, the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals can be directly interconnected.

```
PAL-Equations:

/*                                  */
/*      PAL AM9580 - NS32016        */
/*                                  */

£device 16L8;

£pin /PAV = 1 , /CS9580 = 2,                 /* INPUTS */
     RDY = 3 , /RDTCU = 4,
     /WRTCU = 5 , /HLDA0 = 6,
     ALE = 7,
     /ADSOUT = 12 , /RDOUT = 13,             /* OUTPUTS */
     /WROUT = 14 , /CWAIT = 19,
     /READY = 15 , /WR80 = 16,               /* BIDIRECTIONAL */
     /RD80 = 17,
     GND = 10 , VCC = 20;                    /* POWERSUPPLY */

equations ()
a

   enable(/ADSOUT);
   ADSOUT = PAV + ALE;

   enable(/RDOUT);
   RDOUT = RTCU +
           RD80 * HLDA0

   enable(/WROUT);
   WROUT = WRTCU +
           WR80 * HLDA0

   enable(/CWAIT);
   CWAIT = CS9580 * /READY * RD80 +
           CS9580 * /READY * WR80;

   if (HLAD0) enable(/READY);
   READY = /RDY;

   if (HLAD0) enable(/WR80);
   READY = /WRTCU;

   if (HLAD0) enable(/RD80);
   RD80 = /RDTCU;
```

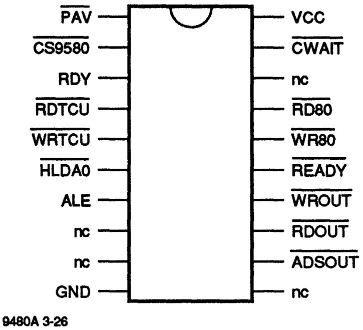


Figure 3-26

The READY signal is an input for the HDC in master mode. The PAL device switches the RDY output of the NS32201 TCU onto this signal so that wait states will be inserted by the Am9580A/90 whenever necessary.

The BREQ signal is being inverted and connected to the HOLD input of the CPU (open collector). The HOLDAQ output of the NS32082 is used to generate the BACK signal for the HDC. This application assumes no further DMA devices.

3.4 Interfaces

3.4.1 The ST506/412 Am9580A/90 Interface

The Am9580A and the Am9590 connect to the various types of disk drives in a very straightforward way. Figure 3-27 shows a typical interface for four ST506 disk drives. The interface consists of two connecters: The control cable carries all the control signals from and to up to four disk drives; the radial cable is exclusive for each drive and carries the data signals driven differentially.

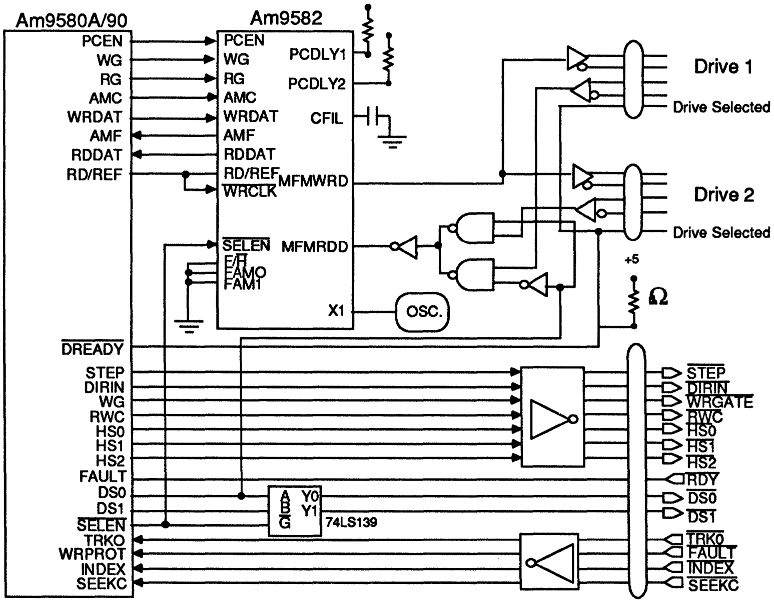


Figure 3-27 Typical ST506 Interface

All the control lines to the disk are directly generated by the HDC. Only 48 mA drivers are required to drive the lines. The signal lines coming from the disk drive need to be inverted by hysteresis receivers like the 74LS14 or equivalent.

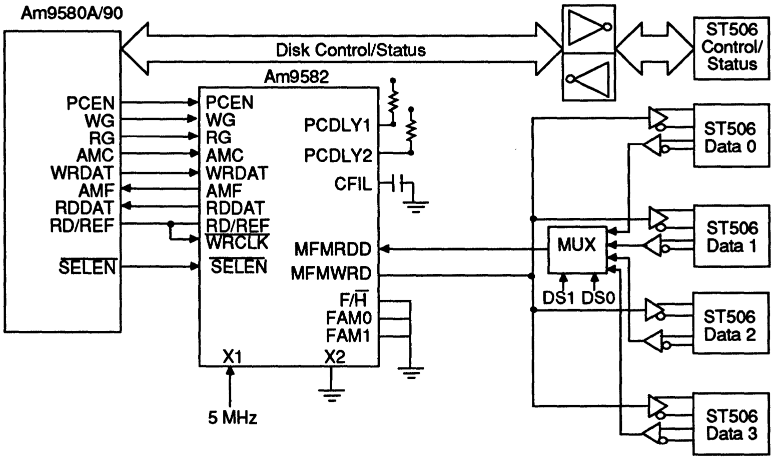
The HDC generates two Drive-Select ($DRSEL_{0,1}$) lines and a Select-Enable ($\overline{SELEN}$) signal. Those can be used to generate the individual select lines for the four possible drives. Note that the Drive-Select lines are static signals and must be qualified with $\overline{SELEN}$. In Figure 3-27 a 74LS139 generates the select signals for the drives.

For an ST506 only application (Figure 3-28), only the Read-Data signals coming from the disk drive must be multiplexed before connecting to the Am9582. This can be done either by using two separate Am26LS32 line receivers or by using two three-state gates as shown.

3.4.2 The ST506/412 and Floppy Disk Interface

If an additional floppy interface is desired, Figure 3-29 shows a possible configuration. Even though different data rates are required for hard disk and floppy mode, a single Am9582 can accommodate both frequencies. The data separator locks onto a new frequency within less than 1 ms. This allows the switching between the two different drive types without adding any delay. This allows the using of a simple multiplexer to select the appropriate clock rate depending on the drive selected.

The same multiplexer (74LS153) can be used to select the Read Data input from the drive. If the floppy drive does not have a $\overline{DREADY}$ output, this signal must be activated whenever a floppy drive is selected. A 74LS38 open collector gate do this task as shown assuming that the floppy drives are located



9480A 3-28

Figure 3-28 Controlling 4 Hard Disks

at drive addresses 2 and 3. The same design can activate SC in floppy mode, if there is now appropriate signal coming from the drive.

3.4.3. The ESDI Interface

Figure 3-30 shows a typical ESDI interface using the Am9590 HDC. All the signals required on the control cable are logically and directly connected to the controller; only buffering is necessary. No special attention needs to be paid to the serial command/ status interface since it is handled by the Am9590.

There are some signals coming from the disk drive that need to be multiplexed. Those signals are: Read Data, Read Reference Clock and Command Complete. Two dual 4-to-1 multiplexers (74LS153) can switch these signals using the Drive Select lines as select signals.

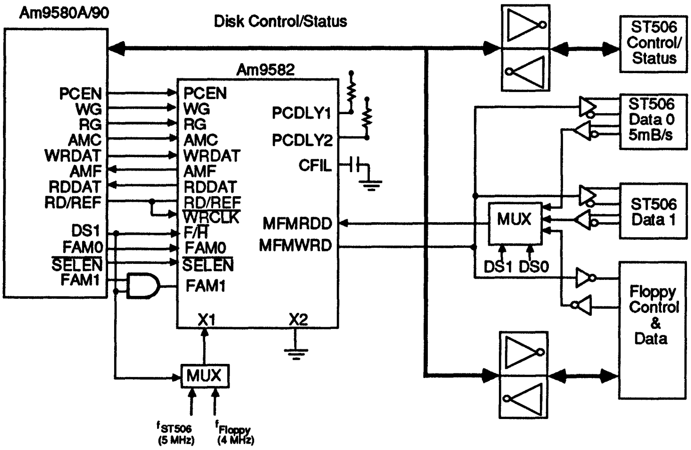
Set-up and hold time requirements on the ESDI interface force the RD/REFCLK to be inverted.

3.4.4 The ESDI and ST506 Interface

The basic effort in combining the ST506 with the ESDI interface is multiplexing different signals. As shown in Figure 3-31 the control connectors in ST506 and ESDI are layed out so that the signals which are common to both interfaces have the same location on the cable. The remaining signals have different meaning depending on the interface. The Am9590 switches these signals the same way, the interface defines them, e.g., the STEP output in ST506 becomes the TRANSFER REQUEST output in ESDI .

The only exception is the $\overline{\text{RG}}$ signal in ESDI. If the ST506 interface is selected, this line carries the Drive Select #4. The easiest way to solve this problem is using the jumper option as shown in Figure 3-31.

The radial cable carries signals that must be multiplexed, as described under the ESDI interface description above. In addition there is some circuitry required to select either the Am9582 as a signal



9480A 3-29

Figure 3-29 Controlling 2 Hard Disks and 2 Floppy Disks

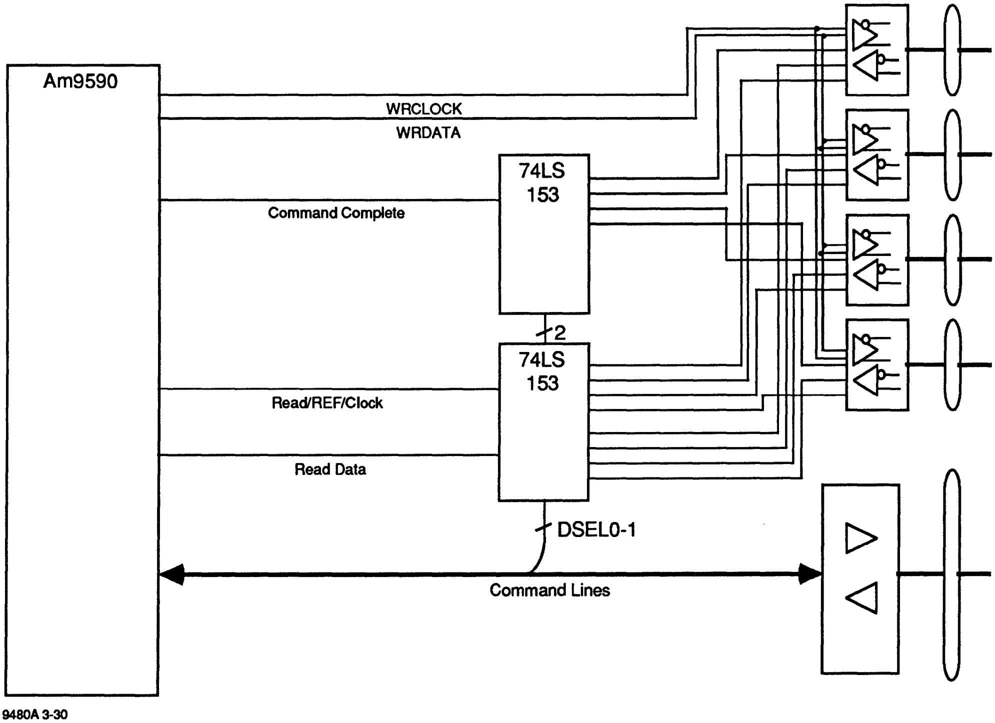
source for the HDC or the ESDI drive. Signals that need to be switched are: NRZ (MFM) Read Data, NRZ Write Data, AMF (Sector), Reference Clock and Command Complete. The fact that the Am9582 three-states all its output signals, when not selected, helps to implement this logic. The multiplexers used (74LS253) can also be three-stated.

A PAL device (AmPAL16L8) uses the inputs DS0, DS1 to decide what drive is selected. Four selectable jumper blocks allow to configure each drive for ST506 or ESDI.

For an ST506 drive, the $\overline{\text{ESDIEN}}$ output will stay inactive, thus enabling the Am9582. The multiplexer outputs REFCLK, COMCOMP and NRZRDDAT will be three-stated as well as the PAL outputs AMF and NRZWRDAT. Therefore, the Am9582 drives all these signals as required in a standard ST506 application.

If an ESDI drive is selected, the Am9582 three-states its outputs since the ESDIEN signal is driven active by the PAL device. All the ESDI signals are active now and directly connect to the Am9590. Please note that the RDDAT lines coming from the disk drives are inverted before the multiplexer. This has been done to use the three-state feature of the 74LS253. The easiest way to achieve the inversion is to swap the inputs of the Am26LS32 differential line receivers.

The application shown in Figure 3-31 also allows the CPU to request the drive type (ST506 or ESDI). This allows the driver software to automatically adapt to the different drives. A Request Drive Type signal—usually a decoded I/O address—enables the Drive Type output. Depending on the status of DS0 and DS1, the Drive Type line will carry the status of one of the jumper inputs. This is possible because the Drive Select lines of the HDC are static signals independent of a disk access. In order to retrieve the correct status, the software has to do a dummy access to the desired drive (e.g., a RESTORE command) and a subsequent read from the Drive Type port.



9480A 3-30

Figure 3-30 ESDI Interface

3.4.5 The ESDI, ST506 and Floppy Disk Interface

This interface (Figure 3-32) works essentially similar to the ESDI + ST506 application; however, there are two modifications required. First, disk drive #3 is designated as a floppy-disk drive and, secondly, the input frequency for the Am9582 must be switched between floppy (4 MHz) and hard disk (5 MHz) speed. Replacing the PAL device AmPAL16L8 with an AmPAL16R4 allows the two clock rates to be generated by dividing an input clock of 20 MHz down. This is possible because the Am9582 does not require a 50% duty cycle clock input as long as the minimum high—and low—time of the clock is not violated

3.4.6. A Storage Module Device (SMD) Interface

This section describes the Am9590 drive interface adapted to a Storage Module Device type of drive (SMD). The Storage Module Device (SMD) interface is one of the oldest standardized Winchester disk drive interfaces available. Because of its larger size (8" and 14"), the Winchester drive has two significant advantages over other types of drives. They are:

1. Larger storage capacity, e.g., in terms of number of bytes. Some 14" drives now boast greater than one Gigabyte of storage capacity.

2. Fast track-to-track access and short average access time.SMDs normally have1.5 milliseconds track-to-track access time, with average seek times of 20 milliseconds or less.

Both factors have led to a migration of SMD drives from large mainframe computers to minicomputers, and, in some cases, high-end microcomputers. When manipulating large data bases such as business transaction files, the fast drive characteristics of the SMD disk drives allow faster system response. This has led disk drive manufacturers to increase SMD performance from 9.6 MHz all the way up to the new HSMD and ESMD interfaces, 19.2 MHz and 24 MHz respectively.

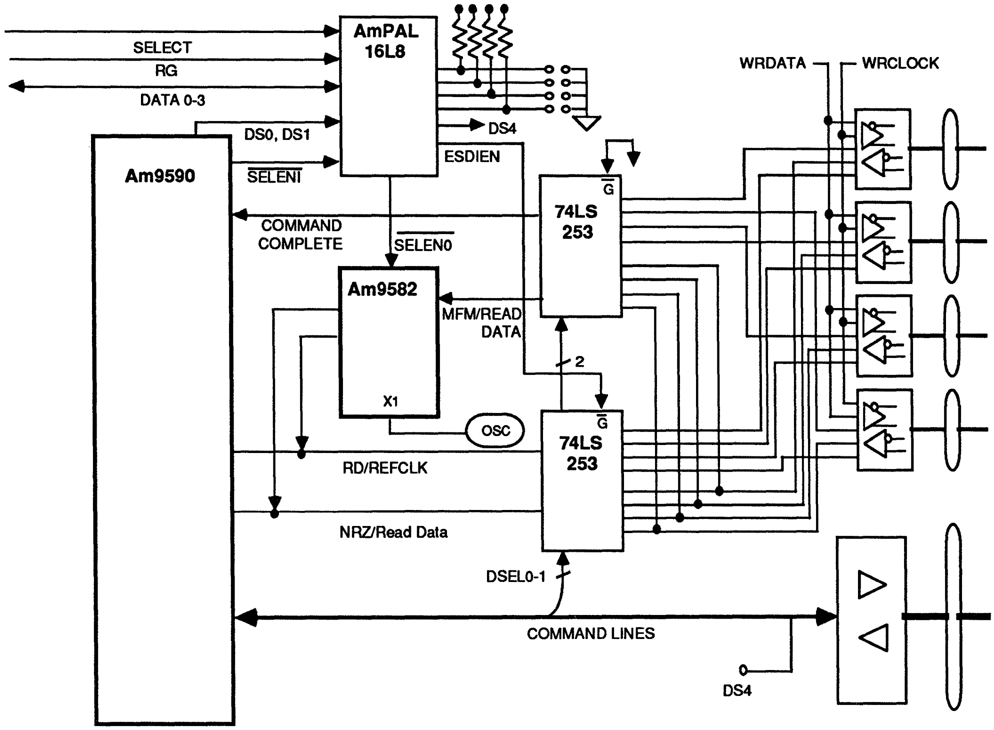
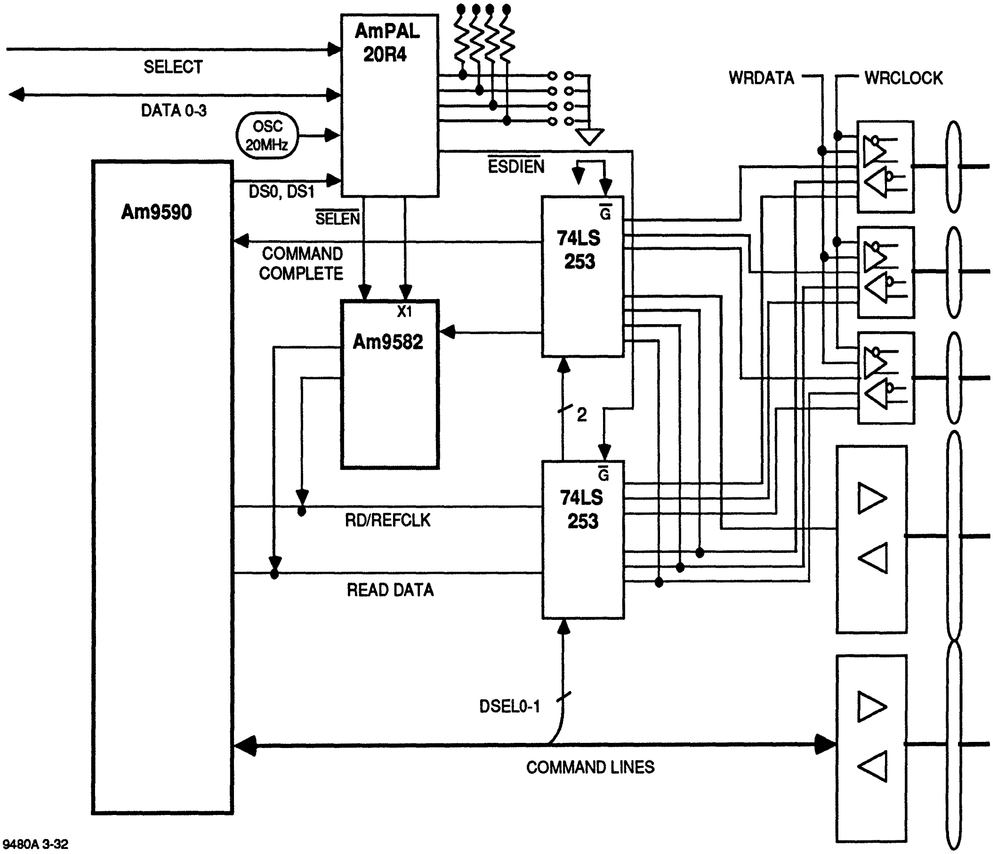


Figure 3-31 ESDI-ST506 Interface

```
PAL Equation AmPAL16L8
       RG     = (input combinatorial)
       S[3:0] = (input combinatorial)
       SELECT = (input combinatorial)
       DSEL[1:0]     = (input combinatorial)
       /SELENI= (input combinatorial)
       /SELENO= (output combinatorial)
       DATA[3:0]     = (output combinatorial, three-state)
       ESDIEN = (output combinatorial)
       DS4    = (output combinatorial)
IF (S[0]•/DS[0]•/DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[1]•DS[0]•/DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[2]•/DS[0]•DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[3]•DS[0]•DS[1]•SELENI) THEN SELENO = TRUE;
IF (/S[0]•/DS[0]•/DS[1]) THEN ESDIEN = TRUE;
IF (/S[1]•DS[0]•/DS[1]) THEN ESDIEN = TRUE;
IF (/S[2]•/DS[0]•DS[1]) THEN ESDIEN = TRUE;
IF (/S[3]•DS[0]•DS[1]) THEN ESDIEN = TRUE;
IF (SELECT) THEN ENABLE  DATA[3:0]
DATA[3:0] = S[3:0];
IF (ESDIEN) THEN DS4 = DS[0]•DS[1];
IF (SELENO) THEN DS4 = RG;
```

Figure 3-31a. ESDI-ST506 Interface PAL Equation



9480A 3-32

Figure 3-32 ESDI-ST506 Floppy Interface

AMD started the development of the Am9580A Hard Disk Controller (HDC) a few years ago. The original Am9580A was meant to address the low-performance, high-volume ST506 and floppy drive markets (both were soft-sectored). Several early design choices, however, easily made the Am9580A a natural to handle the SMD and newer ESDI markets.

Those design choices were:

1. The Am9580A neither generates nor detects Address Mark (AM) but used Address Mark Control (AMC) and Address Mark Found (AMF) handshakes to synchronize its internal data state machine.

2. Read and Write data are not encoded but Non-Return to Zero (NRZ) form is used.

Those two design choices allowed the Am9580A to address new SMD and ESDI markets, both of which require the foregoing capabilities. The Am9580A is for the soft-sector, double-density floppy drives, ST506, and limited SMD drive interfaces. The Am9590 is meant to maintain support for double-density floppy drives and ST506 but adds ESDI, and full SMD support in both soft-sector and hard-sector environments. This section addresses the Am9590 and its interface to SMD drives. The Am9580A was originally designed to support the ST506, which uses the STEP and DIRECTION pins to move the head assembly in and out. SMD, however, uses a parallel address to change its head position; therefore, some changes in the Am9580A were needed. When parallel addressing is selected, the ST506 STEP and DIRECTION pins are disabled and PCEN and STEP pins change their functions to Clock and Data respectively, and the internal cylinder address is clocked out serially in 16 clock cycles.

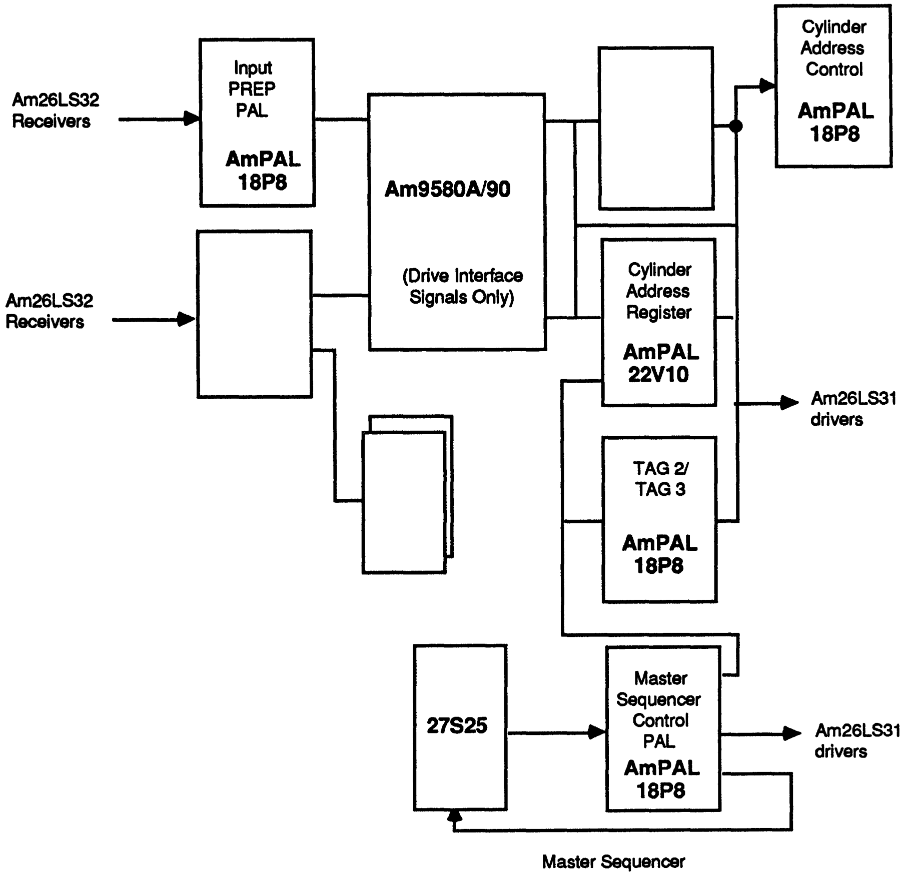
```
PAL Equation AmPAL20R4
       OSC     = (input clock)
       S[3:0]  = (input combinatorial)
       SELECT  = (input combinatorial)
       DSEL[1:0]      = (input combinatorial)
       /SELENI= (input combinatorial)
       /SELENO= (output combinatorial)
       DATA[3:0]      = (output combinatorial)
       ESDIEN  = (output combinatorial)
       CLK2    = (output registered)
       CLK4    = (output registered)
       CLK8    = (output registered)
       CLK16   = (output registered)
       X1      = (output combinatorial)
IF (S[0]•/DS[0]•/DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[1]•DS[0]•/DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[2]•/DS[0]•DS[1]•SELENI) THEN SELENO = TRUE;
IF (S[3]•DS[0]•DS[1]•SELENI) THEN SELENO = TRUE;
IF (/S[0]•/DS0•/DS[1]) THEN ESDIEN = TRUE;
IF (/S[1]•DS[0]•/DS[1]) THEN ESDIEN = TRUE;
IF (/S[2]•DS[0]•DS[1]) THEN ESDIEN = TRUE;
IF (SELECT) THEN ENABLE  DATA[3:0]
DATA[3:0] = S[3:0];
CLK2 = /CLK2;
CLK4 = CLK2•/CLK4 + /CLK2•CLK4
CLK8 = CLK8•/CLK2 + CLK8•/CLK4 + /CLK8•CLK4•CLK2;
CLK16 = CLK16•CLK2 + CLK16•/CLK4 + CLK16•/CLK8
       + /CLK16•CLK8•CLK4•CLK2;
IF (/S[3]•DS[0]•DS[1]) THEN X1 = CLK16;
IF (S[3]•DS[0]•DS[1] + S[2]•/DS[0]•DS[1] + S[1]•DS[0]•/DS[1])
       THEN X1 = CLK4;
```

Figure 3-32a ESDI-ST506 Floppy Interface PAL Equation

SMD Drive Interface

Before looking at the Am9590, the basic SMD interface is examined. The discussion that follows is not meant to be a tutorial on SMD but to introduce the reader to the basics of operation. Each manufacturer has slight variance in some areas, and, in each drive to be interfaced to the Am9590, these basics should be verified.

The SMD drive interface was originally designed with the idea that intelligence, at least in the form of registers, was located on the drive side. Thus, commands are issued to the drive via parallel lines and not serial lines as with floppy drives and ST506. This type of parallel data input is separated into three groups: Cylinder Address, Head Address and General State Lines. Each group is given a name as TAG information and fed to the drive one group at a time, in any order. TAG 1 is Cylinder Address, TAG 2 is Head Address and TAG 3 is General State Lines (e.g., Read Gate, Write Gate, etc.).



9480A 3-33

Figure 3-33 SMD Overview

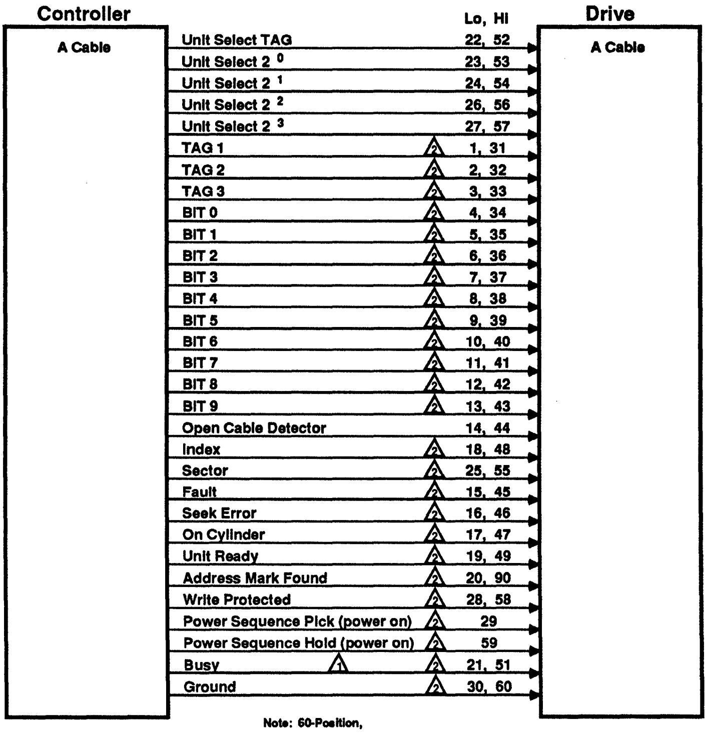
In order to read and write data on the drive, the drive must be addressed by its Drive Select (DRSEL) and must be in TAG 3 mode with SC from Drive Valid. These signals pass through a cable called "A" (or the "daisy-chain" cable). The 60-pin cable (30 pair of signal lines, differentially driven) run from the controller to each drive and then terminate at the end. This cable, however, does not carry high-speed data. High-speed data is carried on the "B" cable (or the "radial" cable). This cable carries the WRITE CLOCK, READ CLOCK (normally derived from the recovered Read data), READ DATA, WRITE DATA and returns the WRITE CLOCK. This is on a 26-pin cable (13 pair of signal lines, differentially driven).

Figures 3-34a and 3-34b show the standard pinouts for a typical SMD interface. Figure 3-35 shows the timings for TAG 1, TAG 2, and TAG 3 operations. Some vendors have TAGs 4 and 5 but these are beyond the scope of this application note.

Am9590 Register Set-up

This section discusses how to set up the registers on the Am9590 and implement SMD using the attached hardware described here.

The register set-up for the SMD driver to the Am9590 requires two registers to be programmed (General Select Byte Register and Data Select Byte Register).



Note: 60-Position,
28 AWG, Shielded Cable.
MAX Length = 100 Ft.
△1 Dual-Channel Units Only.
△2 Gated by Unit Selected.

9480A 3-34a

Figure 3-34a Cable 'A" Pinout

Am9590

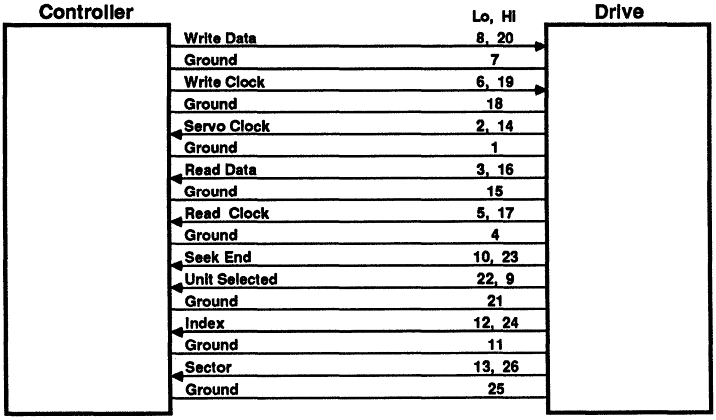
The Am9590 does not restrict drive usage in the Mode Register as does the Am9580A. Drive type and format type are programmed in the Drive Parameter Block only. The drive type is selected in the General Select Byte bits D1 and D0, and must be set to 0 for Parallel Modes. Whether the drive is hard-sectored or soft-sectored , it is selected in the next byte, bits D7 and D6; and this affects how the AMC and AMF lines are handled. In hard-sectored mode, AMC/AMF is replaced with the sector pulse coming from the drive and synchronizes the data state machine. Since the restricted mode is not required with the Am9590, as with the Am9580A, full overlapped seeks, as well as mixed drive types, are easily handled. Additionally, the Data Mark in the IOPB must be 80H or higher.

Theory of Operation

This section breaks the design into smaller entities and describes the interaction of these circuits with the HDC and the SMD drive. Refer to the schematic at the end of this application note in order to see how all the circuits fil together. See Figure 3-36.

"A" Connector Set-up

The "A" connector consists of only input and output signals. There are no bidirectional signals on the "A" cable. The input-only signals go to an Am26LS32 whose inputs are each terminated with a 56 ohm resistor to ground. The output section of the cable is driven by an Am26LS31. (Refer to the Appendix for design considerations using the Am26LS31.) The Am26LS32's pin 12 is grounded at all times to provide continuous drive to the cable. Besides the normally required disk signals, Reset is provided down the cable, as well as the power sequencing signals. The power sequencing relay would be driven by an output port on the local processor attached to the HDC if this feature is used. Pins B8 and B9, on the interface, are driven inverted from their normal conversion while the input is pulled up with a 10K resistor. These signals are only driven during TAG 1 from the Cylinder Address register PAL device where they are inverted before driving B8 and B9. When the Cylinder Address PAL device is three-stated, these signals are forced to a logical "0" on the interface during TAG 2 and TAG 3



Controller	Lo, HI	Drive
Write Data	8, 20	►
Ground	7	
Write Clock	6, 19	►
Ground	18	
◄ Servo Clock	2, 14	
Ground	1	
◄ Read Data	3, 16	
Ground	15	
◄ Read Clock	5, 17	
Ground	4	
◄ Seek End	10, 23	
◄ Unit Selected	22, 9	
Ground	21	
◄ Index	12, 24	
Ground	11	
◄ Sector	13, 26	
Ground	25	

Note: 1. 26-conductor, shielded cable.
MAX Length = 50 Ft.
2. No signals gated by unit selected.

9480A 3-34b

Figure 3-34b Cable "B" Pinout

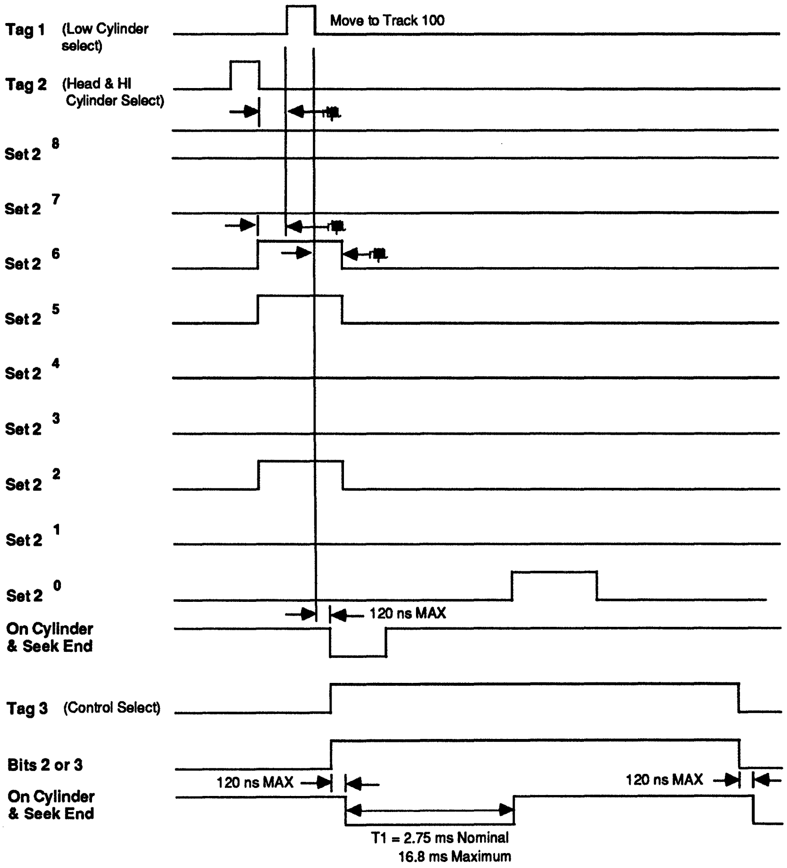
Input Preparation PAL Device

Not all the signals received at the input side of the "A" connector can go straight into the HDC. The signals that are received and qualified are: DRIVE READY, SEEK COMPLETE, and FAULT. These are qualified because a drive has been selected on the "B"cable. Additionally, two Fault lines are provided on the SMD interface and these are "OR'd" together to provide the Fault signal to the HDC. The SC term is set false if, upon select, the Cylinder Address Register was updated on the previous select cycle, or there is a head mismatch. This keeps the HDC from starting an operation too soon before the Master Sequencer can issue TAG 1, TAG 2, and TAG 3 to the drive.

Figure 3-37 shows the PAL device pin assignments and Figure 3-38 shows the PAL device equation in PLPL.

Master Sequencer

The master sequencer consists of two parts. A PAL device that prepares the inputs for the counter and decodes the states of the counter in order to sequence the enabling of specific tags onto the Bus and then the clocks for that tag. The second part is the counter itself. The entire circuit for the master sequencer is shown in Figure 3-39.



9480A 3-35

Figure 3-35 SMD Interface Timing

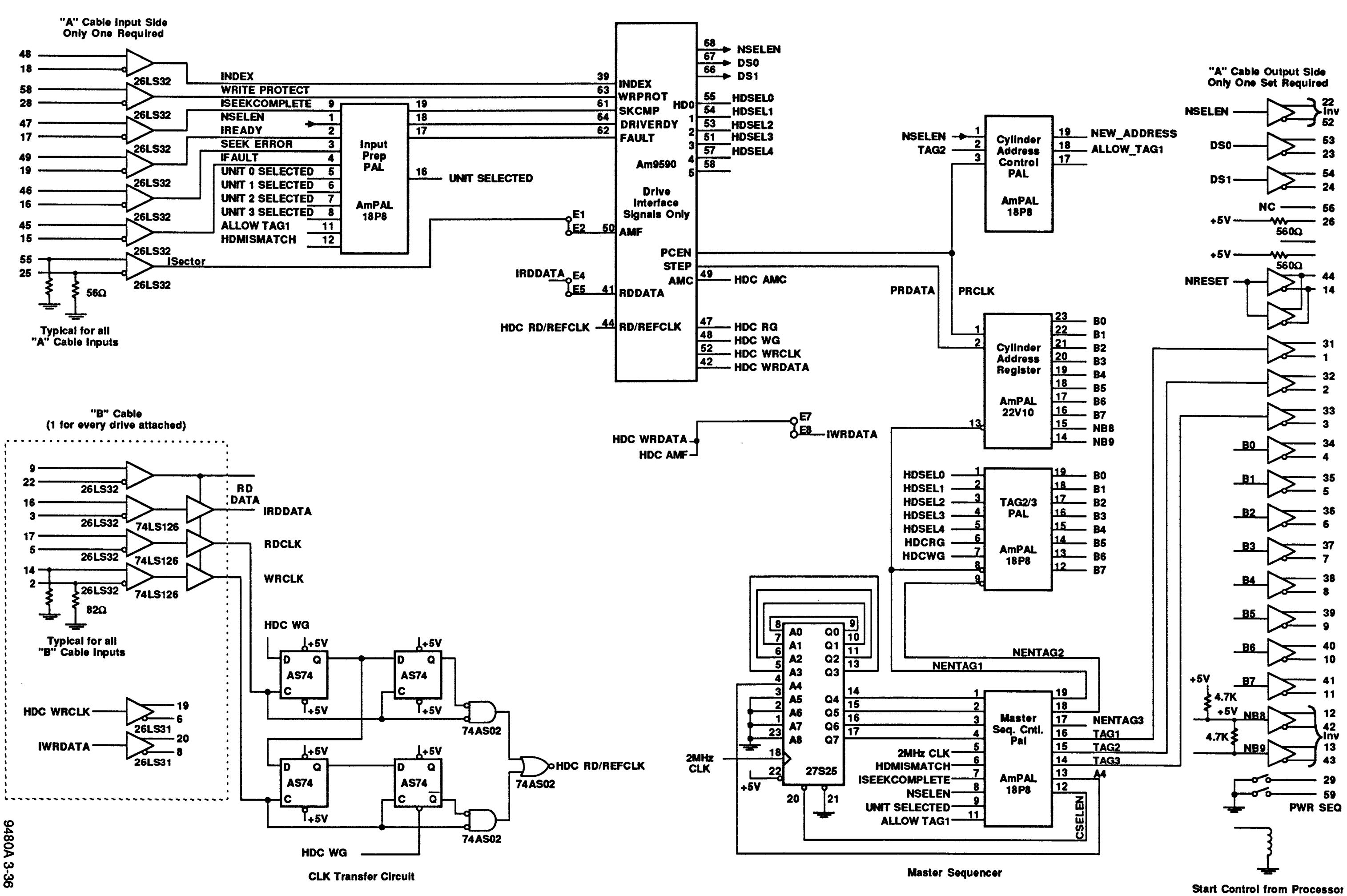
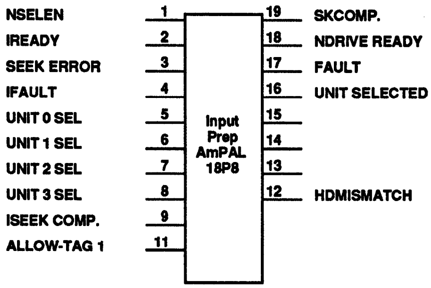


Figure 3-36 Am9590 SMD Interface

Note: All Drivers are 26LS31s with PIN 12 GND for Permanent Enable

9480A 3-36

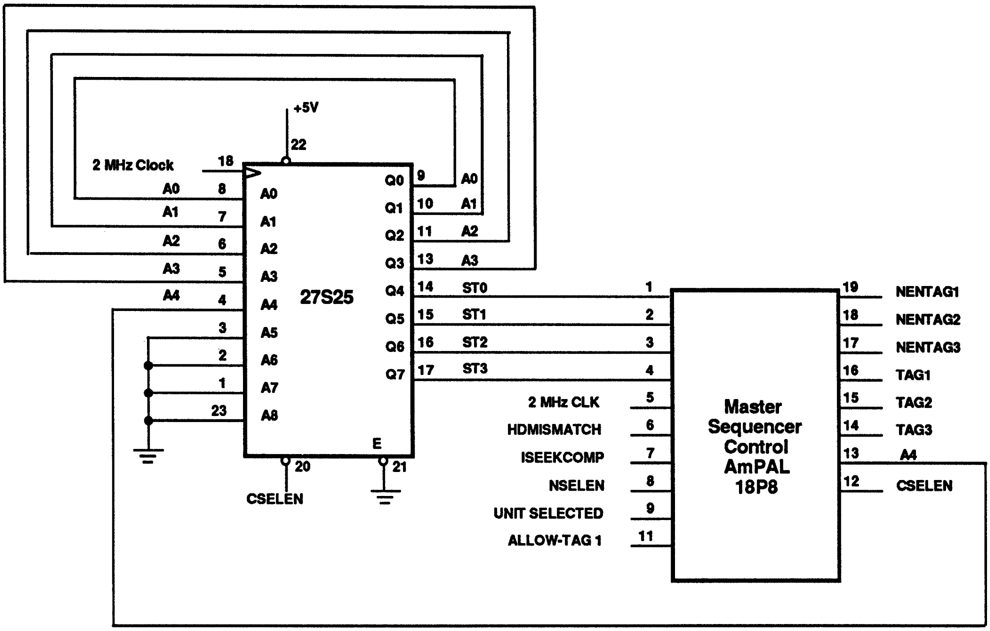


9480A 3-27

Figure 3-37 Input Preparation PAL Device Block Diagram

```
DEVICE  INPUT_PREP_PAL  (AMPAL18P8)

PIN
         /SELEN=1            IREADY=2            SEEKERROR=3            IFAULT=4
         UNIT0SEL=5          UNIT1SEL=6          UNIT2SEL=7             UNIT3SEL=8
         ALLOW_TAG1=11       SKCOMP=19           /DRIVEREADY=18         HDCFAULT=17
         ISEEKCOMP=9         UNITSEL=16          HDMISMATCH=12;

BEGIN

         SKCOMP  =  /ALLOW_TAG1*ISEEKCOMP*UNITSEL*SELEN*/HDMISMATCH;

         DRIVEREADY  =  SELEN*IREADY*UNITSEL;

         HDCFAULT  =  IFAULT*SELEN*UNITSEL  +  SEEKERROR*SELEN*UNITSEL;

         UNITSEL  =  SELEN  *  (UNIT0SEL  +UNIT1SEL  +  UNIT2SEL  +  UNIT3SEL  );

END.
```

Figure 3-38 Input Preparation PAL Equation

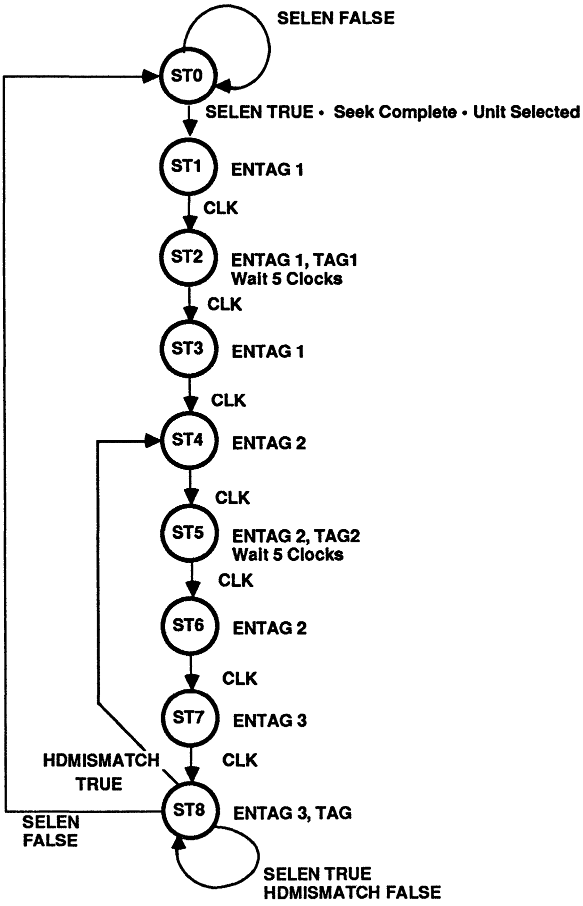


9480A 3-39

Figure 3-39 Master Sequencer Circuit Diagram

The counter is a registered 512 x 8 PROM (Am27S25). Only 32 of the 512 addresses are used. The eight outputs are divided into four address bits and four state control bits. The four address bits go to the A0 - A3 of the PROM; this accounts for the 16 addresses used. The PROM sits in Reset mode (Address = 0) when a drive is deselected (SELEN False). When a drive is selected and the drive returns Interface SC True, the Am27S25 reset term is released, and on each clock the PROM selects the next state. Figure 3-40 shows the actual PROM program and Figure 3-41 is a state flow diagram. The clock is 2 MHz, meeting the drive requirement specification of set-up and hold of TAG data with respect to TAG clock. Basically, the PROM steps to the next address until address F, where the PROM stays until either RESET or A4 is set to a logical True. A4 is set to a logical True by the Head Mismatch PAL device when the HDC changes a Head Address from that issued under TAG 2. When this happens, A4 is set to 1 and the PROM goes to address 1F where a "jump to 18" is found and it reissues TAG 2 to the drive. At address 1C, A4 goes to a logical False and the PROM moves to address 0D on the next clock pulse, and will stay at 0F when it steps to that address.

The companion part to the Am27S25 is the AmPAL18P8. This PAL device preconditions the two control lines, CSELEN 'Am27S25 Reset pin' and A4 so that they are clocked by the 2 MHz clock.



9480A 3-40

Figure 3-40 Am27S25 State Flow Diagram

Additionally, the PAL device decodes the state lines from the Am27S25 to the required TAG 1, 2, and 3 output enables and TAG 1, 2, and 3 clocks. Figure 3-42 shows the pinout and Figure 3-43 shows the PAL device equations in PLPL. The equations are straightforward, the CSELEN starts the state machine and it does not become true until a drive is selected (SELEN True) and a drive has acknowledged (Unit-selected True) and the drive is on cylinder (I SC). One other note is that TAG 1 will only be issued if the Cylinder Address Register PAL device has been updated with a new value and, hence, ALLOW_TAG 1 is True.

Cylinder Address Register

The Cylinder Address Register is contained in two PAL devices. One is the shift register itself. This PAL device is nothing more than a 10-bit shift register with the last two bits inverted. These are inverted so that when the PAL device is three-stated (NENTAG1 False), B8 and B9 are driven to logical False on the interface for TAG 2 and TAG 3 operations. This PAL device is enabled onto the local Bx Bus by NENTAG 1 True. Figure 3-44 shows the pinout and Figure 3-45 shows the PAL device equation.

PROM	D4-D7	D0-D3
Address	State	Next Address
0	0	1
1	1	2
2	2	3
3	2	4
4	2	5
5	2	6
6	3	7
7	4	8
8	5	9
9	5	A
A	5	B
B	5	C
C	6	D
D	7	E
E	8	F
F	8	F

All locations "0" except:

Address	State	Next Address
18	5	9
19	5	A
1A	5	B
1B	5	C
1C	6	D
1D	7	E
1E	8	F
1F	4	8

Goes to 1F if HD MISMATCH

Goes to 0D if A4 goes to a logical False.

9480A 3-41

Figure 3-41 Am27S25 Program

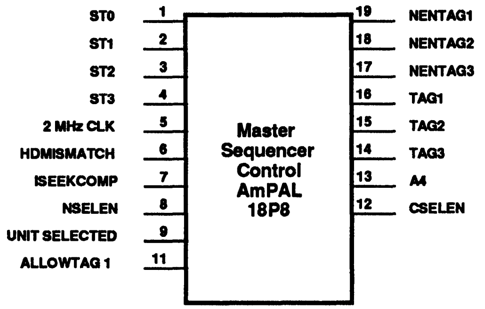


Figure 3-42 Master Sequencer Control PAL Device Pinout

```
DEVICE MASTER_SEQUENCER (AMPAL18P8)

PIN

    ST0=1           ST1=2               ST2=3               ST3=4
    CLK=5           HDMISMATCH=6        ISEEKCOMP=7         /SELEN=8
    UNITSEL=9       ALLOWTAG1=11        /ENTAG1=19          /ENTAG2=18
    /ENTAG3=17      TAG1=16             TAG2=15             TAG3=14
    A4=13           CSELEN=12;

BEGIN

    CSELEN = SELEN*CLK*UNITSEL*ISEEKCOMP +
               CSELEN*SELEN;

    A4 = HDMISMATCH*ST3*/ST2*/ST1*/ST0;

    ENTAG1 = /ST3*/ST2*(/ST1*ST0 + ST1*/ST0 + ST1*ST0);

    ENTAG2 = /ST3*ST2*(/ST1*/ST0 + /ST1*ST0 + ST1*/ST0);

    ENTAG3 = /ST3*ST2*ST1*ST0 + ST3*/ST2*/ST1*/ST0;

    TAG1 = ALLOWTAG1*/ST3*/ST2*ST1*/ST0;

    TAG2 = /ST3*ST2*/ST1*/ST0;

    TAG3 = ST3*/ST2*/ST1*/ST0;
```

Figure 3-43 Master Sequencer Control PAL Device Equation

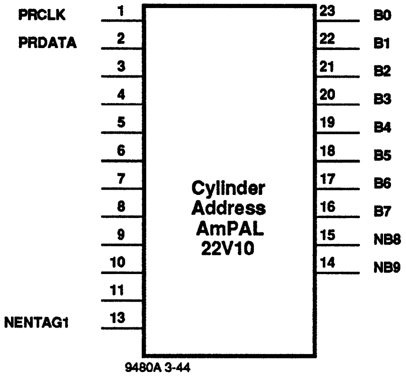


Figure 3-44 Cylinder Address Register Pinout

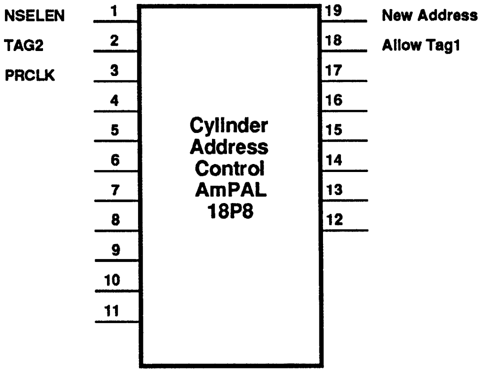
```
DEVICE CYLINDER_ADDRESS_REGISTER (AMPAL22V10)

PIN

    PRCLK=1 PRDATA=2            /ENTAG1=13      B0=23
    B1=22          B2=21           B3=20        B4=19
    B5=18          B6=17           B7=16        /B8=15
    /B9=14;

BEGIN

    IF (ENTAG1) THEN ENABLE (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9);

    B0 = PRDATA;

    B1 = B0;

    B2 = B1;

    B3 = B2;

    B4 = B3;

    B5 = B4;

    B6 = B5;

    B7 = B6;

    B8 = B7;

    B9 = B8;

END.
```

Figure 3-45 Cylinder Address Register Equation

The other PAL device is for the control of the Cylinder Address Register. It monitors the PRCLK line and, if a PRCLK is received with SELEM True, then the Set/Reset flip-flop NEW_ADDRESS is set. The new address is then transferred to the ALLOW_TAG 1 latch when SELEN goes False. This is to keep the ALLOW_TAG 1 synchronized before Drive reselection. This output goes to both the Master Sequencer Control PAL device and the Input Preparation PAL device. Both flip-flops are reset by the TAG 2 input. Figure 3-46 shows the pinout and Figure 3-47 gives the PAL device equations.

TAG 2/TAG 3

The TAG 2/TAG 3 PAL device multiplexes the correct information to indicate a TAG 2 or TAG 3 operation onto the correct Bx signal pins. With TAG 2, the Head Address lines are used, and with TAG 3, the RG/WG lines are driven onto the disk drive. The Fault Clear input is from the local microprocessor and is used for abnormal drive conditions to clear either a Seek error or a Fault. This will be driven to a logical True by the microprocessor and then the HDC would be given an IOPB that was a Seek to



9480A 3-46

Figure 3-46 Cylinder Register Control PAL Device Pinout

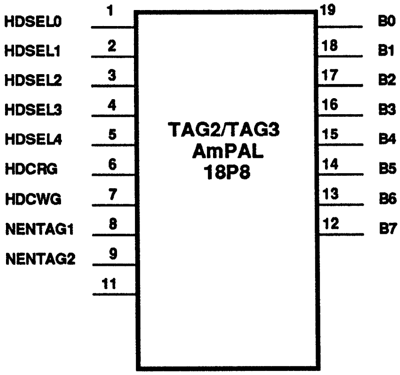
```
DEVICE CYLINDER_CONTROL (AMPAL18P8)

PIN
    /SELEN=1        TAG2=2              PRCLK=3             NEWADD=19
    ALLOWTAG1=18;

BEGIN

    NEWADD = PRCLK + NEWADD*/TAG2;

    ALLOWTAG1 = NEWADD*/SELEN*/TAG2 +
                   ALLOWTAG1*SELEN*/TAG2 +
                   ALLOWTAG1*NEWADD*/TAG2;


END.
```

Figure 3-47 Cylinder Register Control PAL Device Equation

Track Zero command. When the IOPB is completed, the Fault condition will be altered and the drive returns to Track 0. The PAL device is enabled onto the Bx lines when the ENTAG 1 signal is False and the Cylinder Address register PAL device is not enabled. ENTAG 2 is the signal that qualifies the multiplexing of the actual data onto the Bx lines. Figure 3-48 shows the TAG PAL device (TAG 2/ TAG 3) pinout and Figure 3-49 shows the TAG PAL device equations.

"B" Connector Set-up

The "B" connector connects the Radial cable and carries the high-speed clocks and data. Again, Am26LS32s are used to receive data from the cable with each input terminated into an 82 ohm resistor, and half of the Am26LS31s are used to transmit the WRCLK and WRDATA information to the disk drive.



9480A 3-48

Figure 3-48 TAG PAL Device Pinout

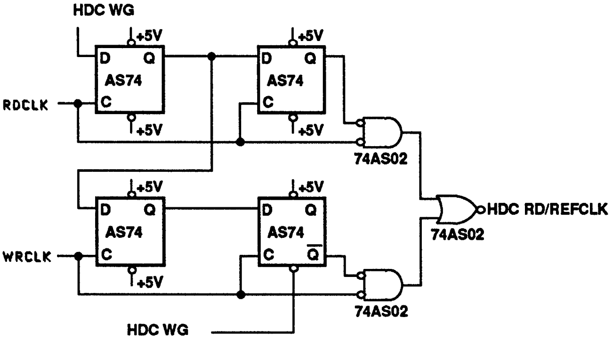
```
DEVICE TAG2_TAG3 (AMPAL18P8)

PIN

    HDSEL0=1       HDSEL1=2            HDSEL2=3            HDSEL3=4
    HDSEL4=5       HDCRG=6             HDCWG=7             /ENTAG1=8
    /ENTAG2=9      B0=19               B1=18               B2=17
    B3=16          B4=15               B5=14               B6=13
    B7=12          FLTCLR=11;

BEGIN

    IF (/ENTAG1) THEN ENABLE (B0,B1,B2,B3,B4,B5,B6,B7);

    B0 = ENTAG2*HDSEL0 +/ENTAG2*HDCWG;

    B1 = ENTAG2*HDSEL1 + /ENTAG2*HDCRG;

    B2 = ENTAG2*HDSEL2;

    B3 = ENTAG2*HDSEL3;

    B4 = ENTAG2*HDSEL4 + /ENTAG2*FLTCLR;

    B5 = ENTAG1;

    B6 = /ENTAG2*FLTCLR ;

    B7 = ENTAG1;

END.
```

Figure 3-49 TAG PAL Device Equation

The 74LS126 provides a common bus for RDCLK, RDDATA and WRCLK on the board so that only one clock transfer circuit is required. The disk drive that returns an Unit Selected signal back down the "B" cable is the one that drives the Read Clock, Read Data, and WR Clock bus; the Unit Select signal enables the 74LS126 onto the Write Bus.

Clock Transfer Circuitry

The Am9580A/90 has only one clock input (RD/REFCLK) pin for the data sequencer. However, SMD drives provide two separate clocks: Read and Write (See Figure 3-50). The RD/REFCLK input on the Am9590 cannot be glitched when changing from READ clock to WRITE clock. The four 74AS74s perform the glitchless crossover as the RD/REFCLK output from the 74AS02 stays High for one whole Read clock and one whole Write Clock before toggling. The clock switch occurs on the HDC/WG rising edge, and the clock will be High on the falling edge for two Read clocks; glitchless in both cases. At the beginning of Interface Select, RD/REFCLK input is ignored for about 12 clock times so that glitches during this time are acceptable.



9480A 3-50

Figure 3-50 CLK Transfer Circuit

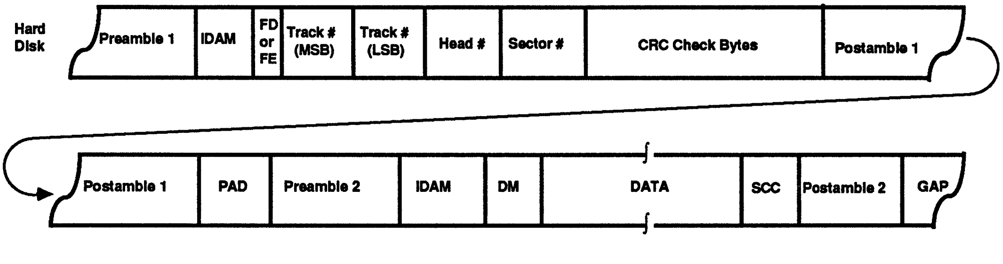
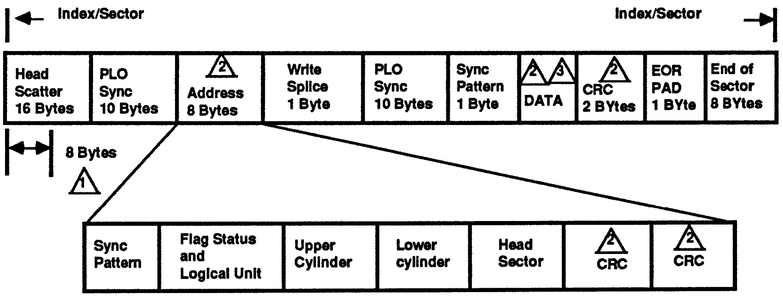


Figure 3-51a ST506 Sector Format



9480A 3-51a & b

Figure 3-51b SMD Hard Disk Sector Format

3.4.7 RLL Encoder-Decoder for the Am9580A/90

In recent years, the need for more storage capacity on disk drives has become more and more relevant. There are several ways to increase the storage capacity on magnetic disks: The number of surfaces can be increased; a higher bit density and therefore higher data rate; or a different encoding scheme; are some of the possible ways to achieve this goal. Where it requires efforts from the disk drive manufacturer to increase the number of surfaces or the bit density (improved media), advances in the encoding scheme falls on the disk controller.

This section shows a new approach to implement the RLL (Run Length Limited) 2,7 Code. The emphasis in this design is to use a method of encoding and decoding the data so that the commonly used and complicated state machines become obsolete. The result is a single PAL device (AmPAL23S8) that implements both the encoder and decoder. A second PAL device (AmPAL16R8) generates and detects the address marks at the beginning of each sector.

The RLL Code

In order to record data onto a magnetic media, NRZ data coming from the disk controller needs to be encoded, so that they translate into flux transitions on the disk drive. Furthermore, clock pulses need to be mixed with data to allow a synchronous retrieval of data from the disk. Figure 3-52 shows the method used for the MFM encoding scheme.

The MFM coding rules are: Insert Clock Pulse only if two consecutive NRZ 0s.

It appears that the output data (mix of data and clock) of the encoder are running at twice the clock speed of the input data.

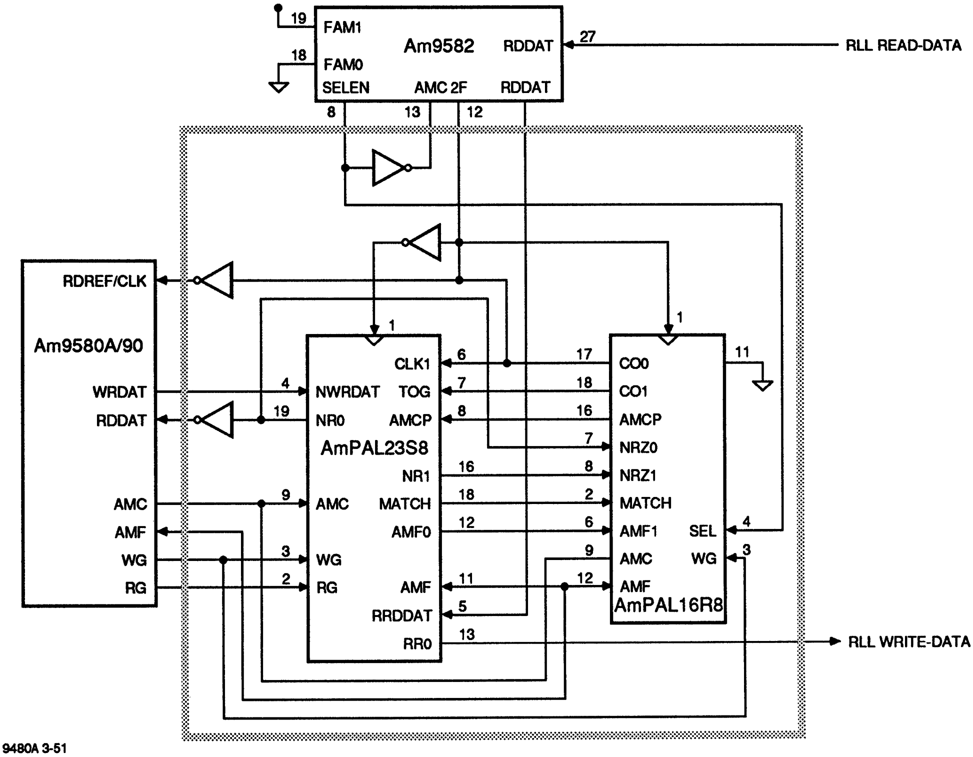


Figure 3-52

The RLL encoding scheme is a very general method to encode data. Basically, "RLL" just defines a minimum and a maximum number of 0s between two 1s in the output data stream. Again, the ouput is running at twice the speed of the input. There are several different RLL codes in use today. They differ in their maximum and minimum numbers and in the actual bit-pattern used to encode a certain input-pattern.

There are certain technical and physical limitation to the values in use for these codes. The maximum number of 0s is limited by the quality of the PLL synchronizing onto the data stream while reading. The minimum number of 0s, on the other hand, is limited by the possible bit density on the magnetic media. The goal is to use a given bit density and increase the number of possible 0s as much as possible. It becomes obvious that this definition of RLL code makes the older MFM coding scheme just a special case of RLL.

The most common RLL code used today is the so called 2,7 Code. These numbers indicate that the minimum number of 0s between two 1s is "two" and the maximum number is "seven". Figure 3-53 shows the possible patterns on both sides of the encoder. The following application implements the RLL 2,7 Code.

Usually a RLL encoder/decoder is realized by a state machine. This state machine clocks the data stream to be encoded or decoded into a shift register. Assuming the encoder is running synchronously, it would clock in the first two bits of NRZ data, and compares them to the two possible values. If there is no match, the logic will shift in one more bit, compare again and perhaps shift in another bit. At this time a match condition must be found, otherwise the state machine is not in sync and must be resynchronized. All these different modes require a rather complicated state machine just to encode the data. If the decoding function is added, even more states are needed, since the shift steps for decoding are different from encoding.

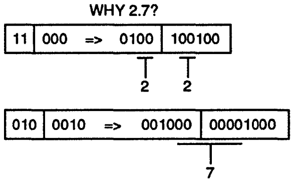
The Implementation

The scheme used in this application does not need any state machine. It uses two shift registers, one 4-bit register for the NRZ data and an 8-bit register for the RLL data. These shift registers are preset with a certain value before clocking data in. The preset value is chosen so that there is an unique stop pattern for each of the bit patterns to be recognized (Figure 3-54). Without a preset, it is not possible to recognize the different input patterns, because some parts are repetetive and the encoder/decoder would not be able to stop at the right point. The scheme works similar for encoding and decoding.

The shift registers and the entire control logic are implemented in one PAL device (AmPAL23S8). The second PAL device AmPAL16R8 provides a clock divider (CO0 and CO1) and logic to generate and detect an address mark.

CODING RULES

NRZ	RLL
10	1000
11	0100
000	100100
010	001000
011	000100
0010	00001000
0011	00100100



9480A 3-53

Figure 3-53

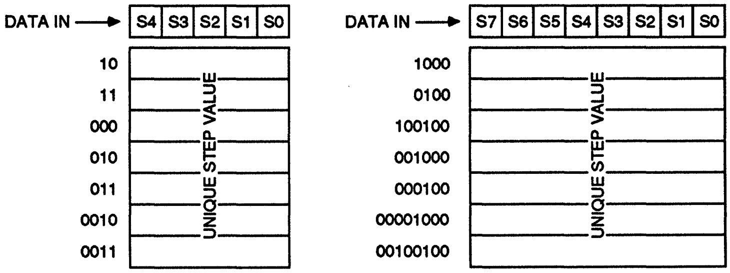
There is no read or write operation performed on the disk drive, both NRZ and RLL shift register are preset with the appropriate values. An inverted SELEN signal is used to assert AMC for the Am9582. The rising edge of this signal clocks the status of the FAM_0 and FAM_1 input into the DDS. For simplicity reasons the following describes only a write operation. As soon as the WG signal becomes active, the NRZ shift register starts shifting in data. This operation is done with half the clock speed of the PAL device. The shift operation continues until a defined NRZ pattern is recognized. At this point the MATCH output of the encoder goes active. This MATCH condition, together with the status of the four NRZ register bits, will cause the output (RLL) shift register to be loaded with the appropriate RLL pattern. At the same time, the input register will be preset with the start value again. There will be no preset on the output register. This process will repeat as long as WG is active.

Before any information is written to a sector, a PLO SYNC field is written to the disk. This synchronization field is required by the PLL (Phase-Locked Loop) during reading. For MFM encoding, a pattern of several 0s is used for this purpose. However, this pattern is not suitable for the RLL 2,7 Code. Figure 3-53 shows that the data "000" are encoded into "100100". Since this is a symmetric pattern, it cannot be used to synchronize properly. Therefore, this application uses the data "11" instead, which translates into the RLL data "0100". The Am9580A/90 sends out a PLO SYNC field of 0s. In order to avoid a special treatment of the sync field in the encoder, all data coming from the HDC are inverted. During a read operation, NRZ data are inverted again before being sent to the HDC.

Whenever an address mark needs to be written, the HDC will assert the signal AMC (Address Mark Control). The AmPAL16R8 generates an AMCP signal of one clock length. This causes the encoder PAL device to write the address mark pattern "00000100". This does not violate the RLL 2,7 encoding scheme, but it is an unique pattern.

The HDC, when starting a read operation, will always asserts AMC to search for an address mark. The decoder will start looking for the sync pattern "0100". This pattern has to occur at least five times before the AmPAL23S8 starts looking for the address mark. The counter for the sync patterns is implemented in the second PAL device. This method insures synchronization at the beginning of a sector.

After an address mark has been recognized, the device waits until the first four bits of NRZ data are available in the NRZ shift register and asserts AMF (Address Mark Found). This causes the HDC to use the data bit at the next rising edge of RD/REFCLK as the first bit of valid data.



9480A 3-54

Figure 3-54

```
DEVICE RLLENCDEC (P23S8)

"
--------------------------------------------------------------------------------
THIS PAL-DEVICE ENCODES AND DECODES RLL 2.7 CODE
IT USES THE Am9582 AS A PLL WHEN READING FROM DISK
EXPECTED ARE THE 2f CLOCK FROM THE DDS AND TWO CLOCK INPUTS
WHICH ARE 2f/2 AND 2f/4.
THE COUNTER FOR CLK1 AND TOG NEEDS TO BE RESET
WITH /RG * /WG OR IF MATCH IS ACTIVE.
THE SIGNAL AMCP GIVES A PULSE OF 2 CLOCKS TO WRITE AN ADDRESS
MARK. THIS SIGNAL IS ALSO ACTIVATED IF THE DEVICE MUST LOOK
FOR AN ADDRESS MARK.

ADVANCED MICRO DEVICES                        JOCHEN POLSTER       3/4/1987
VERSION 1.0
--------------------------------------------------------------------------------
"

PIN
CLK = 1 (Clock)
RG = 2 (INPUT combinatorial)
WG = 3 (INPUT combinatorial)
/NWRDAT = 4 (INPUT combinatorial)
RRDDAT = 5 (INPUT combinatorial)
CLK1 = 6 (INPUT combinatorial)
/TOG = 7 (INPUT combinatorial)
/AMCP = 8 (INPUT combinatorial)
AMC = 9 (INPUT combinatorial)
AMF = 11 (INPUT combinatorial)
AMFO = 12 (IO enable_high active_high registered reg_feedback)
/RR0 = 13 (IO enable_high active_low registered reg_feedback)
NR3 = 14 (IO enable_high active_high registered)
/NR1 = 15 (IO enable_high active_low registered)
MATCH = 16 (IO enable_high active_high registered)
RR5 = 17 (IO enable_high active_high registered)
NR2 = 18 (IO enable_high active_high registered reg_feedback)
NR0 = 19 (IO enable_high active_high registered reg_feedback)
/RR4 = 25 (BREG registered active_low)
/RR2 = 22 (BREG registered active_low)
/RR3 = 23 (BREG registered active_low)
/RR1 = 24 (BREG registered active_low)
/RR7 = 21 (BREG registered active_low)
/RR6 = 26 (BREG registered active_low)
;

BEGIN

ENABLE (NR0,NR3,NR1,NR2,RR5,MATCH,RR0,AMFO) = #B11111111;

NR3       :=         WG * NWRDAT * CLK1                  +   "SHIFT DATA IN"
                     NR3 * /CLK1 * /MATCH                +   "HOLD"
                     WG * MATCH                          +
                     RG * RR7 * RR6 * RR5 * RR4 * /RR3 *
                     /RR2 * RR1 * RR0                    +   "READ CASE 2"
                     RG * RR7 * RR6 * RR5 * RR4 * RR3 *
                     RR2 * /RR1 * /RR0                   +   "READ CASE 5"
                     RG * RR7 * RR6 * RR5 * RR4 * RR3 *
                     /RR2 * RR1 * RR0                    +   "READ CASE 7"
                     /RG * /WG;                              "RESET"
```

Figure 3-55

```
NR2      :=        RG * NR3 * CLK1 * /MATCH                     +    "SHIFT DATA"
                   WG * NR3 * CLK1 * /MATCH                     +    "SHIFT DATA"
                   NR2 * /CLK1 * /MATCH                         +    "HOLD"
                   RG * MATCH * /AMCP *
                   /(RR5 * RR4 * RR3 *
                   /RR2 * /RR1 * /RR0);                              "READ CASE 3"

NR1      :=        /NR2 * CLK1 * /MATCH                         +    "SHIFT DATA"
                   WG * NR1 * /CLK1 * /MATCH                    +    "HOLD"
                   RG * NR1 * /CLK1 * /MATCH                    +    "HOLD"
                   RG * NR1 * /CLK1 * MATCH * /RR3 * /RR2       +    "READ CASE 1 AND 2"
                   WG * MATCH                                   +    "RESET AFTER MATCH"
                   RG * RR7 * RR6 * RR5 * RR4 * RR3 *
                   /RR2 * /RR1 * /RR0                           +    "READ CASE 3"
                   RG * RR7 * RR6 * /RR5 * /RR4 * RR3 *
                   RR2 * /RR1 * /RR0                            +    "READ CASE 4"
                   RG * RR7 * RR6 * RR5 * RR4 * RR3 *
                   RR2 * /RR1 * /RR0                            +    "READ CASE 5"
                   RG * RR7 * RR6 * /RR5 * /RR4 * RR3 *
                   RR2 * RR1 * RR0                              +    "READ CASE 6"
                   RG * RR7 * RR6 * RR5 * RR4 * RR3 *
                   /RR2 * RR1 * RR0                             +    "READ CASE 7"
                   /WG * /RG;                                        "RESET"

NR0      :=        /NR1 * CLK1 * /MATCH                         +    "SHIFT DATA"
                   RG * NR0 * /CLK1 * /MATCH                    +    "HOLD"
                   WG * NR0 * /CLK1 * /MATCH                    +    "HOLD"
                   RG * NR0 * /CLK1 * MATCH *
                   RR7 * RR6 * RR5 * RR4 * RR3 *
                   /RR2 * /RR1 * /RR0                           +    "READ CASE 3"
                   RG * NR0 * /CLK1 * MATCH *
                   RR7 * RR6 * /RR5 * /RR4 * RR3 *
                   RR2 * /RR1 * /RR0                            +    "READ CASE 4"
                   RG * NR0 * /CLK1 * MATCH *
                   RR7 * RR6 * RR5 * RR4 * RR3 *
                   RR2 * /RR1 * /RR0                            +    "READ CASE 5"

                   RG * NR0 * /CLK1 * MATCH *
                   RR7 * RR6 * /RR5 * /RR4 * /RR3 *
                   /RR2 * RR1 * RR0                             +    "READ CASE 1"
                   RG * NR0 * /CLK1 * MATCH *
                   RR7 * RR6 * RR5 * RR4 * /RR3 *
                   /RR2 * RR1 * RR0;                                 "READ CASE 2"


MATCH    :=        WG * CLK1 * NR3 * /NWRDAT * NR2 * NR1 * /TOG   +   "WRITE CASE 1"
                   WG * CLK1 * NR3 * NWRDAT * NR2 * NR1 * /TOG    +   "WRITE CASE 2"
                   WG * CLK1 * /NR3 * /NWRDAT * /NR2 * /NR1 * TOG +   "WRITE CASE 3"
                   WG * CLK1 * NR3 * /NWRDAT * /NR2 * /NR1 * TOG  +   "WRITE CASE 4"
                   WG * CLK1 * NR3 * NWRDAT * /NR2 * /NR1 * TOG   +   "WRITE CASE 5"
                   WG * CLK1 * NR3 * /NWRDAT * /NR2 * NR1 * /TOG  +   "WRITE CASE 6"
```

Figure 3-55 Continued

```
                        WG * CLK1 * NR3 * NWRDAT * /NR2 * NR1 * /TOG   +   "WRITE CASE 7"
                        RG * /AMCP * RR7 * RR6 * RR5 * /RR4 * /RR3 *
                        RR2 * RR1 * /RRDDAT * /AMC * /MATCH       +   "READ CASE 1"
                        RG * /AMCP * RR7 * RR6 * RR5 * /RR4 * /RR3 *
                        RR2 * RR1 * /RRDDAT * AMC * AMF * /MATCH       +   "READ CASE 1"
                        RG * /AMCP * RR7 * /RR6 * /RR5 * /RR4 * /RR3 *
                        RR2 * RR1 * /RRDDAT * /MATCH                      +   "READ CASE 2"
                        RG * /AMCP * RR7 * /RR6 * /RR5 * RR4 * /RR3 *
                        /RR2 * /RR1 * /RRDDAT * /MATCH                    +   "READ CASE 3"
                        RG * /AMCP * RR7 * RR6 * RR5 * RR4 * RR3 *
                        /RR2 * /RR1 * /RRDDAT * /MATCH                    +   "READ CASE 4"
                        RG * /AMCP * RR7 * /RR6 * /RR5 * RR4 * RR3 *
                        /RR2 * /RR1 * /RRDDAT * /MATCH                    +   "READ CASE 5"
                        RG * /AMCP * RR7 * RR6 * RR5 * RR4 * RR3 *
                        RR2 * RR1 * /RRDDAT * /MATCH                      +   "READ CASE 6"
                        RG * /AMCP * RR7 * /RR6 * /RR5 * RR4 * /RR3 *
                        RR2 * RR1 * /RRDDAT * /MATCH                      +   "READ CASE 7"
                        RG * AMCP * RR7 * /RR6 * /RR5 * RR4 * RR3 *
                        RR2 * RR1 * /RRDDAT * /MATCH                      +   "ADDRMARK FOUND"
                        WG * AMCP * /AMFO;


RR7         :=          RG * /RRDDAT                                 +   "SHIFT DATA IN"
                        WG;                                          "ALL WRITE CASES"

RR6         :=          WG * RR7 * /MATCH                            +   "SHIFT DATA WRITE"
                        WG * MATCH                                   +   "WG MATCH"
                        RG * RR7 * /MATCH;                               "SHIFT DATA READ"

RR5         :=          WG * /RR6 * /MATCH                           +   "SHIFT DATA WRITE"
                        RG * /RR6 * /MATCH                           +   "SHIFT DATA READ"
                        WG * MATCH * NR3 * NR2 * /NR1 * /NR0         +   "WRITE CASE 2"
                        WG * MATCH * /NR3 * /NR2 * NR1 * NR0         +   "WRITE CASE 3"
                        WG * MATCH * NR3 * NR2 * NR1                 +   "WRITE CASE 5,7"
                        RG * MATCH                                   +
                        WG * AMCP * MATCH;                               "WRITE ADDR. MARK"

RR4         :=          WG * /RR5 * /MATCH                           +   "SHIFT DATA WRITE"
                        WG * AMCP * MATCH                            +   "WRITE AM"
                        RG * /RR5                                    +   "SHIFT DATA READ"
                        RG * MATCH                                   +
                        WG * MATCH *
                        /(/NR3 * NR2 * /NR0)          *          "WRITE CASE 1,6"
                        /(/NR3 * NR2 * NR1 * NR0)                    +   "WRITE CASE 4"
                        /RG * /WG;

RR3         :=          WG * RR4 * /MATCH                            +   "SHIFT DATA WRITE"
                        WG * MATCH * RR4 *
                        ((/NR3 * NR2 * /NR1 * /NR0)                  +   "WRITE CASE 1"
                        (NR3 * NR2 * /NR1 * /NR0))                   +   "WRITE CASE 2"
                        WG * MATCH *
                        /(/NR3 * NR2 * /NR1 * /NR0)     *            "WRITE CASE 1"
                        /(NR3 * NR2 * /NR1 * /NR0)                   +   "WRITE CASE 2"
                        RG * RR4                                     +   "SHIFT DATA READ"
                        RG * MATCH                                   +
                        /RG * /WG                                    +
                        WG * AMCP * MATCH;
```

Figure 3-55 Continued

```
RR2        :=          WG * RR3 * /MATCH                          +   "SHIFT DATA WRITE"
                       WG * AMCP * MATCH                          +   "WRITE AM"
                       WG * MATCH * RR3 *
                       ((/NR3 * NR2 * /NR1 * /NR0)                +   "WRITE CASE 1"
                       (NR3 * NR2 * /NR1 * /NR0))                 +   "WRITE CASE 2"
                       RG * RR3                                   +   "SHIFT DATA READ"
                       RG * MATCH                                 +
                       WG * MATCH * /NR3 * NR2 * NR1              +   "WRITE CASE 4,6"
                       WG * MATCH * NR3 * NR2 * NR1 * NR0         +   "WRITE CASE 5"
                       /RG * /WG;

RR1        :=          WG * RR2 * /MATCH                          +   "SHIFT DATA WRITE"
                       WG * AMCP * MATCH                          +   "WRITE AM"
                       WG * MATCH * RR2 *
                       ((/NR3 * NR2 * /NR1 * /NR0)  +                 "WRITE CASE 1"
                       (NR3 * NR2 * /NR1 * /NR0)    +                 "WRITE CASE 2"
                       (/NR3 * /NR2 * NR1 * NR0)    +                 "WRITE CASE 3"
                       (/NR3 * NR2 * NR1 * NR0 )    +                 "WRITE CASE 4"
                       (NR3 * NR2 * NR1 * NR0  ))                 +   "WRITE CASE 5"
                       RG * RR2                                   +   "SHIFT DATA READ"
                       RG * MATCH                                 +
                       WG * MATCH * NR2 * NR1 * /NR0              +   "WRITE CASE 6,7"

                       /RG * /WG;

RR0        :=          WG * RR1 * /MATCH                          +   "SHIFT DATA WRITE"
                       WG * MATCH * RR1 *
                       ((/NR3 * NR2 * /NR1 * /NR0)  +                 "WRITE CASE 1"
                       (NR3 * NR2 * /NR1 * /NR0)    +                 "WRITE CASE 2"
                       (/NR3 * /NR2 * NR1 * NR0)    +                 "WRITE CASE 3"
                       (/NR3 * NR2 * NR1 * NR0 )    +                 "WRITE CASE 4"
                       (NR3 * NR2 * NR1 * NR0  ))                 +   "WRITE CASE 5"
                       RG * RR1                                   +   "SHIFT DATA READ"
                       RG * MATCH                                 +
                       WG * MATCH * NR2 * NR1 * /NR0              +   "WRITE CASE 6,7"

                       /RG * /WG                                  +

                       WG * AMCP * MATCH;

AMFO       :=          RG * AMCP * MATCH                          +   "ADDRESS MARK FOUND"
                       RG * AMFO * AMC * /AMF                     +   "HOLD"
                       WG * AMCP * /AMFO                          +   "SET FOR WRITE"
                       WG * AMFO * AMC;                               "HOLD"

END.
```

Figure 3-55 Continued

```
DEVICE RLLEXT (P16R8)

"
-----------------------------------------------------------------------------------------
THIS PAL DEVICE GENERATES THE ADDITIONAL SIGNAL REQUIRED FOR THE
RLLENCDEC PAL.
THE COUNTER FOR CLK1 AND TOG NEEDS TO BE RESET WITH /SELEN
OR IF MATCH IS ACTIVE.

ADVANCED MICRO DEVICES           JOCHEN POLSTER                  3/4/1987
VERSION 1.0
-----------------------------------------------------------------------------------------
"

PIN
CLK = 1 (Clock)
MATCH = 2 (INPUT combinatorial)
WG = 3 (INPUT combinatorial)
/SELEN = 4 (INPUT combinatorial)
AMFI = 6 (INPUT combinatorial)
WRDAT = 5 (INPUT combinatorial)
NRZ[1:0] = 8,7 (INPUT combinatorial)
AMC = 9 (INPUT combinatorial)
/AMF = 12 (OUTPUT registered active_low)
/CNT[2:0] = 15,14,13 (OUTPUT registered active_low)
/AMCP = 16 (OUTPUT registered active_low)
/CO[1:0] = 18,17 (OUTPUT registered active_low)
/AMFINT = 19 (OUTPUT registered active_low)
;

BEGIN


IF(/MATCH * SELEN) THEN BEGIN
CASE(/CO[1:0])
BEGIN   #B00) /CO[1:0] := #B01;
#B01) /CO[1:0] := #B10;
#B10) /CO[1:0] := #B11;
#B11) /CO[1:0] := #B00;
END;
END;

ELSE /CO[1:0] := #B00;


IF(SELEN * AMC * /AMCP * MATCH * NRZ[1:0]) THEN BEGIN
CASE(/CNT[2:0])
BEGIN   #B000) /CNT[2:0] := #B001;
#B001) /CNT[2:0] := #B010;
#B010) /CNT[2:0] := #B011;
#B011) /CNT[2:0] := #B100;
#B100) /CNT[2:0] := #B101;
           #B101) /CNT[2:0] := #B110;
           #B110) /CNT[2:0] := #B111;
END;
END;
```

Figure 3-56

```
ELSE BEGIN
IF(SELEN * AMC * /AMCP * /MATCH) THEN
/CNT[2:0] := /CNT[2:0];

ELSE
/CNT[2:0] := #B000;

END;


AMCP := WG * AMC * AMF * /AMFI * /AMCP                    +  "ONE PULSE FOR WRITE"
          WG * AMC * AMCP * MATCH                         +  "SECOND PULSE"
SELEN * /WG * AMC * /CNT[2] * CNT[1] * /CNT[0]   +  "SET AFTER 00000 DETECT"
SELEN * /WG * AMCP * /MATCH;                           "HOLD UNTIL AM DETECT"


/AMF :=  WG * AMC                                         +  "WRITE AM"
          SELEN * /WG * AMC * AMFINT * /AMCP * MATCH      +  "READ AMF DELAY"
          SELEN * /WG * /AMF * AMC;                                                  "HOLD"


AMFINT := SELEN * /WG * AMCP * MATCH                +
AMFINT * AMC * AMF;

END.
```

Figure 3-56 Continued

CHAPTER 4

4.0 BOARD-LEVEL PRODUCTS

4.1 Am9580A/82 Disk Controller Board for the IBM-PC/AT

Objective of the Board

The disk controller board described below shows that it is possible to build a highly intelligent and compact controller board that supports ST506 as well as ESDI standard. Furthermore, the board also handles the IBM standard double-density floppy-disk format.

Two VLSI chips make it possible to keep the hardware design surprisingly compact for a board of such functionality and performance. Advanced Micro Devices' Hard Disk Controllers (HDC), Am9580A or Am9590, are highly integrated devices that off-load the CPU from handling the disk drives.

The key characterictics of the disk controller are:

- Supports ST506/412 and IBM double-density floppy-disk drives
- Controls up to four drives, any mix of interfaces
- Two on-chip sector buffers of up to 512 bytes, support zero-sector interleaving
- Error checking algorithms supported include
  - CRC/CCITT
  - Single-Burst Reed-Solomon
  - Double-Burst Reed-Solomon
  - External ECC (Error Correction Code)
- Linked-list command and data structure
- On-chip DMA controller supports 32-bit addressing, 8/16-bit data buses and Data Mapping

The difference between the Am9580A and the Am9590 is in the disk interface. As a functional subset of the Am9580A, the Am9590 supports the ESDI standard, in addition to ST506/412 and double-density floppy-disk drives. Both devices are fully hardware and software compatible.

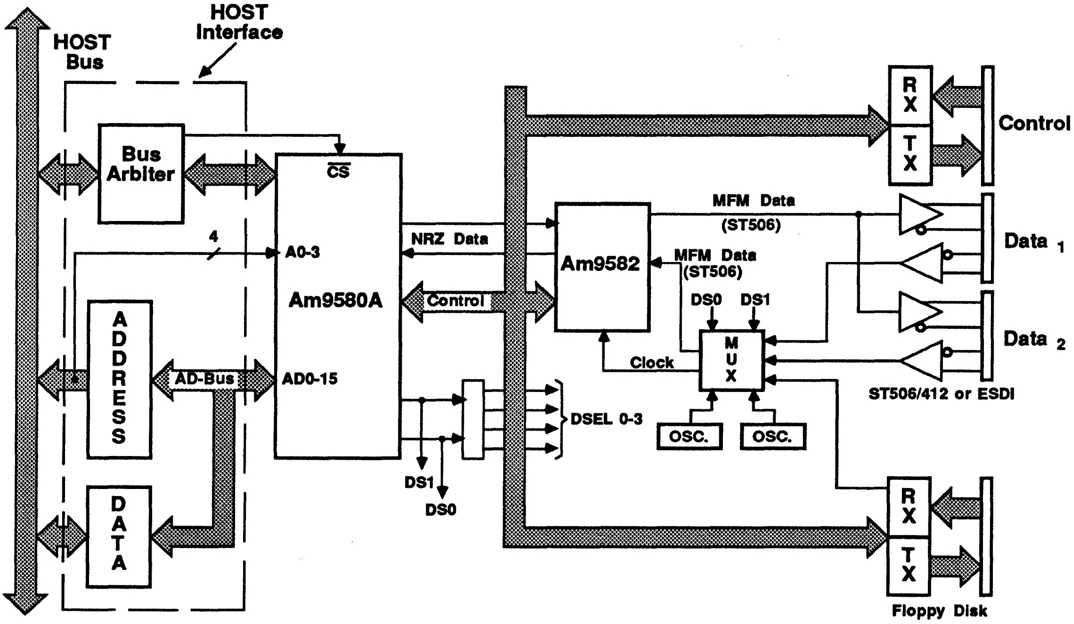
The companion part, Am9582, is a highly integrated Disk Data Separator (DDS). It requires a minimum number of external components. The frequency of the on-chip PLL (Phase-Locked Loop) can be changed dynamically. One DDS can support hard- as well as floppy-disks. Constant linear recording density can be achieved by changing the reference frequency as a function of the track number.

The IBM PC/AT has been chosen as a system interface example because it is the most popular microcomputer with a 16-bit data bus and an open architecture. Furthermore, it differs from the PC and XT, it allows other devices to request the host bus, which is a requirement of all high performance peripheral controllers. The bus exchange and decode logic described here can, however, be easily adapted to any other bus standard.

Hardware Description

The Am9580A/90 provides an independent host and disk interface. The host interface accesses the internal registers of the Am9580A/90 via I/O operations. For all command and data transfers, the HDC requests the host bus and performs the transfers with its internal DMA controller. Two PAL devices adapt the bus signals of the Am9580A/90 to the IBM PC/AT bus.

The disk interface can be configured in two different ways. One mode uses the Am9580A and Am9582 to support two ST506/412 hard-disk drives and two double-density floppy-disk drives (Figure 4-1). The clock frequency of the Am9582 is switched according to the requirements of the selected drive. The DDS generates and detects the address marks on either media. The only external hardware required are a few line drivers and a decoder to select the individual drives.



046JP-001

375-1

Figure 4-1

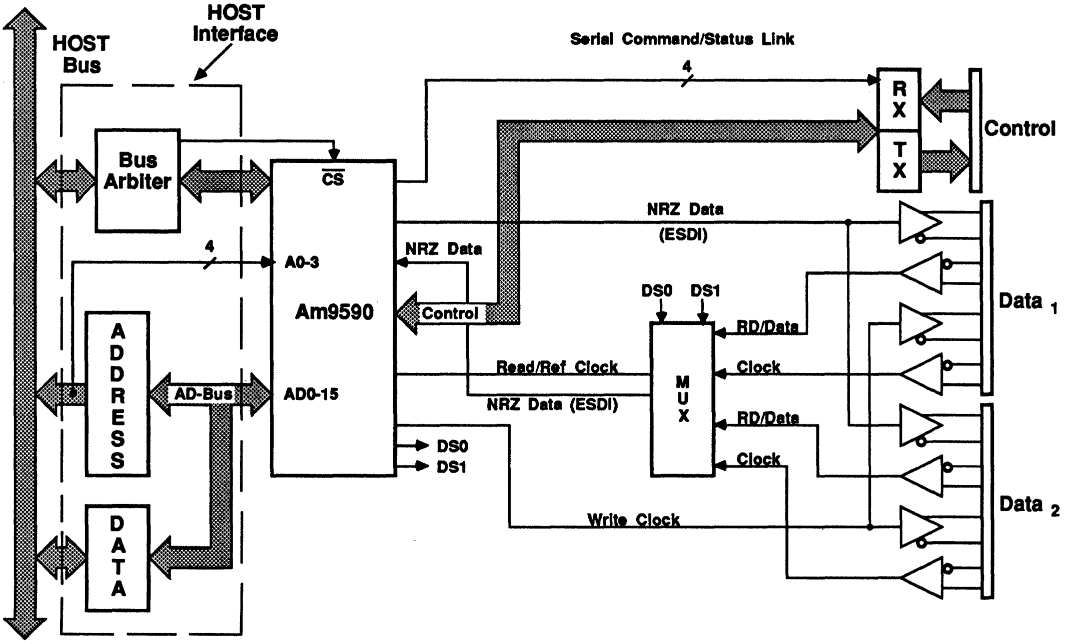


Figure 4-2

375-2

In the other configuration, the new Am9590 Hard Disk Controller replaces the Am9580A (Figure 4-2). The same board, without any hardware changes, is now able to also support the new ESDI interface. The Am9590 fully supports the ESDI standard for soft- and hard-sectored disk drives. The serial command/status link required by ESDI is implemented on-chip. The part automatically generates the ESDI commands (SEEK and RESTORE), necessary for normal disk operations. Therefore, the disk interface is fully transparent to the CPU. Once the initialization has been completed, the processor has no need to know the characteristics and the type of disk media it accesses.

The Am9590 also allows the user to send or receive any other command or status information under CPU control. Thus, the board is fully upward compatible for future ESDI features.

Software Description

Three software packages have been written for this disk controller board. A menu-driven developement software tool helps to debug the disk interface and to adapt different drives to the board. It is also a good tool to get familiar with the various high-level functions of the Am9580A/90.

The other two programs are application examples. A DOS-Driver (Ch.2.2) runs all major programs available for the IBM PC/AT. It is a typical example of an Am9580A/90 software interface to a micro/minicomputer operating system and is written in "C". A BIOS Driver (Ch.2.3) assures full compatibility with all application programs that directly call the ROM BIOS functions. This program is written in assembly language.

Performance

The disk controller described above offers a substantial improvement in speed and software interface compared to conventional controller boards. Zero-sector interleaving and on-chip DMA increase the performance to a point where it is only limited by the physical characteristics of the disk drive itself, its access time, and data rate. Furthermore, a high-level command structure helps to minimize the usually required CPU overhead.

Detailed Hardware Description

The Host Interface

The disk controller board allows two different modes to interface to the IBM-AT. In Slave mode, the host can access the internal registers of the Am9580A. This mode uses the AT IO-addresses from 100_H to 110_H. The Master mode allows the Am9580A to access the AT bus directly. It is used to transfer commands and data between the HDC and the host.

The interface between the Am9580A and the IBM-AT bus requires the HDC address-data bus (AD-bus) to be de-multiplexed. Three Am2956 latches generate the AT address in master mode. An Am2947 and a 74LS646 buffer the 16-bit data-bus of the IBM-AT. The 74LS646 in this application is used only as a bus driver; however, a small modification of the board would allow byte transfers. In the current configuration the board works with word accesses only.

The IBM-AT addresses SA0–SA9 are decoded in U24 (AmPAL16L8)(see Figure 4-6) to generate an Am9580A $\overline{\text{CS}}$ signal in slave mode. A0–A3 select the individual registers inside the HDC. Two PAL devices (U20 and U12) generate all the control signals in master-and slave modes (see Figure 4-7 and Figure 4-8 for the PAL equations). They also convert the two-wire bus exchange interface of the HDC into the three-wire handshake of the IBM-AT.

Jumpers on the controller boards configure the hardware to use different DMA channels and interrupts of the IBM-AT (see the following table).

Jumper	Signal
W10	DREQ7
W11	$\overline{\text{DACK7}}$
W12	DREQ6
W13	$\overline{\text{DACK6}}$
W14	DREQ5
W15	$\overline{\text{DACK5}}$
W16	IRQ14
W17	IRQ6

The default setup is $\overline{\text{DREQ7–DACK7}}$ for the DMA channel and no interrupt.

The Disk Interface

The disk interface of this board allows two configurations. The Am9580A, together with the Am9582, can support up to two hard-disk drives (ST506) and two floppy-disk drives. If the Am9582 is replaced by two PAL devices (U5 and U6), the board can support up to two ESDI hard-disk drives.

In the standard configuration (ST506 and floppy) the Am9582 does the encoding and decoding of the MFM data. One data separator serves both, the hard-disk and the floppy drives. The device is switched between the two modes using the DS1 (Drive Select 1) signal. Furthermore, the clock frequency of the DDS has to be switched between 5MHz (ST506) and 4MHz (Floppy mode 4MHz/16 = 250kHz). The PAL device U7 uses either the 5MHz clock input or the 8MHz clock input devided by two, as a device clock for the Am9582. The AmPAL22V10 also multiplexes the three MFM Read Data inputs (2 hard-disk, 1 floppy) for the DDS. Two line drivers (Am29828) and a 2 to 4 decoder (74LS139) for the individual drive select signals complete the disk interface.

The second configuration (Figure 4-2) of the board supports up to two ESDI hard-disk drives. The Am9582 is removed because the ESDI interface transfers NRZ data between drive and controller. The Am9580A needs only be replaced by the Am9590 to do ESDI.

Jumper description:

Jumper	Function	ST506	ESDI
W1	Analog V_{CC} DDS	Y	Y
W2	DIRIN	Y	N
W3	STEP	Y	N
W4	9580A/90	*	*
W5	Rd/Rdf. Clock	N	Y

* This Jumper was used to allow the Am9580A some extra logic to do ESDI. When using the Am9590 to do ESDI, this jumper is not used.

The Disk Connectors on the board have the following functions:

Connector	Function
J1	Hard Disk Control Cable
J2	Floppy Disk Cable
J3	Radial Cable HD Drive 2
J4	Radial Drive HD Cable 1

The hard-disk drives have to be set up for either DS1 (Hard-disk Drive Select 1) or DS2 (Hard-disk Drive Select 2). This corresponds to Drive 0 and Drive 1 for the HDC parameters. Floppy disk drives have to be configured for DS0 (Floppy-disk Drive Select 0) or DS1 (Floppy-disk Drive Select 1). This corresponds to Drive 2 and Drive 3 for the HDC parameters.

4.2 SCSI Board

Introduction

The Small Computer System Interface (SCSI) is an increasingly popular peripheral communication standard defined by the ANSI X3T9.2 Committee. This standard provides a sophisticated approach to interfacing host systems to intelligent peripheral devices and can significantly improve the overall performance of a computer system.

Although a SCSI device driver is readily available for most commercial operating systems, the most popular hard-disk interface today is either the simple ST506 interface or the high performance ESDI interface. A bridge controller is required to convert between the device-level protocols and the system-level SCSI protocol. Furthermore, a bridge controller allows disk control electronics to be shared by multiple disk drives, and is more cost-effective than providing built-in SCSI electronics for each individual drive.

The availability of off-the-shelf VLSI devices has made it possible to construct a high performance SCSI disk formatter that addresses the need of a new generation of computer systems; yet small enough to be mounted on a 3 1/2" micro hard-disk drive. Further performance enhancement is realized through an on-board 256 kbyte disk cache and hardware features that can drastically reduce file transfer time and system latency.

Objective of the SCSI Disk Formatter Board

The SCSI disk formatter board serves as an example of a highly integrated design that uses state-of-the-art VLSI components to their full advantage. With these VLSI devices, a high-performance application-specific board can be built; even when most of the control software is written in a high-level programming language such as C.

The features of the SCSI board are:

- A 10 MHz 80188 CPU that provides the computing power for SCSI command interpretation, cache memory management, and control of the Am9590 disk controller.
- An Am9590 disk controller that interfaces to two hard-disk drives. Any combination of ST506 /ST412 or ESDI hard-disks are allowed. The Am9590 implements all the control functions required by these industry-standard interfaces and can be customized for different data formats. On-chip dual data buffers permit zero-sector interleaving.
- The Am9582 provides all the data separator functions such as data encoding, decoding, and address mark generation and detection for the ST506 interface.
- The SCSI interface is realized with the Am5380 SCSI controller. It provides hardware support for handling the SCSI protocol and allows an asynchronous data transfer rate of up to 1.25 Mbytes per second for the board.
- 256 Kbytes of dynamic RAM with parity check provide the disk cache memory as well as memory storage required by the operating system. An EPROM of up to 64 kbytes carries the control program for the board.

General Hardware Description

The SCSI board can be logically partitioned into three functional sections (Figure 4-3):

1. The Dynamic RAM (DRAM) interface
2. The SCSI port interface
3. The disk control interface

The activities of these three functional sections are coordinated by the 80188, a highly integrated microprocessor with built-in DMA channels, programmable timers, and an interrupt controller.

The 10 MHz system timing is provided by the CLKOUT signal of the 80188 microprocessor. It drives Am9590 disk controller clock input and is divided by two to provide the clock frequency for the ST506 interface.

The system memory is comprised of a program memory of 64 kbytes in EPROM and a data memory of 256 kbytes in DRAM. The contents of the DRAM are refreshed continuously by the 80188 DMA Channel 0. The integrity of the DRAM storage is maintained through the use of the Am29833 parity transceiver that alerts the CPU of a memory error during a data memory read operation. The DRAM interface consists of two address latches and a programmable logic device that generates the $\overline{RAS}$ and $\overline{CAS}$ control signals to the DRAMs.

SCSI Disk Formatter Board

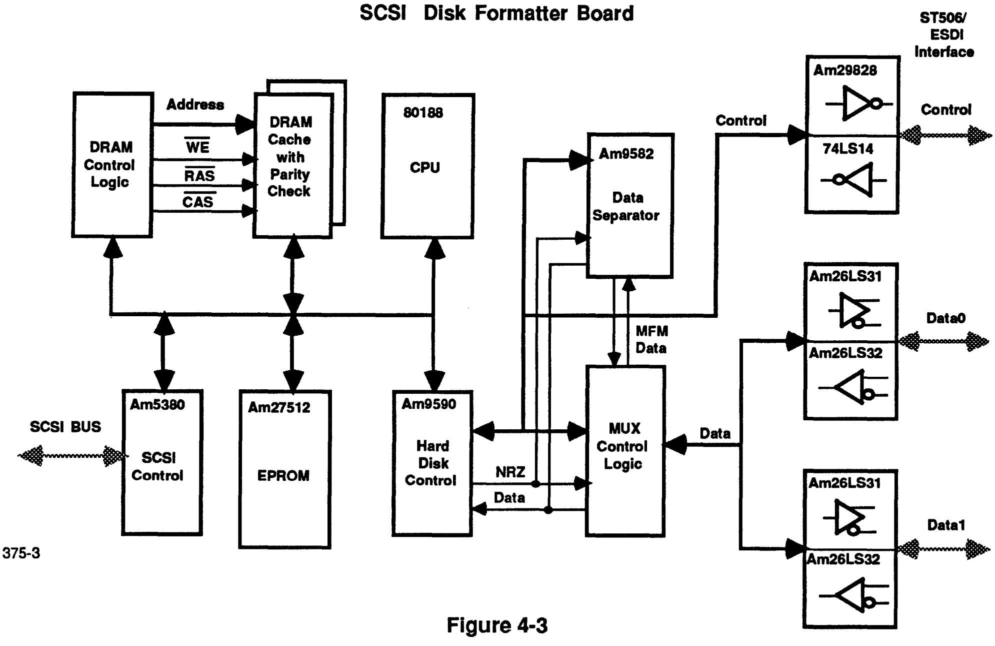


Figure 4-3

The SCSI interface lines are connected directly to the Am5380 SCSI controller. This device has built-in 48 mA open-drain drivers required for the SCSI protocol. The CPU addresses the internal registers of the Am5380 by activating the $\overline{\text{PCS1}}$ line. Data transfers over the SCSI bus are controlled by the 80188 DMA channel 1. When there is a SCSI bus condition that needs the attention of the CPU, the Am5380 will activate the 80188 INT1 line.

The Am9590 disk controller is selected by the 80188 $\overline{\text{PCS0}}$ line. The B/$\overline{\text{W}}$ input is strapped HIGH to configure the Am9590 in the 8-bit interface mode. The Am9590 will output the A16–A31 value on its AD0–AD15 lines with the ALEN signal enabled and will only update these upper addresses when necessary; therefore, a programmable logic device is used to provide the current A16 and A17 values to the DRAM interface when the Am9590 is accessing the DRAMs.

The board is designed to interface to any combination of ST506 or ESDI hard-disk drives using the same connectors. The software senses the jumper settings and configures the drive parameter blocks accordingly. If the selected drive is an ST506 hard-disk drive, the Am9590 receives the Read Data and its Reference Clock signals from the Am9582 DDS. If the selected drive is an ESDI disk drive, the Read Data and its Reference Clock signals is supplied directly through the ESDI data interface. An Am29828 buffer provides the 48 mA drivers for the disk control signals.

An optional Am9582 provides the ST506 support for the board. The Am9582 receives the Write Data from the Am9590 disk controller and formats it as MFM data for the hard-disk drive. When reading data from the disk, the Am9582 retrieve the NRZ data and its Reference Clock from the serial MFM Read Data stream. The Am9582 also provides Address Mark generation and detection functions and the Precompensation function for the MFM Write Data.

General Software Description

The software for the SCSI board can be divided into three functional blocks:

1. SCSI protocol control
2. Am9590 control
3. Disk cache management

Only a small portion of the software is written in assembly language, the majority of the program is written in C. This makes the code easy to understand and modify, so that performance enhancements can be made simply by replacing the various software modules.

The SCSI protocol control software initializes the Am5380 for responding to selection by an initiator. It manages receiving of command bytes from the initiator and initializes DMA Channel 1 for the SCSI data transfer phase. It is also responsible for sending out the status information after a data transfer.

The software is developed for a small system environment similar to the IBM PC-DOS or the Apple Macintosh operating system. The system will wait until the requested sectors of the hard-disk is in the memory before proceeding with another process; therefore, reselection is not supported in this software. The SCSI target, once selected, will continue to control the SCSI bus until the transaction is finished.

The interrupt output of the Am5380 is connected to the 80188 INT0 input. Interrupt service routines support various time-critical activities at the SCSI port such as selection, bus reset, and bus error handling.

The SCSI Common Command Set (CCS) is supported by the software. This allows any host adapter that implements the CCS to communicate with the board. In the case of formatting a ST506 disk drive, extension to the CCS were necessary to pass the drive parameters to the board.

Since the Am9590 accesses memory through its on-chip DMA controller; memory areas have to be allocated for its Input/Output Parameter Blocks (IOPBs), Data Map entries, and Status Result Blocks (SRBs). The software control scheme allows SCSI bus operation, CPU operation, and Am9590 operation to occur simultaneously. The software translates the SCSI commands into the corresponding IOPBs for the Am9590 and puts them in the allocated IOPB area before issuing commands to the Am9590 to initiate the disk access operation. Only one IOPB will be assembled each time because commands are not chained together.

The cache management scheme for the SCSI target is simple. The intention is to provide the benefit of a disk cache without assuming any special feature or any specific environment that a disk drive may have. A user can significantly boost the performance of the SCSI board simply by replacing the cacheing scheme with one suitable for a particular environment.

The cache management software takes advantage of the zero-sector interleaving capability of the Am9590. During a multi-record access, after the Am9590 has accessed the last sector of a track, it will automatically switch to the next head to access the next record. This is called a "head first" access rather than a "track first" access. In this way the drive controller is able to fetch large amount of sequential data from the disk efficiently.

Caching is performed on a track-by-track basis and look-ahead read is performed automatically for any read from the disk. Therefore, significant performance increase results in applications that do not involve a great deal of random data base accesses. Such applications include most personal computer software and time-shared multiuser systems where performance is dependent on the speed of sequential file accesses and repeated access to disk allocation tables and directory information.

The software also makes use of the sector pulse generated by hardware to determine if the first requested sector has been fetched by the Am9590. This allows the software to set up the Am5380 to start transferring data to the initiator, before the total number of logical blocks is read by the Am9590. This drastically reduces the data transaction latency.

A typical application, such as text editing or program compilation will require approximately eight buffers. With caching, based on a whole track and the "look ahead" reading, the software only needs to manage a relatively small number of track buffers to provide a drastic boost in performance for a personal computer environment.

Design Considerations

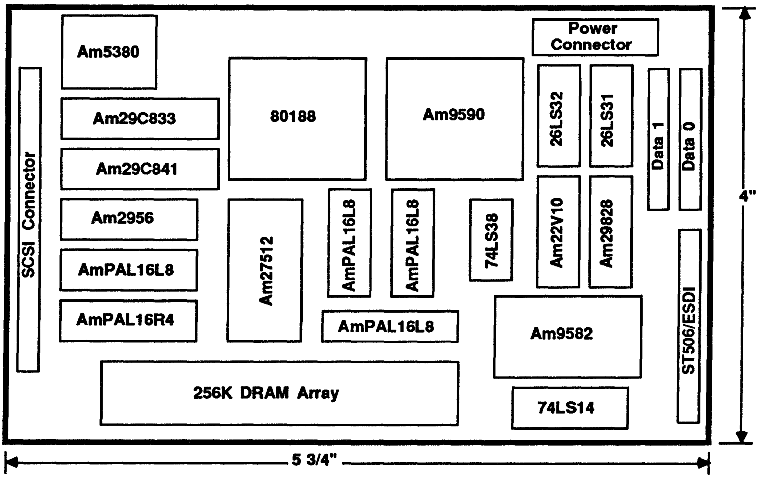
The design goal for the SCSI board is to provide a low-cost, high-performance system with off-the-shelf components in a small form-factor. The use of various types of AMD programmable logic devices contributed to the small chip count required for the board (Figure 4-4).

The design is based on an 8-bit system bus because of the board space constraint. The DRAMs used are the 256k by 1 bit ZIP type DRAMs because of the small package size. Also an 8-bit design reduces the number of transceivers and buffers required for the data bus.

Performance studies shows that, since the SCSI bus is byte oriented, the extension of the system bus to 16-bit does not improve performance enough to justify the added cost and board space. If the application requires more cache memory and faster internal data rate, the design can easily be modified to a 16-bit system by using the 80186 with the Am9590 strapped in Word Mode. However, the board space constraint will require most DIP packages to be replaced by PLCC or other surface-mounted packages; cost-effective only in a mass production environment.

Although the Am9590's interrupt signal is connected to the 80188, the software does not provide service interrupt for the disk controller. The CPU polls the Am9590 status register while running in a loop that coordinates the various board activities. This scheme simplifies the control logic for the board.

The Am9590 is designed to control up to four disk drives. Because of board space constraint, only



375-4

Figure 4-4

057HDC-003

two data cable connectors are put on the board. The only modifications that would be required to control the extra drives are the multiplexers, drivers, and receivers for the additional channels.

Since the Am9590 and the Am9582 are also designed to support floppy disk drives, a multifunction disk controller board can be implemented with little added logic. Floppy disk is not supported by the board because of the extra connector requirement and that the SCSI commands for floppy disk drives have not been standardized.

The DMA controller on the 80188 is able to do a SCSI data read at a source-synchronous rate of 1.25 Mbytes/s; however, writing data to the SCSI port can only be done at 0.8 Mbytes/s because of the limitation of the 80188 source-synchronous DMA transfer. Hardware is provided on board to allow the Am5380 to perform a Block Mode DMA transfer; this way, a SCSI data write operation can be sustained at 1.25 Mbytes/s, using the 80188's string-move operation instead of a DMA write.

User's Guide for the SCSI Disk Formatter

Several jumpers are located on the SCSI board to allow for different configurations. Since the software will read the configurations upon system reset, these jumpers settings should not be modified while the SCSI board is in operation. A system reset is automatically initiated upon power-up, a miniature push-button switch (SW1) on the board allows the user to reset the system after power-up. The descriptions for the jumper function are as follows:

Jumper	Description
W1	Jumper in: Drive 0 = ESDI; out: ST506
W2	Jumper in: Drive 1 = ESDI; out: ST506
W3	Reserved, no jumper inserted
W4	Reserved, no jumper inserted
W5	SCSI Target ID bit 0: Jumper in =1; out = 0
W6	SCSI Target ID bit 1: Jumper in =1; out = 0
W7	SCSI Target ID bit 2: Jumper in =1; out = 0
W8	Jumper in: SCSI Parity Enable; out: disable
W9	Jumper in: Drive 1 = ESDI
W10	Jumper in: Drive 1 = ESDI
W11	Jumper in: Drive 1 = ESDI
W12	Jumper in: Drive 0 = ESDI
W13	Jumper in: Drive 0 = ESDI
W14	Jumper in: Drive 0 = ESDI

Upon system reset, the control software read the values of W1 and W2 to set up the default drive parameter blocks for Drive 0 and Drive 1. The ID bits determine which SCSI selection the SCSI board will respond to, and the Parity Enable bit determines if the Am5380 will generate and check parity on the SCSI bus.

The two drives that can be connected to the SCSI board share the same control signals on connector J2. The drive with its data cable connected to J4 is referred to as Drive 0 (DR0), and the drive with its data cable connected to J3 is referred to as Drive 1 (DR1). The SCSI protocol will refer to DR0 and DR1 as logical unit numbers (LUN) 0 and 1, respectively (Figure 4-5).

The Data Cable Jumpers W9 through W14 are used to prevent contention between an ESDI drive and an ST506 drive. Some ST506 drives in the market use pins 4, 7, and 8 on the data cable instead of leaving them open. Since the same cable is used for ESDI as well as ST506 drives, the connection of these jumpers can cause undesirable signals to be driven into the ESDI interface logic, and may cause the Am9590 to report a fault condition. Jumpers W9 throught W14 should be installed for ST506 drives and left out for ESDI drives. The following table shows the allowable combinations.

The recommended installation for these Data Cable Jumpers are:

DR0	DR1	Install Jumpers
ST506	ST506	None
ST506	ESDI	W12, W13, W14
ESDI	ST506	W9, W10, W11
ESDI	ESDI	W9, W10, W11, W12, W13, W14

There are four double-row header connectors on the SCSI board. The functions of these connectors are:

Connector	Function
J1	50-pin SCSI port
J2	34-pin ESDI/ST506 Control Cable
J3	20-pin Data Cable for Drive 1
J4	20-pin Data Cable for Drive 0

The 50-pin connector J1 is a standard single-ended SCSI port that is terminated by the SIP type resistor packages (RN1, RN2, and RN3). If the SCSI cable is already fully terminated externally, RN1 and RN3 should be unplugged from the board and pin 2 and 4 of RN2 should be removed to avoid overloading the SCSI bus.

The SCSI single-ended cable signal pin assignments are shown in the following:

Pin Number	SCSI Signal
2	$\overline{\text{DB0}}$
4	$\overline{\text{DB1}}$
6	$\overline{\text{DB2}}$
8	$\overline{\text{DB3}}$
10	$\overline{\text{DB4}}$
12	$\overline{\text{DB5}}$
14	$\overline{\text{DB6}}$
16	$\overline{\text{DB7}}$
18	$\overline{\text{DBP}}$
20	GND
22	GND
24	GND
26	No Connection
28	GND
30	GND
32	$\overline{\text{ATN}}$
34	GND
36	$\overline{\text{BSY}}$
38	$\overline{\text{ACK}}$
40	$\overline{\text{RST}}$
42	$\overline{\text{MSG}}$
44	$\overline{\text{SEL}}$
46	$\overline{\text{C/D}}$
48	$\overline{\text{REQ}}$
50	$\overline{\text{I/O}}$

All odd pins, except pin 25, on the SCSI connector J1 are connected to ground. Pin 25 is not used by the SCSI board. The standard SCSI cable connection is to terminate the active SCSI signals externally with 220 ohms to +5V and 330 ohms to ground at the initiator end of the cable, with RN1, RN2, and RN3 installed on the board.

The same 34-pin connector J2 is used for ESDI or ST506 disk control signals. All odd pins on the connector are connected to ground. The pin assignments for the drive control cable are shown in the following:

Pin Number	Disk Control Signal
2	$\overline{\text{Reduced Write Current}}$
4	$\overline{\text{Head Select 2}}$
6	$\overline{\text{Write Gate}}$
8	$\overline{\text{Seek Complete}}$
10	$\overline{\text{Track 0}}$
12	$\overline{\text{Write Fault}}$
14	$\overline{\text{Head Select 0}}$
16	$\overline{\text{ESDI Sector}}$
18	$\overline{\text{Head Select 1}}$
20	$\overline{\text{Index}}$
22	$\overline{\text{Ready}}$
24	$\overline{\text{Step}}$
26	$\overline{\text{Drive Select}}$
28	$\overline{\text{Drive Select 2}}$
30	No Connection
32	No Connection
34	$\overline{\text{Direction In}}$

J3 and J4 are the 20-pin connectors for the data cables for the two disk drives that can be connected to the SCSI board. J4 is used for Drive 0 (DR0) and J3 is used for Drive 1 (DR1). Because the pin assignments for the ESDI and ST506 data cable are slightly different, jumpers W9 through W14 are provided to prevent signal contentions. Some signals are defined only for the ESDI drives. The pin assignments for these connectors are:

Pin Number	Data Cable Signals
1	$\overline{\text{Drive Selected}}$
2	No Connection
3	$\overline{\text{Command Completed}}$ (ESDI)
4	$\overline{\text{Address Mark}}$ (ESDI)
5	No Connection
6	GND
7	+ Write Clock (ESDI)
8	– Write Clock (ESDI)
9	No Connection
10	+ Reference Clock
11	– Reference Clock
12	GND
13	+ Write Data
14	– Write Data
15	GND
16	GND
17	+ Read Data (MFM ST 506) (NR2 ESDI)
18	– Read Data (MFM ST 506) (NR2 ESDI)
19	GND
20	No Connection

P1 is a 4-pin power connector that can be hooked up to the same cable that supplies current to the disk drive. The SCSI board only requires a +5 V power supply; the +12 V power input is not used. Pin 3 is the only ground return line for the board. The board requires about 2 Amps on the +5 V supply. The pin assignments for P1 are:

Pin Number	Voltage
1	Not Used: can be connected to +12 V
2	Not Used: can be connected to GND
3	GND
4	+5 V

System Interface

The 80188 high-integration 8-bit microprocessor (U3) provides the functions that are necessary to control the hardware resources on the SCSI board (Figure 4-5). The 80188 features that are utilized on the board are:

- An enhanced 10 MHz 8088-2 CPU for the computing power required for the board.
- A clock generator that provides the 10 MHz system clock for the Am9590 (U4) and the DRAM control logic. The 5 MHz ST506 frequency for the Am9582 (U18) is derived from the system clock.
- Timer 0 is used to provide a real-time clock for the software, useful for computing time-out intervals.
- Timer 1 is a counter for sector pulses when data is read from the hard disk. By reading this counter, the software can determine the number of sectors being transferred by the Am9590. This reduces the latency time for a SCSI bus transaction.
- Timer 2 is used to generate refresh requests to DMA Channel 0. A DMA data transfer will be initiated by the 80188 when a DMA request is received. The timer is initialized to provide the proper refresh time interval for the DRAM.
- DMA Channel 0 is used to refresh the DRAM (U19–U27). Each DMA cycle refreshes two row addresses of the 256 kbyte DRAMs.
- DMA Channel 1 allows DMA transfer between the Am5380 (U1) and the DRAM.
- The programmable Interrupt Controller also handles interrupt requests for memory error detection and SCSI bus conditions that require the attention of the CPU. It will also service the DMA 0 interrupt, which allows the DMA refresh procedure to be reinitialized.
- The Chip-Select unit activates the proper selection lines for the EPROM, DRAMs, Am9590, and the Am5380. Some select lines are also used to activate hardware on the board.

The 80188 is driven by a 20 MHz crystal (Y1) connected between its X1 and X2 inputs. This frequency is divided by the internal Clock Generator to provide the synchronous 10 MHz CLKOUT output for the system.

When the board is powered-up, or when the system reset button (SW1) is activated, the $\overline{\text{RES}}$ input to the 80188 is driven LOW momentarily. This resets the 80188 and causes the RESET output of the 80188 to go HIGH for an integral number of system clock cycles corresponding to $\overline{\text{RES}}$. The RESET output will generate a system

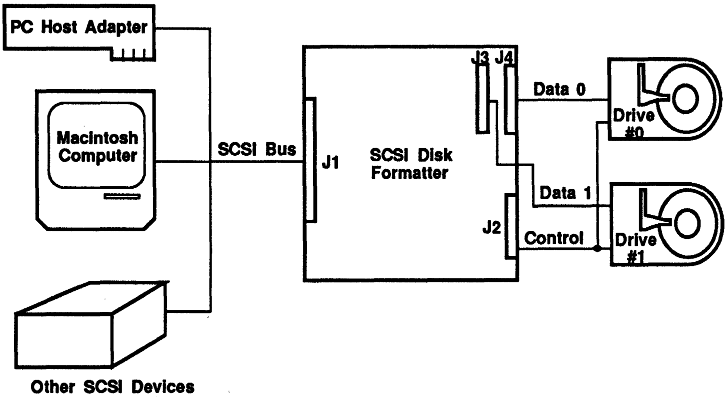


Figure 4-5

reset which resets the Am9590 and the Am5380 and clears the parity flag of the Am29833 Parity Transceiver (U2).

The firmware for the 80188 is contained in an Am27512 EPROM (U9). The EPROM contents are enabled on the data bus only when the $\overline{\text{UCS}}$ and $\overline{\text{RD}}$ lines of the 80188 are activated. Upon reset, the CPU starts the execution of instructions at address 0FFFF0H, which will cause a jump to the beginning of the EPROM space at location 0F0000H. The EPROM occupies a memory address from 0F0000H through 0FFFFFH. The control program at the present time requires less than 32 kbytes, therefore plenty of program space is left for development of enchanced features.

The system RAM storage is comprised of nine 256k by 1 bit ZIP type DRAMs (U19–U27), providing 256 kbytes of system memory with parity check. The ZIP type package is chosen because of its small form factor. To provide a zero wait state data transfer, the DRAM's must have an access time of 150 ns or less.

Parity checking is important for the cache memory because the DRAM contents may be retained for an indefinite period of time, kept refreshed continuously by the DMA Channel 0 of the 80188.

Parity checking will prevent corrupted data caused by soft errors on the DRAMs to be transferred to the Initiator. The Am29833 Parity Transceiver (U2) generates or checks parity on-the-fly during a DRAM access. It also provides buffering between the data bus and the DRAMs.

The DRAMs occupy an address area from 0 to 3FFFFH, activated by the $\overline{\text{LCS}}$ signal of the 80188. The address signals output by the 80188 is latched inside U7 (Am29C841) and U8 (Am2956) when the ALE signal becomes active. Since 18 bits of address lines are necessary to access all the DRAM memory locations, these address bits must be multiplexed into a 9-bit row address and a 9-bit column address for the DRAMs.

The DRAM address control logic is controlled by U16 (Am16R4) which generates the Row Address Strobe ($\overline{\text{RAS}}$) and Column Address Strobe ($\overline{\text{CAS}}$) inputs to the DRAMs. The row addresses latched in U16 are provided to the DRAMs when $\overline{\text{RAS}}$ is active; the column addresses latched in U14 are provided to the DRAMs when $\overline{\text{CAS}}$ is active. U16, together with U12 (AmPAL16L8), also generates the control signals for the Am29833 parity transceiver.

The Am9590 (U4) has a 16-bit multiplexed address/data bus (AD0–AD15). The Am9590 can access a 32-bit physical memory space. During a memory access by the Am9590, the address lines will contain the lower 16 bits of the physical address when ALE is active and the upper 16 bits of the physical address when ALEN is active. The upper address bits are only put out by the Am9590 when it is necessary to access a memory area outside the 64 kbyte window, currently addressable by the 16-bit address lines. These upper address bits are expected to be latched externally when ALEN is active.

Since the Am9590 needs only access 256 kbytes of data, only the upper address signals, A16 and A17, need to be latched when ALEN is active. U15 (AmPAL16L8) is used to latch the A16 and A17 signals put out either by the 80188 or the Am9590. These latched address bits are provided to the DRAMs when $\overline{\text{CAS}}$ is active.

A DMA data transfer for the 80188 consists a fetch cycle and a deposit cycle. Since DMA Channel 0 is used to refresh the DRAMs, the deposit cycle that initiate a write to memory must be suppressed. Instead, logic implemented in U10 and U16 are used to provide the DRAMs with $\overline{\text{RAS}}$-only accesses on two row addresses when a DMA Channel 0 byte transfer is initiated. In this way, the number of DMA cycles per second required for refreshing the DRAMs will be reduced, allowing most of the system bus bandwidth to be used by the CPU or the Am9590.

The Am29833 parity transceiver (U2) generates the parity check bit during a write to the DRAMs and check the returned data during a read from the DRAMs. All DRAM locations are written once upon system reset to guarantee valid parity bits in DRAMs on power-up. The detection of a parity error will activate the NMI input of the 80188, so that the software can inspect the data content or load the data again from the disk.

The Am29833 has to generate a non-maskable interrupt on memory error because DMA activities have higher priority than the Interrupt Controller on the 80188.

The ARDY input to the 80188 is controlled by U10 (AmPAL16L8). This device provides the 80188 with the Am9590 ready signal while the CPU access the internal registers of the disk controller in Slave Mode. During Am5380 Block Mode DMA transfer, the device will provide the 80188 with the Am5380 ready signal in order to control the transfer rate.

The system bus on the board is comprised of the 16-bit Address/Data lines, the latch enable signals, the read/write control signals, and the data control signals. There are five different types of access to the system bus:

1. A DMA refresh cycle is initiated by a DMA Channel 0 transfer. This consists of two $\overline{\text{RAS}}$ only access to the DRAMs. Two row addresses, separated by 256 rows, are refreshed once every 4 ms required by the DRAMs. Since each DMA read or write operation requires four system clock cycles, the total memory bus bandwidth that will be used for memory refreshing will only be approximately 3%. There is a small additional overhead associated with the execution of the interrupt service routine.

2. The CPU access to the Am5380 registers. The 80188 enables the $\overline{\text{PCS1}}$ chip select line and the register address is provided by the Am29C841 latch. No wait state is necessary to read or write to the registers.

3. The DMA access to the Am5380 data register. The 80188 activates the $\overline{\text{PCS2}}$ chip select line in response to a DMA request by the Am5380. This causes the $\overline{\text{DACK}}$ input of the Am5380, controlled by U12, to be asserted. No wait state is necessary for a DMA transfer to the SCSI bus.

4. The CPU access to the Am9590 registers. The Am80188 activates the $\overline{\text{PCS0}}$ chip select line to access the Am9590 registers. The register addresses are provided by the Am29C841 latch. Since the Slave Mode access for the Am9590 can vary between one to sixteen system clock cycles, the $\overline{\text{READY}}$ output of the disk controller is used to control the access time for the registers.

5. The DMA transfer between the Am9590 and the DRAMs. When the Am9590 is ready to start a data transfer, it asserts the BREQ signal that is tied to the HOLD input of the 80188. In response, the 80188 gives up control of the system bus and asserts its HLDA line that is connected to the Am9590's BACK input. The Am9590 takes control of the system bus after it receives the BACK signal. The memory access is the same as an 80188 access; no wait states are required.

SCSI Interface

The SCSI bus interface is provided by the Am5380 SCSI protocol controller (U1). This device has on-chip 48 mA open-drain drivers and is designed to be connected directly to all the SCSI bus signals. The Am5380 is controlled by reading and writing to several internal registers that are addressed as I/O Ports by the 80188.

The Am5380 has built-in hardware that supports the SCSI protocol such as arbitration, parity generation and verification, and SCSI bus reset. The Am5380 will interrupt the 80188 when it detects a SCSI bus condition that requires CPU's attention; the control software will then identify the cause of the interrupt and service it. The SCSI board supports data transfer between the Am5380 and the memory bus in three methods: programmed I/O, normal DMA, and Block Mode DMA.

The 80188 accesses the internal registers of the Am5380 by asserting the $\overline{\text{PCS1}}$ chip select line. The register addresses A0–A2 are provided by the Am29C841 latch (U7). The data lines D0–D7 of the Am5380 are connected directly to the system data bus, but the $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$ lines to the Am5380 are controlled by U10 (Am16L8), which is used to qualify the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals coming from the 80188. In this way, we can set the Am5380 in Block Mode DMA transfer mode, and use the 80188 string-move instruction to transfer data.

The Am5380 registers are mapped into the I/O Port addresses starting at 80H. Bits A3–A6 of the address lines are ignored by the Am5380. The following shows the port addresses of the Am5380 registers:

I/O Port Address	Am5380 Register on Reading/Writing
80	Current SCSI Data/Output Data
81	Initiator Command
82	Mode
83	Target Command
84	Current SCSI Bus Status/Select Enable
85	Bus and Status/Start DMA Send
86	Input Data/Start DMA Target Receive
87	Reset Parity/Interrupt

Upon system reset, the 80188 RESET signal will become HIGH for an integral number of system clock cycles. However, the $\overline{\text{RESET}}$ input of the Am5380 is active LOW, therefore, a Schmitt-trigger inverter (U28) is used to invert the RESET signal to the Am5380. A reset on the Am5380 clears all its internal registers.

The Am5380 supports all SCSI bus signals. The SCSI bus is terminated at the board by the SIP-type resistor packages, RN1, RN2, and RN3. The SCSI $\overline{\text{RST}}$ signal must be terminated, otherwise the Am5380 will continuously generate SCSI bus reset interrupts.

The configuration switches, W1–W8, are buffered by U11 (AmPAL16L8), which enables the logic values onto the data bus when the 80188 reads from I/O Port address 200H. The returned value is used by the control program to initialize the various parameters for the configuration of the SCSI board.

Upon system reset, the Am5380 is initialized to be used as a target SCSI device. The software also reads the configuration jumpers, W5, W6, and W7 to determine the SCSI Identification Number (ID) that the SCSI board corresponds to and initialize the Am5380 Select Enable Register accordingly. The software also reads W7 to determine if SCSI parity should be supported by the Am5380.

When normal mode DMA is used for SCSI transfer, the Am5380 will be programmed to assert its DRQ output whenever a byte is ready to be transferred over the SCSI bus. The 80188 will also be set up such that a DMA byte transfer will be initiated when DRQ goes active and the SCSI data will be transferred through the I/O Port address 100H.

Whenever the 80188 accesses I/O Port address 100H, the $\overline{\text{PCS2}}$ chip select line connected to the Am5380 DMA acknowledge input ($\overline{\text{DACK}}$) of the Am5380 is activated. The $\overline{\text{DACK}}$ line resets the DRQ signal and selects the Am5380 Data Register for data transfers. The End of Process ($\overline{\text{EOP}}$) signal on the Am5380 is not used and is tied HIGH, therefore, the 80188 has to reset the DMA Mode bit of the Am5380 Mode Register to terminate a DMA transfer. The DMA Channel 1 of the 80188 is used for DMA transfers between the memory and the SCSI bus.

For a DMA Write operation of N bytes to the SCSI bus, the software first sets up the 80188 for a DMA transfer of N bytes from memory to the I/O Port Address 100H. The software then writes to the Start DMA Send Register to initiate the transfer. The Am5380 will then transfer N bytes of data to the SCSI host. However, the Am5380 is still in the DMA mode at this point. Therefore when the byte counter of the 80188 DMA controller reaches zero, the software will wait until the DMA Request bit of the Am5380 Bus and Status Register becomes a one, indicating the the the last byte is transferred. When this occurs, the software then resets the DMA Mode bit of the Am5380 Mode Register to terminate the data transfer process.

For a DMA Read operation of N bytes from the SCSI bus, the software first sets up the 80188 for a DMA transfer of (N-1) bytes from the I/O Port address 100H to memory. The software then writes to the Start DMA Receive Register to initiate the transfer. When the DMA byte counter of the 80188 reaches zero, DMA will stop; the software will then wait until the DMA Request bit of the Am5380 Bus and Status Register becomes a one, indicating the last byte is accepted by the Am5380. When this occurs, the software then resets the DMA Mode bit of the Am5380 Mode Register to terminate the DMA process. The last byte of the read operation is then obtained by reading the Am5380 Input Data Register and transferring the content to the memory location, pointed to by the 80188's Destination Pointer..

The hardware on the board also supports Am5380 Block Mode DMA operation. The software first sets the Block Mode and the DMA Mode bits of the Am5380 Mode Register. The software then waits until the DMA Request bit of the Am5380 becomes a one. When this occurs, the software then writes a one to the memory address location 60000H to assert $\overline{\text{DACK}}$. Data transfer can now proceed using the 80188 String Move instructions. The rate of transfer is controlled by the Am5380 READY signal. When the transfer is done, the software needs to write a zero to address location 60000H, in addition to resetting the Block Mode and DMA Mode bits of the Am5380 Mode Register.

Disk Drive Interface

The disk drive interface is comprised of the Am9590 disk controller (U4), the Am9582 disk data separator (U18), and the drive interface logic. Two hard-disk drives, in either the ST506 or ESDI format, can be connected to the SCSI board. The Am9582 is optional and is required only if a ST506 disk drive is attached to the board.

The Am9590 provides the hard-disk control functions for the SCSI board. The B/$\overline{\text{W}}$ input of the Am9590 is strapped HIGH so that the device operates in Byte Mode. All Address/Data lines and data control signals are directly connected to the system bus. When the 80188 HLDA output is HIGH, the Am9590 takes full control over the system bus.

When the 80188 asserts the $\overline{\text{PCS0}}$ chip select line, the Am9590 operates in the Slave Mode. In this mode, the CPU can read or write to the eight 16-bit internal registers. Writing the Am9590 Command Register causes a hard-disk control operation.

The Am9590 internal registers are selected by addresses A0–A3 provided by the Am29C841 latch. Address line A0 is used to select the High or Low byte of a 16-bit register. Since the bus transfer time for the Am9590 can take as long as 16 system cycles, the READY output of the device is used to generate the 80188 Asynchronous Ready (ARDY) input to control the data transfer rate.

The Am9590 registers are mapped into the I/O Port addresses starting at 0. Bits A3–A6 of the address lines are ignored by the Am9590. All internal registers are 16 bits long. Since the Am9590 can address a 32-bit memory space, two registers are required to address the Low and High word of a 32-bit pointer. The following shows the I/O Port addresses of the Am9590 registers:

I/O Port Address	**Am9590 Registers Reading/Writing**
0	Status/Command
2	Mode
4	Next Block Pointer (Low word)
6	Next Block Pointer (High word)
8	Status Result Pointer (Low word)
A	Status Result Pointer (High word)
C	Status Result Length

The Am9590 clock input is provided by the 80188 CLKOUT line. The 80188 RESET output is connected to the Am9590 RESET input. Upon system reset, the Am9590 will perform an internal initialization procedure and allows its READY output to go inactive. This causes the CPU to wait until the initialization is completed and the READY signal becomes LOW, before accessing the content of any Am9590 registers.

When the Am9590 needs to access the system memory, it asserts its BREQ output which is connected to the 80188 HOLD input. The 80188 will then release all the system bus signals and assert its HLDA signal, allowing the Am9590 to take control of the system bus. When the Am9590 is the bus master, it uses its built-in DMA Controller to access the contents of the DRAM.

Because the DRAMs need to be refreshed by the 80188 DMA Channel 0, the Am9590 must be programmed such that the 80188 is allowed access to the memory within the refresh timing limits. This is done by setting the Am9590 DMA dwell time to 16 clock cycles and its DMA burst length to 16 bytes per DMA burst. This will force the Am9590 to release the bus from time to time so that the DMA refresh cycles can proceed.

The Am9590 can control up to four disk drives. Because of board space constraint, the SCSI board can only be connected to two hard-disk drives, DR0 and DR1. Upon reset, the software reads the configuration switches W0 and W1 to determine the type of drives that is connected to the SCSI board. After finding out the drive types, the software puts the default values for the Drive Parameter Block (DRB) of each drives into the DRAM's. A Load Drive Parameter Block command can then be executed to load these drive parameters inside the Am9590.

The hard-disk drive interface consists of one 34-pin drive control connector and two 20-pin drive data connectors. Because of the 48 mA drive requirement for the disk drive interface, an Am29828 buffer (U14) and a 74LS38 open-collector driver (U12) are used to invert the control signals coming out of the Am9590.

The status signals from the disk drives are terminated by resistor package RN7. These signals are then inverted by the 74LS14 (U28) Schmitt-Trigger inverters, before going to the Am9590.

Although the same connector is provided to an ST506 or ESDI disk drive, there are differences in the signal assignments for the two interfaces. The ESDI interface requires more control and data signals than the ST506 interface, therefore, some of the lines not commonly used in an ST506 interface are assigned as ESDI interface signals.

Jumpers W9–W14 are used to prevent contention between the ESDI signal lines used by the SCSI board and the lines that a particular ST506 drive may use. In particular, Pin 4 of the data cable connector, if connected to an ST506 disk drive, should be left open when the other data cable connector is connected to an ESDI disk drive. In addition, Pin 7 and Pin 8 of a data cable connector must be left open if they are connected to an ST506 drive.

The Am9590 asserts SELEN and sets its DRSEL$_0$ output LOW or HIGH to select Drive 0 or Drive 1 respectively. The Am9590 controls the drive interface lines either as ST506 signals or ESDI signals, according to the Drive Parameter Block (DPB) for the selected drive. The software will identify the attached drive as either ST506 or ESDI by checking the jumper settings W1 and W2, then the proper drive parameters are set up in the Drive Parameter Block (DPB) for each drive.

The multiplexer functions implemented in U13 (AmPAL22V10) and U17 (Am16L8) control the routing of data and control signals from the disk drives. If the selected drive is an ST506 drive, MFM signals are used in the drive interface and the Am9582 data separator must be used to convert the signals. If the selected drive is an ESDI drive, Non-Return to Zero (NRZ) signals are used in the drive interface, and these signals can be used directly with the Am9590.

The Am9582 data separator provides encoding and decoding functions for the MFM data if an ST506 disk drive is selected. The 10 MHz system clock is divided by two, using U13, to provide the 5 MHz clock frequency for the Am9582. The Am9582 generates a 5 MHz Reference Clock signal, and is connected to the WRCLK input to provide the frequency for the encoding of MFM data. When the Am9590 selects an ESDI drive, the Am9582 3-states all its output lines so that the Am9590 can communicate directly with the drive interface.

The Am9582 can support hard disk and floppy disk encoding and decoding functions. Since the SCSI board allows hard disk control only, the $F/\overline{H}$ line of the Am9582 is tied to ground. In addition, both the FAM0 and FAM1 inputs are connected to ground to select ST506 operations.

When data is written to an ST506 drive, the Am9590 reads data from system memory and produces the NRZ data that is synchronized to the Reference Clock input from the RD/REF CLK line. The Am9582 then encodes the NRZ data input with WRCLK and generates the MFM data output. The Am9582 also pre-compensates the MFM data output when its PCEN line is driven HIGH by the Am9590. Resistors R5 and R6 are used to set the pre-compensation delay timing when PCEN is activated. The disk tracks that require pre-compensation during a write are determined by the DPB for the selected drive.

When data is received from an ST506 drive, the Am9582 decodes the MFM data with its internal Phase-Locked Loop (PLL) and produces the NRZ data output and the Read Clock. During a Read operation, the RD/REF CLK line of the Am9582 is switched to the Read Clock signal generated by the PLL. This Read Clock is used by the Am9590 to sample the NRZ data on its RDDAT input.

When an ESDI drive is selected, NRZ data are used directly in the drive interface. The Am9590 generates WRCLK that is used by the disk drive to sample the WRDAT output during a write operation. When reading data from the disk, the ESDI drive provides both the NRZ Read Data and the Reference Clock to the Am9590. The Am9590 uses the Reference Clock signal on the RD/REF CLK line to sample the NRZ data on its RDDAT input.

When an ST506 drive is selected, the Am9582 generates the AMF signal to acknowledge to the Am9590 that an Address Mark has been generated during a write operation. During a read operation, the Am9582 also generates AMF to indicate that an Address Mark has been found. However, the ESDI interface generates the AMF signal directly and this signal is buffered by U15 and is supplied to the Am9590.

In this design, the software always reads two tracks from the disk and puts them inside the cache buffer whenever the required data is not already inside the cache. To reduce the latency in accessing the disk contents, logic in U17, in conjunction with a software trick, is used to implement a “sector pulse” feature.

The control program sets up the Am9590 such that it has to read the Data Map information every time the contents of a sector buffer is ready to be transferred to system memory. The Data Map pointer is also arranged to be in a memory space such that address bit 18 is always a one. Since this memory space is outside the 256 kbyte DRAM area, the DRAM interface masks out address bit 18 and provides the Am9590 with valid data that are inside the DRAMs. The Am9590 has only a 16-bit address bus, therefore, a one on the address bit 18 corresponds to the address line AD2 being set to HIGH when ALEN is asserted.

The Am9590 updates the upper 16-bit addresses A31–A16 every time a data map entry is accessed. Therefore, when address line AD2 and ALEN are both HIGH, the Am9590 indicates that a disk sector will be transferred from its internal buffer to the main memory. The logic in U17 makes use of this information to generate a pulse to the 80188 Timer 1 every time a sector transfer begins.

The software resets the 80188 Timer 1 before the Am9590 reads data from the disk. Timer 1 will then be incremented every time the Am9590 begins to transfer a disk sector. In this way, the software only needs to read the content of Timer 1 to find out how many disk sectors have already been transferred. When the Timer 1 count reaches the value two, the software knows that the first requested disk sector has already been transferred to memory, and the 80188 DMA controller can be set up to transfer this information to the SCSI bus. This scheme reduces data access latency drastically and increases the overall performance of the host.

An Am26LS31 differential line driver and an Am26LS32 differential line receiver are used to transfer data signals for the disk drive interface. For the ST506 interface, only one driver for the Write Data output and one receiver for the Read Data input are necessary. For the ESDI interface, an extra driver is required for the Write Clock signal and an extra receiver is required for the Reference Clock signal.

The Am26LS31 and the Am26LS32 are always enabled, therefore, the Write Clock signals destinated for an ESDI drive must be disconnected from the data connector of an ST506 drive to prevent contention. Jumpers W9 and W10 on data connector J3 and W12 and W13 on data connector J4 are used to disconnect the ESDI Write Clock signals.

The status signals from the disk drives are terminated with resistor package RN7 and are then buffered by a 74LS14 Schmitt- to the Am9590 to provide the necessary drive information to the controller.

Because ESDI drives do not support the Reduced Write Current control, bit 5 of the Data Select Byte in the DPB of an ESDI drive must be set to zero. This indicates to the Am9590 that the RWC output is used to represent the Head Select 3 signal, to allow selection of 16 heads in an ESDI drive. In addition, this will prevent the RWC output from being active when a smaller head number is selected.

Memory Organization and Initialization

The 256 kbytes system memory is organized in the following categories:

1. Code and Initialization Data

   All program code will be stored inside the EPROM which will be in high address space. This allows the control program to handle the reset condition. Also included in the EPROM will be a copy of the initialization data. The system reset routine will move this data to the DRAM. This is a requirement for the C compiler.

2. Vectors

   The interrupt vectors of the 80188 will be stored in low memory. The first 32 vectors are reserved by the CPU. No others will be required, therefore the vector area will reserve 128 bytes of DRAM.

3. Assembler Data

   Data that is required by the routines written in 8088 assembly language is stored just above the interrupt vectors.

4. Compiler Data

   The initialized and uninitialized (global and static) data generated by the C compiler will be stored following the data area for the assembly language routines.

5. Stack

   The stack will be maintained just above the compiler data.

6. Buffer Control Blocks

   The buffer control block area will be allocated by the initialization routine just above the stack.

7. Buffer Area

   The track buffers will be stored in the memory area which remains above the stack.

This memory allocation scheme permits all the system memory, with the exception of the track buffers, to lie within a single segment. Therefore only a "small data" compiler model is required. This allows efficient storage and access to the system data. This also minimizes the amount of assembly language programming required because all system data, including the interrupt vectors, are accessible by the compiled programs.

Upon system reset, an assembly language routine will initialize the 80188 registers to have the code segment set at the base of the 64 kbyte EPROM area. The data and the stack segment registers are set to zero. The stack pointer will then be initialized to point to the stack area. The interrupt vectors are then initialized to point to the interrupt service routines, which will call C routines to handle the interrupts. In this way, only minimal assembly language codes are required.

The track buffers can not be accessed directly by the compiled code. However, this is not a constraint because data transfer that involves the track buffers are handled by the DMA Controllers in the 80188 or the Am9590.

Drive Initialization and Format

Upon system reset, the SCSI disk controller will try to obtain the characteristics of the attached drive. If the attached disk drive has already been initialized by the controller, specific drive information will be found on the first track of the first disk cylinder. The stored information includes the dlf the attached disk drive has already been initialized by the controller, specific drive information will be found on the first track of the first disk cylinder. The stored information includes the drive parameter block for the drive and the defect lists.

If the required information is on track 0, the controller will store this information in system memory and initialize the Am9590 accordingly.

If the drive characteristics are not on track 0, and the attached drive is a ST506 drive, then the SCSI controller will not accept any SCSI command except for the Initialize Drive command.

The Initialize Disk command is a private command which allows the SCSI Initiator to pass drive parameters and the manufacturer defect list to the disk controller. The Initialize Disk command is a 6-byte (Group 0) SCSI command which should be issued only for an un-initialized ST506 drive. The format for the Initialize Disk command is:

Command Byte	Content
0	OEH: Vendor unique operation code
1	0 if Drive 0 is selected 20H if Drive 1 is selected
2–5	0

Upon reception of the Initialize Disk command, the SCSI controller will switch to a SCSI Data Out phase, and obtain the Disk Parameter List from the Initiator. The Disk Parameter List format is:

Byte	Description
0–17H	<Drive Parameter Block for the Drive>
18H–1BH	<Primary Defect List Header>
1CH–	<Primary Defect Descriptors>

The format of the Drive Parameter Block is:

Byte	Description
0	<General Select Byte>
1	<Data Select Byte>
2	Track per Surface (Low Byte)
3	Track per Surface (High Byte)
	Number of Heads
5	Sectors per Track
6	Reduced Write Current Track (Low Byte)
	Reduced Write Current Track (High Byte)
8	Seek Dwell (Low Byte)
9	Seek Dwell (High Byte)
A	Step Width
B	Head Settle
C	Pre-compensation Track (Low Byte)
D	Pre-compensation Track (High Byte)
E	Retry Policy
F	Motor On Delay
10	Delay Length
11	Preamble 1 Length
12	Postamble 1 Length
13	Pad Length
14	Preamble 2 Length
15	Error Correction Code Length
16	Postamble 2 Length
17	Gap Length

The recommended value for the General Select Byte is 0AAH, which uses Single-Burst Reed-Solomon Code and the head-first multi-record policy. For a 512-byte sector size, the recommended value for the Data Select Byte is 0AAH when the Reduced Write Current option is used.

The format of the Defect List Header is as follows:

Byte	Description
0	0
1	0
2	Defect List Length (High Byte)
3	Defect List Length (Low Byte)

The Defect List Length specifies the total length in bytes of the Defect Descriptors that follow the Defect List Header. The format of the Defect Descriptor is:

Byte	Description
0	Cylinder number of Defect (Highest Byte)
1	Cylinder number of Defect (Middle Byte)
2	Cylinder number of Defect (Low Byte)
3	Head number
4	Sector number (High Word, High Byte)
5	Sector number (High Word, Low Byte)
6	Sector number (Low Word, High Byte)
7	Sector number (Low Word, Low Byte)

Since a Defect Descriptor is eight bytes long, the Defect List Length specified should be equal to eight times the total number of defects on the disk. Whenever there is a defect on a track, the whole track will be relocated after physical formatting is done.

After the SCSI controller receives the Initialize Disk command, it will format track 0 of head 0 using the Drive Parameters it had received. This track is reserved as the Disk Information Track, and will contain information specific to the disk drive. The Disk Information Track is not accessible directly by the Initiator. The SCSI Controller performs the appropriate mapping such that the Disk Information Track is hidden from the Initiator; SCSI logical block zero actually corresponds to sector zero of track zero, head one on the drive.

The controller will then write the Identification, Format Mark, and the Drive Parameter Block into sector zero of the Disk Information Track. The contents of this ID Sector is:

Byte	Description
0	21H (Special Identification)
1	43H (Special Identification) 1 if the drive has already been formatted 0 if the drive has not been formatted
3	0
4–1BH	<Drive Parameter Block>

Byte 2 of the ID Sector is initialized to zero when the drive is initialized. After the disk has been formatted by the Initiator using the SCSI Format command, this byte will be set to a one to identify the disk as a formatted disk. When the disk is identified as a formatted disk, the Initiator is able to access the SCSI logical blocks on the drive.

After the ID Sector is written to the disk, the SCSI Controller will then write the primary defect information to the disk starting with sector 1 of the Disk Information Track. The first four bytes of the sector will contain the Primary Defect List Header, followed by the Primary Defect Descriptors. When the last primary defect descriptor is recorded, the SCSI Controller will write the Secondary Defect List Header with a zero Defect List Length; this marks the end of the disk defect information.

Any additional defects that are found later on the disk will have to be added to the Secondary defect list. The secondary defect information will be appended to the primary defect list and can grow or shrink with time. The last sector on the Disk Information Track is reserved for system diagnostics only, and will not contain any defect information. The format for the sectors that will contain the defect list is:

Byte	Description
0–3	<Primary Defect List Header>
4–	[<Primary Defect Descriptors>]
	..
	..
	<Secondary Defect List Header>
	[<Secondary Defect Descriptors>]

For an ESDI drive, the disk initialization process is simpler, because the physical characteristics can be obtained from the drive itself. The SCSI Controller accomplishes this by using the ESDI Channel command of the Am9590. If the Disk Information Track does not contain the correct formatted information, the SCSI Controller will automatically send a Request Configuration command to the drive to obtain the drive parameters.

The SCSI Controller will then read the disk's defect list from sector zero of the maximum cylinder for each individual head, as specified in the ESDI specification. If the data on the maximum cylinder is corrupted, the SCSI Controller automatically read sector zero of maximum cylinder minus eight. The defect information will be put into the correct format and written into the Disk Information Track.

To prevent the Initiator from accidentally destroying the defect information on an ESDI drive, the SCSI controller prevents access to any cylinder beyond maximum cylinder minus seven.

```
DEVICE HDC1 (AmPAL16L8);

" U20
ADVANCED MICRO DEVICES
IBM PC AT HDC BUS CONTROLLER
VERSION 1.3 MARCH 10, 1986

---------------------------------------------------------------------------------------------
"

PIN      /AEN = 1  /CSHDC = 2  MAS = 3  /PORTS = 4 /IOR = 7  /IOW = 8
         /DEN = 12  DIR = 13  /MEMW = 14  MASOUT = 15  /MEMR = 16
         /WR = 17  /RD = 18;


BEGIN

MEMR      =           RD;

MEMW      =           WR;

IF (AEN) THEN ENABLE(MEMR,MEMW);

RD        =           IOR;

WR        =           IOW;

/DIR      =           /IOR;

DEN       =           /AEN * CSHDC * (IOR + IOW) +
/AEN * PORTS * (IOR + IOW);

IF (/AEN) THEN ENABLE (RD,WR,DIR,DEN);

/MASOUT   =           MAS;

IF (MAS) THEN ENABLE (MASOUT);

END.
```

Figure 4-6

```
DEVICE HDC2 (AMPAL16R4);

" U12
ADVANCED MICRO DEVICES
IBM PC AT HDC BUS ARBITER
VERSION 1.3 MARCH 10, 1986
-------------------------------------------------------------------------------------------
"
PIN      CK = 1  /DACK = 2  BREQ = 3  /CSHDC = 4  /DEN = 8
         RESET = 9   /EN = 11  /IORDY = 12  /BHE = 13  /AEN = 14
         DRQ = 15   BACK = 16  MAS = 17  /SBHE = 18  /READY = 19;

BEGIN

/DRQ     :=          /BREQ                   +
                     RESET;


/MAS     :=          /BREQ                   +
                     /DACK                   +
                     RESET;

/BACK    :=          /BREQ                   +
                     /DACK                   +
                     RESET;

AEN      :=          BREQ * BACK * /RESET;

BHE      =           SBHE;

IF (/AEN) THEN ENABLE (BHE);

SBHE     =           BHE;

READY    =           AEN;

IF (AEN) THEN ENABLE (SBHE,READY);

IORDY    =           /AEN * CSHDC * DEN * /READY * /RESET;

IF (DEN) THEN ENABLE (IORDY);

END.
```

Figure 4-7

```
DEVICE HDC3 (AMPAL16L8);

" U24
ADVANCED MICRO DEVICES
IBM PC AT HDC I/O DECODER AND DRIVE SELECT MUX
VERSION 1.3  MARCH 10, 1986

-------------------------------------------------------------------------------------------
"

PIN      SA[4:9] = 1:6  /SEEKC = 7  A0 = 8  AENC = 9  /SEEK1 = 11
         DS1 = 13  /PORTS = 14  /STRQ = 15  /CSHDC = 16
         STAT = 17  SEEKCO = 18  /IOCS16 = 19;

BEGIN

CSHDC    = /AENC * /SA[9]* SA[8]*/SA[7]*/SA[6]*/SA[5]*/SA[4];

PORTS    = /AENC * /SA[9]* SA[8]*/SA[7]*/SA[6]*/SA[5]* SA[4] * /A0;

STRQ     = /AENC * /SA[9]* SA[8]*/SA[7]*/SA[6]*/SA[5]* SA[4] * A0;

IOCS16   = CSHDC + PORTS + STRQ;

IF (/AENC * /SA[9]* SA[8]*/SA[7]*/SA[6]*/SA[5]) THEN ENABLE(IOCS16);

/SEEKCO  = /SEEKC * /DS1 * /SEEK1;

/STAT = /SEEKC;

END.
```

Figure 4-8

APPENDIX

The Am9590 has been Improved

The new device number is Am9590-15. The following list five improvements to the device. One of them (**Zero Latency**) will improve the performance of the device considerably. The others are functional improvements that will make the Am9590-15 well-suited for applications that require some disk caching memory.

1. Implementation of Interrupt Features (Figure 1)

Two different Interrupts and their combinations have been implemented:

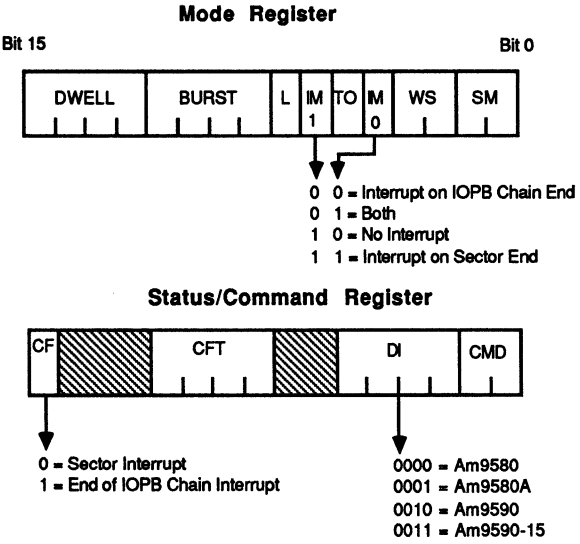
Interrupt on IOPB Chain END
Interrupt on Sector END

The MODE Register can be used to program the device. There are two bits available to indicate which interrupt is selected.

The Status Register is used to distinguish between the two interrupts. Whenever an interrupt occurs, the host CPU must read the Status Register. By examining the CF bit, it can determine if the interrupt was a Sector Interrupt (CF = 0) or an End of Chain Interrupt (CF = 1).

2. Read/Write Long (Figure 2)

To implement the Read/Write Long Commands, an Option Bit in the READ or WRITE VIRTUAL commands is used.



Mode Register

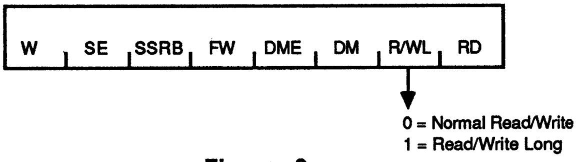
0 0 = Interrupt on IOPB Chain End
0 1 = Both
1 0 = No Interrupt
1 1 = Interrupt on Sector End

Status/Command Register

0 = Sector Interrupt
1 = End of IOPB Chain Interrupt

0000 = Am9580
0001 = Am9580A
0010 = Am9590
0011 = Am9590-15

Figure 1



Option Byte

0 = Normal Read/Write
1 = Read/Write Long

Figure 2

This feature allows the user to read/write the ECC bytes in or out of the SYNDROME RAM. The Read procedure will be:

1. Read single sector with R/WL bit = 1 --> Sector Data will be dumped to memory.
2. Read SYNDROME --> Syndrome RAM with ECC bytes dumped to memory.

The Write procedure will be:

1. Load SYNDROME with desired ECC bytes.
2. Write single sector with R/WL bit = 1 --> Sector Data and ECC bytes written to disk.

3. ISG Field Fix

The Am9590-15 implements a programmable recovery time between WG going inactive and AMC going active. This parameter is located in the Drive Parameter Block of the particular ESDI drive. It shares the location with the RWC parameter for ST506 drives (Bytes 6 and 7).

ISG = 60+(N•8) System Clocks

4. Zero Latency (Figure 3)

To enable the Zero Latency feature of the Am9590-15, the host CPU must enable the ZL bit in the READ or WRITE IOPB. The ZL bit is located in the uppermost bit location of the drive number byte. Therefore, if reading or writing to drive #2 with Zero Latency, the actual drive # written to Byte 8 of the IOPB will be 82_H.

In order to READ or WRITE with Zero Latency, the Am9590-15 will first do a READ ID to determine the current sector the head is passing over. Subsequently the device will issue a READ or WRITE command for the next sector. It will also update the address pointers so that the data is moved to or from the right location. This procedure ensures that the Zero Latency operation is fully transparent to the host. Start address for the buffer will be always where the address that Sector I is located in memory. **Please note that Zero Latency can only be done on a single and entire track base.** Therefore, the sector count in the IOPB should always be equal to the number of sectors per track.

Please note, that in order to do Zero Latency WRITE, the entire track must be present in memory before starting the operation. The Am9590-15 has no means to recognize if data in a particular memory location is valid or not.

5. Data Move IOPB (Figure 4)

The Am9590-15 has a feature to move data from one memory location to another without involving any disk transactions. This command is an option of the Load Buffer command (by setting the Data Move, DM bit) as shown in Figure 4.

Data moves can only be done in multiples of a sector.

Drive # (Byte 8)

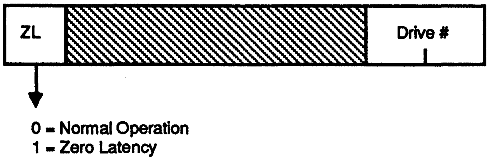
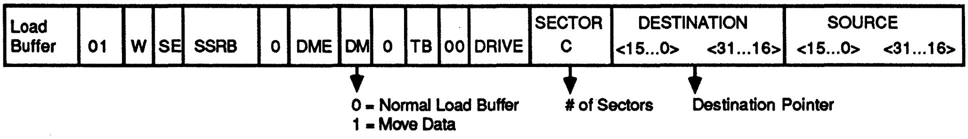


Figure 3



Load Buffer	01	W	SE	SSRB	0	DME	DM	0	TB	00	DRIVE	SECTOR C	DESTINATION <15...0> <31...16>	SOURCE <15...0> <31...16>

Figure 4

DOS Device Driver Header

```
/******************************************************************************
dosdrvr.h            DOS Device Driver Header                              v0.00

Copyright 1985 Advanced Micro Devices, Inc.
Written by Gibbons and Associates, Inc.

This file contains declarations of structures and constant data values which are
used in DOS device drivers.  It is assumed that the compiler will allocate
double words for pointers in the structures that follow. It is also assumed that
the compiler will not insert any padding to align words on even boundaries.

The linkage to DOS device drivers is defined in the IBM document titled "Disk
Operating System Version 3.10 Technical Reference" (IBM document number
6138536).  It is not intended that the contents of this header file should in
any way modify or expand upon the linkages defined by IBM.  IBM's document
should be considered the primary reference for these linkages.


NOTE:

The device header is not included in this file.  There are two reasons for this.
First, the device header contains three pointers, one of which is a double word
and two of which are single words.  This is difficult to represent accurately in
C.  Second, the device header is assumed to be in the same segment as the driver
code.  This is also uncommon in C. For these reasons, the device header itself
and the declarations pertinent to it are left to the assembly language "front
end" of the device driver.
******************************************************************************/



struct BPB
/*
This structure defines the parameter block which DOS requires as a description
of the media on a block device.
*/
{
unsigned
SectSize;                     /*bytes per sector*/

char
AllocSz;                      /*sectors per allocation unit*/

unsigned
RsvdSect;                     /*reserved sectors*/

char
FATCnt;                       /*number of file allocation tables*/

unsigned
RootCnt,                      /*number of root directory entries*/ SectCnt;
               /*total number of sectors*/

char
MediaDsc;                      /*the media descriptor*/

unsigned
SctPrFAT;                     /*sectors per FAT*/
```

```
unsigned
SctPrTrk,                    /*sectors per track*/ HdsPrCyl,
/*heads per cylinder*/ HdnSect;                     /*number of hidden
sectors*/ };


/*
The #define's below define fields withing the media descriptor.  The media
descriptor is useful in distinghishing types of floppy disks and in
distinguishing floppy disks from fixed disks.  All fixed disks have the
descriptor 0xF8.
*/
#define  MED_UPR     0xF8          /*upper bits must all be set*/ #define
MED_REM      (1 << 2)       /*set for removable media*/ #define  MED_8SCT    (1
<< 1)       /*set for 8 sectors per track*/ #define  MED_2SID    1
/*set for 2-sided media*/



#define       BPB_OFST    11                /*offset of BPB in boot sector*/

struct BootForm
/*
This structure defines the format of a (512 byte) boot sector. */
{
char
JmpInst [3],                 /*jump to the bootstrap code*/ OEMName [8];
         /*name of vendor and version*/

struct BPB
BootBPB;                     /*BPB for this media*/

char
BootCode [480];              /*bootstrap code and data*/

unsigned
BootSig;                     /*indentifier for valid boot sector*/ };


/*******************************************************************************
Request header format and values:

The declarations below define the format of the header area of a request (which
is common among all types of DOS requests) and values which may be found in the
header.
*******************************************************************************/


struct ReqHdr
/*
This header defines the static request header which is common to all DOS
requests.  It is always at the beginning of a DOS request.  The format of the
remainder of the request will vary with different command codes. */
{
char
```

```
ReqLen,                        /*the length of the request in bytes*/ UnitCode,
             /*the sub-unit for the operation*/ ReqCode;
/*the request code for the operation*/

unsigned
ReqStat;                       /*the returned status for the operation*/

char
*DosRsvrd [2];                 /*reserved by DOS for future use*/ };


/*
Request code values.
*/

#define  INIT        0              /*initialization (called only once)*/
#define  MED_CHK     1              /*media check - for removable media*/
#define  BLD_BPB     2              /*build parameter block */ #define  IOCT_INP
  3           /*I/O control input*/ #define  INPUT       4             /*I/O
read operation*/ #define  ND_INP      5                 /*non-destructive input, no
wait*/ #define  INP_STAT    6              /*input status inquiry*/ #define
INP_FLSH    7            /*input flush*/
#define  OUTPUT      8              /*I/O write operation*/ #define  OUT_VRFY
9              /*I/O write operation with verification*/ #define  OUT_STAT    10
           /*output status inquiry*/ #define  OUT_FLSH    11
/*output flush*/ #define  IOCT_OUT    12               /*I/O control output*/
#define  DEV_OPN     13             /*device open - for removable media*/
#define  DEV_CLS     14             /*device close - for removable media*/
#define  REM_MEDIA   15             /*removable media*/



/*
Status bits and fields.
*/

#define  ERR_MS      0xff           /*error code mask*/

#define  DONE        (1 << 8)       /*operation done*/ #define  BUSY        (1
<< 9)       /*device busy*/
#define  ERROR       (1 << 15)      /*error in operation*/



/*
Status error values.
*/


#define  WRT_PRT     0              /*write protection violation*/ #define
UNK_UNIT    1            /*unknown unit*/ #define  NOT_RDY     2
/*device not ready*/ #define  UNK_CMND    3              /*unknown command
code*/ #define  CRC_ERR     4              /*CRC (data) error*/ #define  BAD_LEN
   5            /*bad request structure length*/ #define  SK_ERR      6
     /*seek error*/
#define  UNK_MED     7              /*unknown media*/ #define  NT_FND      8
```

```
      /*sector not found*/ #define  NO_PAPER    9              /*printer out of
paper*/ #define  WRT_FLT     10              /*write fault*/
#define  RD_FLT      11           /*read fault*/
#define  GEN_FAIL    12           /*general failure*/ #define  INV_CHG     15
          /*invalid disk change*/

#define  NO_ERROR    0xFF         /*special value for no error*/


/*
File allocation table constants.
*/

#define  BAD_CLST    0xFFF7       /*indicates bad sector*/ /*not part of the
allocation chain*/ #define  CLUST_MS    0xFFF          /*mask for 12-bit FAT
entry*/ #define  CLUST_SH    4            /*12-bit FAT shift factor*/


/*******************************************************************************
Request formats:

The structures below define the formats for specific requests DOS may make.
*******************************************************************************/


struct InitReq
/*
This structure defines the format of data in a DOS INIT request. */
{
struct ReqHdr
InitHdr;                      /*static request header*/

char
UnitCnt;                      /*number of sub-units for this device*/

char
*DrvrEnd;                     /*ending address of the driver*/

unsigned
*InitBPBs;                    /*pointer to initial BPB array*/ /*on input pts to
string from CONFIG.SYS*/

char
UnitNum;                      /*unit number of the first sub-unit*/ };



struct MdChkReq
/*
This structure defines the format of a media check request.  Since this driver
is for non-removable media, it does not use the last field which might pose a
problem to some compilers.
*/
{
struct ReqHdr
MdChkHdr;                     /*static request header*/
char
MdChkDsc,                      /*DOS media descriptor*/ ChgStat;
   /*change status*/

char
*PrevVol;                      /*pointer to prev vol name*/ /*only used for
removable media*/ };


struct BlBPBReq
/*
This structure defines the format of data in a build parameter block request.
*/
{
struct ReqHdr
BlBPBHdr;                      /*static request header*/

char
BlBPBDsc;                      /*DOS media descriptor*/

char
*BlBPBBuf;                     /*pointer to buffer*/

struct BPB
*BPBPtr;                       /*pointer to the BPB*/ };


struct XfrReq
/*
This structure defines the format of all of the following requests:

IOCT_INP       I/O control input
INPUT       read operation
OUTPUT         write operation
OUT_VFRY       write operation with verification
IOCT_OUT       I/O control output
*/
{
struct ReqHdr
XfrHdr;                        /*static request header*/

char
XfrMdDsc;                      /*media descriptor*/

char
*XfrAddr;                      /*transfer address*/

unsigned
XfrSiz,                        /*sector or byte count for transfer*/ StartSct;
                /*starting sector for transfer*/

char
*ErrVolPt;                     /*pointer to volume ID error 15 returned*/ };


struct NDInpReq
/*
```

```
This structure defines the format of a non-destructive input request */
{
struct ReqHdr
NDInpHdr;                     /*static request header*/

char
NextChar;                     /*next character from the device*/ };


/*end of dosdrvr.h*/
```

Am9580 Hard Disk Controller Header

```
/******************************************************************************
Am9580.h                Am9580 Hard Disk Controller Header              v0.00

Copyright 1985 Advanced Micro Devices, Inc.
Written by Gibbons and Associates, Inc.

This file contains declarations of structures and constants which are useful in
writing drivers for the Am9580 hard disk controller in the C language on 8086
class CPUs.  If other CPUs are used, there may be problems related to the
ordering of bytes within 16-bit and 32-bit words.  8086 class CPUs and the
Am9580 are in agreement about these orderings, but other CPUs may have
different conventions.

It should also be noted that the packing of bytes within these structures is
important.  The compiler should not add any "padding."  Since all words are
aligned on even offsets and all longs are aligned on offsets which are even
multiples of four, this should not be too great a burden upon the compiler.

The Am9580's control registers as well as its IOPB and Status formats for
memory are defined in its data sheet.  This header is not intended to modify or
extend that information in any way.

Some of the vocabulary used in this file may be different from that of the
Am9580 data sheet.  For example, the concentric circles on a disk surface where
information is recorded are called tracks.  A group of corresponding tracks on
all surfaces of a disk drive is called a cylinder.  And each track is divided
into records, which are called sectors.  The Am9580 data sheet uses the terms
"sector" and "record" interchangably and mostly uses the word "track" to refer
to what is here called a "cylinder."
******************************************************************************/


/******************************************************************************
Register Definitions:

The declarations below are intended to simplify the use of the 9580's internal
registers.  The register addresses are defined relative to the base address of
the chip.

The fields within the mode register are defined via shift factors and (where
appropriate) unshifted value declarations.  This allows a mode register's
actual value to be defined by a single constant expression. The specific mode
register value used in this example is also defined.

The command register's command field has defined constant values for the
various possible values.

Word addressing of the Am9580 is assumed to be feasible.
******************************************************************************/


/*
Register Addresses:
*/
```

```
#define  HDC_ADDR    0x100

#define  CMND_REG    (HDC_ADDR + 0) /*status/command register*/ #define
MODE_REG    (HDC_ADDR + 2) /*mode register*/ #define  NBP_LO     (HDC_ADDR + 4)
/*next block pointer (low word)*/ #define  NBP_HI     (HDC_ADDR + 6)  /*next
block pointer (high word)*/ #define  SRP_LO     (HDC_ADDR + 8)  /*status result
pointer (low word)*/ #define  SRP_HI     (HDC_ADDR + 10) /*status result
pointer (high word)*/ #define  SRP_LEN    (HDC_ADDR + 12) /*status result
length*/



/*
Command/Status Register Fields and values.
*/


#define  CFT_VALID   (1 << 15)      /*controller fault type field is valid*/

#define  CFT_MAX     16             /*number of different controller fault
types*/

#define  CFT_SH      8              /*shift factor for controller fault type*/
#define  CFT_MS      (0xF << CFT_SH)/*mask for controller fault type*/

#define  NML_CPLT    0              /*normal command completion*/ #define
NULL_FLT    1              /*NBP was zero at start or resume time*/ #define
SRA_OVFL    2              /*overflow of status result area*/ #define  WAIT_STP
  3            /*IOPB completed with Wait Stop set*/ #define  FRC_IDLE    4
          /*idle command while executing chain*/ #define  ERR_STP     5
    /*non-reoveralbe error with SE set*/ #define  ILG_IOPB    6
/*illegal IOPB encountered*/ #define  SRB_STP     7             /*SRB written
with stop on SRB set*/ #define  DATA_TIM    8            /*memory timeout on
data transfer*/ #define  IOPB_TIM    9            /*memory timeout on IOPB
read*/ #define  MAP_TIM    10            /*memory timeout on data map read*/
#define  SRB_TIM     11             /*memory timeout on SRB write*/ #define
WRT_PROT    12             /*write protect violation*/ #define  RST_CPLT    15
          /*reset operation complete*/


#define  CMST_MS     3              /*command/status field mask*/

#define  IDLE        0              /*idle command*/ #define  RESET       1
        /*reset command*/ #define  RESUME      2             /*resume chain*/
#define  START       3              /*start chain*/



/*
Mode Register Fields and values.
*/


#define  DWELL_SH    12             /*shift factor for DMA Dwell clocks*/
#define  DWELL_MS    (0xF << DWELL_SH)/*mask for DMA Dwell clocks*/
```

```
#define  BURST_SH    8              /*shift factor for DMA Burst length*/
#define  BURST_MS    (0xF << BURST_SH)/*mask for DMA Burst length*/

#define  WAIT_SH     2              /*shift factor for wait states*/ #define
WAIT_MS     (0x3 << WAIT_SH)/*mask for wait states*/


#define  LOCKOUT     (1 << 7)       /*HDC cannot become bus master*/ #define
INT_DSB     (1 << 6)       /*HDC interrupts disabled*/


#define  MODE_MS     3              /*mask for seek mode*/

#define  IM_OV_SK    0              /*implied and overlapped seeks*/ #define
IM_SK       1              /*implied seeks only*/ #define  RSTR_SK     2
     /*restricted mode*/ #define  BUF_SK      3              /*buffered mode*/




/*******************************************************************************
Drive Parameter Block.

The structure and constants below are intended to simplify the construction of
a drive parameter block.
*******************************************************************************/

struct DPB
/*
This structure defines format of the drive parameter block. */
{
char
GnrlSlct,                    /*General Select Byte*/ DataSlct;
   /*Data Select Byte*/

unsigned
ClPerDsk;                    /*cylinders per disk*/

char
HdPerCyl,                    /*heads per cylinder*/ ScPerTrk;
  /*sectors per track*/

unsigned
RWCCyl,                      /*Reduced Write Current cylinder*/ SkDwell;
              /*seek dwell timing control*/

char
StepWid,                     /*step width timing control*/ HdSettle;
         /*head settle time control*/

unsigned
PreCmpCl;                    /*precompensation beginning cylinder*/
char
RetryPol,                    /*retry policy byte*/ DlyLen,
 /*index to first sector delay length*/ Pram1Len,
/*preamble 1 length*/ Ptam1Len,                    /*postamble 1 length*/
PadLen,                      /*pad length*/
Pram2Len,                    /*preamble 2 length*/ ECCLen,
 /*ECC length*/
Ptam2Len,                    /*postamble 2 length*/ GapLen,
  /*gap length*/
DPBRes;                      /*not defined*/
};



/*
General select byte fields and values.
*/

#define  AUTO_VEC    (1 << 7)       /*auto vector enable*/ #define  RTZ
(1 << 6)       /*RTZ used for recalibration*/

#define  ERPOL_SH    4              /*error policy shift factor*/ #define
ERPOL_MS    (3 << ERPOL_SH)/*error policy mask*/

#define  MRPOL_SH    2              /*multi record policy shift factor*/
#define  MRPOL_MS    (3 << MRPOL_SH)/*multi record policy mask*/

#define  FORMT_MS    3              /*format mask*/

#define  CRC_16      0              /*16-bit CRC-CCITT*/ #define  EXT_ECC     1
            /*external ECC*/ #define  SGL_ECC     2             /*single-burst
Reed-Solomon ECC*/ #define  DBL_ECC     3             /*double-burst
Reed-Solomon ECC*/



/*
Data select byte values.
*/

#define  SECT_128    0              /*128 byte sector size*/ #define  SECT_256
 1            /*256 byte sector size*/ #define  SECT_512    2
/*512 byte sector size*/



/*
Retry policy byte fields and values.
*/

#define  PRE_ECC     (1 << 7)       /*ECC before any retries*/ #define  ALL_ECC
   (1 << 6)       /*ECC after all retries*/ #define  POST_ECC    (1 << 5)
/*ECC after last retry attempt*/ #define  RTRY_ENB    (1 << 4)       /*retries
enabled*/

#define  RETRY_MS    0xF            /*retry count mask*/
```

```
/*****************************************************************************
IOPB Formats:

The declarations below are intended to simplify the process of constructing and
IOPB.  The layout of bytes and words within the IOPB are defined as structures.
Bit fields are defined via #define's for ease in combining them into single
values.
*****************************************************************************/



struct IOPBHdr
/*
This structure defines the format of the first fourteen bytes of an I/O
parameter block.  This header has the same format for all commands. */
{
long
NextIOPB;                     /*absolute address of next IOPB*/

unsigned
IOPBID;                       /*ID for this parameter block*/

char
Options,                      /*byte holding option bits*/ CmndCode,
        /*I/O command code*/ Drive,                        /*drive number*/
Dummy;                        /*dummy byte*/

unsigned
Cylinder;                     /*(called track in data sheet)*/

char
Sector,                       /*logical or physical or format pattern*/
/*depending on cmnd*/ Head;
};



/*
Command Code values.
*/


#define  READ        0x0C
#define  WRITE       0x0D
#define  VERIFY      0x0F
#define  FORMAT      0x07
#define  RLCT_TRK    0x0B          /*relocate track*/ #define  LD_DPB
0x00          /*load drive parameter block*/ #define  DMP_DPB     0x03
 /*dump drive parameter block*/ #define  RD_PHYS     0x0A          /*read
physical sector*/ #define  RD_ID       0x09          /*read ID*/
#define  LD_BUF      0x01          /*load buffer*/
#define  DMP_BUF     0x02          /*dump buffer*/
#define  LD_SYND     0x04          /*load syndrome*/ #define  DMP_SYND    0x05
         /*dump syndrome*/ #define  CRCT_BUF    0x06          /*correct
buffer*/ #define  SEEK        0x0E
#define  RESTORE     0x08          /*resotre heads to cylinder 0*/
```

```
/*
Options bits.
*/

#define  W           (1 << 7)      /*stop after current IOPB*/ #define  SE
    (1 << 6)      /*stop on error*/ #define  SSRB        (1 << 5)       /*stop
if SRB written*/ #define  DME       (1 << 3)       /*data mapping enabled*/
#define  TV          (1 << 3)      /*track verify enabled (Rd Phys and Seek)*/
#define  LD          (1 << 3)      /*locator dump enabled (Cr Buf and Rd ID)*/
#define  DM          (1 << 2)      /*data mark (normal disk commands)*/


/*
Data mark.
*/

#define  DATA_MRK    0xFE          /*data mark*/


/*
Misc.
*/

#define  INIT_SEC    0x0           /*starting sector number*/


/*
Specific IOPB Formats.
*/


struct StdIOPB
/*
This structure defines the format a standard IOPB.  Not all commands use all of
the fields defined.  A few special commands depart from this format. */
{
struct IOPBHdr
StdHdr;                        /*the standard IOPB header*/

char
SectCnt,                       /*number of sectors for operation*/ DataMark;
               /*for normal operations*/

long
MemAddr;                       /*absolute memory address*/ };


struct FmtIOPB
/*
This stucture defines the format of the IOPB for a format operation. */
{
struct IOPBHdr
FmtHdr;                        /*the standard IOPB header*/

unsigned
TrkCount;                      /*the number of tracks to be formatted*/
```

```
long
MapAddr;                      /*absolute address of sector map*/ };


struct RlctIOPB
/*
This structure defines the format of a relocate track IOPB. */
{
struct IOPBHdr
RlctHdr;                      /*the standard IOPB header*/

unsigned
AltCyl;                       /*the alternate cylinder*/

char
RlctRes1,                     /*unused*/
AltHead,                      /*the alternate head*/ RlctRes2 [2];
  /*unused*/
};



/***************************************************************************
Status/Results Block Definitions:

The declarations below are intended to ease access to the status/results block
and to simplify its interpretation.  The layout of the bytes and words in the
status/results block are defined via structures.  The specific values which may
be found are declared with #define's
***************************************************************************/


struct SRBHdr
/*
This structure defines the format of the first three bytes of the
status/results block.  This header has the same format for all SRBs. */
{
unsigned
SRBID;                        /*ID field for correlation to IOPB*/

char
RsltCode;                     /*identifier for result type*/ };


struct SRBLoc
/*
This structure defines the format of the tail of a status/result block which
specifies a disk location.  It is used in several SRBs. */
{
unsigned
RsltCyl;                      /*cylinder of SRB location*/
```

```
char
RsltHead,                   /*head of SRB location*/ RsltRes1,
    /*reserved*/
RsltRes2,                   /*reserved*/
RsltSect;                   /*sector of SRB location*/ };


/*
Status/Result Code values.
*/

#define  SRER_MAX    31            /*number of different code values*/

#define  MSC_TERM    0x19  /*??*/   /*multi-sector command error termination*/
#define  NO_IDS      0x01           /*no IDs found on track*/ #define  FMT_ERR
  0x00          /*unexpected index pulse encountered*/ #define  SEEK_ERR
0x02          /*ID found, but not as expected*/ #define  FTL_SEEK    0x08
    /*seek error after restore and re-seek*/ #define  RLC_TRK     0x05
/*track read as relocated*/ #define  RLC_NOVEC   0x06         /*relocation
failure - no vector*/ #define  NOT_FND     0x09         /*sector not found on
track*/ #define  NTFND_ER    0x07         /*sector not found, IDs had CRC
errors*/ #define  MSC_OVFL    0x0D         /*multi-sector not allowed across
track*/ #define  NOT_RCVR    0x17         /*HDC could not recover data
requested*/ #define  RCVR_RTR    0x0A         /*data recovered via retries*/
#define  RCVR_ECC    0x03           /*data recovered via ECC*/ #define
SYNC_FLT    0x04           /*ID OK, but no address mark for data*/ #define
DM_ERR      0x0E           /*data mark error*/ #define  SCT_SIZE    0x0F
  /*sector size mismatch*/ #define  VRFY_ERR    0x0B          /*data did not
verify on verify command*/ #define  PHS_RCVR    0x1A  /*??*/   /*physical data
recovered via ECC*/ #define  PHS_UNCR    0x1B  /*??*/   /*physical contained
uncorrected errors*/ #define  ECC_NTSEL   0x10          /*ECC attempted, but
not selected*/ #define  DATA_ECC    0x18          /*an ECC error was detected
in the data*/ #define  DM_PHYS     0x1C  /*??*/   /*data mark error read phys*/
#define  ID_CRC      0x1D  /*??*/   /*CRC error in ID field*/ #define  SEEK_FLT
  0x12          /*FAULT line asserted while seeking*/ #define  RSTR_FLT
0x16          /*HDC could not restore the drive*/ #define  HDSL_FLT    0x13
     /*HDC could not select specified head*/ #define  DRSL_FLT    0x11
 /*HDC could not select the specified drive*/ #define  DRST_TRP    0x1E  /*??*/
 /*unexpected drive status change*/ #define  MEM_TIM     0x0C
/*memory time-out*/ #define  NO_RLCT     0x14          /*relocate track
illegal on sgl dens flpy*/ #define  END_MAP     0x15          /*unexpected end
of data map*/


/*
Drive Status bits
*/

#define  DREADY      (1 << 9)      /*the drive ready status line*/ #define
FAULT       (1 << 10)      /*the hardware fault line*/ #define  SEEKCOM     (1
<< 11)      /*the seek complete line*/ #define  WRTPROT     (1 << 12)
/*the write protect line*/ #define  TRK0        (1 << 13)      /*the track 0
status line*/
```

```
/*
Specific SRB formats
*/

struct MTrmSRB
/*
This structure defines the format of the status result block indicating that a
multi-record command has been terminated at the specified location and count.
*/
{
struct SRBHdr
MTrmHdr;                      /*the standard SRB header*/

char
MTrmCnt;                      /*count of sectors processed???*/

struct SRBLoc
MTrmLoc;                      /*location of the termination*/ };


struct StdSRB
/*
This structure defines a format which encompasses several actual SRBs.  Not all
of the fields will always contain vaild information. Consult the Am9580 data
sheet for details.
*/
{
struct SRBHdr
StdHdr;                       /*the standard SRB header*/

char
StdRes;                       /*reserved*/

struct SRBLoc
StdLoc;                       /*the standard SRB location fields*/

/*add two more bytes here???  so that it can be used with sizeof()?? */

};


struct SeekSRB
/*
This structure defines the format of the SRBs which are generated when seek
errors occur.
*/
{
struct SRBHdr
SeekHdr;                      /*the standard SRB header*/

char
SeekRes;                      /*reserved*/
```

```
unsigned
CurCyl,                       /*current cylinder*/ DsrdCyl;
/*desired cylinder*/

char
DsrdHead,                     /*desired head*/ CurHead;
/*current head*/ };


struct RlctSRB
/*
This structure defines the format of the SRB which is generated when a track is
read as relocated and the HDC auto-vectors to the new track.
*/
{
struct SRBHdr
RlctHdr;                      /*the standard SRB header*/

char
RlctRes3;                     /*reserved*/

unsigned
RlctCyl,                      /*cylinder which was relocated*/ NewCyl;
            /*new cylinder to be used*/

char
NewHead,                      /*head which was relocated*/ RlctHead;
        /*new head to be used*/ };



struct RcvrSRB
/*
This structure defines the format of the SRBs which are generated when data is
recovered either via retries or via the ECC. */
{
struct SRBHdr
RcvrHdr;                      /*the standard SRB header*/

char
RetryCnt;                     /*the retry count*/

struct SRBLoc
RcvrLoc;                      /*location of data recovered*/ };


struct MarkSRB
/*
This structure defines the format of the SRBs which are generated when a data
mark error occurs.  The location fields are not valid in the case of a data
mark physical error.
*/
{
struct SRBHdr
MarkHdr;                      /*the standard SRB header*/
```

```
char
MarkFnd;                      /*the data mark found*/

struct SRBLoc
MarkLoc;                      /*the location of the error*/ };

struct SizeSRB
/*
This structure defines the format of the SRB which is generated if a sector
size mismatch is occurs between the header actually found and the DPB.
*/
{
struct SRBHdr
SizeHdr;                      /*the standard SRB header*/

char
SizeFnd;                      /*the sector size code found*/ };

struct FltSRB
/*
This structure defines the formats of SRBs which are generated when a hardware
fault or an unexpected change in drive status occurs. The location fields
indicate the cylinder sought if the fault occured during a seek or the head
desired if the fault occured during a head select.

???Is it true that status is not included in fault while head select??? */
{
struct SRBHdr
FltHdr;                       /*the standard SRB header*/

char
DrvStat;                      /*the drive status lines*/

struct SRBLoc
FltLoc;                       /*location sought when fault occured*/ };

=
struct TimeSRB
/*
This structure defines the format of the SRB which is generated when a data
timeout occurs.
*/
{
struct SRBHdr
TimeHdr;                      /*the standard SRB header*/

char
TimeRes;                      /*reserved*/

unsigned
BlkCnt;                       /*size of block where error occured*/

long
BlkAddr;                      /*physical address of block start*/ };

/*end of Am9580.h*/___________________________
```

DOS Device Driver for the Am9580

```
/*******************************************************************************
drvr9580.c           DOS Device Driver for the Am9580              v0.00

This file contains the C programs which constitute the bulk of the loadable DOS
device driver for the Am9580.  A small amount of assembly language is required
for interface to DOS.  There are also external files containing DEBUG routines.

Upon initialization, 1 of n pre-defined DPB's may be selected by specifying a
DPB selection number (0 to n-1) in the "device" statement of the CONFIG.SYS
file.  For instance,

device = Am9580.exe 2

selects DPB parameter set 2.  If no parameter selection is included, the default
is 0.

This driver assumes that only one drive, which is non-removable, is attached to
the Am9580.
*******************************************************************************/


#include "dosdrvr.h"                    /*declarations for DOS device drivers*/
#include "Am9580.h"                     /*declarations for Am9580 use*/

#ifdef   DEBUG

#include "dbg.h"                        /*declarations for debug use*/

#endif


/*
Error translation codes
*/

#define  CHK_SRB        0xFE        /*look into SRB for details on error*/
#define  RW_ERROR       0xFD        /*read/write error*/ #define  SRB_MAX
16          /*number of SRB in SRA*/

/*
Initial value of mode register
*/

#define  MODE_VAL    ((0x1 << DWELL_SH) | (0xF << BURST_SH) | INT_DSB \ | (0x0
<< WAIT_SH) | IM_SK)     /* 0x1f41 */


extern
EndDrvr ();                        /*to get end of the driver*/

unsigned
DPBIndex;
```

```
struct DPB
DPBArea[8] =                    /*drive parameter block array*/ {0x3A,
                         /*general select byte*/
       2,                             /*data select byte*/  0x140,
                         /*cylinders per disk*/ 4,
          /*heads per cylinder*/ 0x11,                          /*sectors per
track*/ 0x50,                         /*reduced write current cylinder*/ 0x10,
                         /*seek dwell timing control*/ 0x10,
    /*step width timing control*/ 0x20,                   /*head settle time
control*/ 0x45,                     /*precompensation biginning cylinder*/
       0x5f,                  /*retry policy byte*/ 0xf0,
          /*index to first sector dalay length*/ 0x10,
          /*preamble 1 length*/ 3,                              /*postamble 1
length*/ 0x10,                         /*pad length*/
0x10,                    /*preamble 2 length*/ 9,
          /*ECC length*/
3,                             /*porstamble 2 length*/ 8,
                /*gap length*/
0xf0};                          /*not defined*/


#ifdef NO_HDWR

struct StdSRB                      /*storage for Am9580 status/results*/
SRBArea[SRB_MAX]
= {4096, 4, 0, 3, 3, 0, 0, 1};   /*for simulation only*/

struct BPB
BPBParm                         /*disk BIOS parameter block*/ = {0x200, 4, 1,
2, 0x200, 0x110, 0xF8, 1, 0x11, 4, 0};

#else

struct StdSRB                      /*storage for Am9580 status/results*/
SRBArea[SRB_MAX];

struct BPB
BPBParm;                        /*disk BIOS parameter block*/

#endif

unsigned
BPBAddr;                        /*word offset of BPB*/

/*
This table translates controller fault type values to DOS error codes. */

unsigned
XlatCFT[CFT_MAX] =
{NO_ERROR,                    /*NML_CPLT*/
GEN_FAIL,                    /*NULL_FLT*/
GEN_FAIL,                     /*SRA_OVFL*/
NO_ERROR,                     /*WAIT_STP*/
GEN_FAIL,                     /*FRC_IDLE*/
CHK_SRB,                      /*ERR_STP*/
GEN_FAIL,                     /*ILG_IOPB*/
CHK_SRB,                      /*SRB_STP*/
GEN_FAIL,                     /*DATA_TIM*/
GEN_FAIL,                     /*IOPB_TIM*/
GEN_FAIL,                     /*MAP_TIM*/
GEN_FAIL,                     /*SRB_TIM*/
WRT_PRT,                      /*WRT_PROT*/
GEN_FAIL,                     /*reserved*/
GEN_FAIL,                     /*reserved*/
NO_ERROR},                    /*RST_CPLT*/

/*
*This table translates status result block codes to DOS error codes. */

XlatSRB[SRER_MAX] =
{RW_ERROR,                     /*FMT_ERR*/
RW_ERROR,                     /*NO_IDS*/
NO_ERROR,                     /*SEEK_ERR*/
NO_ERROR,                     /*RCVR_ECC*/
RW_ERROR,                     /*SYNC_FLT*/
RW_ERROR,                     /*RLC_TRK*/
RW_ERROR,                     /*RLC_NOVEC*/
NT_FND,                       /*NTFND_ER*/
SK_ERR,                       /*FTL_SEEK*/
NT_FND,                       /*NOT_FND*/
NO_ERROR,                     /*RCVR_RTR*/
WRT_FLT,                      /*VRFY_ERR*/
GEN_FAIL,                     /*MEM_TIM*/
RW_ERROR,                     /*MSC_OVFL*/
RW_ERROR,                     /*DM_ERR*/
UNK_MED,                      /*SCT_SIZE*/
GEN_FAIL,                     /*ECC_NTSEL*/
GEN_FAIL,                     /*DRSL_FLT*/
SK_ERR,                       /*SEEK_FLT*/
SK_ERR,                       /*HDSL_FLT*/
GEN_FAIL,                     /*NO_RLCT*/
GEN_FAIL,                     /*END_MAP*/
SK_ERR,                       /*RSTR_FLT*/
CRC_ERR,                      /*NOT_RCVR*/
CRC_ERR,                      /*DATA_ECC*/
RW_ERROR,                     /*MSC_TERM*/
NO_ERROR,                     /*PHS_RCVR*/
CRC_ERR,                      /*PHS_UNCR*/
RW_ERROR,                     /*DM_PHYS*/
RW_ERROR,                     /*ID_CRC*/
GEN_FAIL};                    /*DRST_TRP*/
```

```
MoveMem (Src, Dest, Cnt)
/*
This routine copies Cnt bytes from Src to Dest.  It doesn't perform correctly in
the event of overlap.

Normally, a library routine would be used for this function.  It is included
here to avoid dependence on a particular library's linkage convention.
*/

char
*Src,
*Dest;

unsigned
Cnt;

{
for ( ; Cnt > 0; *Dest++ = *Src++, Cnt--)
;
return;
}


long
PhysAddr (MemAddr)
/*
This routine converts segmented address into absolute physical address.

It definitely needs to be recoded if some other compiler is used!!! */

char
*MemAddr;

{
/* return (((MemAddr & 0xFFFF0000l) >> 12) + (MemAddr & 0x0FFFFl)); */ return
((long)MemAddr);
}


unsigned
LogSect (Cylinder, Head, Sector)
/*
This routine converts cylinder, head, sector into logical sector number. */

unsigned
Cylinder,
Head,
Sector;

{
unsigned
LogSec;                         /*storage for the logical sector number*/
LogSec = ( ((Cylinder * BPBParm.HdsPrCyl + Head) * BPBParm.SctPrTrk) + Sector -
INIT_SEC - BPBParm.HdnSect); return (LogSec);
}


unsigned
GetRmSec (ReqPtr, SRBPtr)
/*
This routine checks on which sector the error has occurred, and figures out the
remaining sector count for the DOS return status. */

struct XfrReq
*ReqPtr;

struct StdSRB
*SRBPtr;

{
unsigned
CrntPos,                      /*current position where error occurred*/ RmCnt;
                    /*remaining sector count*/

switch (SRBPtr->StdHdr.RsltCode)
{
case MSC_TERM:                /*the calculations will be done only for*/ case
NOT_FND:                 /*those with disk location information*/ case NTFND_ER:
case MSC_OVFL:
case NOT_RCVR:
case RCVR_RTR:
case RCVR_ECC:
case SYNC_FLT:
case DM_ERR:
case VRFY_ERR:
{
CrntPos = LogSect (SRBPtr->StdLoc.RsltCyl, SRBPtr->StdLoc.RsltHead,
SRBPtr->StdLoc.RsltSect); RmCnt = ReqPtr->XfrSiz - (CrntPos - ReqPtr->StartSct);
break;
}
default:                      /*for others, assume none were transfered*/ {
RmCnt = ReqPtr->XfrSiz;
break;
}
}
/*   return (RmCnt); */
return (ReqPtr->XfrSiz);
}


SetIOPB (ReqPtr, IOPBPtr, Command)
/*
This routine converts the parameters from MSDOS request block to a format that
fits into the IOPB required by the Am9580. This includes translating logical
sector number to the cylinder, track, and sector form,  and converting segmented
address to absolute physical address.
```

```
It is usually called from disk transfer routines, e.g. read, write, verify. */

struct XfrReq
*ReqPtr;

struct StdIOPB
*IOPBPtr;

unsigned
Command;

{
unsigned
Cylinder,                   /*temporary variables*/ Track,
Head,
Sector;

Sector = (ReqPtr->StartSct + BPBParm.HdnSect) % BPBParm.SctPrTrk + INIT_SEC;
Track = (ReqPtr->StartSct + BPBParm.HdnSect) / BPBParm.SctPrTrk; Cylinder =
Track / BPBParm.HdsPrCyl;
Head = Track % BPBParm.HdsPrCyl;

IOPBPtr->StdHdr.NextIOPB = 0;
IOPBPtr->StdHdr.IOPBID = 4096;   /*???*/
IOPBPtr->StdHdr.Options = SSRB;
IOPBPtr->StdHdr.CmndCode = Command;
IOPBPtr->StdHdr.Dummy = 0;
IOPBPtr->StdHdr.Drive = 0;
IOPBPtr->StdHdr.Cylinder = Cylinder;
IOPBPtr->StdHdr.Sector = Sector;
IOPBPtr->StdHdr.Head = Head;
IOPBPtr->SectCnt = ReqPtr->XfrSiz;
IOPBPtr->DataMark = DATA_MRK;
IOPBPtr->MemAddr = PhysAddr (ReqPtr->XfrAddr);
return;
}


RtnStat (ReqPtr, Code, SecCnt)
/*
This routine sets the return status for MSDOS.  It will always set the done bit.
If an error has occurred, the error code, error bit, and the remaining count
will be set accordingly.
*/

struct XfrReq
*ReqPtr;                    /*pointer to a request block*/

unsigned
Code,                       /*error code, 0xFF for none*/ SecCnt;
        /*remaining count*/
```

```
{
ReqPtr->XfrHdr.ReqStat = DONE;
ReqPtr->XfrSiz -= SecCnt;
if (Code != NO_ERROR)
{
ReqPtr->XfrHdr.ReqStat |= (ERROR | (Code & ERR_MS)); }
return;
}


unsigned
XlatErr (CFTErr, RWErr, ReqPtr)
/*
This routine translates the error returned by Am9580 into MSDOS error code.
This includes translating from controller fault type directly, or looking into
the SRB for more details.

RWErr is the DOS error code to be returned if RW_ERROR is found. This routine
also calls the RtnStat() to prepare the return status for MSDOS.
*/

unsigned
CFTErr,                        /*fault type in status register*/ RWErr;
            /*value to be returned if RW_ERROR*/

struct XfrReq
*ReqPtr;

{
unsigned
DOSErr,
RemSect;                       /*number of remaining sectors*/

DOSErr = XlatCFT[CFTErr];
RemSect = ReqPtr->XfrSiz;         /*assume error, and none were transfered*/ if
(DOSErr == CHK_SRB)
{                              /*need to look into SRB*/

#ifdef   NO_HDWR
GetStat (&SRBArea[0]);
#endif

DOSErr = XlatSRB[SRBArea[0].StdHdr.RsltCode];
if (DOSErr != NO_ERROR)
{                           /*check return location*/ RemSect = GetRmSec (ReqPtr,
&SRBArea[0]);
}

#ifdef   DEBUG
if (DbgCtrl & SHOW_STA)
{
ShowSRB (&SRBArea[0]);
}
#endif
}
if (DOSErr == RW_ERROR)
{
DOSErr = RWErr;
}
if (DOSErr == NO_ERROR)
{
RemSect = 0;                    /*correct remaining sectors if no error*/ }
else
{
switch (ReqPtr->XfrHdr.ReqCode)
{
case INPUT:
case OUTPUT:
case OUT_VRFY:
{
break;
}
default:
{
RemSect = 0;              /*remaining sectors is only meaningful*/ break;
       /*   for transfer requests*/ }
}
}
RtnStat (ReqPtr, DOSErr, RemSect);
return (DOSErr);
}


unsigned
ExCmnd (Command, IOPBPtr)
/*
This routine executes a 9580 command.  It then waits for the controller to
become idle and reads the resulting command/status word.  It returns controller
fault type in the status register.

This routine will always updates the value of status result pointer register. */

unsigned
Command;                        /*the command word*/

struct IOPBHdr
*IOPBPtr;                       /*for debug purpose*/

{
long
Addr;

unsigned
Cnt,
Status,
CFType;
```

```
#ifdef   DEBUG
if (DbgCtrl & SHOW_CMD)
{
PutStr ("IOPB command:\n");
ShowIOPB (IOPBPtr);
}
#endif

if (IOPBPtr->Cylinder >= DPBArea[DPBIndex].ClPerDsk) return (0x0E);
              /*for sector not found*/ Addr = PhysAddr (IOPBPtr);
   /*initialize next block reg*/ OutWord (NBP_LO, (unsigned)Addr);
OutWord (NBP_HI, (unsigned)(Addr >> 16));
Addr = PhysAddr (&SRBArea[0]);           /*initialize SRB reg*/ OutWord (SRP_LO,
(unsigned)Addr);
OutWord (SRP_HI, (unsigned)(Addr >> 16));
OutWord (SRP_LEN, SRB_MAX);      /*size of area in terms of blocks*/ OutWord
(CMND_REG, Command);
/* some delay may be needed */
Status = InWord (CMND_REG);
while ((Status & CMST_MS) != IDLE)
{
/*wait till it becomes idle*/
Status = InWord (CMND_REG);
}

CFType = (Status & CFT_MS) >> CFT_SH;

#ifdef   DEBUG
if (DbgCtrl & SHOW_STA)
{
sprintf (DbgStr, "Status register:  %4x\n", Status); PutStr (DbgStr);
}
#endif

return (CFType);                 /*error code is returned*/ }


RVCmnd (ReqPtr, Command)            /*TEMPORARY!!*/
/*
This routine's operation is very similar to that of the ExCmnd()'s except that
it is specially designed to handle the data_recovered_with_ECC error.  When the
controller stops on a data_recovered_with_ECC error, this routine will send out
a correct buffer command, and then send out another read/verify request starting
at the next sector. */

struct XfrReq
*ReqPtr;                      /*dos request block pointer*/


unsigned
Command;                      /*the command word*/

{
struct StdIOPB
XferIOPB;
```

```
unsigned
FaultTyp,                    /*fault in stats register*/ RWErr,
      /*code to be returned if RW_ERROR*/ CrntSct,                    /*sector
where RCVR_ECC occurred*/ XfrCnt,                     /*number of sectors
transfered*/ TotalXfr;                      /*total number of sectors
requested*/

ReqPtr->XfrHdr.ReqStat = 0;      /*initial status to be no error*/ RWErr =
WRT_FLT;                 /*value to be returned if RW_ERROR*/ if (Command ==
READ)
{
RWErr = RD_FLT;              /*return read fault if input request*/ }
XfrCnt = 0;
TotalXfr = ReqPtr->XfrSiz;

/*loop until all sectors are transfered unless error occurred*/

while (XfrCnt < TotalXfr && (ReqPtr->XfrHdr.ReqStat & ERROR) == 0) {
SetIOPB (ReqPtr, &XferIOPB, Command);
FaultTyp = ExCmnd (START, &XferIOPB);
if (XlatCFT[FaultTyp] == CHK_SRB)
{

#ifdef   NO_HDWR
GetStat (&SRBArea[0]);
#endif

if (SRBArea[0].StdHdr.RsltCode == RCVR_ECC) {
CrntSct = LogSect (SRBArea[0].StdLoc.RsltCyl, SRBArea[0].StdLoc.RsltHead,
SRBArea[0].StdLoc.RsltSect); XfrCnt += (CrntSct - ReqPtr->StartSct);  /*sectors
transfered*/ XferIOPB.StdHdr.NextIOPB = 0;            /*do correct buffer*/
XferIOPB.StdHdr.IOPBID = 4096;
XferIOPB.StdHdr.Options = SSRB;          XferIOPB.StdHdr.CmndCode = CRCT_BUF;
XferIOPB.StdHdr.Drive = 0;
XferIOPB.MemAddr = 0; /*????*/
FaultTyp = ExCmnd (START, &XferIOPB);
XlatErr (FaultTyp, RWErr, ReqPtr);
if ((ReqPtr->XfrHdr.ReqStat & ERROR) == 0) {
XfrCnt++;                 /*include the recovered one*/ while
(ReqPtr->StartSct <= CrntSct)
{
ReqPtr->XfrAddr += BPBParm.SectSize;      ReqPtr->StartSct++;
ReqPtr->XfrSiz--;
}
}
}
else
{
XlatErr (FaultTyp, RWErr, ReqPtr);
XfrCnt += ReqPtr->XfrSiz;            /*add up all xfr sectors*/ }
}
else
{
XlatErr (FaultTyp, RWErr, ReqPtr);
XfrCnt += ReqPtr->XfrSiz;              /*add up all xfr sectors*/ }
}
ReqPtr->XfrSiz = XfrCnt;       /*use total sectors xfered*/ return;
}


Init (ReqPtr)
/*
This routine is called only once.  It does the following:

1. Initialize the hardware Am9580, e.g. setting the status result pointer
register, status result length register, mode register, etc.  It also scans a
number from the invocation line which is an index to a list of DPB structures
and the chosen DPB is used as the initial value.

2. Read the BPB from the boot sector of the disk and return a pointer to the
BPB.

3. Return the number of units (1).

4. Return the first available byte of memory above the resident driver. */

struct InitReq
*ReqPtr;

{
struct StdIOPB
XfrIOPB;

unsigned
Cnt,
Rtn;

char
*Ptr;

struct PtrAddr
{
unsigned
Ofst,
Seg;
} EndAddr, *EndPtr;


/*query initial value for DbgCtrl*/

#ifdef   DEBUG
sprintf (DbgStr, "Address of DbgCtrl: %08lx\n", &DbgCtrl); PutStr (DbgStr);
sprintf (DbgStr, "Enter the initial DbgCtrl value (hex): "); PutStr (DbgStr);
GetStr ();
sscanf (DbgStr, "%x", &DbgCtrl);
#endif

/*initialize the Am9580*/
```

```
OutByte (0xd6, 0xc2);           /*init DMA*/
OutByte (0xd4, 0x02);
OutWord (CMND_REG, RESET);      /*reset the chip first*/ OutWord (MODE_REG,
MODE_VAL);
sscanf (ReqPtr->InitBPBs, "Am9580.exe%d", &DPBIndex); for
(Cnt=sizeof(XfrIOPB),Ptr = (char *)&XfrIOPB; Cnt >0; Cnt--) *Ptr++ = 0;
XfrIOPB.StdHdr.NextIOPB = 0;
XfrIOPB.StdHdr.IOPBID = 2000;
XfrIOPB.StdHdr.Options = SSRB;
XfrIOPB.StdHdr.CmndCode = LD_DPB;
XfrIOPB.StdHdr.Drive = 0;
XfrIOPB.MemAddr = PhysAddr (&DPBArea[DPBIndex]);
if ((Rtn = XlatErr (ExCmnd (START, &XfrIOPB), GEN_FAIL, ReqPtr)) == NO_ERROR) {
/*read the BPB from the boot sector*/

XfrIOPB.StdHdr.NextIOPB = 0;
XfrIOPB.StdHdr.IOPBID = 2002;
XfrIOPB.StdHdr.Options = SSRB;
XfrIOPB.StdHdr.CmndCode = RESTORE;
XfrIOPB.StdHdr.Drive = 0;
if ((Rtn = XlatErr (ExCmnd (START, &XfrIOPB), GEN_FAIL, ReqPtr)) != NO_ERROR) {
sprintf (DbgStr, "REAL BAD WHEN RESTORE!!!!!\n");   PutStr (DbgStr); return;
}
XfrIOPB.StdHdr.NextIOPB = 0;
XfrIOPB.StdHdr.IOPBID = 2001;
XfrIOPB.StdHdr.Options = SSRB;
XfrIOPB.StdHdr.CmndCode = READ;
XfrIOPB.StdHdr.Drive = 0;
XfrIOPB.StdHdr.Cylinder = 0;
XfrIOPB.StdHdr.Sector = INIT_SEC;
XfrIOPB.StdHdr.Head = 0;
XfrIOPB.SectCnt = 1;
EndDrvr (&EndAddr);
XfrIOPB.MemAddr = PhysAddr (EndAddr.Ofst, EndAddr.Seg); if ((Rtn = XlatErr
(ExCmnd (START, &XfrIOPB), GEN_FAIL, ReqPtr)) == NO_ERROR) {
/*make a copy of BPB and return a pointer to it*/

#ifndef   NO_HDWR
MoveMem (EndAddr.Ofst+BPB_OFST, EndAddr.Seg, &BPBParm, sizeof (struct BPB));
#endif
BPBAddr = (unsigned)&BPBParm;
ReqPtr->InitBPBs = &BPBAddr;
/*return dword end address by assigning offset and segment*/ EndPtr = (struct
PtrAddr *)&ReqPtr->DrvrEnd; EndPtr->Ofst = EndAddr.Ofst;   EndPtr->Seg =
EndAddr.Seg; ReqPtr->UnitCnt = 1;       /*only one unit is supported*/
ReqPtr->UnitNum = 0xF8;    /*media descriptor byte*/ }
}
return;
}


MediaChk (ReqPtr)
/*
It is always assumed that the media has not changed. */

struct MdChkReq
*ReqPtr;
```

```
{
ReqPtr->ChgStat = 1;
RtnStat (ReqPtr, NO_ERROR, 0);
return;
}


BldBPB (ReqPtr)
/*
This routine is called whenever that a preceeding media check call indicates
that the disk has been changed.

It returns a pointer to a BPB.
*/

struct BlBPBReq
*ReqPtr;

{
ReqPtr->BPBPtr = &BPBParm;
RtnStat (ReqPtr, NO_ERROR, 0);
return;
}


IoctlInp (ReqPtr)
/*
This routine will return the contents of the status/command register and
possibly a status result block at the transfer address. */

struct XfrReq
*ReqPtr;

{
unsigned
Status,
*BufPtr;

/* while (((Status = InWord (CMND_REG)) & CFT_VALID) == 0) */ /* some delay may
be needed */
while (((Status = InWord (CMND_REG)) & CMST_MS) != IDLE) /*TEMPORARY!!*/ {
/*wait till it becomes idle*/
}
BufPtr = (unsigned *)ReqPtr->XfrAddr;
*BufPtr++ = Status;
MoveMem (&SRBArea[0], BufPtr, sizeof(struct StdSRB)); RtnStat (ReqPtr, NO_ERROR,
0);
#ifdef   DEBUG
if (DbgCtrl & SHOW_STA)
{
sprintf (DbgStr, "Status register:  %4x\n", Status); PutStr (DbgStr);
ShowSRB (&SRBArea[0]);
}
#endif
return;
}
```

```
Intrpt (ReqPtr)
/*
This routine performs according to the specifications of an "interrupt" routine
for a DOS device driver.  The assembly language interface routine provides it
with its input parameter.
*/

struct ReqHdr
*ReqPtr;                          /*pointer to the request from DOS*/

{
struct StdIOPB
XferIOPB;                      /*IOPB for the HDC transfer operations*/

#ifdef   DEBUG
if (DbgCtrl & SHOW_REQ)
{
PutStr ("Driver entry:\n");
ShowReq (ReqPtr);
}
#endif
((struct XfrReq *)ReqPtr)->XfrSiz &= 0x00ff; /*only handle byte xfr count*/
switch (ReqPtr->ReqCode)
{
case  INIT:
{
Init (ReqPtr);
break;
}
case  MED_CHK:
{
MediaChk (ReqPtr);
break;
}
case  BLD_BPB:
{
BldBPB (ReqPtr);
break;
}
case  INPUT:
{
RVCmnd (ReqPtr, READ);
break;
}
case  OUTPUT:
{
SetIOPB (ReqPtr, &XferIOPB, WRITE);
XlatErr (ExCmnd (START, &XferIOPB), WRT_FLT, ReqPtr); break;
}
case  OUT_VRFY:
{
SetIOPB (ReqPtr, &XferIOPB, WRITE);
XlatErr (ExCmnd (START, &XferIOPB), WRT_FLT, ReqPtr); if (ReqPtr->ReqStat &
ERROR)
{
break;
}
RVCmnd (ReqPtr, VERIFY);
break;
}
case  IOCT_INP:                /*I/O control input*/ {
/*return status register*/ IoctlInp (ReqPtr);        /*and possibly a SRB*/
break;
}
case  IOCT_OUT:                /*I/O control output*/ {
/*used to write an IOPB*/ struct XfrReq
*XfrPtr;

unsigned
Rtn;

XfrPtr = (struct XfrReq *)ReqPtr;
XlatErr (ExCmnd (START, XfrPtr->XfrAddr), GEN_FAIL, ReqPtr); break;
}
case  ND_INP:                  /*requests not supported*/              case
INP_STAT:
case  INP_FLSH:
case  OUT_STAT:
case  OUT_FLSH:
case  DEV_OPN:
case  DEV_CLS:
case  REM_MEDIA:
{
RtnStat (ReqPtr, NO_ERROR, 0);          /*just return OK status*/ ((struct
XfrReq *)ReqPtr)->XfrHdr.ReqStat |= BUSY;       /*and show non-removable*/
break;
}
default:
{
RtnStat (ReqPtr, UNK_CMND, 0);          /*unknown command*/ break;
}
}
#ifdef   DEBUG
if (DbgCtrl & SHOW_REQ)
{
ShowReq (ReqPtr);
PutStr ("Driver exit:\n\n");
}
#endif
return;
}

/*end of drvr9580.c*/
```

Using Am26LS31/32 in High Speed Transmission Line Environments Application Note

April 1988

APPLICATION NOTE By: David Stoenner

ABSTRACT

This article presents the use of high speed serial data communications in a transmission line environment using Am26LS31 and Am26LS32 differential drivers and receivers. The first section describes how a transmission line works and how to properly drive a terminated transmission line, and how to arrive at the proper circuit for an Am26LS31 driver. The final section solves a real world problem of how to drive the serial clock and data at speeds of 15 to 20 MHz using the Am26LS31 and the Am26LS32 with additional support circuitry.

INTRODUCTION

What do a telephone line 10,000 feet long, a coaxial cable 100 feet long, and a PC trace one foot long have in common? First, all of these can transmit voltage and current to an end point. In most cases involving the transmission of voltage and current, dynamic information is present, and the desired result from a system's standpoint is that this information be transmitted with as little distortion, hence error, as possible to the receiving end. Therefore, if the frequency of interest is such that information from the first cell of information can corrupt the second cell, the above three instances are treated as transmission lines.

If in all cases the transit time of the incident wavefront (that is, the wavefront moving away from the transmitter and going towards the receiver) is equal to or greater than 1/20 the time from the first to the second cell of information, then careful attention must be given to how energy is absorbed at the receiver. In free space, electromagnetic energy travels at roughly 1 ns per foot. In a transmission line this value slows down due to the constriction of the EM wavefront (a good rule of thumb is 1.5 ns per foot). In the first example, the voiceband of the telephone is 300 Hz to 3 KHz, thereby, making the time from the first to the second packet of information 333 µs; 1/20 of 333 µs at 1.5 ns per foot causes a telephone cable of 11,100 feet to have problems if it is not terminated properly.

Maximum power transfer also has an effect on the signal to noise ratio of the receiving end, and hence the ability to extract reliable information from all the incident and reflected waves on the line. Maximum power transfer by definition is accomplished when the impedance of the receiver exactly matches the impedance of the transmitter.

The following is a review of the ideal transmission line and its characteristics. The intent is to give the reader a working knowledge and fuel for "If I vary this cause, this effect might happen", rather than a strict mathematical derivation of transmission line theory.

TRANSMISSION LINE BASICS

The ideal transmission line is a differential pair of lines that connect a source (transmitter) with a receiver. Figure 1 shows a schematic of a system assuming that the transmission lines are a pair of wires that exist in free space and that the transmitter is a step function generator $U_0(t)$.

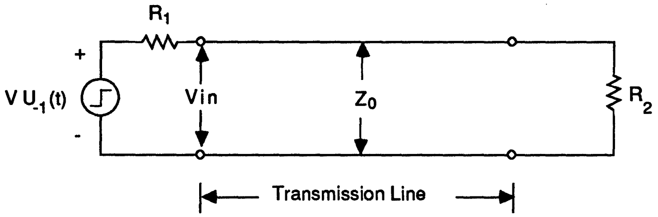


Figure 1

Since the line is made up of real components, the wire will exhibit an inductance in its length and a capacitance across itself as it goes along. By long derivation, the line will have a characteristic impedance given by the formula in Equation 1:

$$Z_0 = \sqrt{\frac{\text{L (inductance per unit length)}}{\text{C (capacitance per unit length)}}}$$

Equation 1

The transit time constant is given in Equation 2:

$$\text{Time/unit length} = \sqrt{\text{L (per unit length)} \cdot \text{C (per unit length)}}$$

Equation 2

The two equations above exclude the idea of series resistance and shunt resistance as loss factors from the equation making the Zo of the transmission line a pure resistance. Series resistance and shunt resistance enter into the equation and cause a phase shift, i.e., Z_0 is a complex impedance in real life and serves to attenuate the signal at the receiving end. This attenuation is normally specified in a value called nepers/meter and is given as a db value.

In reference to the schematic example in Figure 1, what happens when the step function generator steps? The step is impressed across R_1 and enters the line at the input. The signal starts to propagate down the cable at approximately 1.5 ns per foot until it reaches R_2. If this step were made infinitely long, the final value of the voltage on R_2 would be given by Equation 3:

$$V_{final} = \frac{V_{step} \cdot R_2}{R_1 + R_2}$$

Equation 3

Equation 4 shows R1=R2 needed for maximum power transfer:

$$V_{final} = \frac{V_{step} \cdot R_1}{R_1 + R_1} = \frac{V_{step} \cdot R_1}{2R_1} = \frac{V_{step}}{2}$$

Equation 4

The incident wave charging the transmission line must now be considered, as the transmission line is active at the beginning but not active in the final value. Looking at the line as the step function generator stepped, it can be seen that although there is a voltage at the input, there is no voltage at R2; therefore, the voltage impressed on the input of the transmission line is given by Equation 5:

$$V_{IN} = \frac{V_{step} \cdot Z_0}{Z_0 + R_1}$$

Equation 5

Again for maximum power transfer Z0 should be equal to R_1; therefore, Vin = VSTEP/2. When the wavefront moves to R_2, it is at its final value and hence is fully absorbed, and in one transit time of the cable, the cable has reached its steady state final value. This means that for an ideal case, the generator output impedance (R_1) is equal to the cable characteristic impedance (Z_0) which is equal to the termination impedance (R_2), and hence no distortion of the received signal.

In contrast to the preceding theoretical example, the following discussion of a real world consideration will be limited to resistive values and changing R_2. It can be seen that if R_1 equals Z_0 but does not equal R_2, then the cable must assume the final value by hit and miss, which it does by reflections. When the incident wave gets to the termination and it isn't correct, it turns around (reflected) and goes back towards the transmitter. This bouncing back and forth goes on until it reaches the final value and it takes an increment on that final value at a rate of two times the transit time of the cable. This works for one pulse of infinite length, but the situation deteriorates when the pulses become repetitious. The longer the line, the worse it gets and it's previous information past determinate.

In R.F. transmitters which only deal in sine waves, this value is defined as Voltage Standing Wave Ratio (VSWR) and is a measure of how much power is transmitted to and how much actually gets out of the antenna, hence a VSWR of 1:1 is ideal. A VSWR of 1:2 indicates transmission of 100 watts with 50 watts getting out the antenna. In digital signal transmission and reception, this is called ρ(rho), and ρ can be a value from +1 to –1 with ρ=0 being perfect. The equation for ρ is (assuming R_1=Z_0) given in Equation 6:

$$\rho = \frac{\frac{R_2}{Z_0} - 1}{\frac{R_2}{Z_0} + 1}$$

Equation 6

This is a measure of the ratio of the value of the reflected wave value to the incident wave voltage. From this it can be seen that if R_2 is greater than Z_0 then $\rho > 0$, or if R_2 is less than Z_0 then $\rho < 0$.

Plotting these reflections, it can be seen that if $R_2 > Z_0$, the line steps from 0 to .5, and then incrementally to a higher value. Since this looks similar to a charging capacitor, the line is said to look as an excess capacitor whose value is given by Equation 7:

$$\Delta C = \frac{\text{Length of cable} \cdot \text{transit time/unit length}}{Z_0}\left(1 - \frac{Z_0^2}{R_2^2}\right)$$

Equation 7

Conversely, if R2 < Z0, then the line looks like an inductor spike and hence is said to look like an excess inductor whose value is Equation 8:

$$\Delta L = \text{Length of cable} \cdot \text{transit time/unit length} \cdot Z_0 \cdot \left(1 - \frac{Z_0^2}{R_2^2}\right)$$

Equation 8

In some unusual cases, these characteristics can be used to an advantage. It should also be noted that the previous discussion was for a differential transmission line such as a twisted pair or a coaxial cable. The same discussions are also valid for a single ended transmission like a 1one wire above a ground plane (e.g., a PC trace), except that the return path is the ground plane itself which has an extremely low inductance. The Appendix gives equations for Z_0 for various physical conditions of wires and PC traces.

A SPECIFIC LOOK AT TWISTED PAIR TRANSMISSION LINES

The predominant usage of the Am26LS31 and Am26LS32 is in the twisted pair type of transmission lines. A transmission line of two wires running parallel to each other has an impedance which can be calculated as shown in Appendix Figure B. However, the field coupling is not perfect as it is in a coaxial cable. If two signals run close, then the parallel line can cross talk. By twisting the cable over its distance, then the fields are more loosely linked and the cable cross talk is much reduced. This twisting has another effect in that the actual distance is longer and the impedance is controlled by the insulation of the conductors.

The mechanical distances are smaller and yet they still vary. This means that the impedance of the cable varies slightly from point to point but so does its velocity factor. The velocity factor amounts to a cable skew which is a percentage of the cable length times the nominal velocity per unit length and is proportional to the impedance variation. A good rule of thumb seems to be 5%. For example, two cables each 100 feet long would have a skew of 100 x 1.5 x .05 = 7.5 ns of uncertainity as to which wavefront would get there first. For the rest of this discussion, this 5% number will be used in the calculations.

An additional factor is that the actual wire length of a cable is longer than the physical cable length of twisting. For example, if a 28 gauge twisted pair cable is constructed using 12 twists per foot, then the actual wire length is not one foot but one foot plus 12 times the one wire diameter or 12.4 inches. This represents a 3% growth in the cable.

Am26LS31/Am26LS32 INTRODUCTION

The Am26LS31 and Am26LS32 were developed by AMD in 1978 in order to meet the requirements of a then new engineering standard RS-422. During that time, RS-232 was the prevailing standard for terminal communications. It was an old standard and therefore, a single-ended interface and as such was limited to 50 feet per the original RS-232 specification. Terminals were mostly 1200 baud with a few at 9600 baud so a new standard at 1 megabits and 1000 feet was a fantastic idea. RS-422 found plenty of places to operate and work but it never did replace the RS-232, since 9600 and 19.2K band seemed all the market really needed. The disk world was another story. Disks have an insatiable appetite for frequency and what was 5 to 9.2 MHz in 1978 is now 9.2 to 20 MHz today. However, the first law of thermodynamics cannot be violated and what was satisfactory in 1978 at 9.2 MHz cannot be tolerated today in a design for 15-20 MHz.

The rest of this discussion will consider how to make the Am26LS31 and Am26LS32 work reliably in an area that they were not designed to work. (The design max for the Am26LS31/32 was 10 MHz, that is, 10 times the max RS-

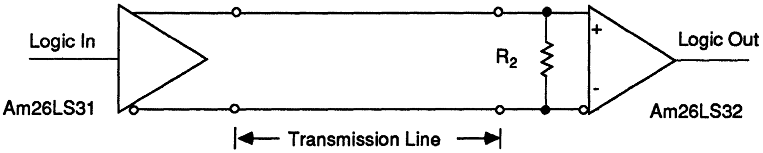
422 spec). Concentration will be on the Storage Module Device (SMD) interface although the discussion can easily be extended to any application of high frequency data with a separate clock signal or where data and clock are combined such as in Manchester encoding.

APPLYING Am26LS31 AND Am26LS32 TO THE TRANSMISSION LINES

Before applying the previous transmission line theory to make the Am26LS31/32 work correctly, both devices will be considered and characteristics reviewed – the Am26LS31 as a driver and the Am26LS32 as a receiver.

The Am26LS32 is basically a differential voltage amplifier with a front end attenuator (voltage divider) which can tolerate a common mode voltage of + or –7 volts and still amplify a differential signal of 100 millivolts and produce and logic 1 or 0 depending on which direction the + and – inputs are with respect to each other. With an input impedance guaranteed to be above 6.8K Ω, it will not load any transmission's line termination of the usable 100-200 Ω. It is basically a very passive device that needs no relative ground reference.

The Am26LS31, on the other hand, is a much more complicated device from the analog standpoint which takes an input logic level of 1 or 0 and has two outputs that work out of phase of each other. When a 1 is placed at the input, the + output is at a high TTL level and the other is at a low TTL level. When a 0 is at the input, then the reverse is true at the output. The output stages of the Am26LS31 are the basic TTL totem pole outputs which although good for logic, is a poor transmission line driver. The reason is that when the output is going from a 0 to a 1 level, it is being driven by an emitter follower whose output impedance is on the order of 50 Ω, while the driver going from 1 to a 0 is about 5 Ω, that of a saturated switch. This discontinuity in impedance makes driving the transmission line a problem. It should also be noted that the rise time is slower than the fall time. An equivalent circuit concept is shown in Figure 2, the real life circuit and Figure 3, the equivalent circuit in standard voltage generator concepts.



#10716A-002

Figure 2

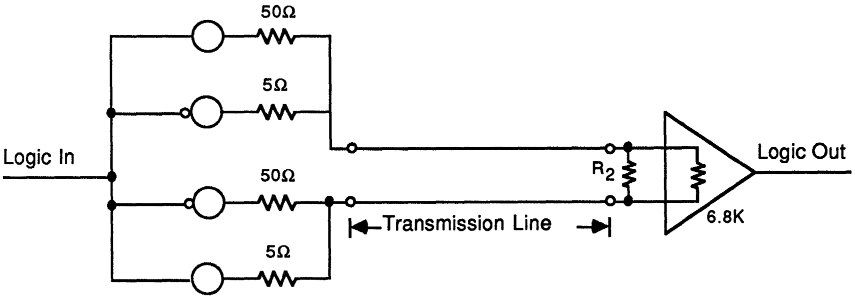


Figure 3

From the preceding discussions, an R2 could easily be selected that would terminate the line in its proper Z_0, but how can the driver be made to conform to the proper value for maxium power transfer and minimum reflections? First, it should be noted that most transmissions lines in use, that is, twisted lines, are in a range of 100 to 130 Ω for their Z_0. In order to match the cable, maximum power transfer indicates that source impedance is needed to be equal to the transmission line impedance. Since the output impedance is less than all the Z_0s , a series resistor can be added to each leg to balance the value. The Thevenin Theorem states that the lumped resistor to both legs can be split and achieve the same effect. The equation for that series resistor would be as given in Equation 9:

$$R_{\text{series}} = \frac{Z_0 - (R_{\text{low to high}} + R_{\text{high to low}})}{2}$$

Equation 9

Therefore, if a Z0 of 100 is assumed, a series resistor of 22.5 or 22, the nearest 5% 1/4 watt resistor value in each leg would be added, thus making the new circuit look as follows in Figure 4.

This solves the reflection problem and allows the cable to be driven so that the limiting factors are now the drivers and receivers themselves; however, this has been done at the expense of signal attenuation. What used to be a 3 volt differential signal is now been reduced to a 2.0 volt signal, that is, if the cable is 0 feet. As the cable is extended in length, this attenuation gets worse. It should also be noted that a twisted pair cable over about 10 feet long starts appearing capacitative regardless of how it is terminated, so the 22 Ωs and capacitance entered into a theoretical maximum upper limit is due to the cable alone, as shown in Equation 10.

$$F_{\text{max}} = \frac{160\text{K}}{2 \times 22 \times \text{Capacitance (in microfarads)}} \text{(Hz)}$$

Equation 10

For example, if the cable reached as high as 100 picofarads, Fmax would be 36 MHz. Although this sounds high, it represents a significant phase shift at 20 MHz and this phase shift value will be needed in the next section.

Another effect of the series resistors should be noted. As discussed before, the rise and fall time of the Am26LS31 driver is different because of the two different types of drivers, emitter follower versus saturated switch. This rise and fall time is exaggerated even more if the output of the driver is connected directly to the transmission line resulting in a difference in skew at the receiver called an "eye" pattern. The series resistors help in giving a more uniform rise and fall time at the cable input and thereby closing up the eye which in effect gives a better signal to noise ratio on the line.

USING THE Am26LS31 AND Am26LS32 IN CLOCK AND DATA APPLICATIONS

The previous discussion was for one signal. If that signal was data only, then the clock would have to be extracted from that signal and regenerated at the receiver end. In many applications and SMD in particular, clock and data are sent over separate sets of wires. In this case, a method of eliminating time skew from the system is needed. The best method is to clock data onto the transmission line with one edge of the clock and clock data off the transmission line with the other edge of the clock. The scheme is shown in Figure 5.

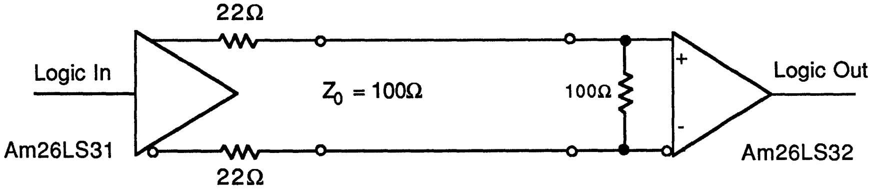


Figure 4

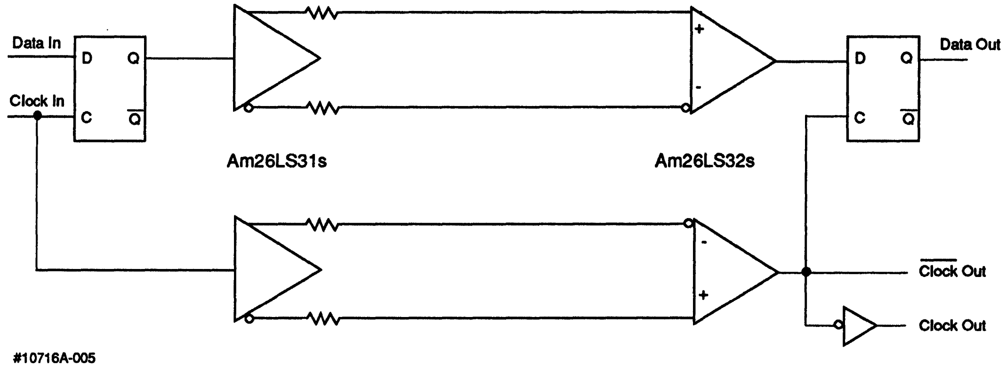


Figure 5

At the transmitter, the rising edge of the clock moves data from D to Q on the flip-flop. Clock and data then propagate across the Am26LS31 down the transmission line and are received by the Am26LS32s at the other end. Note that the clock receiver is inverted in polarity at the Am26LS32 from that of the data receiver which effects clocking the data on the clock-in's falling edge into the receiver flip-flop. For this system to work, the set-up time requirement of the receiver flip-flop must be met. However, skew is introduced by each stage along the way, thus reducing the 1/2 clock time by significant values. There is a clock to Q time of the flip-flop, a skew of the driver, a cable skew, a receiver skew, and finally the set-up of the flip-flop. These must all add up to or be less than the minimum 1/2 clock high time. The maximum delay is given in Equation 11:

$$\frac{\text{CLK Period Min}}{2} \geq \left\{\begin{matrix}\text{clk to Q}\\ \text{Flip Flop}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{skew}\\ \text{Driver}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{skew}\\ \text{Cable}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{skew}\\ \text{Receiver}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{ setup}\\ \text{Flip Flop}\end{matrix}\right\}$$

Equation 11

To examine the following equation, let's assume a 74AS74 flip-flop at each end and that a 10 MHz clock is used. A 74AS74 CLK to Q is 9 ns and set up is 4.5 ns. A 10 MHz clock period is 100 ns, its 1/2 period is 50 ns and its minimum is usually 10%, thereby making it 45 ns. This now makes Equation 11 reduce to Equation 12:

$$45 \geq 9\text{ns} + \left\{\begin{matrix}\tau\text{ skew}\\ \text{Driver}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{ skew}\\ \text{Cable}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{ skew}\\ \text{Receiver}\end{matrix}\right\} + 4.5\text{ ns}$$

OR

$$31.5 \geq \left\{\begin{matrix}\tau\text{ skew}\\ \text{Driver}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{ skew}\\ \text{Cable}\end{matrix}\right\} + \left\{\begin{matrix}\tau\text{ skew}\\ \text{Receiver}\end{matrix}\right\}$$

Equation 12

The AMD data sheet sets the skew time of the driver and the receiver to be 6 ns each if they are in the same package leaving the skew time of the cable to be 19.5 ns. This 19.5 ns is divided between the impedance difference between two pairs of lines and the previously calculated pole frequency of the 22 Ω source termination resistors and the cable capacitor. If it is assumed that the impedance variation between lines is 5%, then the transmit time varies 5% also, so that delta is a function of length as shown in Equation 13:

$$\tau\text{skew}_{\text{Cable}} = \text{Length of Cable} \bullet \text{Transit Time} \bullet \text{Impedance Tolerance}$$

Equation 13

The second factor is the capacitance phase shift. In this example with a pole at 34 MHz, the phase shift at 10 MHz is a 16 degree phase shift which represents 1.4 ns skew, thereby making the maximum cable length at 10 MHz to be 240 feet at 5%, or 120 feet at 10% cable variations.

A similar calculation could be done at 20 MHz showing that the parts are marginal with the skew time of the drivers and receivers as the predominant contributor. By careful screening of this parameter, the Am26LS31 and Am26LS32 could be used up to 20 MHz. Beyond 20 MHz no driver or receiver will really handle the job in TTL, and ECL should be considered as the method of choice. However, an improvement can be made on the previous circuit. In examining the circuit it can be noted that the clock drives the cable before the flip flop's output has had a chance to reach its own driver. The 9 ns is a significant value at 10 to 20 MHz as a compensation series of gates could be added in the clock line to balance the clock to Q time of the flip-flop. This would take the form shown in Figure 6.

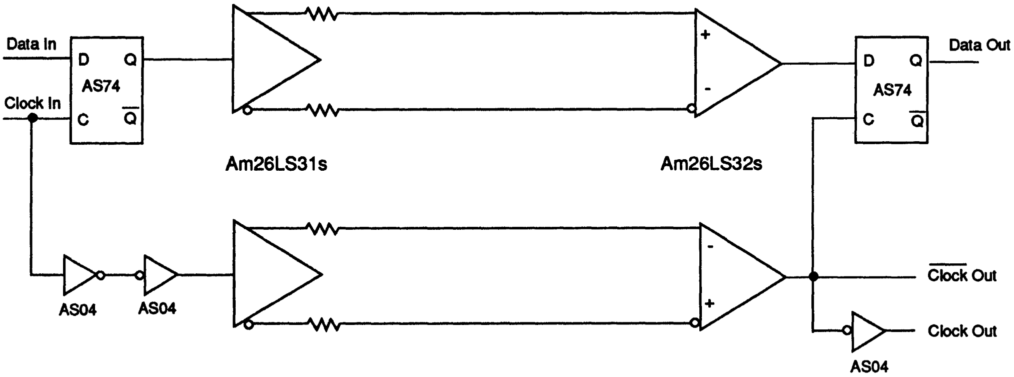


Figure 6

Now the original equation is modified by the fastest time of the series gates allowing a longer cable at a higher frequency. The 74AS04's minimum time is 3 ns each, allowing that the maximum cable skew be 25.5 ns, and allowing reliable operations of the circuit at 20 MHz with a 56 foot cable. Without the 74AS04s, it would not work at all in the worst case.

GREEN WIRE CONSIDERATIONS IN A SYSTEM

All the above examples deal with one driver driving one or more receivers over a twisted pair cable. Due to FCC requirements, if that cable leaves the cabinet and goes to another cabinet, it will more than likely have to be shielded to reduce any radiated EMI. Because of UL, the cabinets must be tied directly to the green wire of the incoming power cable. That green wire represents earth ground. If the cabinets are in close proximity, there is a good chance that the green wire in both cabinets are on the same power circuit and hence at the same potential voltage. But if they are separated by too great a distance, then a good possibility exists that they are not on the same power circuit and hence may have a potential different voltage between them. When this happens, a fault current flows between the two cabinets down the shield and this can be a fairly large value, for example exceeding 1/2 amp. However there is a simple matter of just breaking the DC path and adding a series capacitor of .1 microfared in series with the shield. This eliminates the 60 Hz fault current but provides termination for the high frequency AC shield needed for FCC.

However, designers should be aware of another path. UL requires also that the circuitry of a computer system be returned to green wire ground which is normally accomplished by connecting in the power supply green wire ground with logic ground. This is fine in most cases, but in reviewing the schematic in Figures 4 and 5, the signal is sent over differentially and not referenced to logic ground. The green wire fault voltage now appears as a common mode voltage to the Am26LS32 receivers, and since its input impedance is at least 6 KΩ, nothing really happens.

There are cases where multiple Am26LS31s are 3-stated onto a cable from various sources to one or more receivers. In such cases, care must be taken since there is no special isolation. At about 5 volts RMS green wire fault voltage, the Am26LS31s break down. Current again is only limited by series resistance of the cable except now it is not flowing harmlessly down a braided shield, but down the semiconductor itself in a manner current was never meant to flow.

Figure 7 shows the breakdown path. Any series resistors added as shown in Figures 4, 5 or 6 would limit the current below harmful limits. It must also be pointed out that Electrostatic Discharge (ESD) can also cause the same problem on a single driver situation alone. The best system solution in this environment is to have only one green wire to logic ground connection in the system and all logic grounds tied together by a single wire in the distributed cable. This eliminates any fault current in the Am26LS31s. ESD can then be handled by adding a balun transformer as necessary.

CONCLUSION

In summary, this article has shown that by applying a few simple rules and careful worst casing in the use of lines between devices, a reliable system can be produced using readily available technology. One must take advantage of the laws of physics and not ignore them.

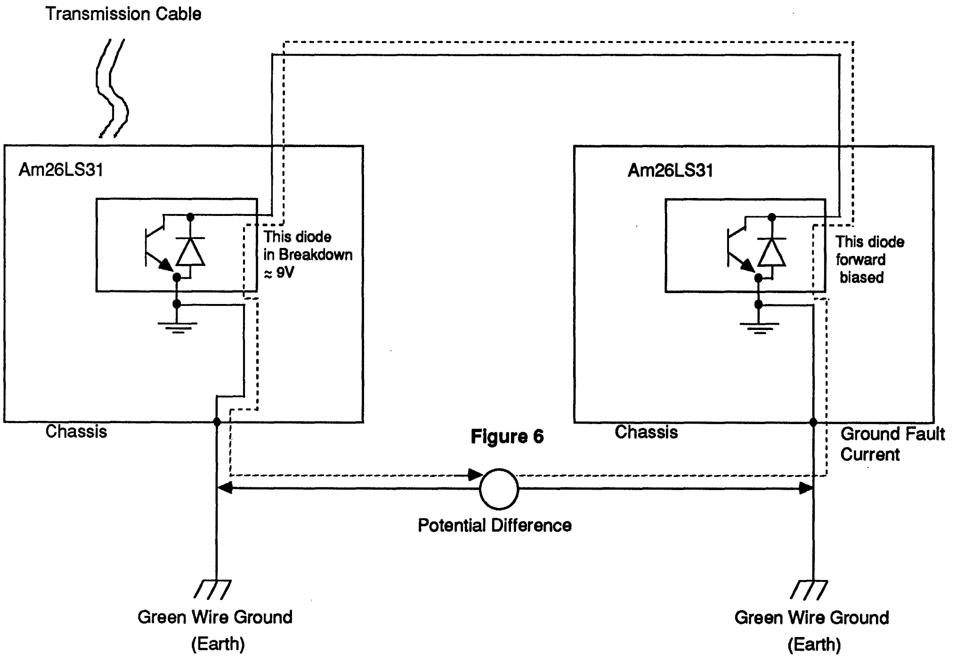


Figure 7

REFERENCES

1. *Pulse Digital And Switching Waveform*, Millman and Tabb, McGraw Hill, 1965, Chapter 3, Sections 12 through 19.

2. *Electronics,* March 6, 1967, pp. 165-168.

3. *Electronic Design*, May 24, 1976, "Technology, Wiring For High Speed Circuits".

4. *EDN*, June 14, 1984, "Transmission Line Effects Influence High Speed CMOS".

5. *Electronics Engineers Handbook*, Donald Finic, McGraw Hill, 1975 pp. 6-35

Physical Constraints of Free Space

INDUCTANCE $M_0 = 4 \cdot \pi \cdot 10^{-7}$ HENRIES/METER

CAPACITANCE $E_0 = 8.85 \cdot 10^{-12}$ FARADS/METER

Transmission Line Equations

$$Z = \sqrt{\frac{L}{C}}$$

where L = Inductance/Unit length
where C= Capacitance/Unit length

$$T = 3.3 \cdot E_r^{1/2} \cdot E\text{-}9$$

where T = Time in meters/seconds

E_r = Relative dielectric constant

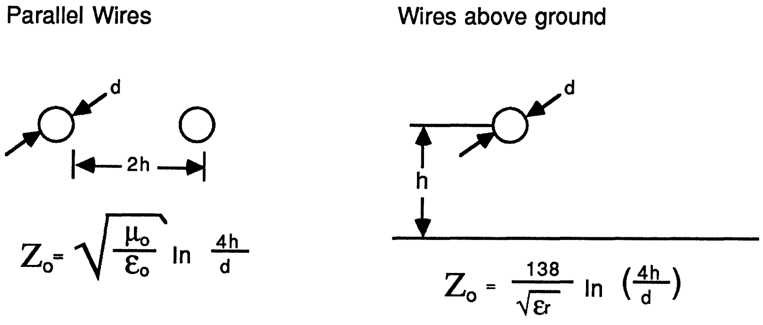
$$E_r = \frac{\text{Actual dielectric constant}}{E_0}$$

E_r for various materials

FR4 glass pc board	4.7
Mica	5.4 - 8.7
Glass	7.2
Polypropylene	2.2
Nylon	3.5
Polystrene	1.2

#10716A-008

Figure A

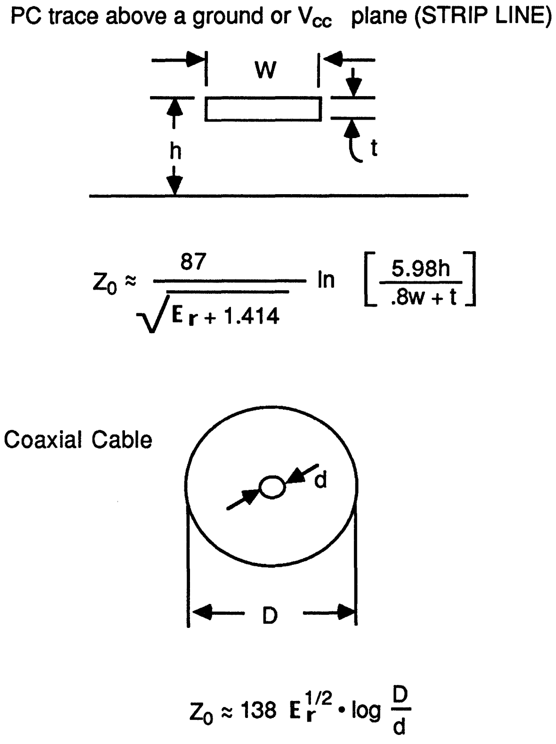


$$Z_o = \sqrt{\frac{\mu_o}{\varepsilon_o}}\ \ln \frac{4h}{d}$$

$$Z_o = \frac{138}{\sqrt{\varepsilon r}}\ \ln \left(\frac{4h}{d}\right)$$

#10716A-009

Figure B



$$Z_0 \approx \frac{87}{\sqrt{E_r + 1.414}}\ \ln \left[\frac{5.98h}{.8w + t}\right]$$

#10716A-010

$$Z_0 \approx 138\ E_r^{1/2} \cdot \log \frac{D}{d}$$

Figure C

DESIGN APPLICATIONS

ELECTRONIC DESIGN EXCLUSIVE

Constant-density recording comes alive with new chips

Mark S. Young
Advanced Micro Devices Inc., 901 Thompson Pl., Sunnyvale, CA 94086; (408) 749-3409.

Designers of computer systems keep shooting for the highest possible disk capacity. Engineers are exploring various recording techniques, such as data encoding schemes and special disk-track formats. Constant-density recording, one type of disk-track formatting, can boost raw disk capacity, or capacity minus overhead, by 15 to 35%. More important, this gain is on top of any other techniques applied, like run-length-limited encoding.

Until now, constant-density recording has been little used because of its high cost and the difficulty of putting it to work. Among those difficulties are the requirement for a phase-locked loop (PLL) capable of operating at any one of several different frequencies, the need to change the disk's rotation speed, the narrow frequency range of read-channel circuits, and the ability to use a disk controller with disk drives that do not have a built-in data separator.

With fresh chips on the scene, disk data-recording capacity increases over 25%. It takes just one peripheral card and a handful of parts.

Those difficulties are overcome through use of the Am9582 disk data separator (DDS) chip. With that chip, plus the Am2971 digital frequency-generator circuit, designers can add constant-density recording to disk controllers at minimal cost, while boosting data capacity by over 25%.

Most disk controllers apply constant-angular velocity recording. For that type of recording, the disk controller writes data to the disk at a constant frequency, giving each data-bit cell the same angular velocity. Although each cell has the same angular displacement, they have different linear velocities on each track to make up for the physical size differences in the individual bit cells.

As a result of its particular physical position, each track holds data bits of different sizes (Fig. 1). The data on the disk is read or written at a fixed frequency and all tracks store the same amount of data. In constant-angular velocity recording, the minimum distance must be calculated for the innermost track, where the data-bit cells will be closest together. As the read-write heads move toward the outer tracks, the data bits move farther apart, reducing the average bit density. Constant-angular velocity recording is popular because it simplifies the disk drive and controller electronics, as well as the system software.

Constant-density recording, on the other hand, varies the recording frequency on each track to maintain a constant physical-data density. This is done by varying either the recording frequency or the disk platter rotation speed. The physics of magnetic recording, however, dictate the maximum flux transition rate, or rather, the minimum distance between adjacent data bit-cell transitions.

With constant-density recording, the distance between data-bit cells is the same for all tracks, maintaining denser, more efficient data packing than constant-angular velocity recording. The amount of data on each track is therefore different, with the outer tracks containing the most data and the inner tracks containing the least. This technique is more efficient than constant-angular velocity recording because it comes closer to recording to the maximum capacity of the disk. With constant-density recording, 15 to 35% increases over the capacity of standard constant-angular velocity recording can be achieved.

PROBLEMS OF CONSTANT DENSITY

While it uses the disk more efficiently than constant-angular velocity recording, constant-density recording creates a number of problems that can dramatically complicate the design and cost of the disk controller. The largest obstacle is the creation of either a variable-frequency PLL or one that operates at several discrete frequencies. The most common technique calls for the designer to multiplex separate PLLs operating at the re-

Reprinted from Electronic Design, Nov. 13, 1986.

quired frequencies.

Because the data frequencies in floppy disks are so low, designers sometimes build digital PLL circuits that are tunable to different frequencies. Another technique is to change the disk's rotation speed and to operate the PLL at one frequency. Unfortunately, since it takes so long for the rotation speed to change, the disk I/O rate is affected.

Head-flying characteristics make it undesirable to alter the motor speed of Winchester disk drives. In addition, with constant-speed ST-506 drives (the specification for most of the new 5¹/₄ in. Winchester disks), this technique cannot be used. As a result, constant-density recording has been used more extensively in floppy drives than in Winchesters.

Constant-density recording can be included only in a disk controller with disk drives that do not have a built-in data separator. Unless that type of recording is installed by the drive manufacturer, it is not easily applied to enhanced small device interface (ESDI) or storage module device (SMD) disks. Accordingly, only ST-506 interface-compatible Winchester and floppy drives can easily use such a circuit, because the disk controllers on those drives include the data separator circuit. SCSI-compatible disk drives with the entire disk controller built onto the drive, including the data separator, can also take advantage of this circuit.

The read-channel circuits, with their narrow frequency range, may not have the bandwidth necessary to handle the higher frequency requirements of constant-density recording. For example, the ST-506 interface requires operation at only 5 MHz, and constant-density recording may push the data frequency up to 8 MHz or more. Fortunately, the technology being used in many Winchester disks today is sufficiently advanced to support those frequencies.

Since the flying heads in Winchester drives are optimized for a particular height of operation, using constant-density recording with them may pose a problem. Toward the outer edge of the disk, the data density is higher than with constant-angular velocity recording. When constant-density recording is employed, the read-write heads tend to fly higher because of the design of the air bearing, reducing their data recording density capability. Moreover, the manner in which the oxide magnetic medium is sputtered onto the disks causes the coating to be thinner at the outer tracks. This also may affect the ability of the disk to store the higher data density.

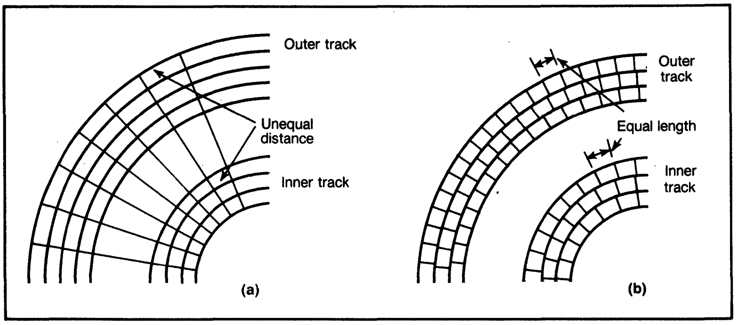
CONSTANT-DENSITY CLOCK GENERATOR

A complete disk controller circuit for constant-density recording requires only a few chips, with the Am9580A hard disk controller (HDC) and the data disk separator forming the controller's core (Fig. 2). Fewer than 12 ICs are required to build a complete Winchester controller that fits onto a half-size IBM PC AT peripheral card. The system interface requires eight additional ICs at most, mostly data and address transceivers.

To overcome the limitations of previous techniques, the data-separator circuit must operate at the various recording frequencies needed for constant-density recording. The first critical element in such a design is the PLL synchronizing the data read from the disk. It must operate at several different frequencies without a complex set of external analog components. Since most PLLs operate at one frequency with those analog components, the multiple-frequency requirement calls for some form of analog multiplexer to connect those different components to the PLL. The separator chip includes a PLL that operates at different frequencies without using any external analog components.

The PLL in the separator operates at any frequency between 4 and 16 MHz for Winchester disks, and from 125 kHz to 1 MHz for floppy-disk drives, with no external components that depend on frequency. The user supplies only a reference input clock and must wait less than 3 ms to allow the PLL to stabilize at the new frequency.

Instead of requiring a complex and expensive variable-frequency analog reference clock generator, this constant-density recording circuit includes a digital clock-generator circuit, the Am2971 programmable event generator (PEG) chip (Fig. 3). The chip, a general-purpose timing device, acts as a programmable timing and waveform generator. It digitally synthesizes multiple waveforms, with up to 32 10-ns segments. A total of 12 programmable output pins are available. Even with such a ver-



1. In constant-angular velocity recording, the physical size of the data bits on the inner tracks is smaller than those on the outer tracks (a). With constant-density recording, data on all tracks has the same density (b). Overall, constant-density recording boosts disk capacity by up to 35%.

satile clock generator, the number of clock frequencies must be reduced to a manageable number.

Rather than each track having a slightly different recording frequency, the disk surface is divided into recording zones. Each zone consists of a group of tracks, with every track in that zone recorded at the same frequency. Although the density of each track is extended as its track radius increases, the change in physical data space as the read-write heads move from one track to the next is relatively small. The change is only about 0.1 to 1% of the track capacity.

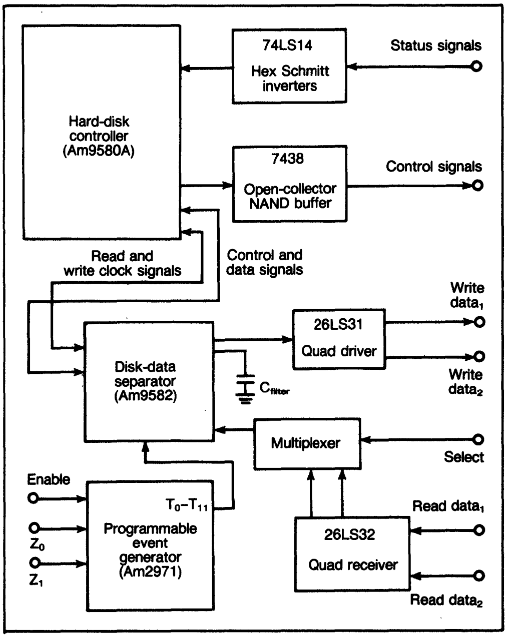
Disk controllers and computers organize data on a disk track in data blocks, or sectors, each containing 256 bytes or more of data. Any gain in stored data that is less than a full sector cannot be used. A useful simplification is to divide the disk surface into zones (groups of contiguous tracks). That supplies enough additional space to let the disk controller add one or more new sectors to each track in the zone. This zoning technique dramatically reduces the number of required frequencies to only a few (usually eight or less).

In this application, the surface of the disk is divided into four equal zones. Assuming that the innermost zone (zone 0) operates at 5 MHz, all other zones will be changed in frequency according to the greater recording space available in the outer zones. The highest allowable frequency for each zone is proportional to its radius. This design optimizes the density improvement by approaching the highest allowable zone data-recording frequencies as closely as possible.

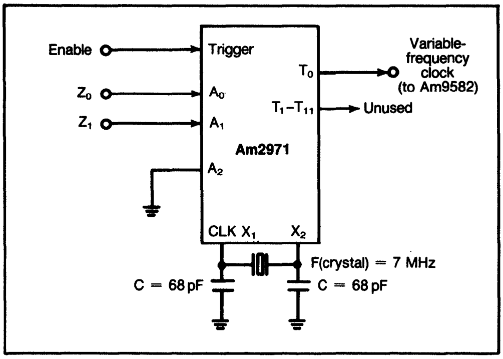
Several restrictions were imposed on this design to simplify the circuit and to limit costs. One limits the maximum operating frequency of the clock-generator circuit. Although the frequency-generator chip operates internally at frequencies up to 100 MHz, the external crystal frequency was limited to 20 MHz. A clock multiplier in the chip allows internal operation to reach 100 MHz. The problems of high-frequency operation are solved within the PEG chip, including timing skew problems between signals.

In this application, a 7-MHz crystal frequency creates a 35-MHz operating frequency in the frequency-generator chip. Correctly programming the chip's outputs generates clock waveforms at frequencies that closely approach the theoretical maximums for each recording zone. The large frequency range of the frequency-generator chip gives the user great flexibility in choosing an operating frequency up to 70 MHz.

Within the maximum 32-step sequence of the frequency-generator chip, four different waveforms can be programmed into the TO output (Fig. 4). From this pin, depending on the zone selected, one of the four waveforms can be selected. Those waveforms are generated by a PROM sequencer with up to 12 programmable outputs that can be set at either a 1 or a 0. In this application, four different sequences are programmed into the address sequencer of the PEG. Once the PEG chip is locked into one of the four sequences, it will continue until interrupted or halted. Accordingly, as the PEG chip steps through one of the four sequences, it supplies a constant-frequency clock



2. When using the Am9580A hard disk controller and the Am9582 disk data separator as a core, all the chips required for a Winchester drive fit on a half-size IBM PC peripheral card.



3. A variable-frequency generator chip (Am2971) supplies programmable output frequencies. Those outputs are digitally synthesized and referenced to a crystal oscillator. They are programmed by selecting the appropriate input lines.

signal output.

This design automatically allows a choice for either floppy disks or Winchester devices. When the disk controller tells the disk data separator that a floppy drive has been selected, an internal clock divider changes the reference input frequency to meet the operating frequencies of the floppy drive. Some changes, however, may be required to the frequency generator chip's internal programming because individual floppy drives operate at different frequencies.

Besides the Am2971 chip, the user must supply three bits of a parallel output port. Three signals, Enable, Z0, and Z1 operate the constant-density recording circuit. The Z0 and Z1 pins select the correct zone frequency, and the Enable signal shows when the zone selection signals are valid and enable the clock generator logic.

SOFTWARE CONSIDERATIONS

Dividing the disk surface into several zones does impose some restrictions on the normal operation of the disk drives. The computer operating system must be made aware of the various storage capacities of the different tracks. As a result, before performing a disk I/O operation, the disk device driver software must set up the proper zone value on the zone-selection port and adjust the disk controller parameters to the proper sector count per track in the disk controller.

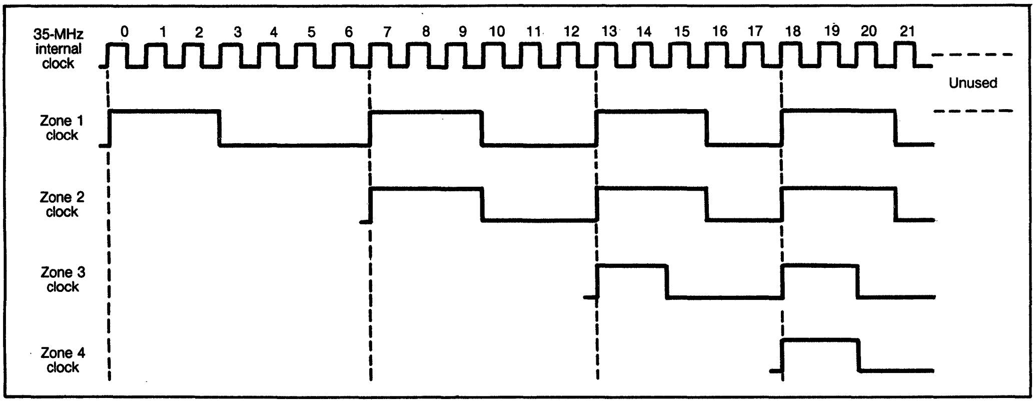
Moreover, the disk operation must be checked to ensure that it does not change zones. Disk controllers such as the Am9580A can process up to 256 sectors in one command, spanning multiple tracks and heads. If the disk I/O operation crosses a zone boundary, the disk command must be broken into two separate commands. This allows the system microprocessor to change the zone values between the commands. The likelihood of that occurrence can be reduced by modifying the computer's file system to avoid creating files that straddle the zone boundaries.

Several other restrictions occur in a constant-density system, including zone changes that require PLL stabilizing time, and continuous alteration of the disk controller operating parameters. Changing zones and, therefore, the operating frequency in a constant-density recording design requires that the PLL in the data separator be stabilized at the zone frequency. This usually takes about 1 to 5 ms.

The greater the difference between the previous operating frequency and the new zone frequency, the greater the stabilizing time. In many disk controller systems the amount of time required to continuously update the parameters is insignificant. All that is required is some additional software.

The PLL in the Am9582 requires substantially less than 3 ms (worst case) to change frequencies. To prevent the disk I/O time from being affected by this switching time, the control software should always set the new zone operating frequency prior to performing a "seek" (or read-write head movement) to a new zone track. Since the mechanical seek time on most disk drives is much longer than 3 ms, the PLL stabilization time should not actually affect the disk I/O functions of the controller. The controller designer should check that this PLL stabilization time requirement is met for the drive's different uses. □

Mark S. Young is a product planning section manager at AMD. His responsibilities include preparing specifications for and designing microprocessor peripheral chips. Young received a BA in computer science from the University of California, Berkeley.

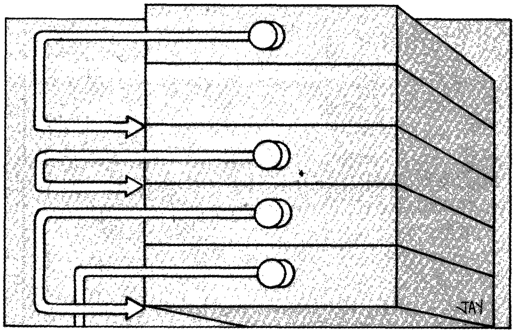


4. The digital clock synthesizer generates four different clock waveforms. Each waveform is repeated within each zone. The 35-MHz clock frequency comes from the crystal oscillator.

CONTROLLERS WRING PEAK PERFORMANCE OUT OF DISK DRIVES

Elevator sorting, disk cache, and random data storage techniques, implemented with the help of advanced disk controllers, minimize performance limitations due to head positioning time.

by Mark S. Young



Today's disk subsystems provide large amounts of memory and fast access with minimal cost, at levels designers could only dream about a few years ago. Continual increases in track and linear bit density—and, as a result, in transfer speeds—combine with decreases in access times to improve performance of disk products. Disk controllers and operating systems have also enjoyed steady gains that increase speed. Improvements in processor speeds, however, have led system integrators to demand still higher throughputs.

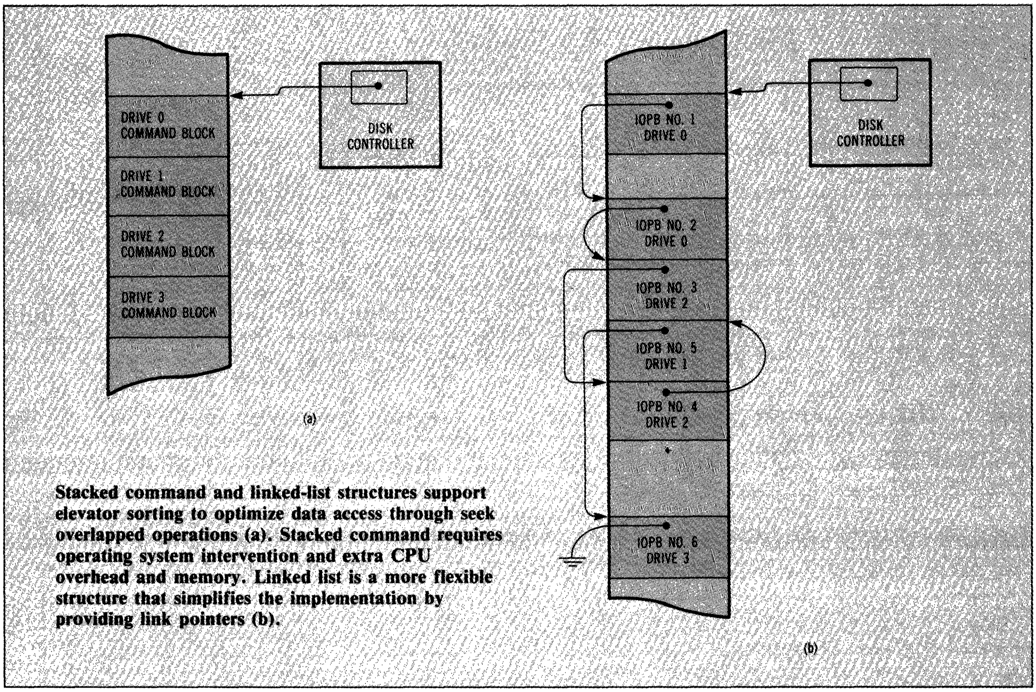
Yet, several design techniques can still yield better overall data throughput. Typical disk drives require between 90 and 30 ms for a seek operation. Since the read/write head positioning, or seek, consumes the most time in a disk I/O operation, controller and operating system designers have focused their efforts there.

In a multidrive system, controller design can change the sequential series of disk access operations into parallel ones. A typical 5¼-in. Winchester, using the ST506 interface, can accept head movement pulses about once every 10 ms (1 pulse means to move the read/write heads from one track to the next). A read/write head requires up to 3 ms to move from track to track, depending on how far it has to move. Most small Winchester disks do not require selection while read/write head positioning is taking place. After the controller issues a complete head movement command, therefore, it can turn its attention to another drive while the positioning operation occurs.

Optimizing seek operations

To execute disk operations on different drives more or less in parallel, the controller looks at multiple disk commands from the system and isolates the commands to the different drives. After determining the operations that are required, the controller issues necessary commands to the drive. It

Mark S. Young is a product planning engineer at Advanced Micro Devices (Sunnyvale, Calif). He holds a BA in computer science from the University of California at Berkeley.



Stacked command and linked-list structures support elevator sorting to optimize data access through seek overlapped operations (a). Stacked command requires operating system intervention and extra CPU overhead and memory. Linked list is a more flexible structure that simplifies the implementation by providing link pointers (b).

then processes another drive's seek operations while waiting for the current drive's command to be completed. Average access time to all the disks is substantially improved. In a four-drive system, this could be a gain of as much as 400 percent.

The success of seek-overlap operations, as these parallel seek operations are called, depends on random disk file request. If the computer's operating system tends to access files on a single disk consistently (in a multidrive system), then seek-overlap has little advantage. Maximum benefit is gained when files are stored on a randomly chosen disk. In some Unix operating systems, for example, the disk file manager randomly selects a disk on which to store a newly created file. This ensures the disk controller will receive I/O access requests that are distributed among the various drives.

Since files are not usually accessed in a truly random manner, the randomizing technique may require adjustments. By keeping track of all file accesses over a certain period, the operating system can redistribute the files among different disks. Although extra disk accesses and additional computation overhead are required, access time improvement can be significant in a heavily loaded system.

Another technique, called elevator accessing, can also reduce the average seek time. This technique sorts track access operations into two groups: seeks that move steadily toward the disk spindle (towards the inner tracks) and those that move steadily away from the spindle (towards the outer tracks). In this grouping, each successive access is in the same direction. The read/write heads move first in one direction and then the other, accessing requested data as they go. This eliminates back and forth motions of the head. The sorting function generally is handled by either the disk file controller or the computer's operating system.

A typical series of disk seek requests, if executed as received, might require a total seek distance of some 556 tracks. Sorting the incoming requests numerically could reduce this to about 283 tracks—a 50 percent reduction in seek distance and time.

File dependencies, however, mean full performance benefits cannot always be gained from optimal elevator sorting. If one request requires a write operation on a track and a later operation requires a read, the elevator sort must maintain the correct sequence of events. Such file dependencies can be controlled by designating which accesses must occur first in multiple data requests from the same file. One method is to time stamp disk access requests and always give priority to the request with the earliest time.

The elevator sorting algorithm, like seek-overlap operations, will only work if the controller can queue commands long before they will be needed. The stacked command structure is a commonly used technique that supports these functions. An area in system memory is used to store disk I/O commands, usually one command per drive. The controller can easily initiate a primitive seek-overlap algorithm by allowing up to four disks to have commands pending. Elevator sorting must be performed in the file manager or a separate disk I/O processor. To guarantee continuous use of the controller, the file manager must also update the command structure whenever a disk is free. A disadvantage of the stacked command structure is that operating system intervention is required each time a disk I/O command is completed. Thus the controller would introduce extra CPU overhead.

The linked list command chain offers a more flexible command structure. A disk command chain, created using disk command blocks (I/O parameter blocks), is placed either in consecutive memory locations or is strung together as a linked-list data structure. Although this structure imposes some extra overhead, such as pointers to commands in the list, it maximizes system flexibility when integrating the disk controller with the operating system.

Zero-interleaved data transfers improve data I/O operations, as well as speed disk access times.

The operating system creates the command structure and the disk controller executes it. The linked-list structure allows IOPBs to be placed in the command chain well before execution without imposing size limitations. Since a queue of disk I/O commands is available to the disk controller, it simplifies the implementation of a seek-overlap function in the disk controller.

Moreover, it allows the disk controller to start searching the command list for the command that starts the next seek as soon as one I/O operation finishes. This keeps the drive constantly busy without system intervention. The linked-list structure also speeds execution of the elevator access algorithm. Because the operating system can create the commands and then sort the seeks without moving commands around in system memory, only the link pointers have to be altered.

Disk I/O operations can also be improved by continuously transfering data to and from a track. This method, called zero-interleaved data transfers, sequentially accesses physical sectors on the disk. Many controller designs require sectors on a track to be interleaved. Interleaving calls for specific amounts of physical space between logically adjacent track sectors. It also provides time between processing of the data to (or from) the disk and the arrival of the next desired sector on the track.

Data buffering and high speed control capability can eliminate the need for interleaving. One method uses first in, first out buffers to allow zero-interleaved operations. FIFOs, however, are limited in size and allow underflow/overflow. A number of simple, low performance system FIFOs in various sizes (8, 32, or 128 bytes) are used in VLSI controllers to minimize hardware costs, while still allowing zero-interleaved transfers.

Alternate data transfer methods

Another method uses a pair of sector or toggle buffers. This allows disk data to fill or empty one buffer while the system empties or fills the other. Continuous data transfer is provided by switching buffers at the correct time. System memory can be completely decoupled from the disk drives, since all disk data from one transfer is contained in one of two buffers. Toggle buffers also prevent data underflows/overflows and allow on-controller correction of data errors.

A dual-ported RAM can also implement the buffering needed to perform zero-interleaved operation. These RAMs are expensive, and require close coordination of timing control and addressing. A track buffer can hold all data from a single disk track. This RAM, however, should be designed as a variation of the dual-ported RAM to ensure continuous throughput between the disk and the system.

Whatever method is used, the zero-interleaved operation significantly improves disk access times. A typical Winchester disk rotates at 3600 rpm, requiring about 16.6 ms per revolution. With a controller interleaving of degree five on each track (logically adjacent sectors are physically separated by five sectors), at least six revolutions (or 100 ms) are required to read or write data to an entire track. This actual data transfer time, when added to the time required to move the read/write heads to a track (80 to 90 ms), results in a total I/O access time that is twice that using zero interleaving.

A variation of zero-interleaved transfers, a technique called "nonsequential" sector access, further boosts overall performance. Data is stored on the disk tracks in blocks called sectors. These sectors are logically accessed in a sequential manner for all multisector operations (ie, a controller first accesses the sector numbered A, then A + 1, A + 2, etc). A normal controller logically accesses several consecutive sectors on a track, by searching for the first desired logical sector and reading or writing data from (and to) subsequent sequential sectors.

Statistically, however, the controller will miss the desired first sector half of the time. This means that about half the amount of time a rotation takes must be added to the "access" time of every new track operation. This delay, about 8 ms on 5 1/4-in. Winchesters, is called "rotational latency" time. On fast disk drives (with average access time of 30 ms), rotational latency can increase the average access time of the disk by about 15 percent (ie, 30 ms for seeking, 16 ms for read/write, and 8 ms for rotational latency).

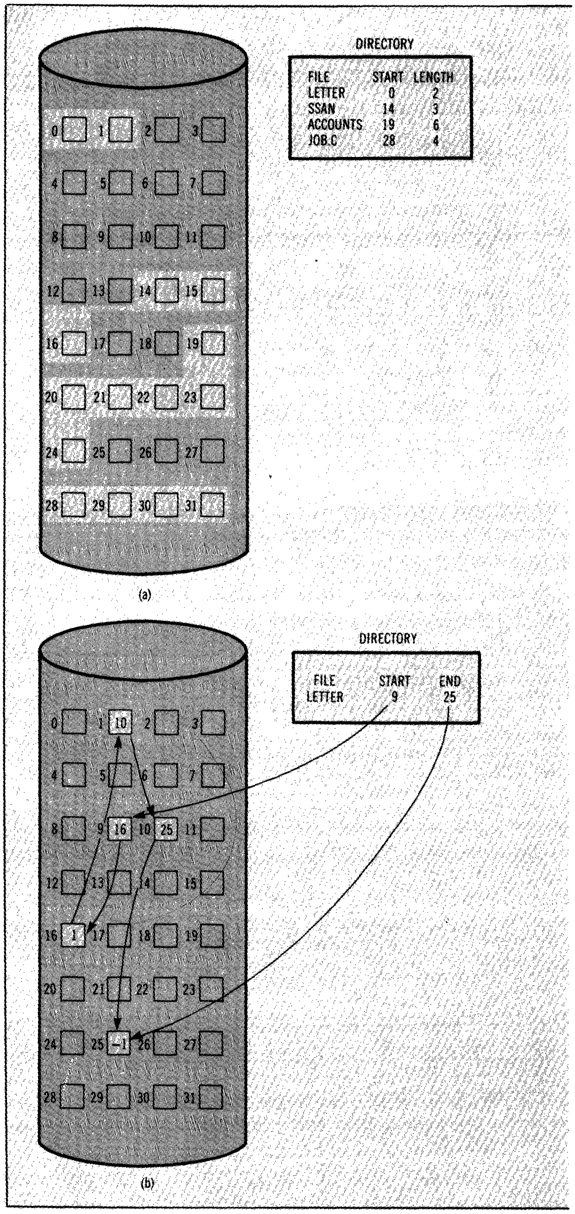
The nonsequential sector access method minimizes rotational latency by accessing the requested sectors in whatever order they are found. As soon as any desired sector (A, A + 1, A + 2, etc) is located, the disk controller performs the necessary I/O operation on that sector. Thus, the maximum time necessary to access any amount of data is no more than that required for one rotation of the disk plus one sector. The nonsequential access method is implemented by using a track buffer to hold the contents of the track during I/O operations. The controller, however, requires extra hardware for the logic necessary to recognize when a desired sector passes under the read/write heads.

Boosting disk performance

A disk data cache provides an increasingly common method of boosting disk performance. Disk cache functions are much like those of caches used for CPUs. As data is read or written from or to the disks, copies are stored in the cache. If the operating system later requests the same data, the data can be taken from cache instead of from disk. Disk caching boosts disk I/O performance from two to nine times because head positioning delays are not involved. Since cache is not disk dependent, slow, low cost disks can be used with a cache controller to give performance approaching that of expensive, high performance drives.

Disk caching can be done either by using the operating system, or by building a dedicated cache controller. Having the operating system handle cache requires cache management software and reduces system memory space (since it must be allocated to the cache buffers). The other popular method of implementing disk caching is through dedicated cache controller hardware. This involves adding RAM, a cache manager (usually a microprocessor and ROM), software, some form of interface to the operating system, and the disk controller.

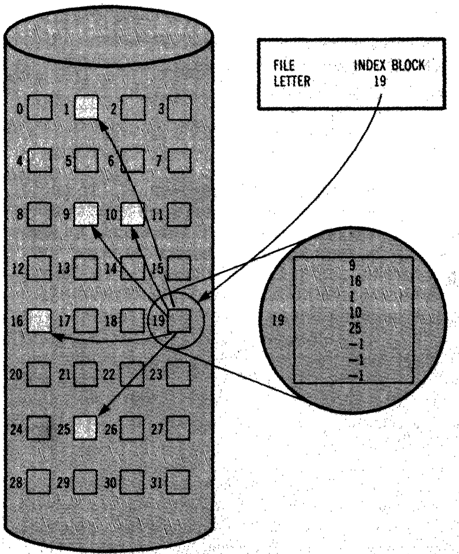
With this method, whenever a disk I/O request is initiated by the operating system, it is sent to the cache controller instead of to the disk controller. If the requested data is in the cache, it is passed to the system without disk controller intervention. If the data is not in the cache, the cache controller passes the command to the disk controller. When the data is returned from the disk, the cache controller makes a copy of the data and puts it in cache as it is sent to the system.



FILE	START	LENGTH
LETTER	0	2
SSAN	14	3
ACCOUNTS	19	6
JOB.C	28	4

FILE	START	END
LETTER	9	25

Organization of files on a disk can also affect the access time. Normally, all read/write heads on a head assembly are positioned on the same track,



(c)

Certain file organizations can improve disk access times. Contiguous allocation allows file data to be stored compactly and retrieved quickly on the disk, but suffers from space allocation and garbage collection problems (a). Linked-list allocation overcomes contiguous allocation's disadvantages, but introduces extra overhead, speed problems, and vulnerability to file damage. Moreover, it does not allow random file data access (b). The indexed allocation scheme, widely used in Unix systems, overcomes most of the problems in a and b, but has some speed trouble, extra complexity, and can impose file size limits (c).

although they are on different surfaces. Operating systems can allocate multiple tracks for file storage on either the same or multiple surfaces. A preferred strategy is to switch read/write heads until a particular track on all surfaces is used before moving the read/write heads to another track. This is the fastest method, since the time required to switch from one read/write head to another usually requires only microseconds, while changing tracks requires several milliseconds.

A recurring problem with many disk-based operating systems is file fragmentation, which occurs when files are updated (ie, increased or decreased in size). One reason for the popularity of the Berkeley version of Unix (4.2 BSD) is its use of an improved file management system to increase file access speed and reduce fragmentation.

Files are stored on disks using either linear (continuous), linked-list allocation, or indexed allocation. The linear method stores files in single, contiguous blocks on the disk, minimizing the time needed to read the files. To increase or decrease file size, data must be moved to another area on the disk that is equal to or greater than the new file. Because files are allocated randomly, free space is broken into a large number of pieces. Frequent compaction is required to maximize the disk's storage capacity, usually at a great cost in disk processing time.

The linked-list file structure incorporates a pointer in each sector to indicate the next sector belonging to the file. Fragmentation is avoided because every sector on the disk can be used in disk files. Despite the advantages of using complete disk space and the flexibility in allocating space, the linked structure requires extra overhead in each sector for the pointers. The structure is also more vulnerable to damage; if the link in one sector is damaged, the rest of the file is lost. The operating system's inability to make direct accesses into the file is also a shortcoming.

The indexed allocation method uses a pointer table built into the beginning of the sector to define all sectors in the file. This method, like the linked list, minimizes fragmentation on the disk because all sectors can be used and direct access is allowed. The space required for the file pointer table, however, is usually fixed at file creation and must serve for the life of the file. Since estimating the table size is usually difficult, overestimation is the rule, costing storage space. The space required for the table frequently exceeds that used for the linked-list method. And unless the index tables are kept in the controller, the elevator sort algorithm cannot be properly exploited.

Programmable features in modern disk controllers combine the best features of all three file storage methods while minimizing their disadvantages. Keeping the different file techniques distinct from one another presents the biggest problem when methods are combined. An advanced controller, such as the Am9580, provides ability to flag individual sectors, allowing the sectors to determine how they are being used. For example, a combination of the linear method with the linked-list file allows allocation of a single block of memory when the file is created.

But it still allows sectors to be added without resulting in fragmentation or excessive file movement.

Ondisk file caching also boosts overall disk/system throughput. This method stores frequently used files in the most rapidly accessible part of the disk. Normally, files are stored on the disk randomly (based on such things as space requirements and partitioning of the disk data space).

Second-generation VLSI disk controllers

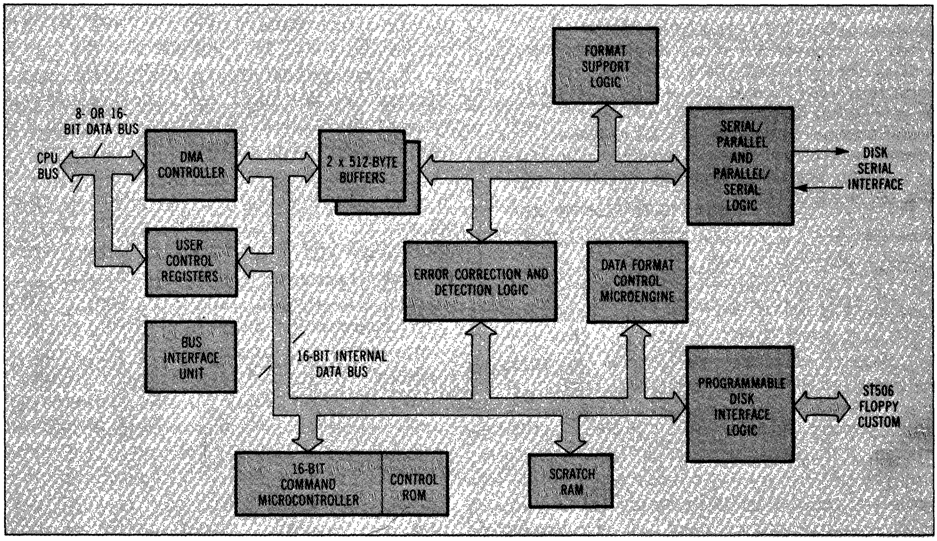
A primary requirement of second generation VLSI disk controllers is maximum disk and system performance. The Am9580 hard disk controller's dual-buffer architecture with integrated DMA controller boosts disk system performance in two ways. First, it fully decouples the disk serial interface (up to 16 Mbits/s) from the system bus. This allows the system to operate without being tied into the peculiarities of the disk timings. Second, it provides efficient data transfers and cuts software overhead. The DMA controller can transfer data at rates up to 5 Mbytes/s and supports 8- or 16-bit interfaces. Programmable bus throttling regulates bus activity of the hard disk controller on the system bus.

Another facet of second generation disk controllers is the large amount of integrated software. The Am9580 has two different microengines operating in parallel and executing specialized disk data control algorithms. One of the microengines, the data format controller, handles all the serial transfer of data to and from the disk.

In addition to performing track formatting and sector reads and writes, it handles special data recovery algorithms, and floppy or Winchester disk formats. The second microcontroller, the command sequencer, is responsible for interpreting the 16 different disk commands, coordinating DMA activity, and handling the disk control interface. Command, status, and data transfers are handled by the controller with minimal CPU intervention. The hard disk controller uses a linked-list command structure. Command blocks (called I/O parameter blocks) are set up as a linked list in system memory. The disk controller automatically fetches and executes the commands without additional CPU overhead. Data transfers between the disk and system are handled by the onboard DMA controller. An additional control structure, called data map, allows the Am9580 to break up or combine data scattered throughout system memory and collect it into contiguous blocks.

Status result information is stored in a specially designated area of system memory called the status result area. Errors resulting from abnormal disk behavior are reported to the system in the status result area along with an identification code that describes which command caused the error. Users can program how the controller will handle errors and when it will abort a command.

Finally, the Am9580 supports several different disk drive control interfaces. The industry standard ST506 Winchester interface is fully implemented. For floppy drives, a floppy-like control interface is supported. To accommodate custom disk interfaces, a special programmable option allows different portions of the disk control interface to be selectively disabled. This allows the user to implement (externally) any desired control interface.



Optimal disk access is achieved, however, by centering read/write heads around the middle tracks on the platter because the average distance to any location (assuming random disk I/O requests) is minimized. To ensure this happens, the disk file manager must keep track of how frequently different files on the disk are accessed. Then the file manager must rearrange the files so those most frequently accessed are located on the center tracks. In paged virtual memory systems, spare data blocks allocated on the center tracks should be used to accommodate data that is constantly being swapped in and out of system memory.

Ondisk caching can be put into the operating system's normal disk I/O routines without degrading system throughput. During medium and high system load periods, the operating system should only statistically track disk I/O. Periods of low use (about 25 percent or less) can then be used by the disk sorting routines to move files around (based on current statistics) and to clean up fragmented files and put them into linear blocks. These routines would move least recently used files to outer or inner tracks, and migrate more frequently used files to center tracks.

Special options would allow the operating system designer to force certain files to middle tracks or prevent the ondisk caching routines from moving them. Since many systems have long periods of low loading, daily updates of the ondisk cache would not degrade users' response time. Thus, overall throughput of the disk drives would increase without any system overhead that is visible to the user.

Disk controllers employing these techniques are not simple systems based on one or two VLSI chips. Instead, the disk controller is built up around powerful disk controller ICs, such as the Am9580 controller and the Am9582 disk data separator, and supported by a microprocessor, random logic, RAM (large amounts for caching), and ROM to contain all the necessary software algorithms. Although single-user, single-tasking microcomputers do use these techniques, newer machines and Unix-based systems require very high performance disk systems to perform adequately in most multi-user, multitasking environments.

Reprint by permission from EDN. Vol.31,No.19.
Copyright 1986 Cahners Publishing Co..

PLDs implement encoder/decoder for disk drives

By using software to define programmable logic devices as run-length-limited encode/decode systems, you can design disk-drive systems that have 50% more data-storage capacity than drives that implement the MFM code.

Arthur Khu and Rudy Sterner,
Advanced Micro Devices Inc

When you're designing a disk-drive system, you can implement run-length-limited (RLL) 2,7 encoding/decoding circuitry in your design by using only three programmable-logic devices (PLDs) and two shift registers. You design the encoder and decoder as state machines and use the timing diagrams to determine what the timing and control signals must be.

To create a disk controller with encoding/decoding features, you can use three AmPAL22V10 PLDs and a disk controller such as the Am9580/Am9582 chip set **(Fig 1)**. The Am9580 hard-disk controller and the Am9582 disk-data separator perform all the general disk-control functions. The PLDs have appropriate architectures for implementing the encoding and decoding state machines. Further, you can reprogram the PLDs to implement higher density ratios for data encoding, and you can increase your system's speed simply by using faster PLDs.

An RLL code is a code in which the number of zeros between ones—the run length—is definite. The "2,7" designation means that the code for the binary data string has at least two and at most seven zeros separating the ones. RLL codes increase the density of data stored on a disk by reducing the number of recorded pulses (ones) necessary to represent a given amount of data. This reduction allows the disk-drive circuitry to pack the ones closer together, increasing the amount of data on the disk.

In comparison with the (de facto) industry-standard approach, MFM, the RLL 2,7 code increases by 50% the amount of data you can store on a disk drive (see **box,** "RLL 2,7 code vs MFM code"). In addition, RLL

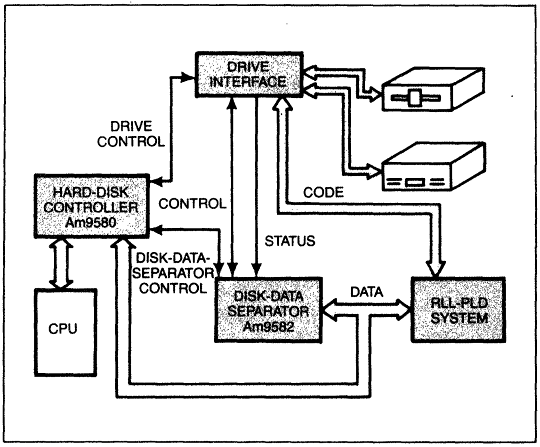

DRIVE INTERFACE
DRIVE CONTROL
HARD-DISK CONTROLLER Am9580
CONTROL
CODE
STATUS
DISK-DATA-SEPARATOR CONTROL
CPU
DISK-DATA SEPARATOR Am9582
DATA
RLL-PLD SYSTEM

Fig 1—A ***complete disk controller*** *requires only two VLSI ICs, a PLD-based encoding/decoding system, and a drive interface. The hard-disk controller and the disk-data separator provide generic disk-drive control, and the PLDs provide RLL 2,7 encoding and decoding, which increases disk storage capacity.*

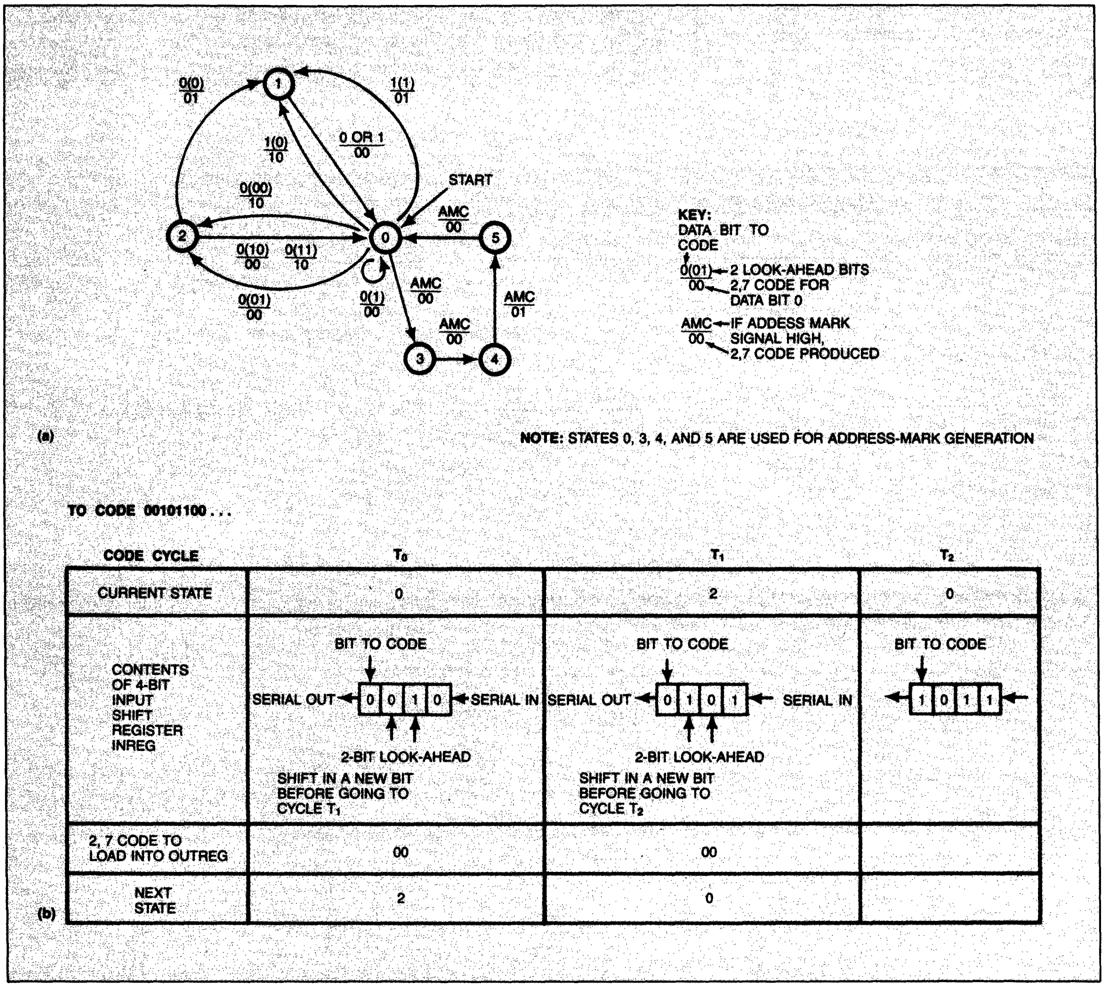
TABLE 1 — RLL 2,7 CODING RULES

DATA	2,7 CODE
10	1000
11	0100
000	100100
010	001000
011	000100
0010	00001000
0011	00100100

2,7 decoding circuitry recovers quickly from code-detection errors.

As **Table 1** shows, RLL 2,7 encoding circuitry translates seven data strings into 2,7 code. You can break any binary non-return-to-zero (NRZ) data string into combinations of the seven data strings. To obtain the decoded data string, the circuitry matches the 2,7 code patterns with the seven 2,7 code strings in **Table 1.**

Because 2,7 code strings have variable lengths (they can be 2, 3, or 4 bits long), your design will need control logic that controls the output from the encoder/decoder as translation takes place. Encoding and decoding state machines **(Figs 2 and 3)** implement this control logic from the code in **Table 1** (see **box,** "Convert RLL 2,7 code to a state machine"). The encoding state machine



TO CODE 00101100 . . .

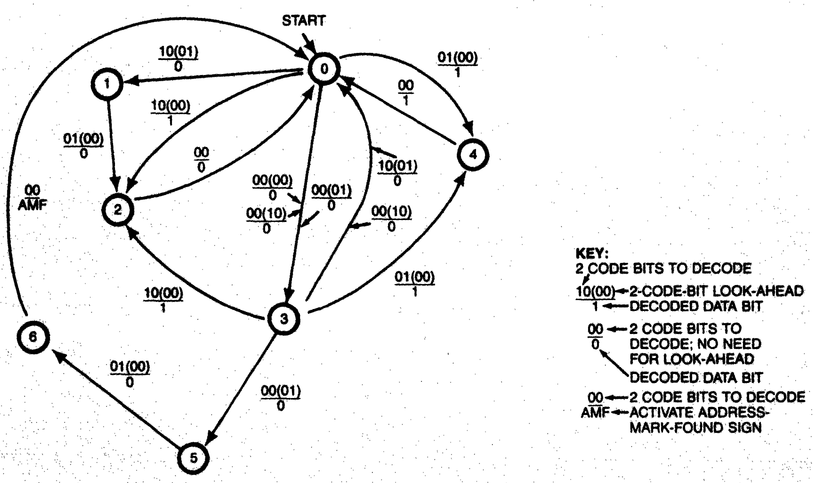
CODE CYCLE	T_0	T_1	T_2
CURRENT STATE	0	2	0
CONTENTS OF 4-BIT INPUT SHIFT REGISTER INREG	BIT TO CODE SERIAL OUT ← 0 0 1 0 ← SERIAL IN 2-BIT LOOK-AHEAD SHIFT IN A NEW BIT BEFORE GOING TO CYCLE T_1	BIT TO CODE SERIAL OUT ← 0 1 0 1 ← SERIAL IN 2-BIT LOOK-AHEAD SHIFT IN A NEW BIT BEFORE GOING TO CYCLE T_2	BIT TO CODE ← 1 0 1 1 ←
2,7 CODE TO LOAD INTO OUTREG	00	00	
NEXT STATE	2	0	

Fig 2—The encoder state machine (a) *describes the translation of input data into RLL 2,7 code. The first two cycles in the encoding of the data string 00101100* ***(b)*** *demonstrate how the encoder determines the correct code by examining both the bit to be encoded and the next two (look-ahead) bits.*

During decoding, circuitry matches the RLL 2,7 code patterns with the seven RLL 2,7 code strings to obtain the corresponding decoded data string.

produces two 2,7 code bits for each data bit received. The decoding state machine, on the other hand, produces one data bit for every two bits of 2,7 code. The clocking scheme in these encoding/decoding state machines is simpler than the familiar table-look-up method, which requires suspended operation during encoding and decoding.

You can implement both of these encoding/decoding 2,7 state machines with one PLD device (IC_1) and two shift registers (INREG and OUTREG) **(Fig 4)**. To



(a)

NOTE: STATES 0, 3, 5, AND 6 ARE USED WHEN DETECTING ADDRESS MARKS

TO DECODE 00100010 . . .

DECODE CYCLE	S_0	S_1	S_2
CURRENT STATE	0	3	2
CONTENTS OF 4-BIT INPUT SHIFT REGISTER INREG	2 CODE BITS TO DECODE SERIAL OUT ← 0 0 1 0 ← SERIAL IN 2-CODE-BIT LOOK-AHEAD SHIFT IN TWO NEW CODE BITS BEFORE GOING TO CYCLE S_1	2 CODE BITS TO DECODE SERIAL OUT ← 1 0 0 0 ← SERIAL IN 2-CODE-BIT LOOK-AHEAD SHIFT IN TWO NEW CODE BITS BEFORE GOING TO CYCLE S_2	2 CODE BITS TO DECODE ← 0 0 1 0 ←
DECODED DATA BIT TO LOAD INTO OUTREG	0	1	
NEXT STATE	3	2	

(b)

***Fig 3—This decoder state machine** (**a**) represents the translation of RLL 2,7 code into an output data stream. To determine the correct output data, the decoder examines four code bits—two bits to be decoded and two look-ahead bits, as shown in the example in **b**.*

Because the RLL 2,7 code contains two bits for every one bit of data, the timing circuitry divides the clock signal for the coded data, producing a data clock.

synchronize the encoder/decoder with the hard-disk controller, you use two other PLDs (IC_2 and IC_3) to provide clock-generation and address-mark-control logic. IC_1 is a AmPAL22V10, which has sufficient capacity to implement the two state machines. IC_2, also an AmPAL22V10, utilizes the PLD's large capacity and programmable output cells to implement the address-mark-control circuitry. An AmPAL16HD8 (IC_3), which is sufficient for implementing the clock circuitry and the random logic, completes the design.

To understand the state-machine implementation of the RLL 2,7 encoder, consider the state machine in **Fig 2.** The encoding state machine begins in state zero; the first four bits of the data to be encoded are in the shift register INREG. For the data stream in the figure, the encoder, beginning in the first cycle (T_0) reads the first bit in the stream as 0 and sees that the next two bits (the look-ahead bits) in INREG are 0 and 1, respectively. According to the state diagram, the encoder produces a 00 output because, as **Table 1** shows, input data starting with 001 translates to a character string that starts with 00. The encoder then enters state 2, shifts the encoded 0 out of INREG, and shifts in the next (fifth) bit of the data to be encoded.

RLL 2,7 code vs MFM code

Although the RLL 2,7 and MFM coding methods can both increase a disk drive's capacity, RLL 2,7 code is more compact. A disk drive that implements the RLL 2,7 code can, therefore, store 50% more data than can a drive that implements the MFM code.

On the magnetic medium in the disk drive (hard-disk or floppy-disk drives), binary data appears as a change in flux (representing a one) or as no change in flux (representing a zero). Because the disk density is limited by the minimum distance between flux transitions, the maximum data density depends on how the data is encoded.

MFM is an RLL code with the designation 1,3;1,2;1. RLL 2,7 code (its full designation is 2,7;1,2;3) allows a minimum of two zeros between each one, and MFM code allows a minimum of one zero. RLL 2,7 code can store as much as 50% more data in a given number of flux changes than can MFM code; therefore, RLL 2,7 code can store as much as 50% more data on a given section of magnetic medium.

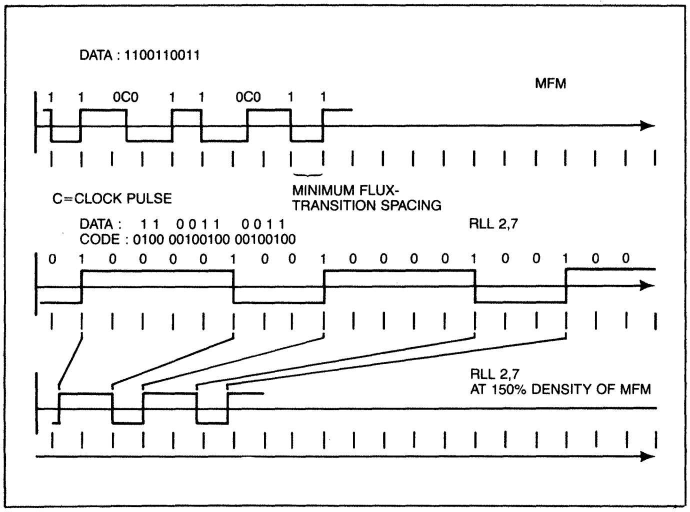


***Fig A—Data storage that's 50% more dense** is the principal advantage of RLL 2,7 code over the de facto standard MFM code. Because it needs fewer transitions to describe data, the RLL 2,7 code takes up only 67% of the disk storage space that MFM code occupies.*

Consider the example in **Fig A.** The data stream 1100110011 is represented by eight transitions in MFM code, and by five transitions in RLL 2,7 code. When the disk drive uses the minimum distance between transitions to record the RLL 2,7 code, the RLL 2,7 code takes 67% of the space that the MFM code takes, so it has space available to store 50% more data. In most applications, the actual storage increase is between 35% and 40%, because some disk space is reserved for sector and data-field markers.

In the second cycle (T_1) of the encoding process, the encoder sees that the data bit is 0 and the two look-ahead bits are 1 and 0, respectively. According to the state diagram, when the encoder is in state 2 and sees 010, its output is 00. As before, the encoder then moves to the next state (in this case, state 0) and shifts a new bit into INREG. The rest of the encoding process proceeds similarly.

The decoding process is similar to the encoding process. The decoder, which is described as a state machine (**Fig 3**), accepts two input bits in RLL 2,7 format and sees the next two coded bits as look-ahead bits. In contrast to the encoder, the decoder produces one output bit for every two input bits.

To implement this encoder and decoder circuitry with PLDs, you can use PLD-design tools such as CUPL from Personal CAD Systems Inc (San Jose, CA) and Abel from Data I/O Corp (Redmond, WA). These software tools include syntaxes that you can use to describe state machines as well as general logic equations.

To control the rate at which IC_1 (in **Fig 4**) receives data and code, IC_2 and IC_3 implement the timing signals shown in **Fig 5.** Three signals (Read, Write, and Code_Clock) from the hard-disk controller and disk-data separator form the basis of the timing signals.

First, the timing circuitry produces a clock signal, Rd_Clk, from the Code_Clock signal that originates at the disk-data separator's FDDAM output. Because the RLL 2,7 code contains two bits for every one bit of

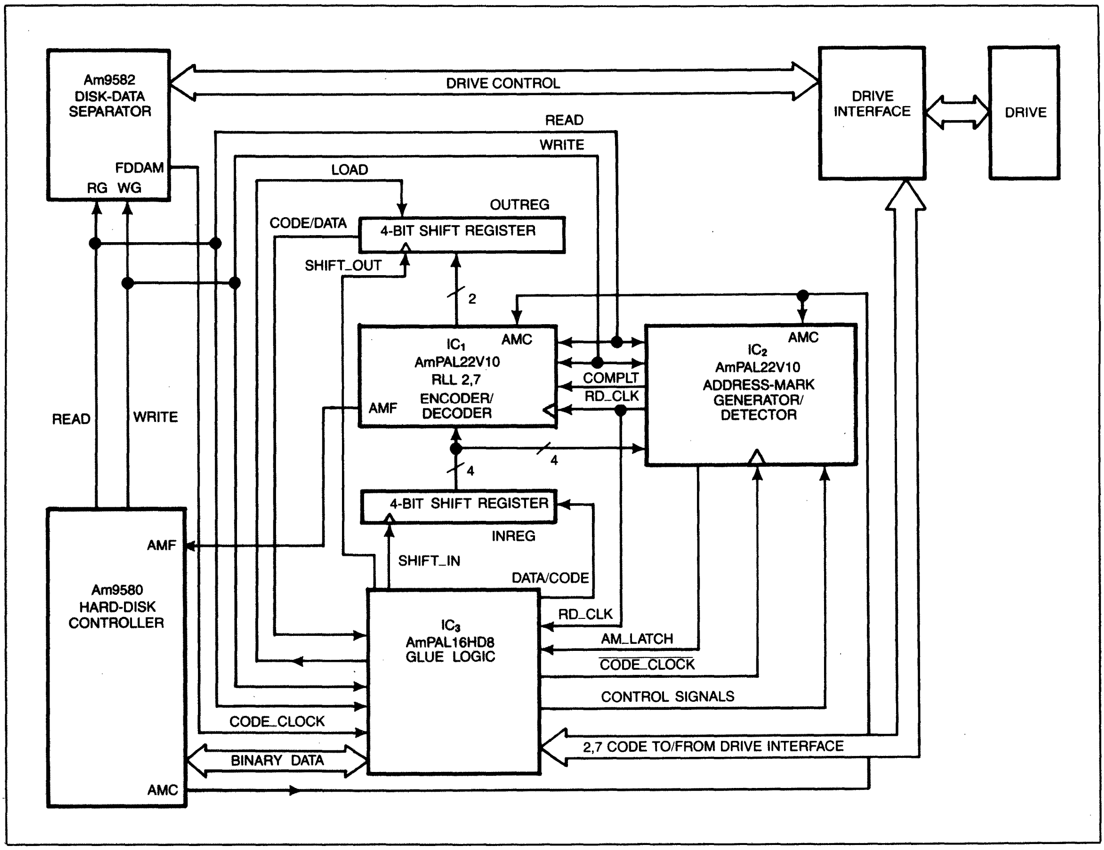


***Fig 4—The encoding/decoding circuitry** includes three PLDs. IC_1 implements the encoding and decoding state machines. IC_2 monitors IC_1 and provides special timing and input signals. IC_3 implements glue logic and timing signals.*

Before the disk-drive system can decode data on the disk, it must synchronize itself with the disk's data clock to find out when the data bits begin.

data, the timing circuitry divides the clock signal for the coded data (Code_Clock), producing a data clock signal, Rd_Clk. The timing circuitry then uses Code_Clock and Rd_Clk to control the timing of shift-register control signals Shift_Out and Shift_In.

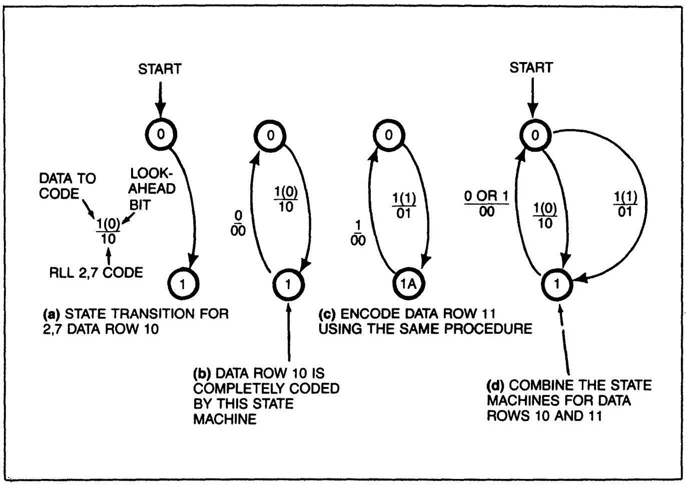
To obtain the equations you need to program the PLDs, examine the timing diagrams. The timing diagrams show that the state-machine design requires only a few timing signals to implement the controller. Because the timing circuitry loads and shifts data produced by IC$_1$ at the points indicated on the timing diagram in **Fig 5,** the only clock signal that IC$_1$ requires to code or decode data in INREG is the Rd_Clk signal.

To control the transfer of data and code, the Shift_In signal latches data or code into INREG, and the Shift_Out signal controls the output of OUTREG. For example, when encoding data, the encoder produces two bits of code on each rising edge of Rd_Clk. Because

Convert RLL 2,7 code to a state machine

You can use a simple algorithmic procedure to convert a code table to an encoding state machine. For each row of the code table, you create a simple 2-state expression for the conversion of the data into the code. Then you combine the expressions into one state machine for the entire table.

For the code in **Table 1** of the accompanying article, you consider each data bit and the associated pair of bits of RLL 2,7 code. For example, the first row of the table contains two data bits: one, which is associated with the code 10, and zero, which is associated with the code 00. The state machine for this row begins in state zero (which is arbitrarily assigned) and changes state when it sees that the first data bit is a one and the next bit (the look-ahead bit) is a zero. This change of state, which appears graphically in **Fig Aa,** results in the output 10.



(a) STATE TRANSITION FOR 2,7 DATA ROW 10

(b) DATA ROW 10 IS COMPLETELY CODED BY THIS STATE MACHINE

(c) ENCODE DATA ROW 11 USING THE SAME PROCEDURE

(d) COMBINE THE STATE MACHINES FOR DATA ROWS 10 AND 11

*Fig A—**Creating a state machine** from a code table is a 2-part process. First, you create a simple state machine for each row in the table, and then you combine the state machines into one state machine for the whole table.*

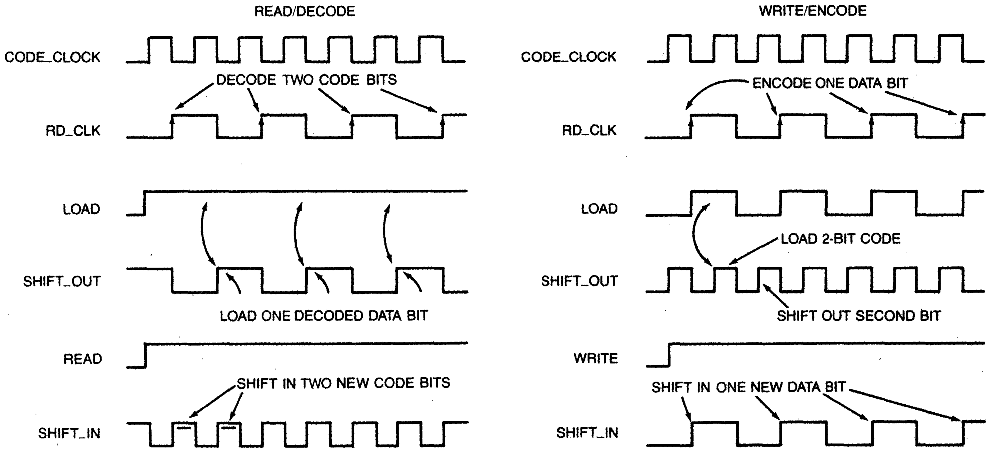
Note that the state machine must evaluate the look-ahead bit. If this bit is a one instead of a zero, the input data will be 11, corresponding to the second row of the table, so your state machine must generate a 01. For this code table, no more than two look-ahead bits are necessary for encoding any data bit.

If the state machine is in state one, and the input data is zero, the state machine produces a 00 output and returns to state zero (**Fig Ab.**) Note that the data bit encoded in this state is the look-ahead bit from the previous state and that no look-ahead bit is necessary for encoding because the zero is the last bit in data row 10.

You use this procedure to create state machines for the other table rows as well. **Fig Ac,** for example, contains the state machine for data row 11. Because all the state machines have the same beginning state (state zero), you can combine them to form the complete encoding state machine. Furthermore, the state machines may share other states, as shown in the combination of the two data rows **(Fig Ad).**

the Load signal is high at the same time that Rd_Clk is high, the bits are loaded into the shift register. Because the frequency of Shift_Out is twice that of Rd_Clk, Shift_Out shifts both encoded bits out before the next encoded bits are loaded. Shift_In presents the next data bit to the encoder at the same time that the second encoded bit is shifted out, starting the next encode cycle.

Before the disk-drive system can decode data on the disk, it must synchronize itself with the data clock from the disk and find out when the data bits begin. To assist the circuitry in synchronization and initialization, you can place synchronization signals, as well as a marker indicating the beginning of data, at the beginning of the RLL code. When the RLL circuitry reads these patterns, it's ready to decode RLL data.



INREG AND OUTREG LOAD WHEN LOAD AND SHIFT ARE HIGH;
THEY SHIFT WHEN LOAD IS LOW AND SHIFT IS HIGH

LOGIC EQUATIONS:

RD_CLK := /RD_CLK (A REGISTERED SIGNAL FROM IC_2)
SHIFT_IN = READ*/CODE_CLOCK+WRITE*RD_CLK
SHIFT_OUT = READ*/RD_CLK+WRITE*CODE_CLOCK
LOAD = READ+WRITE*RD_CLK

CODE_CLOCK : CODE-RATE CLOCK SIGNAL FROM FDDAM PIN ON Am9582 DISK-DATA SEPARATOR

READ,WRITE : READ/WRITE MODE SIGNALS FROM Am9580 HARD-DISK CONTROLLER

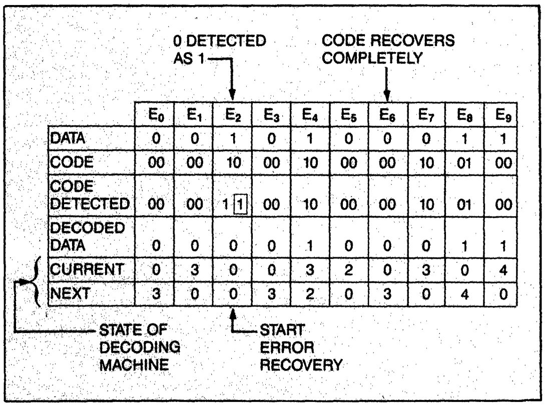
RD_CLK : DATA-RATE CLOCK DERIVED FROM CODE_CLOCK

SHIFT_IN : COMBINATORIAL SIGNAL FROM IC_2

SHIFT_OUT, LOAD : COMBINATORIAL SIGNALS FROM IC_3

Fig 5—The timing signals from IC_2 and IC_3 control *the loading and shifting of code and data in the INREG and OUTREG registers. Note that the clock and shifting signals for the coded bits have twice the frequency of those for the data bits, a situation that corresponds to the 2:1 density ratio between code and data.*

If the decoder detects any pattern before it detects the address marker, the circuit resets itself and begins the initialization sequence anew.



	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
DATA	0	0	1	0	1	0	0	0	1	1
CODE	00	00	10	00	10	00	00	10	01	00
CODE DETECTED	00	00	1 $\boxed{1}$	00	10	00	00	10	01	00
DECODED DATA	0	0	0	0	1	0	0	0	1	1
CURRENT	0	3	0	0	3	2	0	3	0	4
NEXT	3	0	0	3	2	0	3	0	4	0

***Fig 6—The ability to recover from an error** is a useful feature of RLL 2,7 code. An error at cycle E_2 results in incorrectly translated data until the machine recovers fully.*

The synchronization field comprises a series of patterns that allow the phase-locked loop in the disk-data separator to synchronize to the frequency of the incoming data. These patterns represent the maximum frequency input. For RLL 2,7 code, that input is 100100, because this code has the fewest permissible zeros between ones. Using the highest possible frequencies minimizes the synchronization time, so the patterns are completed quickly.

A marker that follows the synchronization field indicates the beginning of data. This marker, the address mark, must be distinct from all other code words. For example, you can use a pattern that violates the code rules, such as a string of zeros (this string is called "dc erase" because it removes all flux transitions from the recording medium). Alternatively, you can use a pattern that obeys the code rules but is not a defined code pattern. In the examples in this article, the pattern 00000100 identifies the beginning of data. Although this pattern is not a defined code word, it doesn't violate the RLL 2,7 code rules as long as it's preceded by no more than two zeros in a row.

When the 100100 pattern is synchronizing the circuitry, the circuitry produces the Rd_Clk signal, which is in phase with the pattern. In order for the decoder to operate properly, the Rd_Clk signal must rise with the first one in the pattern and fall with the second one. However, because of the repetitive nature of the pattern, the circuitry could produce a falling edge of Rd_Clk on the first one in the pattern and a rising edge on the second one, in which case the circuitry would not be in synchronization with the beginning of the data.

The encoding circuitry resolves this synchronization problem by inserting the pattern "0100" eight times between the synchronization field and the address mark. The phase-locked-loop system locks onto the 0100 pattern, and the Rd_Clk control circuitry in IC_2 sets itself to decode the data correctly.

To put the synchronization and marker data into the RLL 2,7 code on the disk, you must design the RLL 2,7 encoder to produce the signals. When the signal WG (from the hard-disk controller IC_3) goes high, the data is encoded to all zeros. The encoder translates the zeros as the synchronization pattern 100100, which is then stored on disk.

Next, the address-mark-control (AMC) signal from the hard-disk controller sets an internal latch in IC_2, producing the output Am_Latch, which forces IC_3 to put ones in the INREG register. The encoding circuitry codes the ones as the string 0100. After the eighth 0100 pattern, IC_2 sets the Complt signal high. On sensing Complt, IC_1 writes the address mark.

The encoding circuitry must have the first four data bits in INREG by the time the address mark is written. IC_1 writes two patterns for the address mark. At the start of the second address-mark pattern, IC_1 sets AMF (Address Mark Found) High and resets Am_Latch low. The assertion of AMF signals the hard-disk controller to begin shifting data into INREG for encoding. Because four Rd_Clk cycles occur while the address-mark pattern is being written, the first four bits are shifted into INREG by the time the second address mark is written.

The decoder circuit has two safeguards that prevent it from identifying the address-mark pattern incorrectly. First, the decoder must detect at least five 0100 patterns before it acknowledges the address-mark patterns. Second, if it detects any pattern before it detects the address mark, the circuit resets itself and begins the initialization sequence.

IC_2 implements these two safeguards by monitoring the RLL code that IC_1 decodes. When the hard-disk controller asserts the RG and AMC signals, IC_2 sets Complt and Am_Latch low. IC_2 sets Complt high when it detects the fifth consecutive 0100 pattern, enabling IC_1 to detect the address mark. If the input to IC_1 is anything other than a 0100 pattern or an address mark, IC_2 sets Complt low, and the initialization begins anew.

If IC_1 successfully detects the address-mark pattern, it sets AMF high. The assertion of AMF causes IC_2 to set Am_Latch high, thus enabling the output of OUT-

REG to send decoded data to the hard-disk controller. Am_Latch remains high as long as RG is high.

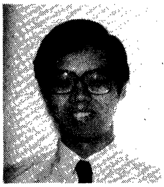
RLL 2,7 encoding also provides for error recovery. Whenever a disk-drive system reads code from a disk, the read/write heads or the transmission cables can cause transmission errors. One of the properties of RLL 2,7 code is that any single-bit error (for example, a coded one detected as a zero) will correct itself after a run of at most 16 correctly detected bits.

In short, whenever a 2-bit pattern doesn't match any of the expected patterns for a particular state of the decoding circuitry, the decoding state machine returns to state zero and generates a zero as a translation for the erroneous code bits. Then the decoder shifts the next two code bits into the INREG register and continues decoding from state zero.

In such cases, the decoding state machine can correctly decode coded data, but it may not recover immediately upon returning to state zero. As **Fig 6** shows, although the code bits that the decoder detects may be valid, the decoded data is incorrect because the error has forced that state machine into the wrong state. In this example, the decoder doesn't recover until 6 code bits later (in cycle E6), when it again falls into the correct state and accurately decodes the data. **EDN**

Authors' biographies

Arthur Khu is a product planning engineer at Advanced Micro Devices Inc (Sunnyvale, CA). He is involved in the research and development of architectures for advanced programmable-logic devices, and he has developed a general logic compiler for advanced PLDs. Art holds a BS in Math/Computer Science and an MS in Computer Science from Santa Clara University. He lists racquetball and astronomy among his interests.



Rudolph J Sterner, an engineer at Advanced Micro Devices Inc (Sunnyvale, CA), is engaged in the development of disk-drive-related products. Prior to his two years at AMD, Rudy worked at IMI Corp and Sperry Corp. He holds a BSEE from San Jose State University, and in his spare time he enjoys photography.

NOTES

NOTES

ADVANCED MICRO DEVICES' NORTH AMERICAN SALES OFFICES

ALABAMA	(205) 882-9122
ARIZONA	(602) 242-4400
CALIFORNIA,	
Culver City	(213) 645-1524
Newport Beach	(714) 752-6262
San Diego	(619) 560-7030
San Jose	(408) 452-0500
Woodland Hills	(818) 992-4155
CANADA, Ontario,	
Kanata	(613) 592-0060
Willowdale	(416) 224-5193
COLORADO	(303) 741-2900
CONNECTICUT	(203) 264-7800
FLORIDA,	
Clearwater	(813) 530-9971
Ft Lauderdale	(305) 776-2001
Melbourne	(305) 729-0496
Orlando	(407) 830-8100
GEORGIA	(404) 449-7920
ILLINOIS,	
Chicago	(312) 773-4422
Naperville	(312) 505-9517
INDIANA	(317) 244-7207
KANSAS	(913) 451-3115

MARYLAND	(301) 796-9310
MASSACHUSETTS	(617) 273-3970
MICHIGAN	(513) 549-7174
MINNESOTA	(612) 938-0001
MISSOURI	(913) 451-3115
NEW JERSEY	(201) 299-0002
NEW YORK,	
Liverpool	(315) 457-5400
Poughkeepsie	(914) 471-8180
Woodbury	(516) 364-8020
NORTH CAROLINA	(919) 878-8111
OHIO	(614) 891-6455
Columbus	(614) 891-6455
Dayton	(513) 439-0470
OREGON	(503) 245-0080
PENNSYLVANIA,	
Allentown	(215) 398-8006
Cherry Hill	(609) 662-2900
TEXAS,	
Austin	(512) 346-7830
Dallas	(214) 934-9099
Houston	(713) 785-9001
WASHINGTON	(206) 455-3600
WISCONSIN	(414) 792-0590

ADVANCED MICRO DEVICES' INTERNATIONAL SALES OFFICES

BELGIUM,		
Bruxelles	TEL	(02) 771 91 42
	FAX	(02) 762 37 12
	TLX	61028
FRANCE,		
Paris	TEL	(1) 49-75-10-10
	FAX	(1) 49-75-10-13
	TLX	263282
WEST GERMANY,		
Hannover area	TEL	(05143) 50 55
	FAX	(05143) 55 53
	TLX	925287
München	TEL	(089) 41 14-0
	FAX	(089) 406490
	TLX	523883
Stuttgart	TEL	(0711) 62 33 77
	FAX	(0711) 625187
	TLX	721882
HONG KONG	TEL	852-5-8654525
	FAX	852-5-8654335
	TLX	67955AMDAPHX
ITALY, Milano	TEL	(02) 3390541 (02) 3533241
	FAX	(02) 3498000
	TLX	315286
JAPAN,		
Kanagawa	TEL	462-47-2911
	FAX	462-47-1729
Tokyo	TEL	(03) 345-8241
	FAX	(03) 342-5196
	TLX	J24064AMDTKOJ
Osaka	TEL	06-243-3250
	FAX	06-243-3253

KOREA, Seoul	TEL	82-2-784-7598
	FAX	82-2-784-8014
LATIN AMERICA,		
Ft. Lauderdale	TEL	(305) 484-8600
	FAX	(305) 485-9736
	TLX	5109554261 AMDFTL
NORWAY,		
Hovik	TEL	(02) 537810
	FAX	(02) 591959
	TLX	79079
SINGAPORE	TEL	65-2257544
	FAX	65-2246113
	TLX	RS55650 MMI RS
SWEDEN, Stockholm	TEL	(08) 733 03 50
	FAX	(08) 733 22 85
	TLX	11602
TAIWAN	TLX	886-2-7122066
	FAX	886-2-7122017
UNITED KINGDOM,		
Manchester area	TEL	(0925) 828008
	FAX	(0925) 827693
	TLX	628524
London area	TEL	(04862) 22121
	FAX	(0483) 756196
	TLX	859103

NORTH AMERICAN REPRESENTATIVES

CANADA	
Burnaby, B.C.	
DAVETEK MARKETING	(604) 430-3680
Calgary, Alberta	
VITEL ELECTRONICS	(403) 278-5833
Kanata, Ontario	
VITEL ELECTRONICS	(613) 592-0090
Mississauga, Ontario	
VITAL ELECTRONICS	(416) 676-9720
Quebec	
VITEL ELECTRONICS	(514) 636-5951
IDAHO	
INTERMOUNTAIN TECH MKGT	(208) 888-6071
INDIANA	
ELECTRONIC MARKETING CONSULTANTS, INC.	(317) 253-1668
IOWA	
LORENZ SALES	(319) 377-4666
KANSAS	
LORENZ SALES	(913) 384-6556

KENTUCKY	
ELECTRONIC MARKETING CONSULTANTS, INC.	(317) 253-1668
MISSOURI	
LORENZ SALES	(314) 997-4558
NEBRASKA	
LORENZ SALES	(402) 475-4660
NEW MEXICO	
THORSON DESERT STATES	(505) 293-8555
NEW YORK	
NYCOM, INC	(315) 437-8343
OHIO	
Centerville	
DOLFUSS ROOT & CO	(513) 433-6776
Columbus	
DOLFUSS ROOT & CO	(614) 885-4844
Strongsville	
DOLFUSS ROOT & CO	(216) 238-0300
PENNSYLVANIA	
DOLFUSS ROOT & CO	(412) 221-4420
UTAH	
R^2 MARKETING	(801) 595-0631

Advanced Micro Devices reserves the right to make changes in its product without notice in order to improve design or performance characteristics. The performance characteristics listed in this document are guaranteed by specific tests, guard banding, design and other practices common to the industry. For specific testing details, contact your local AMD sales representative. The company assumes no responsibility for the use of any circuits described herein.

ADVANCED MICRO DEVICES 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE TOLL FREE: (800) 222-9323 • (408) 749-5703

Printed in the U.S.A. TDC-CP-5M-5/88-0