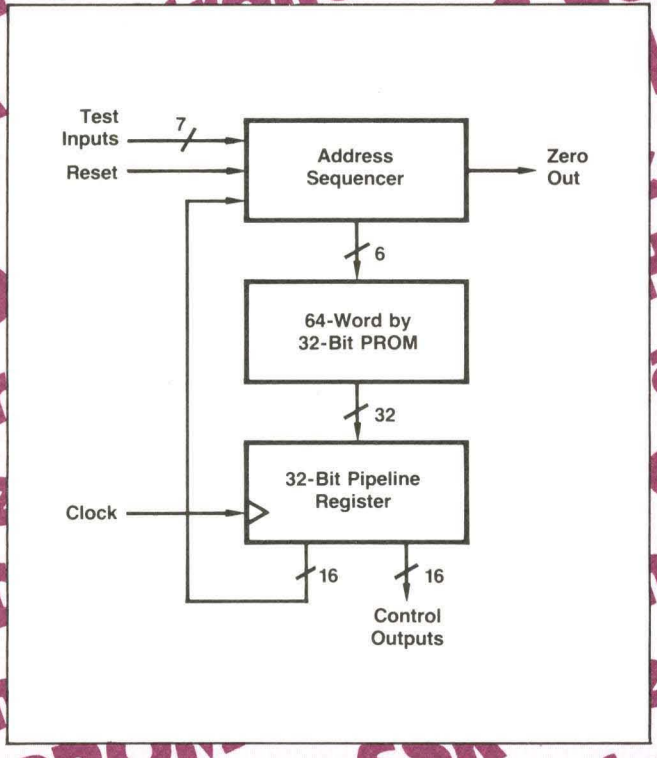
Am29PL141
Fuse Programmable Controller

Handbook



ADVANCED MICRO DEVICES

Advanced Micro Devices

Am29PL141
Fuse Programmable Controller

Handbook

06591A

Contributors to the Am29PL141 Fuse Programmable Controller Handbook:

Rajesh Tanna, Headquarters Applications, Sunnyvale, CA (Chapters 1 and 4) MS 151
Om Agrawal, Product Planning (Chapters 1, 2, and 3)
William Chen, Product Planning (Chapters 2 and 3)
Arthur Khu, Product Planning (Chapter 1)
Rick Purvis, FAE, Austin, TX (Chapters 5 and 6)
David Stoenner, FAE, Newport Beach, CA (Chapters 7 and 8)
Robert O'Hara, FAE, Dorsey, MD (Chapter 9)
Stephen L. Belechak-Becraft, former AMD FAE (Chapter 9)
Dan Overman, Dibec, Inc. (Chapter 9)
Frank Hudziak, Jr., FAE, Itasca, Il (Chapter 10)
Philip Freiden, Product Planning Manager

Technical Writer:

Erland Kyllonen, Senior Technical Writer, Headquarters, MS71

TABLE OF CONTENTS

TABLES

CHAPTER 1

FUSE PROGRAMMABLE CONTROLLER OVERVIEW

1.1 DESIGN CHOICES

Sequential state machine design is normally approached using one of two general methods: the traditional random logic and flip-flop approach, or microprogramming. Until recently, traditional methods were used for state machines with relatively few states (e.g. dynamic memory controllers), while microprogramming was used for applications with many states (e.g. CPUs). The area in between was handled with a hodge-podge of techniques ranging from ad-hoc use of counters and shift registers to PROM-based sequencers. Now, Advanced Micro Devices has introduced the Am29PL141 Fuse Programmable Controller (FPC) to allow cost effective application of microprogramming techniques to fairly small state machines.

Traditional design methodology generally uses state diagrams to define machine behavior, followed by derivation of appropriate J-K flip-flop excitation equations. This approach typically results in very high speed state machine implementations which are highly optimized for a particular task. Unfortunately, this technique is at best tedious, and can be essentially unusable for large state machines.

The microprogramming approach to state machine design consists of storing machine cycle control sequences in memory locations. These instructions are fetched and executed sequentially. Microprogramming is similar to assembly language programming of other processors. It is typically register oriented, with subroutines, loops, and structured programming constructs.

1.2 Am29PL141 ARCHITECTURE OVERVIEW

The Advanced Micro Devices Am29PL141 is a single-chip Fuse Programmable Controller (FPC). It combines, in one chip, powerful address sequencer logic and a memory to store a microprogram based on an instructions set of 29 microinstructions including a repertoire of jumps, multiple branches, and subroutine calls. These instructions can be executed conditionally depending on the level of one of seven external input lines or one internal condition. A Serial Shadow Register (SSR) helps designers diagnose system troubles at the individual IC component level. A pipeline register permits fetching the next instruction at the same time that the current instruction is being executed. This Chapter provides a general description of the FPC. For a detailed description, refer to Appendix F, the data sheet.

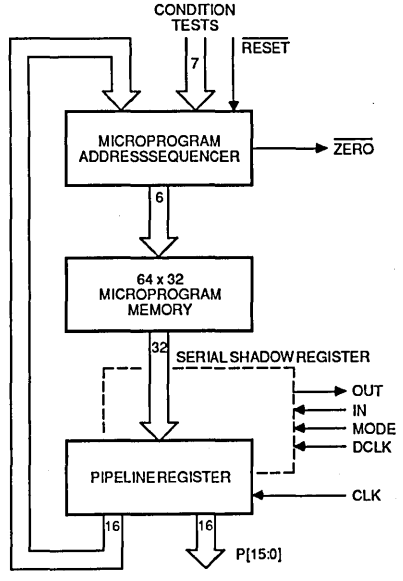
The Am29PL141 consists of four major architectural blocks:

- Address sequencer control logic
- Branch control/condition code select logic
- Instruction decode logic
- 64 x 32-bit microprogram memory with a Pipeline Register and Serial Shadow Register

1.2.1 Address Sequencer

As shown in Figure 1-1, FPC control sequences, stored in the 64 word by 32 bit on-chip programmable memory, are fetched under control of the address sequencer and clocked into the pipeline register. Figure 1-2 shows a more detailed block diagram of the Am29PL141.

The address sequencer inputs consist of seven external condition code inputs and sixteen bits of



06591A 1-1

Figure 1-1. Am29PL141 Block Diagram

the 32 bit instruction currently in the pipeline register. These 16 bits are wrapped around internally in the chip. (The remaining 16 bits go off chip to control the remainder of the state machine.) The test field in the 16-bit microcode input to the address sequencer tells the sequencer which condition code input to test. The results of the condition code test determines whether the sequencer will process the next instruction in the sequence or fetch an instruction from the address specified in the data field of the 16-bit input, from one of the two stack registers, or from the external world via the test inputs.

Within the address sequencer control logic, a 6-bit wide, four-to-one address multiplexer supplies the next state address (refer to Figure 1-2). This next state address can be one of the following:

- Current address (for repeat or hold instructions)
- Incremented PC state (for sequential and continue instructions)
- Subroutine register (SREG) value (for nesting and repeat loops)
- Output of the GO-TO branch control logic

The Program Counter contains the address of the current state (the current instruction being executed). Allowing the address multiplexer to select the current state as the next state allows execution of loops and wait-until-condition-true type instructions. The PLC can thus simply insert wait states until a particular event becomes true. This function of intelligent state machines is needed to interface with various microprocessors and peripherals.

The incremented Program Counter address is the normal next address when no jumps, branches, or subroutine calls are active.

In addition to the Program Multiplexer and Program Counter, the address sequencer contains a dedicated subroutine block and a counter block. The subroutine block has a 6-bit subroutine register (SREG) and a three-to-one multiplexer as the source for the SREG. When a microprogram calls a subroutine, the subroutine register (SREG) supplies the return address.

The counter block is used for timing. It has a 6-bit register (CREG) and a four-to-one multiplexer as the source for the CREG. To perform iterative loops, the controller first loads CREG with the value of the number of iterations required. Every iteration of the loop decrements the count. When the count reaches zero, iterations stop. The zero condition is detected by the zero detect logic on the chip.

The two internal registers, SREG and CREG, are used respectively as a 1 address stack and 6-bit counter. In addition, they can be used together as either a two deep stack, or as nested counters. The ZERO* output indicates that the internal CREG value is zero. The RESET* input initializes the FPC to address 63. An additional operating mode allows use of Serial Shadow Register (SSR) diagnostic techniques.

1.2.2 Branch Control/Condition Code Select

The branch control logic provides the address for multiple branching and for conditional statements such as IF-THEN-ELSE. The condition code select logic selects the condition to be tested which the user can specify for each microprogram instruction. This allows monitoring of both external (7) and internal (1) events.

The user-defined microcode can set the polarity control to test on either true or false conditions without the need for external hardware invertors.

The branch control logic accepts six bits of external test inputs and six bits of branch address from the microinstruction to be used as either an address value or a counter value. It also receives six mask bits from the microinstruction to mask some of the test inputs or the branch address. The resulting masked value specifies the branch address. This provides easy multibranching based on external inputs.

A flexible next address instruction set provides powerful conditional branch, multibranching, subroutine, and loop structures. These instructions are listed below, and explained in the data sheet.

Branch – CONT, WAIT, GOTOPL, GOTOPLZ, GOTOTM, FORK
Call – CALPL, CALPLN, CALTM, CALTMN
Push – PSH, PSHN, PSHPL, PSHTM
Return – RET, RETN, RETPL, RETPLN
Loop – LPPL, LPPLN
Load – LDPL, LDPLN, LDTM, LDTMN
misc. – DECPL, DECGOPL, DECTM, DEC, CMP

1.2.3 Instruction Decode Logic

The instruction decoder decodes 15 of the upper 16 bits of the microinstruction. These include the 5-bit opcode field, the polarity bit, the 3-bit test field, and the 6-bit data field. The test field specifies the condition code input that determines if a branch is to be taken. For conditional branches, if the condition is true (or false if the polarity bit is set to 1), a branch is taken to the

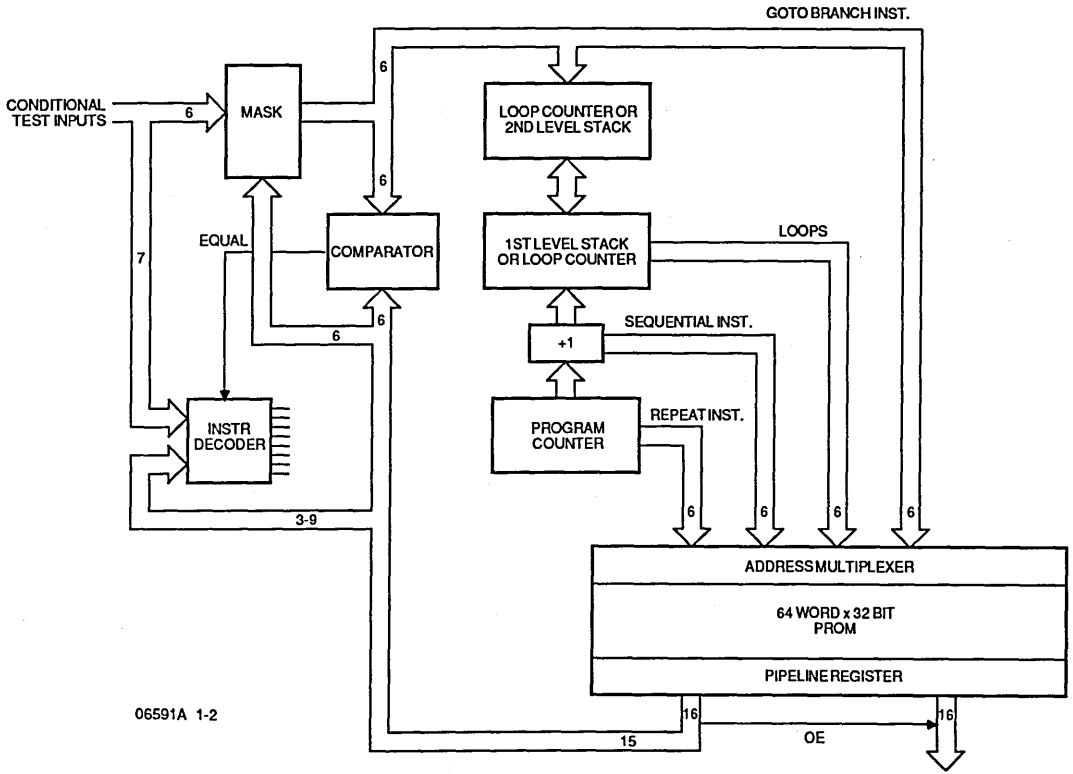


Figure 1-2. Am29PL141 Detailed Block Diagram

branch address specified in the data field. The 16th bit is an output enable line that enables the 16 output lines of the FPC.

1.2.4 Microprogram Memory and Pipeline Register

Conceptually, each memory location can be thought of as defining a particular state of the state machine, with each address corresponding to the number of this state. Seven external test inputs (T0, T1, T2, T3, T4, T5, and CC) and one internal test (EQ) are included to allow conditional state transitions based on external inputs. Typical microcode consists of testing one of the test inputs and branching if the condition tested is true.

The 64 by 32-bit fuse-programmable memory stores the microprogram of up to 64 microinstructions. It stores all state transitions. Each micro-instruction specifies the state of each of the 16 output lines used to control peripherals and other devices. The remaining fields in the micro-instruction have been described above. The internal PROM is programmed using commercially available PROM programmers.

The pipeline register associated with the memory is 32 bits wide and contains the microinstruction currently being executed. It allows concurrent execution of the current micro-instruction and fetching of the next instruction. Its upper 16 bits form the state sequencing and internal control logic. The low order 16 bits are used as general purpose, user definable control outputs. Of these user-controlled bits, the upper eight bits can be tristated by output enable bit (OE) in the microinstruction. If more than 16 output control bits are needed, Am29PL141s can be cascaded quite simply.

The FPC operates in two modes: normal and diagnostic. In the normal mode, a microinstruction is executed for every clock cycle. When the FPC is programmed to use the diagnostics feature, a 32-bit Serial Shift Register (SSR) is activated. This provides a simple, straightforward method of in-system testing to isolate problems to the individual IC level.

A 32-bit Serial Shadow Register (SSR) simplifies device and system-level diagnostics. To test a chip, an instruction is shifted serially into the SSR and then loaded in parallel into the pipeline. As a result, the instruction is executed and its results are transferred back from the pipeline into the SSR. From there, it may be shifted out for diagnosis.

1.3 MICROCODE

Microinstructions are 32 bits long. Up to 64 microinstructions can be stored in the 64 by 32-bit FPC memory. This Section discusses the micro-instruction formats, the instructions, and the SSR diagnostic option. For more detailed information, refer to the data sheet.

1.3.1 Mlicroinstruction Format

Each microinstruction is partitioned into fields. There are two microinstruction formats: the general microinstruction format and the Compare micro-instruction format. The low order 16 bits in each format contain 16 user-controlled output signals that appear on FPC outputs P[15:0].

In the general microinstruction format, the upper 16 bits are assigned as follows:

Bits	Description
16–21	Data (A conditional branch address, test input mask, or counter value)
22–24	Test (specifies which one of eight input signals to use for the condition code)
25	Polarity (specifies whether to test input for true or false)
26–30	Opcode (identifies microinstruction to execute)
31	Output Enable (when set to 0, it 3-states output lines P[15:8]

In the Compare microinstruction format, the upper 16 bits are assigned as follows:

Bits	Description
16–21	Data (A 6-bit mask for masking the T[5:0] inputs)
22–27	Constant (specifies a 6-bit constant for comparison with T*M for the condition code)
28–30	Opcode (identifies microinstruction to execute)
31	Output Enable (when set to 0, it 3-states output lines P[15:8]

1.3.2 Mlicroinstructions

The FPC microinstruction set consists of 29 instructions. These opcodes can be grouped into three groups:

Looping
Conditional
Unconditional

Looping. The four looping instructions use the counter CREG to execute loops. CREG is loaded prior to entering the loop. When CREG is not zero, CREG is decremented on each pass through the loop and a branch is performed to the beginning of the loop as specified in the data field. When CREG becomes zero, program execution continues at the next instruction following the loop.

One variation of the loop instruction loads the CREG with the value specified in the data field or the test inputs when CREG is zero. Another loop instruction tests both CREG and a test condition and branches to a specified location when either CREG becomes zero or the condition is true.

High-level language constructs such as REPEAT-UNTIL, WHILE-DO, and FOR are used to apply structured programming techniques to improve code readability and documentation. These same benefits can be realized in the FPC by implementing these constructs in microcode as follows:

REPEAT-UNTIL (condition) can be performed using one of the loop or branch instructions as the UNTIL condition test. WHILE-DO is implemented similarly except that the microcoded condition test is performed at the beginning of the loop instead of at the end.

An equivalent FOR construct uses a loop instruction to load CREG and then test for zero status. If it is zero, execution continues with the next instruction following the loop. If CREG is not zero, it is decremented by one and a branch is performed to the top of the loop. As with the other constructs, the designer can choose to do the CREG test at the top or at the end of the loop.

Conditional. Conditional instructions depend on the results of a test. If the test condition is not true, the action such as branch, push, load, or decrement is not performed.

The branch opcodes use the data field in the microinstruction to specify a branch address or subroutine location. In addition, multi-way branches or subroutine calls can be implemented by using the GOTO T*M or CALL T*M opcodes. In these instructions, the address is specified by the test (T) inputs masked by a mask value (M) in the data field. This allows a branch through the test inputs to a vector which can be used for interrupt servicing.

Other conditional instructions are pushing the program counter value onto the stack, loading CREG with a value, or decrementing CREG. Each of these conditional instructions is performed only if a test condition is true (or false if the Polarity bit is 1).

Unconditional. The two unconditional instructions are: CONTINUE and COMPARE. CONTINUE is used to generate the output bit and proceed to the next instruction. COMPARE unconditionally compares the test inputs with a user-defined constant and sets an internal equal bit (EQ) accordingly. This instruction is useful in searching for a character.

1.3.3 SSR Diagnostics

In the diagnostics mode, four of the device pins are redefined. These include one input and three output pins. The CC input is redefined as Serial Data In (SDI). The three output pins are redefined as Diagnostics Clock (DCLK), MODE, and Serial Data Out (SDO).

The shadow register can be serially loaded from the SDI pin under control of DCLK with MODE LOW. It can be parallel-loaded from the pipeline register with MODE HIGH, SDI LOW. The pipeline register is loaded from the microprogram memory during normal operation. During diagnostics, the pipeline register is loaded from the shadow register when MODE is High on the LOW to HIGH transition of CLK.

To run the diagnostics, a test pattern is serially loaded into the shadow register. From there, it is transferred to the pipeline register in parallel causing it to be executed. The results can then be clocked into the pipeline register, transferred to the shadow register, and serially shifted out for diagnosis.

1.4 Am29PL141 SOFTWARE SUPPORT

Designing complex state machines and intelligent controllers requires good software support.

The Am29PL141 software package simplifies the complete design process. Three major software modules—the Assembler, the Test Vector Generator, and the Simulator—are available. The Assembler is used to permit specifying the design in a symbolic language. The Test Vector Generator and the Simulator are used to simulate and verify the design.

1.4.1 Am29PL141 Assembler

The Am29PL141 Assembler converts design specifications written in a symbolic language into a JEDEC fuse map which can be used by other modules such as the Simulator.

The assembler allows data to be defined as bytes or words, permits forward label references, and allows assignment of values to bits in binary, octal, decimal, and hexadecimal format.

High level language constructs such as IF-THEN-ELSE and WHILE are directly supported by the assembler providing program structure and clear documentation for the designer.

The Assembler is described in detail in Chapter 2.

1.4.2 Am29PL141 Test Vector Generator

The Am29PL141 Test Vector Generator takes the test vector file (function table) generated by the designer and generates a JEDEC standard format test vector file. This output is used as an input by the Simulator.

1.4.3 Am29PL141 Simulator

Device simulation is based on a test vector file, generated by a Test Vector Generator from the test vectors specified by the designer. The Am29PL141 Simulator uses the JEDEC fuse map file (generated by the Am29PL141 Assembler) and the test vector file as its inputs. The Simulator generates computed output signals. These are compared with expected output values as specified in the test vector file. A printout of the output shows the differences if any.

The Simulator also provides an interactive mode allowing the designer to interactively preload or change any or all of the Am29PL141's internal registers. Single-step and breakpoints provide further control. For details, refer to Chapter 3.

1.5 AN OVERVIEW OF THIS TECHNICAL MANUAL

Chapter 2 describes the Am29PL141 Assembler. The assembler lets the user write microcode in a higher level language using mnemonics for addresses and using several number bases to represent numbers.

Chapter 3 describes in detail an Am29PL141 Fuse Programmable Controller Simulator. The simulator makes it possible to check out the logic of the microprogram before it is entered into the Am29PL141 chip memory.

Chapter 4 provides a simple, tutorial example of an Am29PL141 application. It is a coffee machine controller. This example shows not only the hardware but also the microprogram required.

Chapter 5 describes the realistic use of an Am29PL141 as an interface for the DEC PDP-11 Unibus. The complexity of Unibus handshaking is such that microprogramming is a reasonable design technique, but use of a separate sequencer, control memory, and pipeline register is not economical. Since the FPC contains a sequencer, memory, and pipeline, an interface for the DEC PDP-11 Unibus can be readily designed using the Am29PL141 FPC. It fits this class of problem rather well. The PDP-11 was chosen for this example because it has a well documented protocol which is familiar to many engineers.

Chapter 6 describes the use of an Am29PL141 as a controller for the DEC Q-Bus. The problem addressed is to design an interface between the Q-Bus and a generic device to allow the following operations:

- DATI/DATO with device as slave
- Device interrupt request (single level)
- Device Direct Memory Access request
- DATI/DATO with device as master

The control logic is implemented using the Am29PL141 FPC. Its microprogram implements a state machine to control both device and Q-Bus handshaking.

Chapter 7 describes the use of the Am29PL141 as a dual port memory arbitrator in a Starlan system. The Am29PL141 controls the DMA transfers to and from the relatively slow speed communication lines freeing the CPU to perform other tasks.

Chapter 8 describes the use of an IBM/PC to run diagnostic tests on a device containing a Serial Shadow Register (SSR). The Am29PL141 controls the flow of data to and from the SSR.

Chapter 9 describes an Am29PL141–QIC-02 and SCSI interface. This interface links tape drives with a CPU. It permits the backup of large hard disk drives on quarter-inch magnetic tape.

Chapter 10 Describes a high speed DMA controller using the Am29PL141.

The appendixes include the JEDEC Standard Number 3; an alphabetical listing of the Assembler Error Messages; the QIC-02 and SCSI timing diagrams; References; Glossary; Am29PL141 data sheet; and an Index.

CHAPTER 2

Am29PL141 ASSEMBLER

2.1 INTRODUCTION TO THE Am29PL141 ASSEMBLER

This section discusses the Am29PL141 assembler. It describes the features, the installation procedures, the assembler execution statements, the system requirements, and the assembler language elements.

2.1.1 ASSEMBLER FEATURES

The Am29PL141 Assembler provides higher level support for developing microprograms for the Am29PL141. This assembler accepts data defined as either bytes or words, allows forward references, and assigns values to bits in different formats (binary, octal, decimal or hexadecimal).

With the inclusion of high level language constructs such as IF-THEN-ELSE and WHILE, the microprogrammer's task is greatly simplified since the microcode is written in a logical and more natural flowing syntax. In addition, documentation of code is significantly enhanced since the microcode is expressed in a more readable and easy to follow format.

The assembler features include:

- High level language constructs
  - IF-THEN-ELSE
  - WHILE
- Binary, octal, decimal, and hexadecimal numbers are recognized
- Jump/branch to labels
- Logic equations for control outputs
- Error detection and diagnosis
- Default test condition
- JEDEC standard fuse map output
- Symbol table output

2.1.2 ERROR DETECTION AND DIAGNOSIS

Much effort has been made to provide relevant syntax error detection and diagnosis messages in order to facilitate debugging of errors occurring in the microcode. Note that one error may cause spurious errors to propagate through the assembler source file because the compiler logic is based on the expected sequence of symbols. The compiler does not understand the micro-code's intent or purpose. Correcting the first error and other meaningful errors (ex: variable name not defined in DEFINE section) will erase the spurious errors (ex: ',' symbol not defined).

The programmer can choose a default test condition to reduce the amount of microcoding since any conditional statement should refer to the default test condition as the condition. (Refer to Section 2.3)

The assembler will check the input file to determine that no conflicts exist in the use of input pins which double as SSR diagnostic pins.

2.1.3 SYSTEM REQUIREMENTS

The following hardware and software are required to use the assembler:

Hardware:

- An IBM PC/XT or other PC-compatible with at least 256K bytes of RAM memory
- Two double-sided, double-density floppy disk drives
- RS232 serial port and a cable to connect to a logic device programmer

Software:

- PC-DOS Version 2.0 or higher or MS-DOS 2.11 or higher
- A word processor to create the assembler source file. Any word processor which produces standard ASCII output files is acceptable. Example: Wordstar operating in Non-document mode.
- The following files are on the Am29PL141 Assembler disk:

FILENAME	DESCRIPTION
ASM141.EXE	Am29PL141 assembler
PL141	Database file for Am29PL141
COFFEE.EXP	Source file for coffee machine example
MAKE_CPY.BAT	Batch file for making copies and backups

2.1.4 MAKING BACKUPS

Before using the Am29PL141 assembler, make a backup copy of the distribution disk using the following procedure.

For two drive systems:

Put master diskette in drive A and a formatted blank diskette in drive B and type "MAKE_CPY B"

All the files in the distribution disk will be copied to the diskette in drive b.

For Hard disk systems :

1. Turn on the computer and boot up with DOS
2. Put the master/distribution disk in drive A
3. Set the system prompt to drive A by typing

   C>A: <CR>

4. Type "MAKE_CPY C" and press return.

Check that a directory "FPC" does not exist. "MAKE_CPY C" will create a directory called "FPC" and all the files on the distribution disk will be copied on to the hard disk in this directory. This batch file may be modified if the files are to be copied to a directory other than "FPC".

Store the distribution disk in a safe place. The copy just created is the working copy.

The recommended procedure is to make a backup of the program disk and use the backup copy as the working copy. In the event something happens to the working copy, a new working copy can be created by repeating the above procedure.

2.2 USER'S GUIDE

2.2.1 NOTATION

The Backus-Naur format (BNF) is used to describe the syntax of an action expression used in a statement. BNF is a short-hand notation with the following rules:

:= means "is defined as"

' ' literals must be enclosed in single quotes. High-lighted characters and characters in single quotes are literals and are required.

< > angle brackets enclose identifiers

[ ] square brackets enclose optional items

| | items separated by vertical bars indicate a choice between the items

2.2.2 RUNNING THE ASSEMBLER

The assembler takes an input source file written in the AM29PL141 syntax (described in Section 2.3) and produces a JEDEC fuse map which can be sent to a programmer.

Two system files are required to run the assembler:

The executable file ASM141.EXE and the database file PL141.

After a source file is created, it is assembled using the following command:

A> ASM141 -I <assembler file> [-O <fuse map file>] [-B <PROM bit file>] [-E <error message file>] [-T <symbol table file>] [-S]

where:

-I <filename> specifies an input file

-O <filename> specifies a destination file for the fuse map generated by the assembler

-E <filename> specifies a file to hold the assembler's error messages

-B <filename> displays and stores the PROM bit pattern into a file

-T <filename> displays a symbol table file

-S removes the SSR fuse from the JEDEC fuse map

Options O, E, B, T, and S are optional. The error messages and fuse map will always be displayed on the screen. The options do not have to be capitalized.

Examples :

A> ASM141 -I MYINPUT
ASM141 will process the input file named MYINPUT

A> ASM141 -I MYINPUT -O MAPOUT
ASM141 will process the input file MYINPUT and output the fusemap to the file named MAPOUT

A> ASM141 -I MYINPUT -B PROMBIT

ASM141 will process the input file named MYINPUT and store the PROM bit pattern into the output file named PROMBIT

A> ASM141 -i COFFEE.EXP -o COFFEE.JED -b COFFEE.BIT
ASM141 will process the assembler file named COFFEE.EXP and output a JEDEC fuse map file named COFFEE.JED and output a PROM bit pattern file named COFFEE.BIT.

2.2.3 ASSEMBLER OUTPUT

JEDEC Standard Fuse Map

ASM141 produces a fuse map file which follows the standards set forth by the Joint Electronic Device Engineering Council (JEDEC) for programmable devices. The fuse map file can be sent to a programmer via a communications program. Programmers from different manufacturers may have varying setup and communications parameters and procedures. Consult the programmer manual for more details. Information regarding the fields in the JEDEC fuse map file is detailed in Appendix A.

PROM Bit Pattern

When the 'B' option is specified, ASM141 displays the bit pattern for every word in the PROM that is translated into the JEDEC format.

Each word in the PROM is preceded by its decimal PROM address. Words are displayed from the lowest location to the highest (maximum of 63 for the Am29PL141).

The fields (e.g. DATA, OPCODE, TEST CONDITION) in the bit pattern are marked. This allows the microprogrammer to quickly check the contents of a field in a particular word.

2.3 LANGUAGE REFERENCE

The Am29PL141 Assembler language is used to program the Fuse-Programmable Controller Am29PL141. Logic designs and state-machines are described in this high-level language and translated into a format that can be loaded into a programmable logic device programmer. The device programmer then programs the Am29PL141 with this information.

This section describes the elements and structure of the assembler language. It is arranged as follows

2.3.1 *Language Elements*
Describes the elements used in the language.

2.3.2 *Assembler Program Structure*
Explains how the language elements are put together to describe a logic design.

2.3.3 *Statement Format*
Describes the general assembler statement format

2.3.4 *Statements Available for the Am29PL141*
Lists the statement forms which correspond to data sheet opcodes of the Am29PL141

2.3.5 *Quick Reference Guide*
Shows the flowcharts for the different statements and opcodes.

2.3.1 LANGUAGE ELEMENTS

The language consists of keywords and user-defined identifiers which are put together to form statements describing a logic design/state machine. These statements correspond to Am29PL141 machine level opcodes used to implement the state machine.

A source file must conform to the following rules and restrictions:

1. Comments are allowed in the assembler file for readability. Comments are enclosed between double quotes and can span more than one line. Comments cannot be nested.

2. Keywords and identifiers are separated by at least one space.

3. Keywords and identifiers can be in upper or lower case. No distinctions are made between the two alphabetic cases.

4. A line in the source file must not be more than 80 characters long. This is the normal width of the screen on a computer monitor.

Keywords

The following words are assembler keywords and cannot be used as variables:

```
DEVICE          T2      PUSH    CONTINUE
DEFAULT         T3      RET     OE
DEFINE          T4      DEC     OD
TEST_CONDITION  T5      WAIT    PL
SSR             CC      LOOP    TM
BEGIN           EQ      TO      SREG
END             GOTO    NESTED  CREG
T0              CALL    NEST    DEFAULT_OUTPUT
T1              LOAD    CMP      .ORG
```

Note: T0 to T5, CC and EQ are test conditions.

Identifiers

Identifiers are user-defined names identifying control output patterns, test pins, and labels.

The following rules apply to names and numbers used in the language:

- Variables, labels and constant names are limited to 29 characters in length. The first character should be an alphabetic character ('A' to 'Z' or 'a' to 'z') and the remainder can be alphanumeric characters or the underscore sign '_'.
- Numbers should be terminated with a '#n' where n is either B (binary), O (octal), D (decimal), or H (hexadecimal). If '#n' is left out, then the number is assumed to be decimal.

2.3.2 ASSEMBLER PROGRAM STRUCTURE

An assembler program source file describing a logic design or state machine contains seven sections which must appear in the following order:

(1) DEVICE
(2) SSR
(3) DEFAULT
(4) DEFINE
(5) DEFAULT_OUTPUT
(6) TEST_CONDITION
(7) Main body

DEVICE Section

This section must be specified for each file. It consists of the keyword DEVICE (need not be in uppercase) followed by the part name to be programmed in parenthesis.

Example:

```
DEVICE ( PL141 )
```

SSR Section

This is is an optional section which instructs the assembler to check for test conditions made unavailable during SSR diagnostics mode. The messages generated by the assembler will indicate which test condition pins have to be left solely for SSR diagnostics. This option will set the SSR fuse in the JEDEC fuse map. Default is SSR = 0 or no diagnostics.

Example:

```
SSR = 1; "a semicolon is necessary"
```

Note: Some device programmers may require the designer to blow the SSR fuse externally; i.e, SSR cannot be specified in the fuse map. In this case, use the "-S" option to remove the SSR fuse from the JEDEC fuse map.

The control output pins P[7] and P[6] are also used for SSR diagnostics. Because these are output pins to the user in normal mode, they are not flagged as errors if the user assigns a control output value using these two pins. If the SSR option is chosen, P[7] and P[6] will have undefined values.

DEFAULT Section

If a DEFAULT = 0 is specified, unspecified fuses will be blown, thus programming unspecified microcode words and fields at a logic level 0. If no DEFAULT statement is used or if a DEFAULT= 1 is specified, unspecified fuses will remain unblown, thus leaving unspecified microcode words and fields at a logic level 1.

DEFINE Section

Any variable name specified in this section can be assigned to a defined test condition or to a number by using the '=' sign. The last definition should end with a semicolon. This optional section is not needed if user defined names are not created.

Example:

```
DEFINE first = 1 "assign first the
                      decimal value 1"

       second = 2
       third = 3
       test = t0  "assign test to be
                    t0"
                  "condition T0"
       output1 = 45#H
       last_one = 0101001111#b;
```

DEFAULT_OUTPUT Section

This section is used to specify a default control output. This default output value is used if no control output expression is specified for an assembler statement.

Example:

```
DEFAULT_OUTPUT = FF0F#h;

Begin
"line1"          , if ( test = 0 ) then
                     goto pl (stateN) ;
```

```
"line2"   FF#h , if ( cond = 1 ) then
                   load pl(value) ;
    ....
    ....
end.
```

In the above example, the statement at line 1 uses the default control output. Statement 2 will use 'FF#h' as its control output.

TEST_CONDITION

A default test condition can be specified if only one test condition is being tested by the device. This reduces the number of 'IF <cond> THEN' strings because if a default test condition is specified, the aforementioned string is automatically concatenated with the action (ex. GOTO, CALL, LOAD).

The test condition works with assembler statements that use the form: 'IF (cond) THEN <action>' where condition is one of the eight test conditions T0 to T5, CC, or EQ .

Example:

```
TEST_CONDITION = t0; "specify the
                      default test
                      condition as
                      T0"

Begin
     ....
output_pattern1, ret;
     ....
output_pattern2, if ( cond = 1 ) then
                 goto pl
                 (a_defined_label);
     ....
     ....
End .
```

In the statement prefaced by output_pattern1, this statement becomes OUTPUT_PATTERN1, IF (t0) THEN RET; after default test condition replacement.

The default test condition can also be overridden in the same file by typing out the IF-THEN string with a different test condition (see line prefaced by output_pattern2).

Note that the default test conditions are limited to the name only; no comparisons or complements (<> , = or NOT) are allowed.

MAIN BODY

The main body must be enclosed by a single BEGIN-END block. Any number of statements as described in Section 2.3 can be inside this BEGIN-END block as long as the number does not exceed the total PROM size of the part being used (for the Am29PL141 the maximum number of statements is 64).

Example:

```
Begin
LABEL1: output1, if ( t0 = 0 ) then
        load pl(data);
        output2, while ( creg <> 0 )
        loop to pl(LABEL1);

End .   "terminate the block with a
          '.' "
```

2.3.3 STATEMENT ELEMENTS

A statement consists of the following elements:

- an optional label
- an output value
- a statement form

Example:

```
LABEL1: output, IF ( cond = 0 ) THEN
        GOTO PL(data);
```

Both the colon separating the label from the rest of the statement and the comma separating the output part from the statement are necessary to distinguish the elements from each other.

All statements are terminated by a ';' symbol.

SPECIFYING ADDRESSES

The assembler defaults to starting assembly at location 0. Successive statements go into successive locations. The pseudo operation .ORG followed by an address value can be used to change this. See the examples in the following section on Labels.

LABELS

Labels are names followed by a colon. Labels are permitted in the body of the program. This allows

Table 2-2. Am29PL141 Microprogram Instruction Set

	Opcode	Mnemonics	Assembler statement
(1)	19	GOTOPL	IF (cond) THEN GOTO PL(data)
(2)	0F	GOTOTM	IF (cond) THEN GOTO TM(data)
(3)	0B	GOTOPLZ	IF (CREG = 0)THEN GOTO PL(data)
(4)	18	FORK	IF (cond) THEN GOTO PL(data) ELSE GOTO (SREG)
(5)	1C	CALPL	IF (cond) THEN CALL PL(data)
(6)	1D	CALPLN	IF (cond) THEN CALL PL(data),NESTED
(7)	1E	CALTM	IF (cond) THEN CALL TM(data)
(8)	1F	CALTMN	IF (cond) THEN CALL TM(data),NESTED
(9)	04	LDPL	IF (cond) THEN LOAD PL(data)
(10)	05	LDPLN	IF (cond) THEN LOAD PL(data),NESTED
(11)	06	LDTM	IF (cond) THEN LOAD TM(data)
(12)	07	LDTMN	IF (cond) THEN LOAD TM(data),NESTED
(13)	15	PSH	IF (cond) THEN PUSH
(14)	17	PSHN	IF (cond) THEN PUSH,NESTED
(15)	14	PSHPL	IF (cond) THEN PUSH,LOAD PL(data)
(16)	16	PSHTM	IF (cond) THEN PUSH,LOAD TM(data)
(17)	02	RET	IF (cond) THEN RET
(18)	03	RETN	IF (cond) THEN RET,NESTED
(19)	00	RETPL	IF (cond) THEN RET,LOAD PL(data)
(20)	01	RETPLN	IF (cond) THEN RET NESTED,LOAD PL(data)
(21)	09	DEC	IF (cond) THEN DEC
(22)	0C	DECPL	WHILE (CREG <> 0) WAIT ELSE LOAD PL(data)
(23)	0E	DECTM	WHILE (CREG <> 0) WAIT ELSE LOAD TM(data)
(24)	1B	DECGOPL	IF (cond) THEN GOTO PL(data) ELSE WHILE (CREG <> 0) WAIT
(25)	1A	WAIT	IF (cond) THEN GOTO PL(data) ELSE WAIT
(26)	08	LPPL	WHILE (CREG <> 0) LOOP TO PL(data)
(27)	0A	LPPLN	WHILE (CREG <> 0) LOOP TO PL(data) ELSE NEST
(28)	0D	CONT	CONTINUE
(29)	10 - 13		CMP CMP TM(data) TO PL(data)

CHAPTER 3

Am29PL141 SIMULATOR and TEST VECTOR GENERATOR

3.1 OVERVIEW

The Am29PL141 Simulator, Test Vector Generator, and the Am29PL141 Assembler provide complete high level software support for the Am29PL141 device. Both the simulator and the test vector generator are designed specifically for the Am29PL141.

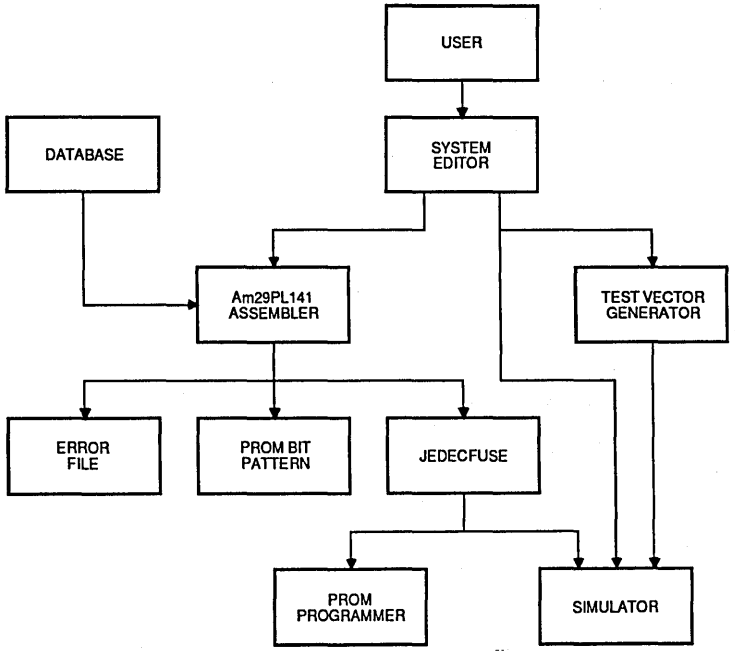
3.1.1 SIMULATOR FEATURES

The Am29PL141 simulator provides high level interactive simulation capability for the Am29PL141 microprograms. Along with the Assembler and Test Vector Generator, it helps to verify Am29PL141 designs completely before a device is programmed. The simulator supports functional simulation only. It does not provide any timing simulation.

The Am29PL141 simulator uses the Jedec fuse map file (generated by the Am29PL141 Assembler) and the test vector file (generated by the Am29PL141 Test Vector Generator) as its inputs (Figure 3-1). Based upon the contents of the Jedec fuse map and the test vector file, it generates "computed output signals" and compares these against expected output values as specified in the test vector file or interactively by the user. If any differences are detected, it flags the errors by displaying a "?" under the unmatched output signals. For any outputs in the test vector for which the computed results or contents of the Jedec fuse map file (omitting the output signals) are desired, "X" or "N" must be specified.

3.1.2 Am29PL141 SIMULATOR DISTINCTIVE CHARACTERISTICS

- Allows the user to preload or change all internal registers (interactively)
- Displays complete status information including



06541A 3-1

Figure 3-1. Simulator/Test Vector Environment

all input pin signals, computed and expected output signals, contents of all internal registers
- Break point capability
- Single step capability
- Simulates SSR diagnostic mode
- Default values for test vectors
- Interactive mode of operation
- Jedec fuse map file used as simulated micro program memory
- Another program can be executed during simulation

3.1.3 SIMULATOR REQUIREMENTS

The following steps are required to run the simulator:

A. Write and Assemble a microprogram source file

Write a micro-program using the Am29PL141 Assembler language. Then use the Am29PL141 Assembler to assemble the program. The Assembler will generate the corresponding Jedec fuse map file to be used by the simulator. Refer to Chapter 2 for details about writing and assembling a micro-program.

B. Create Test Vectors File

The source test vectors file can be written in a symbolic format. This test vector source file is transformed into the Jedec standard, structured functional test information format by the Am29PL141 test vector generator. The output of the Am29PL141 test vector generator is called the Test Vector file. (Please refer to the Am29PL141 Test vector generator description in Section 3.2 for details.

Keeping micro program source and test vector files separate allows one simulation model to have a set of different test vector files.

C. Execute simulation

After the source program is assembled and the test vectors file has been generated, the simulator is ready to run. The details of running the simulator are presented in Section 3.3.

The simulator model is designed to reflect the Am29PL141 device as much as possible. Initially applying a software asserted RESET signal to the simulator is the same as applying a RESET to the Am29PL141 device. On the next rising clock edge after a RESET, a value of 63 is loaded into the microprogram counter; the microinstruction at location 63 is loaded into the pipeline register and the EQ flag is cleared. However, if the RESET signal is not asserted in the beginning, the simulator assumes its stage right after a RESET.

Please note, that Am29PL141 simulator provides functional simulation only—no timing simulation. The simulator assumes 0 propagation delay. However, the clock pulse must be specified as one of the inputs in the test vectors to get register transfers and to compute outputs.

3.2 Am29PL141 TEST VECTOR GENERATOR

3.2.1 INTRODUCTION

This section describes the test vector generator program, TEST41, and the syntax of the function table. The source test vector file is defined as the function table created by the user (using a text editor). The function table is written in a symbolic format which the Am29PL141 Test Vector Generator (TEST41) can transform into JEDEC standard test vector format. The syntax of the function table is quite similar to that of PLPL. The output of TEST41 is in the JEDEC standard format. It can be used as the input file to the Am29PL141 simulator or sent to the device programmer.

The function table enables the user to easily specify his own set of test vectors to verify his microprogram design.

3.2.2 FUNCTION TABLE SYNTAX

The function table has the following format:

```
                [Table heading]

(PIN)

        Pin declaration
        ;

(VECTORS)

        (IN) Input pin names ;
        (OUT)Output pin names ;

       (BEGIN)

         Test vectors

       (END.)
```

Keywords are enclosed in parentheses. The optional fields are enclosed in brackets.

Table Heading: The heading comprises the first

arbitrary number of lines before the keyword "PIN". The table heading is provided as design documentation.

Pin Declaration: The purpose of pin declarations is to let users specify symbolic names for device pins, so that user-defined identifiers can be equated to physical device pins. The first five character of the specified name are displayed vertically on the simulator screen. Note, the pin names in Figure 3-2 and the resulting simulation screen display.

The pin declaration begins with the keyword "PIN" and is terminated with a semicolon. All the pin assignments appear within the keyword PIN and the semicolon.

The syntax for a single pin assignment is as follows:

pin_name = pin_number
Example : CLK = 27

The difference in pin declarations between PLPL (Programmable Logic Programming Language) and Am29PL141 software supporting package TEST41 is that TEST41 can not support vector-type pin declaration. Thus, the following pin assignment is illegal in TEST41 environment:

T[5:0] = 20:25

The correct form is:

T5 = 20
T4 = 21
T3 = 22
T2 = 23
T1 = 24
T0 = 25

Vector Table Body: The vector table body begins with the keyword "VECTORS". A list of signal declarations follows the keyword VECTORS. This list specifies that all signals in test vector body be identified in the same sequence as they appear in this list. TEST41 will identify and display the vectors according to the order in the names are given in the list.

```
HEADER (Am29PL141)
Test for Instr PSH

PIN     clk = 27
        /reset = 19
        t5 = 20          t4 = 21
        t3 = 22          t2 = 23
        t1 = 24          t0 = 25
        cc = 26          p15 = 18
        MEMRQ = 17       p13 = 16
        p12 = 15         p11 = 13
        IORD = 12        IOWR = 11
        p8 = 10.         p7 = 9
        p6 = 8           p5 = 7
        p4 = 6           p3 = 5
        p2 = 4           p1 = 3
        p0 = 2
        /zero = 1 ;

VECTORS

IN      clk /reset t5 t4 t3 t2 t1 t0 cc ;
OUT     p4 p5 p6 p7 p15 MEMRQ p13 p12 p11 IORD IOWR p8 p3 p2 p1 p0 /zero ;

BEGIN
"                                                    "
"       c  /                  M                 /    "
"       l  r                  E    II           z    "
"       o  e                  M    OO           e    "
"       c  s  ttttttc  pppp 1R11 1RWp  pppp  r    "
"       k  e  543210c  4567 5Q32 1DR8  3210  o    "
"                                                    "
"  TEST INSTR FOR FAIL CONDITION                     "
"                                                    "
1       C  0  1000000  LHHL LHHL LLHH  LLHH  X    ;
2       C  1  011100X  HHLL LLHH LLHL  LLHL  X    ;
3       C  1  1001100  HHLL LLHH LLHH  LLHH  X    ;
4       C  1  000101X  HHLL LLHH HLLL  HLLL  X    ;
5       C  1  1001000  HHLL LHHH HHHH  HLLH  H    ;
6       C  1  1XXXXXX  HHLL LLHH LHHL  LHHL  H    ;
END.
```

Figure 3-2. TEST41 Input File (Function Table) Example

There are two fields in this signal declaration list—IN, and OUT. The IN field contains the input signals to the device. The OUT field contains the output signals of the device. The signals are displayed in the simulation screen display in the order in which they are listed in the IN and OUT fields. Figure 3-2 shows the IN and OUT declarations. Refer to Figure 3-3 for the resulting simulation display.

Test Vector Format: The test vectors are embodied between the keyword "BEGIN" and "END.". Each test vector starts with a vector number and ends with a semicolon. The vector number can be any decimal integer with 4 or fewer digits. The test vectors in the TEST41 output file are in JEDEC standard format and have the same vector numbers specified in the source function table by the user. This makes cross reference easier.

The test vectors must contain only valid JEDEC test conditions:

0 – drive input low
1 – drive input high
C – drive input low, high, low
N – power pins and outputs not tested
L – test output low
H – test output high
Z – test output for high impedance
X – don't care

There is a direct one to one correspondence between each entry in a test vector and the pin signal within the test vector table. For example, in Figure 3-2, the first signal 'C' in the test vector table body corresponds to the CLOCK input signal while the second test signal bit corresponds to the RESET signal. The simulator maps test signals to their corresponding pre-defined pin locations. The test vector signals must be in the same order as the listing of the pins in the "IN" and "OUT" list.

The signals not included in the "IN" and "OUT" list are treated as don't cares(i.e. 'X') in the JEDEC format. There is a direct one to one correspondence between each test signal and the "IN" and "OUT" list.

Regardless of the order in which the vector signals are displayed, the output vector in the pipeline is

```
-------------------------------------------------------------------------------
V0001          INPUT                |              OUTPUT
          /                         |              M
          R                         |              E           I  I
Pin     C E                         |           P  M  P  P  P  O  O
Name  : L S  T  T  T  T  T  T  C    | P  P  P  P 1  R  1  1  1  R  W  P  P P P
        K E  5  4  3  2  1  0  C    | 4  5  6  7 5  Q  3  2  1  D  R  8  3 2 1
                                    |
Pin # : 27 19 20 21 22 23 24 25 26  | 6  7  8  9 18 17 16 15 13 12 11 10 5 4 3
Vector: C  0  1  0  0  0  0  0  0   | L  H  H  L L  H  H  L  L  L  H  H  L L H
                        Computed :  | L  H  H  L L  H  H  L  L  L  H  H  L L H

CREG = 0 ,    SREG = 0 ,     PC = 63,     EQ = 0
     #X00          #X00          #X3F

Pipeline :    OE    OPCODE    POL    TEST    DATA      OUTPUTS
              1     #X0F      1      6       63        #B0110001101100011
                                             #X3F      #X6363
OPCODE MNEMONICS : GOTOTM
Current PL Contents loaded from ROM address 63

-------------------------------------------------------------------------------
V0002          INPUT                |              OUTPUT
          /                         |              M
          R                         |              E           I  I
Pin     C E                         |           P  M  P  P  P  O  O
Name  : L S  T  T  T  T  T  T  C    | P  P  P  P 1  R  1  1  1  R  W  P  P P P
        K E  5  4  3  2  1  0  C    | 4  5  6  7 5  Q  3  2  1  D  R  8  3 2 1
                                    |
Pin # : 27 19 20 21 22 23 24 25 26  | 6  7  8  9 18 17 16 15 13 12 11 10 5 4 3
Vector: C  1  0  1  1  1  0  0  X   | H  H  L  L L  L  H  H  L  L  H  L  L L H
                        Computed :  | H  H  L  L L  L  H  H  L  L  H  L  L L H

CREG = 0 ,    SREG = 0 ,     PC = 32,     EQ = 0
     #X00          #X00          #X20

Pipeline :    OE    OPCODE    CONSTANT    DATA      OUTPUTS
              1     #B100     28          63        #B0011001000110010
                              #X1C        #X3F      #X3232
OPCODE MNEMONICS : CMP
Current PL Contents loaded from ROM address 32
```

Figure 3-3. Simulator Output File Example

```
V0003              INPUT                    |              OUTPUT
                  /                         |                 M
                  R                         |                 E                  I  I
Pin            C  E                         |                    P  M  P  P  P   O  O
Name  : L  S  T  T  T  T  T  T  C           | P  P  P  P  1  R  1  1  1  R  W  P  P  P  P
        K  E  5  4  3  2  1  0  C           | 4  5  6  7  5  Q  3  2  1  D  R  8  3  2  1

Pin # : 27 19 20 21 22 23 24 25 26          | 6  7  8  9 18 17 16 15 13 12 11 10 5  4  3
Vector: C  1  1  0  0  1  1  0  0           | H  H  L  L  L  L  H  H  L  L  H  H  L  L  H
                        Computed :          | H  H  L  L  L  L  H  H  L  L  H  H  L  L  H

CREG = 0 ,    SREG = 0 ,     PC = 33,       EQ = 1
    #X00          #X00           #X21

Pipeline :    OE    OPCODE    POL    TEST    DATA      OUTPUTS
              1     #X0F      1      6       63        #B0011001100110011
                                              #X3F      #X3333
OPCODE MNEMONICS : GOTOTM
Current PL Contents loaded from ROM address 33

-------------------------------------------------------------------------------
V0004              INPUT                    |              OUTPUT
                  /                         |                 M
                  R                         |                 E                  I  I
Pin            C  E                         |                    P  M  P  P  P   O  O
Name  : L  S  T  T  T  T  T  T  C           | P  P  P  P  1  R  1  1  1  R  W  P  P  P  P
        K  E  5  4  3  2  1  0  C           | 4  5  6  7  5  Q  3  2  1  D  R  8  3  2  1

Pin # : 27 19 20 21 22 23 24 25 26          | 6  7  8  9 18 17 16 15 13 12 11 10 5  4  3
Vector: C  1  0  0  0  1  0  1  X           | H  H  L  L  L  L  H  H  H  L  L  L  H  L  L
                        Computed :          | H  H  L  L  L  L  H  H  H  L  L  L  H  L  L

CREG = 0 ,    SREG = 0 ,     PC = 38,       EQ = 1
    #X00          #X00           #X26

Pipeline :    OE    OPCODE    POL    TEST    DATA      OUTPUTS
              1     #X16      0      7       63        #B0011100000111000
                                              #X3F      #X3838
OPCODE MNEMONICS : PSHTM
Current PL Contents loaded from ROM address 38

-------------------------------------------------------------------------------
V0005              INPUT                    |              OUTPUT
                  /                         |                 M
                  R                         |                 E                  I  I
Pin            C  E                         |                    P  M  P  P  P   O  O
Name  : L  S  T  T  T  T  T  T  C           | P  P  P  P  1  R  1  1  1  R  W  P  P  P  P
        K  E  5  4  3  2  1  0  C           | 4  5  6  7  5  Q  3  2  1  D  R  8  3  2  1

Pin # : 27 19 20 21 22 23 24 25 26          | 6  7  8  9 18 17 16 15 13 12 11 10 5  4  3
Vector: C  1  1  0  0  1  0  0  0           | H  H  L  L  L  H  H  H  H  H  H  H  H  L  L
                      Computed :            | H  H  L  L  L  L  H  H  H  L  L  H  H  L  L
                    Unmatched :             |                ?           ?  ?

CREG = 5 ,    SREG = 39,     PC = 39,       EQ = 1
    #X05          #X27           #X27

Pipeline :    OE    OPCODE    POL    TEST    DATA      OUTPUTS
              1     #X0F      1      6       63        #B0011100100111001
                                              #X3F      #X3939
OPCODE MNEMONICS : GOTOTM
Current PL Contents loaded from ROM address 39

-------------------------------------------------------------------------------
V0006              INPUT                    |              OUTPUT
                  /                         |                 M
                  R                         |                 E                  I  I
Pin            C  E                         |                    P  M  P  P  P   O  O
Name  : L  S  T  T  T  T  T  T  C           | P  P  P  P  1  R  1  1  1  R  W  P  P  P  P
        K  E  5  4  3  2  1  0  C           | 4  5  6  7  5  Q  3  2  1  D  R  8  3  2  1

Pin # : 27 19 20 21 22 23 24 25 26          | 6  7  8  9 18 17 16 15 13 12 11 10 5  4  3
Vector: C  1  1  X  X  X  X  X  X           | H  H  L  L  L  L  H  H  L  H  H  L  L  H  H
                        Computed :          | H  H  L  L  L  L  H  H  L  H  H  L  L  H  H

CREG = 5 ,    SREG = 39,     PC = 36,       EQ = 1
    #X05          #X27           #X24

Pipeline :    OE    OPCODE    POL    TEST    DATA      OUTPUTS
              1     #X15      1      5       0         #B0011011000110110
                                              #X00      #X3636
OPCODE MNEMONICS : PSH
Current PL Contents loaded from ROM address 36

-------------------------------------------------------------------------------
Simulation Completed    3 simulation error(s) found
```

Figure 3-4. Function Table Example 2

given in ascending order from right to left by pin number.

3.2.3 RUNNING TEST41

Format: `TEST41 [-e] [-o OUTFILE] INFILE`

Meaning:

-e	Suppress the output to the CRT
-o OUTFILE	Write the test vector generator output to the file named "OUTFILE"
INFILE	The test vector source file (Function Table) Refer to Figure 3-2 .

3.3 EXECUTING SIMULATIONS

3.3.1 INVOCATION COMMAND OPTIONS AVAILABLE

This section describes the options which can be supplied to the simulator in the invocation command. Option flags are prefaced with a minus "–" character. Options can be specified in upper case or lower case characters. Some options require a parameter. The parameter following the option flag must be typed. The order of appearance of options is not significant as long as they are situated between the command name and the JEDEC fuse map file name.

The invocation command line format is:

```
sim41 [-edsr] [-b adr] [-x val] [-o
out] [-p val] -t tstvec -ntstin
jedfile
```

Where:

-e Suppress the output to the CRT.

-d Simulate in the SSR diagnostic mode.

-s Suppress single step mode. Without this option, simulator pauses after each test vector is simulated until the user gives a command (carriage return) to let it continue. During the pause, users can enter commands interactively. Refer to Section 3.3.5 for details.

-r Suppress displaying the contents of all internal registers.

-b adr Set break point at microprogram memory address adr. When the microinstruction at address adr is loaded into pipeline register and executed, the simulator pauses and waits for commands from users. The address adr must be a decimal integer from 0 to 63.

-x val Set default value for "X" in test vectors. Val is either 1 or 0. Without this option, the default value of "X" is 0.

-o out Write the simulator output to the file name "out".

-p val Preload internal registers. Which internal register is preloaded depends on the first character of "val" as follows:

First Character	Register Preloaded
P	PC
C	CREG
S	SREG
E	EQ

The next character following E is the decimal integer 0 or 1. For all other registers, the first character is a value from 0 to 63.

-t tstvec Specifies the test vector file, the output file of test vector generator program TEST41. It is required.

-n tstin Specifies the input source file also used in the test vector generator program TEST41. It is required.

jedfile The last command line argument is not optional. It is the Jedec fuse map file name.

Some examples of simulation command lines which demonstrate correct invocation syntax are:

```
sim41  -sr  -t  cntr.t1  -n  cnta
          cntr.jed
sim41  -t cntr.t2 -n cnta cntr.jed
sim41  -es -o tmpfile -x 0 -r -t
          cntr.t3 -n cnta cntr.jed

sim41  -esr -t tmpfile 0 -n cnt
          cntr.jed
sim41  -p p21 -o tmpfile -n cnt
          cntr.jed
sim41  -p p0 -p c63 -p s0 -t cntr.t1
       -n cnt cntr.jed
```

The files cntr.t1, cntr.t2, and cntr.t3 are TEST41 output vector files. The files cnta and cnt are TEST41 input files. Both the source name and the output file name are required. The file cntr.jed contains the JEDEC fuse map.

3.3.2 SIMULATOR OUTPUT

After each test vector is simulated, the simulator outputs a snapshot. The output snapshot contains the result of the simulation for the test vector, the test vector input with pin number and pin name, and a vector number of the vector last fetched. Each vector number with an initial character of "V" is a vector fetched from the test vector file.

The contents of internal registers are also displayed. If the single step mode is not suppressed, the simulator will prompt the user for commands by displaying "!". The user can either enter commands or press <CR> to continue the simulation. Among the commands is a help command "H" which, when invoked, shows all available commands to the user. An example of simulation output is shown in Figure 3-3.

The test vector number "V0001" appears on the upper left corner of Figure 3-3. This means that the test vector "V0001" in the test vector file was used as the input. The vector numbers preceded by a 'V' are the same as the vector numbers fetched by the simulator from the test vectors file.

The first line displays the vector file number and the "INPUT" and "OUTPUT" column titles. The next five lines give the pin names displayed vertically. Shown below the pin names are pin numbers. The pin names and numbers appear in the order they are specified in the TEST41 source file in the "IN" and "OUT" list. Refer to Section 3.2. Below the pin numbers are the test vectors given to the simulator. Below the test vectors are the output signals computed by the simulator. If the user specifies expected output signals ('H' or 'L'), the simulator compares the expected output signals with the computed output signals and shows any unmatched signals by displaying '?' under them.

Unless users apply "-r" option when invoking the simulator, the simulator displays the contents of all internal registers, OPCODE mnemonics of the current OPCODE field in the pipeline register, and the source of the contents of pipeline register. For CREG, SREG, PC, and DATA field of pipeline register, both decimal and hexadecimal values are displayed. The hexadecimal values are displayed below their decimal values. For the number representations, please see Section 3.3.6.

The simulator output file shown in Figure 3-3 is the result of running the file shown in Figure 3-2 along with a related Jedec map. In the first test (V0001), the CC input (pin 26) is tested for a low signal and since it is a Zero, a branch is taken to the address in T[5:0] which is decimal 32 or hex 20. In the next step, the value 32 is in the PC. The opcode calls for a compare with the constant value 28. Since the T[5:0] input is 011100 which is hex 1C or decimal 28, the compare is true and the EQ flag is set as seen in the next display (V0003).

The PC is incremented to 33. The value of the test input of vector V0003 is 38. The condition to be tested is test 6 (CC) for LOW or false. Since bit 26 is zero, the branch is taken to address 38.

The next display (V0004) shows that the EQ bit is still set to ONE and the PC is at address 38. The EQ is not reset to Zero unless EQ is the condition in a branch or a reset is executed. In this step, the EQ bit is tested for true as indicated by the test number (7) with the polarity bit set at Zero. As a result, the address PC + 1 (39) is pushed into the SREG. The CREG is loaded from the T[5:0] field masked by the data field (#X3F) placing the value 5 into the CREG. V0005 shows these register values. The EQ bit remains at "1" because it was the condition in a PUSH instruction, not a branch instruction.

The pipeline is loaded from address 39 for V0005. The condition to be tested for false is test 6 (CC). Since CC (pin 26) is Zero, the branch is taken to the address specified by the input T[5:0] masked by the data field giving an address of 36 decimal or 24 hexadecimal.

There are three errors in the V005 output vector as shown by the three question marks in the "unmatched" line. This did not affect the simulation results but would affect the application control signals.

The last test vector shown in this example (V0006) tests the T5 input. Since it is a ONE and the polarity bit is set to One (test for false), no branch is taken. This test vector is not the last vector in the application being simulated but it is the last vector in this simulator run. Therefore, a summary of the number of errors found is displayed following this vector.

3.3.3 TERMINATING THE SIMULATION

When not in the single-step mode, the simulation is terminated when the simulator has read all of the

lines in the test vector file. When in the single-step mode, the user terminates the simulation by entering the command EX. After the simulation is terminated, the simulator tells the user how many unmatched output signals were detected by displaying the total number of such errors as seen in Figure 3-2.

3.3.4 SSR DIAGNOSTICS SIMULATION

To choose th SSR diagnostics option, one must specify the "-d" option in the simulator invocation command. In addition, MODE, DCLK, and SDI input signals must be specified in test vectors file. The simulated internal registers include the Shadow Register on this option.

3.3.5 INTERACTIVE COMMAND SET

To use interactive commands, one must be in the single-step mode. Single-step mode is specified in the invocation command by not specifying the -S option which suppresses the single-step mode. An interactive command is defined to be the contents of a single text line. Only one command is allowed on a single text line. The simulator prompts for commands with the prompt "!". There is no difference between the upper and the lower case characters.

Each command line begins with the name of the command. Some commands require arguments. If the user does not enter the argument in the command line, the simulator will prompt for the argument. The argument for interactive commands may be a binary, octal, decimal, or hexadecimal number. Usually, the argument specifies the contents of a register or a PROM address. The argument should not exceed the range of values that a register can hold. If a user specifies an argument beyond the range, the argument will not be accepted. All invalid arguments leave the contents of the internal register intact. Note: the simulator refers to interactive commands as subcommands to distinguish them from the invocation command.

The space between commands and arguments is optional.

The following commands are currently available:

LP [arg]	Load argument into Program Counter (PC).
LC [arg]	Load argument into Count Register (CREG).
LS [arg]	Load argument into Subroutine Register (SREG)
LL	Load pipeline register. The simulator will prompt for each field's contents.
SQ	Set EQ flag.
RQ	Reset EQ flag.
SS	Set single step mode. It is used to resume after reaching a breakpoint in single step mode.
CS	Cancel single step mode.
SB [arg,..]	Set break point at PROM address specified by argument.
CB [arg,..]	Cancel break point at PROM address specified by argument. If the break point is not set yet, this command has no effect.
CBA	Cancel all break points.
DB	Display break points currently set.
RUN [prog]	Run another program. The full device and pathname for COMMAND.COM must be given by an entry in the environment, with "COMSPEC". This may be checked by using the DOS command "SET" at the system command level (not at the interactive command level).
EX	Terminate simulation and exit to OS when entering interactive commands from the keyboard in the single-step mode.

Command Examples:

Shown below are some examples of the usage of interactive commands. The underlined characters are entered by the user. The other characters are prompts by the simulator.

```
! lp 10

! lp
PC = 10

! lc#b0000000

! lc
CREG = #X1a

! run dir
```

```
! run type myfile

! sb 1,2,3,4

! cb
Break Point = 2,3

! db
Break point(s) : 1,4

! cba

! db
Break point(s) :
```

3.3.6 NUMBERS

Much of the information passed between the simulator and the user is expressed in numeric form. Input numbers are typed by the user, and accepted by the simulator. Output numbers are generated by the simulator, and viewed by the user. Each number is in one of four bases: 2, 8, 10, or 16.

On input, the user must specify the intended number base. This is done by either an explicit number prefix, or by following the prompt of the simulator.

The simulator output format is fixed. The user can neither change the format nor the number base. However, if the internal form of an output data has a bit width more than or equal to 6, the simulator displays the data (with the exception of opcodes) in both decimal and hexadecimal form. Opcodes are only displayed in hexadecimal form. The output field of the pipeline register is displayed in both binary and hexadecimal forms.

Uppercase characters do not differ from lowercase characters in number representation.

Binary Numbers: Input and output binary numbers are represented as a string of '0' and '1' digits prefaced with the string "#B".

Octal Numbers: Input and output octal numbers are represented as a string of the digits '0' through '7' prefaced with the string "#O".

Decimal Numbers: Input and output decimal numbers are represented as a string of the digits '0' through '9'. Decimal numbers may optionally be prefaced with the string "#D".

Hexadecimal Numbers: Input and output hexadecimal numbers are represented as a string of the digits '0' through '9' and the letters 'A' through 'F' prefaced with the string "#X".

Number Examples:

#d123	(decimal)
123	(decimal)
#B0110	(binary)
#b101	(binary)
#O077	(octal)
#xa1	(hex)
–#b100	(invalid, the simulator does not accept negative numbers)

CHAPTER 4

COFFEE MACHINE CONTROLLER USING Am29PL141

This section is a tutorial to show designers how to go from a design requirement to Am29PL141 microcode. The coffee machine application was chosen because it is easy to understand. Obviously, the Am29PL141 will never be used for such a slow application.

The following example describes the hardware and the programming required. A flow diagram of the program is included. The assembler program for the coffee vending machine example is called COFFEE.EXP. The Am29PL141 assembler produces two outputs, the JEDEC fuse map output file (COFFEE.JED) and the PROM bit pattern output file (COFFEE.BIT. First, the problem is defined.

The coffee machine controller waits for a coin before dispensing the beverage selected by the customer. The choices are indicated as combinations of buttons.

Design requirement:

Design a coffee machine controller that works as follows:

1. Do nothing until a coin is detected.
2. On coin detection turn on busy light and wait for selection:

   i. coffee
   ii. chocolate
   iii. soup
   iv. coin return

3. If coin return is detected, return coin, turn off busy light and wait for next coin.
4. If coffee, chocolate or soup is detected, drop a cup.
5. The cup has 1.5 seconds to get into place.
6. Turn on water for 1 second prior to release of powders.
7. Water will remain on continuously for a total of 10 seconds.
8. Busy light will remain on until end of sequence.
9. Depending on selection, either coffee, soup or chocolate will be dispensed:

coffee	2.5 seconds
soup	2.0 seconds
chocolate	3.5 seconds

10. If coffee was selected, check to see if cream and/or sugar are selected. If yes, cream 2.0 seconds, sugar 1.5 seconds.
11. After water has completed filling the cup, allow 3.5 seconds for cup removal before testing for presence of next coin.
12. Clock rate is 10 Hz.

As can be seen, there are six possible beverages:

i. coffee black
ii. coffee with sugar
iii. coffee with cream
iv. coffee with cream and sugar
v. chocolate
vi. soup

The conditions that need to be tested are:

i. coin drop
ii. coffee
iii. cream
iv. sugar
v. chocolate
vi. soup
vii. return (coin return)

Control signals that need to be generated from the controller are:

i. busy light on (busy)
ii. cup drop (cup)
iii. water on (water)
iv. coffee on (coffee)
v. cream on (cream)
vi. sugar on (sugar)
vii. chocolate on (choclat)
viii. soup on (soup)
ix. coin return (coin_return)
x. clear inputs (clr_inp)

Figure 4-1 represents the hardware required for the controller. The inputs need to be synchronized and latched hence the PAL device (16R8). Once latched, the clr_inp signal from the Am29PL141 clears the external registers within the Pal at the end of each sequence. The Am29PL141 has seven external test inputs. These are used to test the seven conditions. Since all but one of the Am29PL141 instructions are conditional, unconditional jumps must be implemented by a 'forced pass'. The 'EQ' flag internal to the Am29PL141 is a test condition not

being used in this design. It can therefore be used to allow 'unconditional' instructions. The state of the 'EQ' flag is always known since it is unused for any other purpose. (The 'EQ' flag is cleared on reset).

Figure 4-2 is the flow diagram for the program. It describes the logical flow of events required by the design. The rectangular boxes in the flow diagram show the value of the control field for that state. The diamond shaped boxes imply a conditional test to decide the next state. A pair of rectangular and diamond shaped boxes indicate a conditional microcode line. A rectangular box not followed by a diamond shaped box implies that the instruction is a continue or an unconditional branch.

The Am29PL141 is used to develop the microcode. Figure 4-3 is a listing of the Assembler source code used. It is assumed that the reader is familiar with the Am29PL141 assembler (described in Chapter 2) supplied by Advanced Micro Devices. Note that all timing is in 0.5 second increments. At 10 Hz, 0.5 second corresponds to 5 clocks.

Each box in the flow diagram can be directly translated into one or more lines of microcode. One important convention needs to be remembered. Each microcode line specifies the state of the control outputs and the branch address for the NEXT instruction. Hence in the flow diagram, the decision box follows the output field box. The flow diagram indicates the microcode line numbers corresponding to each box.

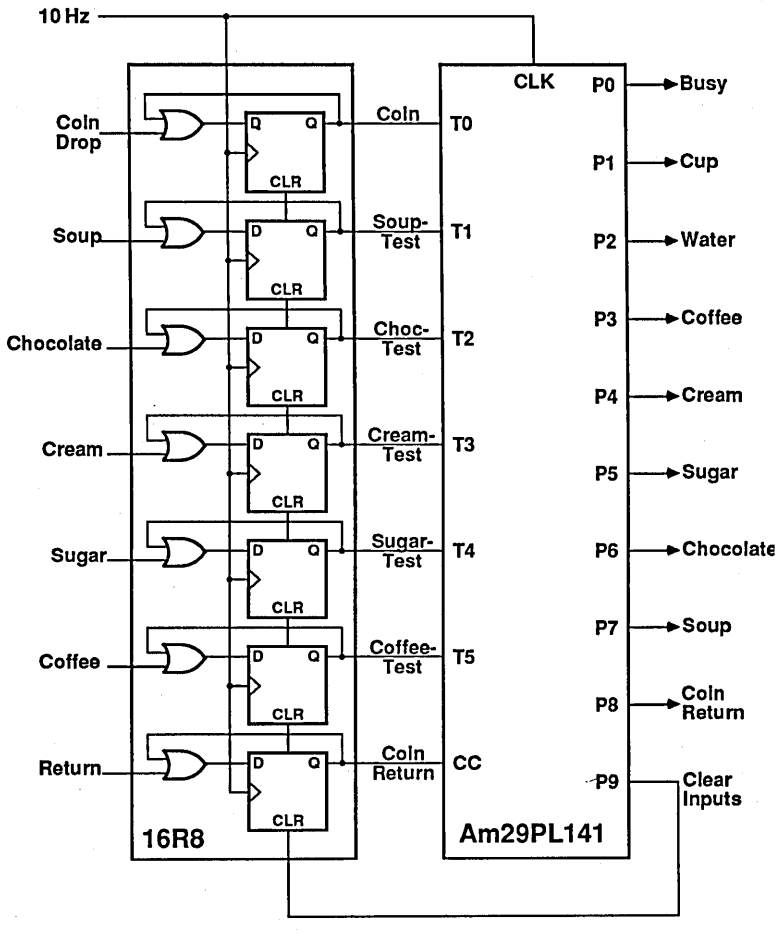
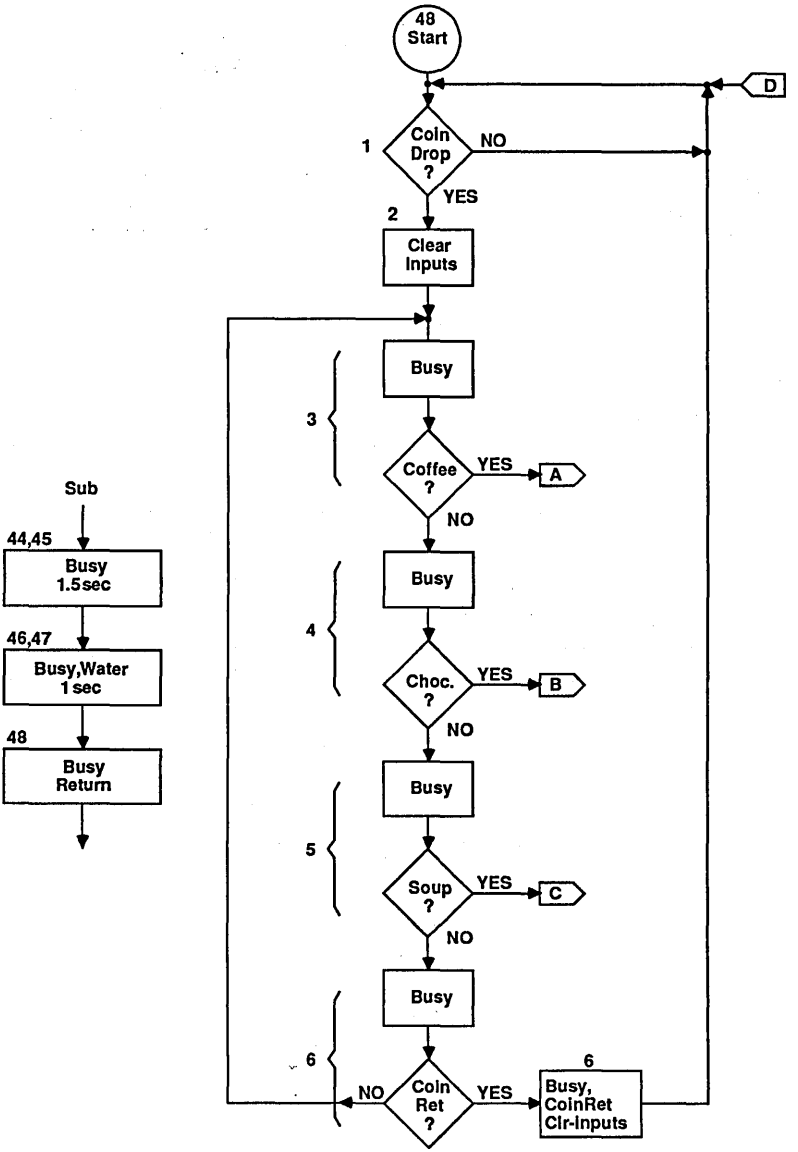
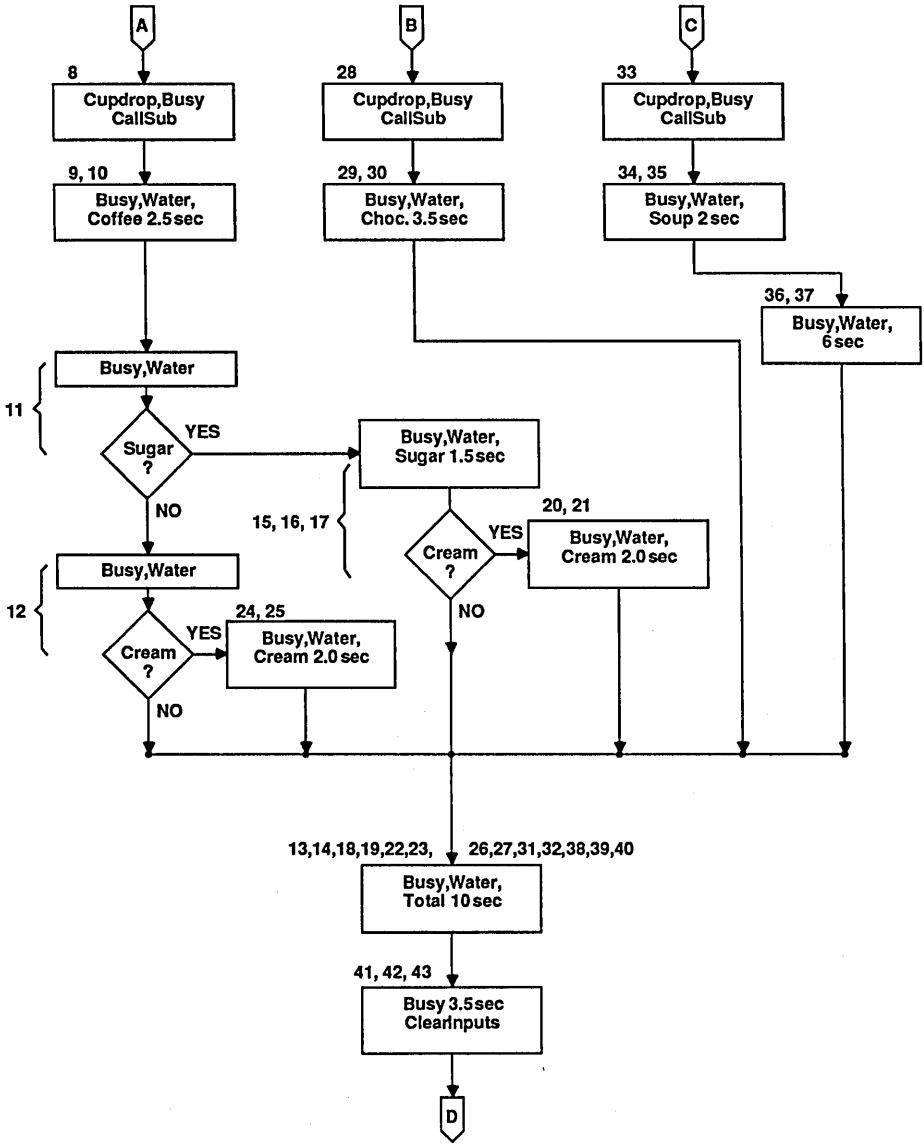


Figure 4-1. Coffee Machine Hardware



06591A 4-2

Figure 4-2. Coffee Machine Program Flow Diagram (Sheet 1 of 2)



06591A 4-2b

Figure 4-2. Coffee Machine Program Flow Diagram (Sheet 2 of 2)

Special care needs to be taken to ensure that water is on continuously for 10 seconds. Six possible paths lead to the microcode line labeled "last". At the end of each of these paths, the CREG is loaded with a value equal to 10 seconds minus the time in seconds for which water has already been on. Note that the value loaded into the CREG is one less than the expected value. This is because the value 0 in the CREG needs to be accounted for as the Am29PL141 checks the CREG and then decrements.

For example, coffee needs to be turned on for 2.5 seconds if selected. At 10 Hz., this translates into 25 clock periods. Coffee is on for one clock period during the instruction when the counter (CREG) is loaded with a countdown value (line 9 of the microcode). The counter therefore needs to be loaded with a countdown value of 23 which corresponds to coffee being on for 24 clock periods before the counter counts down to zero. The total time for which coffee is on is therefore 24 + 1 = 25 clock periods or 2.5 seconds.

On reset, the 'EQ' flag is cleared. For a 'pass' to occur, the flag must, therefore, be tested for a '0'. Hence the 'not fail' in each of the unconditional microcode lines instead of the more obvious 'pass'. Also on reset, the Am29PL141 executes the instruction on line 63. In this example, this line is an unconditional branch to line 1 of microcode. This is a wasted microcode line. If efficient coding is required to preserve microcode lines, the coin test on line 1 of the microcode could be placed on line 63 thus saving one line of microcode.

To assemble this file, type:

A> ASM141 -i COFFEE.EXP -o COFFEE.JED -b COFFEE.BIT

When the file COFFEE.EXP is assembled, two output files are created, COFFEE.JED and COFFEE.BIT. The JEDEC fuse map output is sent to the file COFFEE.JED (Figure 4-4). The PROM bit pattern is sent to the file COFFEE.BIT. See Figure 4-5 for a listiing of this file.

```
DEVICE (PL141)

DEFAULT = 1 ;

DEFINE "test inputs are given name assignments"

       coin = t0
       soup_test = t1
       choc_test = t2
       cream_test = t3
       sugar_test = t4
       coffee_test = t5
       coin_ret = cc
       fail = eq

       "output/control bits are given name assignments"

       off = 0#h
       busy = 01#h
       cup = 02#h
       water = 04#h
       coffee = 08#h
       cream = 10#h
       sugar = 20#h
       choclat = 40#h
       soup = 80#h
       cn_ret = 100#h
       clr_inp = 200#h;

BEGIN
"wait for a coin to drop and check selection after coin detect"
"1" zero:off,                    if (not coin) then goto pl(zero);
"2"      clr_inp,                continue;
"3" test:busy,                   if (coffee_test) then goto pl(cofe);
"4"      busy,                   if (choc_test) then goto pl(choc);
"5"      busy,                   if (soup_test) then goto pl(sup);
"6"      busy,                   if (not coin_ret) then goto pl(test);
"7"      busy + cn_ret + clr_inp,if (not fail) then goto pl(zero);
```

Figure 4-3. Coffee Machine Source Program Listing (Sheet 1 of 2)

```
hex <dec>     OE  OPCODE POL  TEST   DATA         OUTPUT
000 <  0>   [ 1 | 11001 | 1 | 000 | 000000 | 0000000000000000 ]
001 <  1>   [ 1 | 01101 | 1 | 111 | 111111 | 0000001000000000 ]
002 <  2>   [ 1 | 11001 | 0 | 101 | 000111 | 0000000000000001 ]
003 <  3>   [ 1 | 11001 | 0 | 010 | 011011 | 0000000000000001 ]
004 <  4>   [ 1 | 11001 | 0 | 001 | 100000 | 0000000000000001 ]
005 <  5>   [ 1 | 11001 | 1 | 110 | 000010 | 0000000000000001 ]
006 <  6>   [ 1 | 11001 | 1 | 111 | 000000 | 0000001100000001 ]
007 <  7>   [ 1 | 11100 | 1 | 111 | 101011 | 0000000000000011 ]
008 <  8>   [ 1 | 00100 | 1 | 111 | 010111 | 0000000000001101 ]
009 <  9>   [ 1 | 01000 | 1 | 111 | 001001 | 0000000000001101 ]
00A < 10>   [ 1 | 11001 | 0 | 100 | 001110 | 0000000000000101 ]
00B < 11>   [ 1 | 11001 | 0 | 011 | 010111 | 0000000000000101 ]
00C < 12>   [ 1 | 00100 | 1 | 111 | 111100 | 0000000000000101 ]
00D < 13>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
00E < 14>   [ 1 | 00100 | 1 | 111 | 001100 | 0000000000100101 ]
00F < 15>   [ 1 | 01000 | 1 | 111 | 001111 | 0000000000100101 ]
010 < 16>   [ 1 | 11001 | 0 | 011 | 010011 | 0000000000100101 ]
011 < 17>   [ 1 | 00100 | 1 | 111 | 101110 | 0000000000000101 ]
012 < 18>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
013 < 19>   [ 1 | 00100 | 1 | 111 | 010010 | 0000000000010101 ]
014 < 20>   [ 1 | 01000 | 1 | 111 | 010100 | 0000000000010101 ]
015 < 21>   [ 1 | 00100 | 1 | 111 | 011010 | 0000000000000101 ]
016 < 22>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
017 < 23>   [ 1 | 00100 | 1 | 111 | 010010 | 0000000000010101 ]
018 < 24>   [ 1 | 01000 | 1 | 111 | 011000 | 0000000000010101 ]
019 < 25>   [ 1 | 00100 | 1 | 111 | 101000 | 0000000000000101 ]
01A < 26>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
01B < 27>   [ 1 | 11100 | 1 | 111 | 101011 | 0000000000000011 ]
01C < 28>   [ 1 | 00100 | 1 | 111 | 100001 | 0000000001000101 ]
01D < 29>   [ 1 | 01000 | 1 | 111 | 011101 | 0000000001000101 ]
01E < 30>   [ 1 | 00100 | 1 | 111 | 110100 | 0000000000000101 ]
01F < 31>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
020 < 32>   [ 1 | 11100 | 1 | 111 | 101011 | 0000000000000011 ]
021 < 33>   [ 1 | 00100 | 1 | 111 | 010010 | 0000000010000101 ]
022 < 34>   [ 1 | 01000 | 1 | 111 | 100010 | 0000000010000101 ]
023 < 35>   [ 1 | 00100 | 1 | 111 | 111010 | 0000000000000101 ]
024 < 36>   [ 1 | 01000 | 1 | 111 | 100100 | 0000000000000101 ]
025 < 37>   [ 1 | 00100 | 1 | 111 | 000111 | 0000000000000101 ]
026 < 38>   [ 1 | 11001 | 1 | 111 | 100111 | 0000000000000101 ]
027 < 39>   [ 1 | 01000 | 1 | 111 | 100111 | 0000000000000101 ]
028 < 40>   [ 1 | 00100 | 1 | 111 | 100000 | 0000000000000001 ]
029 < 41>   [ 1 | 01000 | 1 | 111 | 101001 | 0000000000000001 ]
02A < 42>   [ 1 | 11001 | 1 | 111 | 000000 | 0000001000000001 ]
02B < 43>   [ 1 | 00100 | 1 | 111 | 001101 | 0000000000000001 ]
02C < 44>   [ 1 | 01000 | 1 | 111 | 101100 | 0000000000000001 ]
02D < 45>   [ 1 | 00100 | 1 | 111 | 000111 | 0000000000000101 ]
02E < 46>   [ 1 | 01000 | 1 | 111 | 101110 | 0000000000000101 ]
02F < 47>   [ 1 | 00010 | 1 | 111 | 111111 | 0000000000000101 ]
03F < 63>   [ 1 | 11001 | 1 | 111 | 000000 | 0000001000000000 ]

Where:
     Oe         = Synchronous output enable for P[15:8]
     OPCODE     = Five-bit field for selecting one of the 29
                  microinstructions
     POL        = Test condition polarity select field
                     0 = Test for true (HIGH) condition
                     1 = Test for false (LOW) condition
     TEST       = Binary value of input line to be tested
          Value     Input condition  Value     Input Condition
          000       T0              100       T4
          001       T1              101       T5
          010       T2              110       CC
          011       T3              111       EQ
     DATA       = 6-bit conditional branch microaddress, test input mask,
                  or counter value field designated as PL in
                  microinstruction mnemonics (P[21:16])
     Output     = 16-bit user output control signals (P[15:0])
```

Figure 4-5. PROM File for Coffee Machine Application

CHAPTER 5

DEC PDP-11 UNIBUS CONTROLLER

5.1 THE DESIGN PROBLEM

This paper discusses the use of the Am29PL141 Fuse Programmable Controller (FPC) as a DEC PDP-11 Unibus* interface controller.

Designing an interface for the Unibus is typical of the problems which can be readily solved using the Am29PL141 FPC. The complexity of Unibus handshaking is such that microprogramming is a reasonable design technique, but use of a separate sequencer, control memory, and pipeline register is not economical. Since the FPC contains a sequencer, memory, and pipeline; it fits this class of problem rather well. The PDP-11 was chosen for this example because it has a well documented protocol which is familiar to many engineers. An Overview of the Unibus is included.

The problem this application note solves is to:

Design an interface between the Unibus and a generic I/O device to allow the following operations:

- Interface to handle all Unibus protocol for
- DATI/DATO with device as slave
- Device BR (interrupt)
- Device NPR (direct memory access)
- DATI/DATO with device as master
- Interface to handle synchronous parallel transfers with device

5.2 DEC UNIBUS OVERVIEW

The DEC PDP-11 Unibus is an asynchronous bus which supports programmed I/O, prioritized interrupts, and Direct Memory Access (DMA) in a memory mapped I/O environment. All bus transfers are between a bus master and bus slave, and are controlled by the master. A bus arbitrator grants bus mastership to requesting devices.

The six basic types of transfers allowed are:

DATO – word data transfer from master to slave
DATOB – byte data transfer from master to slave
DATI – word data transfer from slave to master
DATIP – word data transfer slave to master, inhibit restore cycle
NPR – Non Processor Request. DMA device wants to become bus master.
BRi – Bus Request. Interrupt request at level i (4,5,6,or 7).

The following control signals are used during transfers:

MSYN	master sync—timing control
SSYN	slave sync—timing control
C0, C1	data transfer type
BRi	interrupt bus request level i
BGi	interrupt bus grant level i (note 1)
INTR	interrupt vector strobe
NPR	DMA bus request
NPG	DMA bus grant (note 1)
SACK	select acknowledge
BBSY	bus busy

Note 1: These signals are daisy chained to form a physical priority level at each separate logical priority level (npg, bg4, bg5, bg6, bg7).

5.3 INTERFACE HARDWARE DESIGN

As shown in Figure 5-1, the architecture chosen for this interface consists of three main sections—Unibus signal buffering, address decoding, and control logic. Data, address, and control signal buffers provide proper Unibus levels and are implemented using DS8641 Quad Unified Bus Transceivers. The address decoder detects whether the device is addressed as a slave or master during Unibus DATI and DATO transfers, and is best implemented using Am29806 decoders. The control logic is a microprogrammed state machine which handles both Unibus and device handshaking.

The heart of the control logic is the Am29PL141 Fuse Programmable Controller. Its user-defined microprogram implements a state machine which handles both device and Unibus handshaking. Test inputs are synchronized with the FPC clock using an AM29821A 10-bit register. Five of these inputs go directly to the FPC, while the other five go through a multiplexer which expands the FPC conditional test capability from seven to fourteen signals. Two D flip-flops and OR gates are used to

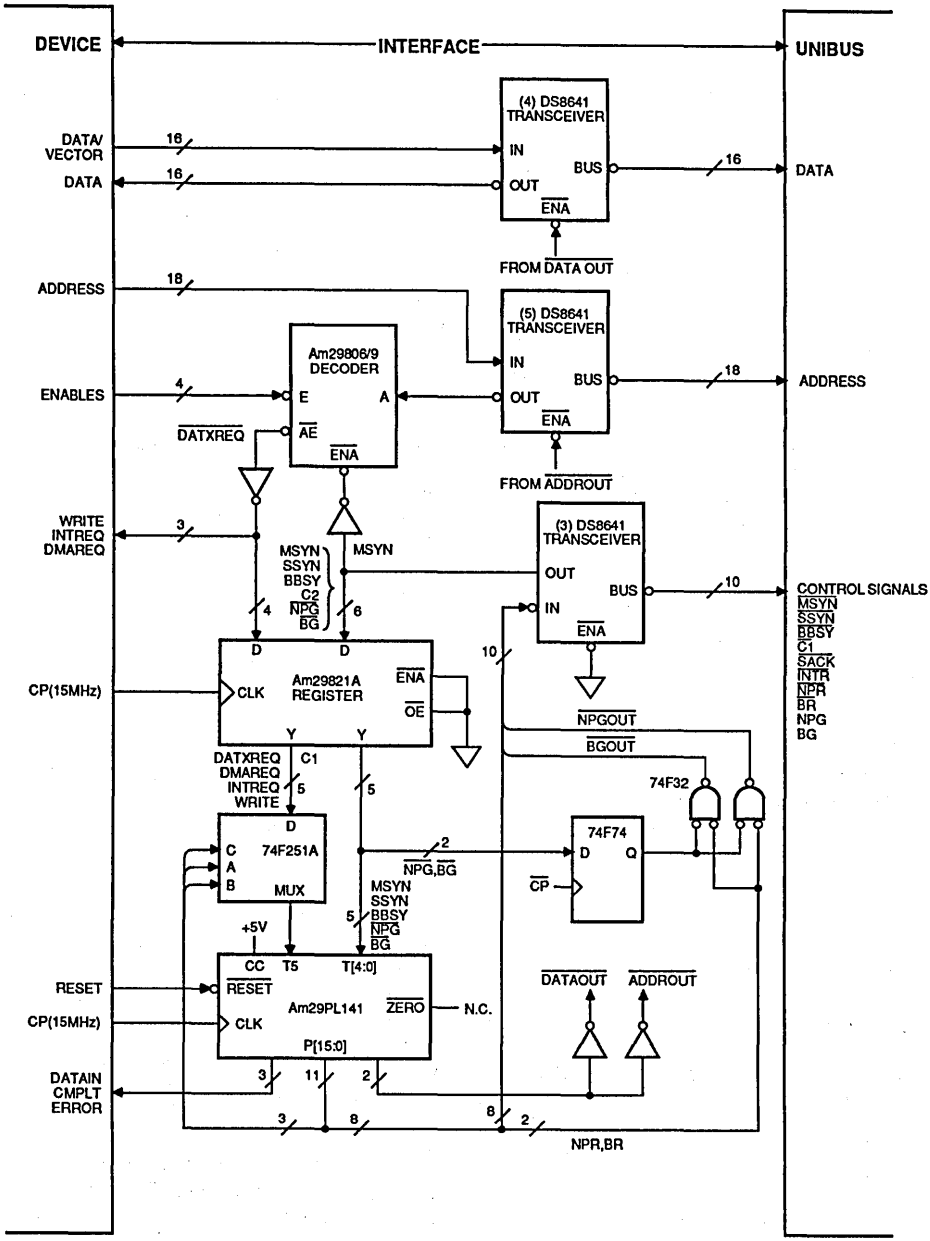


Figure 5-1. Unibus Interface Block Diagram

implement the Unibus request/grant handshaking. Because the clock period must be at least 64.5 ns, a clock frequency of 15 MHz is appropriate (66.6 ns). A detailed control logic timing analysis is shown in Figure 5-2.

5.4 MICROWORD FORMAT

The microword organization for this application of the FPC is shown in Figure 5-3. The 32-bit microword is subdivided into fields of various sizes and functions. The 16 most significant bits are used during next address generation within the FPC, while the lower 16 bits are tailored to the application.

OE is a synchronous output enable for output bits 15 through 8. The 5-bit OPCODE field contains the FPC next address instruction.

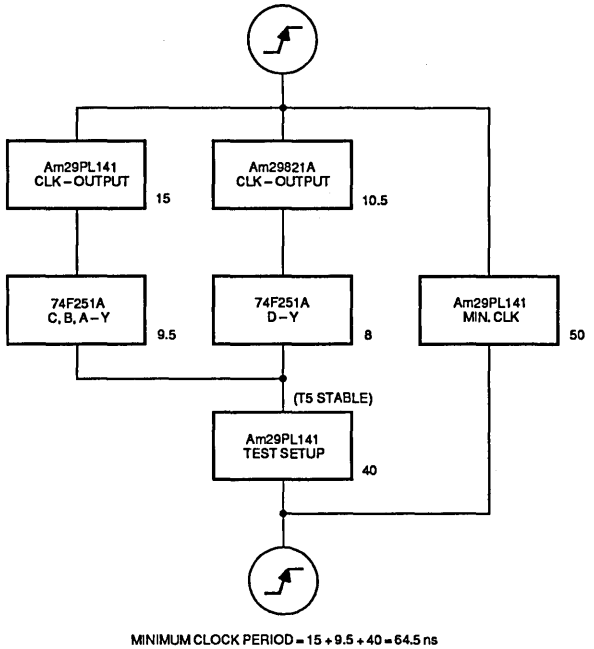
POL controls polarity of the test condition selected by the 3-bit TEST field.

DATA is a 6-bit address, test mask, or counter value; depending on the OPCODE used.

ERROR is an interface timeout indication to the peripheral device.

AUX TEST is a 3-bit field which controls the external multiplexer for additional test inputs. The TEST field must have a value of 5 to use the test selected by AUX TEST.

The 12 COMMAND outputs are single bit control signals. ADDROUT and DATAOUT enable Unibus address and data buffers. DATAIN clocks Unibus data into peripheral device registers. COMPLT indicates to the device that an interrupt or DMA operation has been completed. The remaining outputs are Unibus control signals described in Section 5.2.



06591A 5-2

Figure 5-2. Control Logic Timing

Microword Format:

```
: 31 :  30 - 26  :  25 : 24,23,22 : 21 - 16 :   15  : 14,13,12 :  11 - 0   :
:....:..........:.....:..........:........:.......:..........:...........:
: oe :   opcode  : pol :   test   :   data  : error :  aux tst :  command  :
:....:..........:.....:..........:........:.......:..........:...........:
```

oe: output enable
(31)

opcode:
(30-26)

29PL141 command

00 - RETPL	08 - LPPL	10 - CMP	18 - FORK
01 - RETPLN	09 - DEC	11 - CMP	19 - GOTOPL
02 - RET	0A - LPPLN	12 - CMP	1A - WAIT
03 - RETN	0B - GOTOPLZ	13 - CMP	1B - DECGO/C
04 - LDPL	0C - DECAL	14 - PSHPL	1C - CALPL
05 - LDPLN	0D - CONT	15 - PSH	1D - CALPLN
06 - LDTM	0E - DECTM	16 - PSHTM	1E - CALTM
07 - LDTMN	0F - GOTOTM	17 - PSHN	1F - CALTMN

pol:
(25)

test is test polarity (1 = negate)

test:
(24,23,22)

conditional test input select

0 - msyn	4 - npg
1 - ssyn	5 - aux tests
2 - bbsy	6 - pass
3 - bg	7 - equal flag

data:
(21-16)

branch address, test input mask, or counter load value

error:
(15)

timeout error indication to device

aux test:
(14,13,12)

additional test inputs when test = 5

0 - datxreq	4 - c1
1 - dmareq	5 - spare
2 - intreq	6 - spare
3 - write	7 - spare

command:
(11-0)

```
:  11  :  10  :   9  :  8  :  7  :   6  :  5 :  4  :   3  :   2  :   1  :  0   :
:......:......:......:.....:.....:......:...:.....:......:......:......:......:
: addr : data : data : com : c1  : intr : br : npr : sack : bbsy : ssyn : msyn :
: out  : out  : in   : plt :     :      :    :     :      :      :      :      :
:......:......:......:.....:.....:......:...:.....:......:......:......:......:
```

06591A 5-3

Figure 5-3. Microword Organization

5.5 UNIBUS CONTROLLER MICROCODE

Two things always happen during execution of a microinstruction—the address of the next microinstruction is determined using the OPCODE, POLARITY, TEST, and DATA fields; while concurrently, the Unibus and device interfaces are controlled by signals from the COMMAND field.

The microcode which controls the FPC was written using the Am29PI141 assembler available from AMD. The mnemonics used in the source code are shown in Figure 5-4. Note that these definitions are consistent with the microword definition of Figure 5-3. Figure 5-4 also contains the source code for the FPC. Figure 5-5 shows the FPC PROM contents. Note that one line of source generates one PROM word. The general source format is:

```
<label>: <outputs>, <FPC
         instruction>; "comment"
```

Outputs may be either mnemonic or constants, and may be logically "ANDed" or "ORed" together. The FPC assembler instructions are included in Chapter 2. The following paragraphs describe the code written for this FPC application. It is helpful to refer to the microcode source program listing (Figure 5-4) and the timing diagrams (Figures 5-6, 5-7, 5-8, and 5-9).

After reset to address 63, the program branches to address 0 (label TOP) and loops until one of the external conditions DATXREQ, DMAREQ, or INTREQ is asserted. For example, at TOP, if auxiliary test condition DATXREQ is asserted, the subroutine DATX is called. Otherwise, the next sequential instruction is executed.

DATXREQ true indicates that a Unibus master has initiated a DATO or DATI transfer with the interface and causes a branch to the subroutine at label DATX, with the return address being saved in the FPC SREG. Unibus signal C1 is tested to determine direction, and then a DATO or DATI slave sequence is completed beginning at label DATO or DATI. At DATI, Unibus signal SSYN is asserted and data gated onto the Unibus using DATAOUT, until test MSYN is negated. The next instruction has no control signals asserted (OFF), and returns from the subroutine by branching to the address saved in SREG. DATO processing is similar.

DMAREQ indicates that the device is requesting a Direct Memory Access cycle, which causes a branch to label NPRX. The program waits at NPRX until NPG is de-asserted. NPR is then asserted and the program loops at NPR1 until NPG is reasserted. SACK is asserted, and the program loops at NPR2 until the three signals NPG, BBSY, and SSYN are unasserted. Note how the compare instruction masks the test inputs with the constant NPG_BBSY_SSYN and compares the result to 0. This allows concurrent testing of three inputs in only two microcycles. BBSY is asserted, making the interface bus master, and WRITE is tested to determine DMA direction. If a DATI cycle is to occur, we fall through to NPRDATI.

Front-end 150 ns de-skewing is done at NPRDATI and WAIT1, concurrent with loading the FPC CREG with 31 hex for a 15 microsecond slave timeout. WAIT2 is the top of the timeout loop. If the slave Unibus device asserts SSYN within 15 microseconds, the program branches to pass1 for tail-end 75 ns de-skew. Otherwise it falls through to the error exit at ERROR1. DATO processing is similar to DATI, and begins at NPRDATO.

INTREQ is asserted when the device wants to interrupt the Unibus CPU, causing execution to continue at INTR0. Interrupt request/grant processing occurs at INTR0 and INTR1. SACK is then asserted and the program loops at INTR2 until BG, BBSY, and SSYN are unasserted. The device supplied interrupt vector is gated onto the Unibus data lines at INTR3, and the interrupt handshake is finished at WAIT0.

5.6 CONCLUSION

One of the advantages of microprogrammed design is that it is relatively easy to change. In this application, Unibus DATOB and DATIP transfers were not differentiated from DATO and DATI transfers. This could be easily accommodated by modifying the DATX microcode to test Unibus signal C1 by adding a few words of additional code. Another change to be considered is to change the device interface to a less rudimentary protocol. Additional control signals could be provided by adding a decoder at the FPC output, and encoding eight signals using only 3 microword bits. Spare multiplexer inputs could be used for additional device status lines. Additional control signals can also be provided by adding another FPC.

```
"     Unibus Controller microcode using Am29PL141 assembler          "

"     Version 1.2                  R. Purvis, 19 December 85         "

device (PL141)
default = 1 ;

define
"       **********   DEFINITION OF TEST INPUTS  ***********  "

        tmsyn = t0              " test Unibus signal MSYN     "
        tssyn = t1              "                    SSYN     "
        tbbsy = t2              "                    BBSY     "
        tbg   = t3              "                    BG       "
        tnpg  = t4              "                    NPG      "
        aux   = t5              " auxiliary test conditions   "
        pass  = cc              " unconditional pass          "
        bg_bbsy_ssyn  = 0e#h    " test mask                   "
        npg_bbsy_ssyn = 16#h    " test mask                   "

"       **********  DEFINITION OF OUTPUTS  ****************  "

"       AUXILIARY TEST CONDITIONS                             "
        datxreq = 0000#h      " Unibus DATI or DATO request   "
        dmareq  = 1000#h      " device DMA request            "
        intreq  = 2000#h      " device Interrupt request      "
        write   = 3000#h      " device write request          "
        tc1     = 4000#h      " Unibus signal c1              "
                              " aux tests 5 - 7 are unused    "

"       CONTROL SIGNALS                                       "
        off     = 0000#h       " no signals active          "
        error   = 8000#h       " error flag to device       "
        addr    = 0800#h       " gate address onto Unibus   "
        dataout = 0400#h       " gate data onto Unibus      "
        datain  = 0200#h       " strobe data in from Unibus "
        complt  = 0100#h       " complete flag to device    "

        c1      = 0080#h       " assert Unibus signal C1    "
        intr    = 0040#h       "                      INTR  "
        br      = 0020#h       "                      BR    "
        npr     = 0010#h       "                      NPR   "
        sack    = 0008#h       "                      SACK  "
        bbsy    = 0004#h       "                      BBSY  "
        ssyn    = 0002#h       "                      SSYN  "
        msyn    = 0001#h ;     "                      MSYN  "

        test_condition = cc;  " default test condition        "

begin       "   ***************   Source Code   ***************      "
            "   Unibus Controller V1.2                               "

"           ************************************************************   "
"           *    MAIN LOOP  -  Loop at TOP until external condition    *   "
"           *    DATXREQ, DMAREQ, or INTREQ is true.                   *   "
"           ************************************************************   "

top:        datxreq, if (aux) call pl(datx);
            dmareq, if (aux) call pl(nprx);
            intreq, if (not aux) goto pl(top);


"           ************************************************************   "
"           *     INTERRUPT SERVICE ROUTINE  -  Device interrupt service *   "
"           *     request.  Perform Unibus interrupt handshake.          *   "
"           ************************************************************   "
```

Figure 5-4. Unibus Controller Source Program Listing (Sheet 1 of 2)

```
intr0:      off, if (tbg) goto pl (intr0);       " request/grant handshake "
intr1:      br, if (not tbg) goto pl(intr1);
            br + sack, continue;
intr2:      br + sack, cmp tm(bg_bbsy_ssyn) to pl(0);
            br + sack, if (not eq) goto pl(intr2);
intr3:      sack + bbsy + intr + dataout, continue;     " interrupt vector "
wait0:      bbsy + intr + dataout, if (not tssyn) goto pl(wait0);
            complt, goto pl(top);


"           ************************************************************   "
"           *     PROGRAMMED I/O ROUTINE  -  Unibus master accessing       "
"           *     device.  Perform Unibus DATO/DATI handshake.             "
"           ************************************************************   "

datx:       tc1, if (aux) goto pl(dato);

dati:       ssyn + dataout, if (tmsyn) goto pl(dati);    " unibus slave DATI "
            off, ret;

dato:       ssyn + datain, if (tmsyn) goto pl(dato);   " unibus slave DATO "
            off, ret;

"           ************************************************************   "
"           *     DMA SERVICE ROUTINE  -  Device DMA service request.      "
"           *     Perform Unibus DMA handshake.                            "
"           ************************************************************   "

nprx:       off, if (tnpg) goto pl(nprx);      "  request/grant handshake  "
npr1:       npr, if (not tnpg) goto pl(npr1);
            npr + sack, continue;
npr2:       npr + sack, cmp tm(npg_bbsy_ssyn) to pl(0);
            npr + sack, if (not eq) goto pl(npr2);
            bbsy + write, if (aux) goto pl(nprdato);      " bus master now "


"           DMA READ ROUTINE  (unibus master DATI)                          "

nprdati:    bbsy + addr, load pl(31#h);
wait1:      bbsy + addr, if (tssyn) goto pl(wait1);
wait2:      bbsy + addr + msyn, if (tssyn) goto pl(pass1);   " 15 us       "
            bbsy + addr + msyn, if (tssyn) goto pl(pass1);   " timeout     "
            bbsy + addr + msyn, if (tssyn) goto pl(pass1);
            bbsy + addr + msyn, while (creg<>0) loop to pl(wait2);
error1:     bbsy + addr + error, ret;                      "  timeout error  "
pass1:      bbsy + addr + complt + datain, ret;            "  normal exit    "


"           DMA WRITE ROUTINE  (unibus master DATO)                         "

nprdato:    bbsy + addr + c1 + dataout, load pl(31#h);
wait3:      bbsy + addr + c1 + dataout, if (tssyn) goto pl (wait3);
wait4:      bbsy + addr + c1 + dataout + msyn, if (tssyn) goto pl(pass2);
            bbsy + addr + c1 + dataout + msyn, if (tssyn) goto pl(pass2);
            bbsy + addr + c1 + dataout + msyn, if (tssyn) goto pl(pass2);
            bbsy + addr + c1 + dataout + msyn, while(creg<>0)loop to pl(wait4);
error2:     bbsy + addr + c1 + error, ret;                 "  timeout error  "
pass2:      bbsy + addr + c1 + complt, ret;                "  normal exit    "
            .org 63#d
            off, goto pl(0);                          "  hardware reset here. "

end.
```

Figure 5-4. Unibus Controller Source Program Listing (Sheet 2 of 2)

PROM Contents are :

hex	<dec>	OE	OPCODE	POL	TEST	DATA	OUTPUT
000	< 0>	1	11100	0	101	001011	0000000000000000
001	< 1>	1	11100	0	101	010000	0001000000000000
002	< 2>	1	11001	1	101	000000	0010000000000000
003	< 3>	1	11001	0	011	000011	0000000000000000
004	< 4>	1	11001	1	011	000100	0000000000100000
005	< 5>	1	01101	1	111	111111	0000000000101000
		OPCODE		CONSTANT		DATA	
006	< 6>	1	100	000000		001110	0000000000101000
007	< 7>	1	11001	1	111	000110	0000000000101000
008	< 8>	1	01101	1	111	111111	0000010001001100
009	< 9>	1	11001	1	001	001001	0000010001000100
00A	< 10>	1	11001	0	110	000000	0000000100000000
00B	< 11>	1	11001	0	101	001110	0100000000000000
00C	< 12>	1	11001	0	000	001100	0000010000000010
00D	< 13>	1	00010	0	110	111111	0000000000000000
00E	< 14>	1	11001	0	000	001110	0000001000000010
00F	< 15>	1	00010	0	110	111111	0000000000000000
010	< 16>	1	11001	0	100	010000	0000000000000000
011	< 17>	1	11001	1	100	010001	0000000000010000
012	< 18>	1	01101	1	111	111111	0000000000011000
		OPCODE		CONSTANT		DATA	
013	< 19>	1	100	000000		010110	0000000000011000
014	< 20>	1	11001	1	111	010011	0000000000011000
015	< 21>	1	11001	0	101	011110	0011000000000100
016	< 22>	1	00100	0	110	110001	0000100000000100
017	< 23>	1	11001	0	001	010111	0000100000000100
018	< 24>	1	11001	0	001	011101	0000100000000101
019	< 25>	1	11001	0	001	011101	0000100000000101
01A	< 26>	1	11001	0	001	011101	0000100000000101
01B	< 27>	1	01000	0	110	011000	0000100000000101
01C	< 28>	1	00010	0	110	111111	1000100000000100
01D	< 29>	1	00010	0	110	111111	0000101100000100
01E	< 30>	1	00100	0	110	110001	0000110010000100
01F	< 31>	1	11001	0	001	011111	0000110010000100
020	< 32>	1	11001	0	001	100101	0000110010000101
021	< 33>	1	11001	0	001	100101	0000110010000101
022	< 34>	1	11001	0	001	100101	0000110010000101
023	< 35>	1	01000	0	110	100000	0000110010000101
024	< 36>	1	00010	0	110	111111	1000100010000100
025	< 37>	1	00010	0	110	111111	0000100110000100
03F	< 63>	1	11001	0	110	000000	0000000000000000

Figure 5-5. FPC PROM Contents

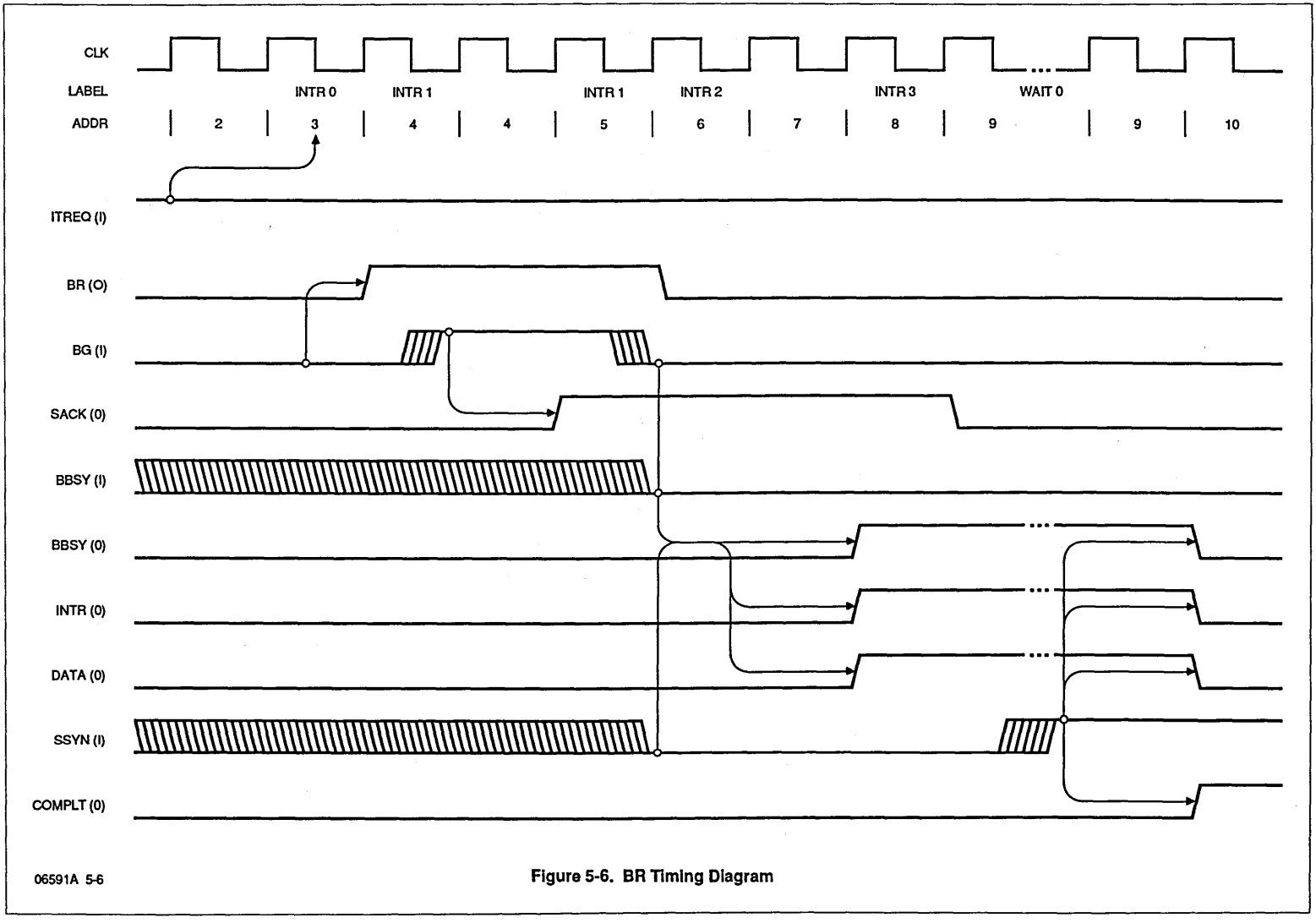
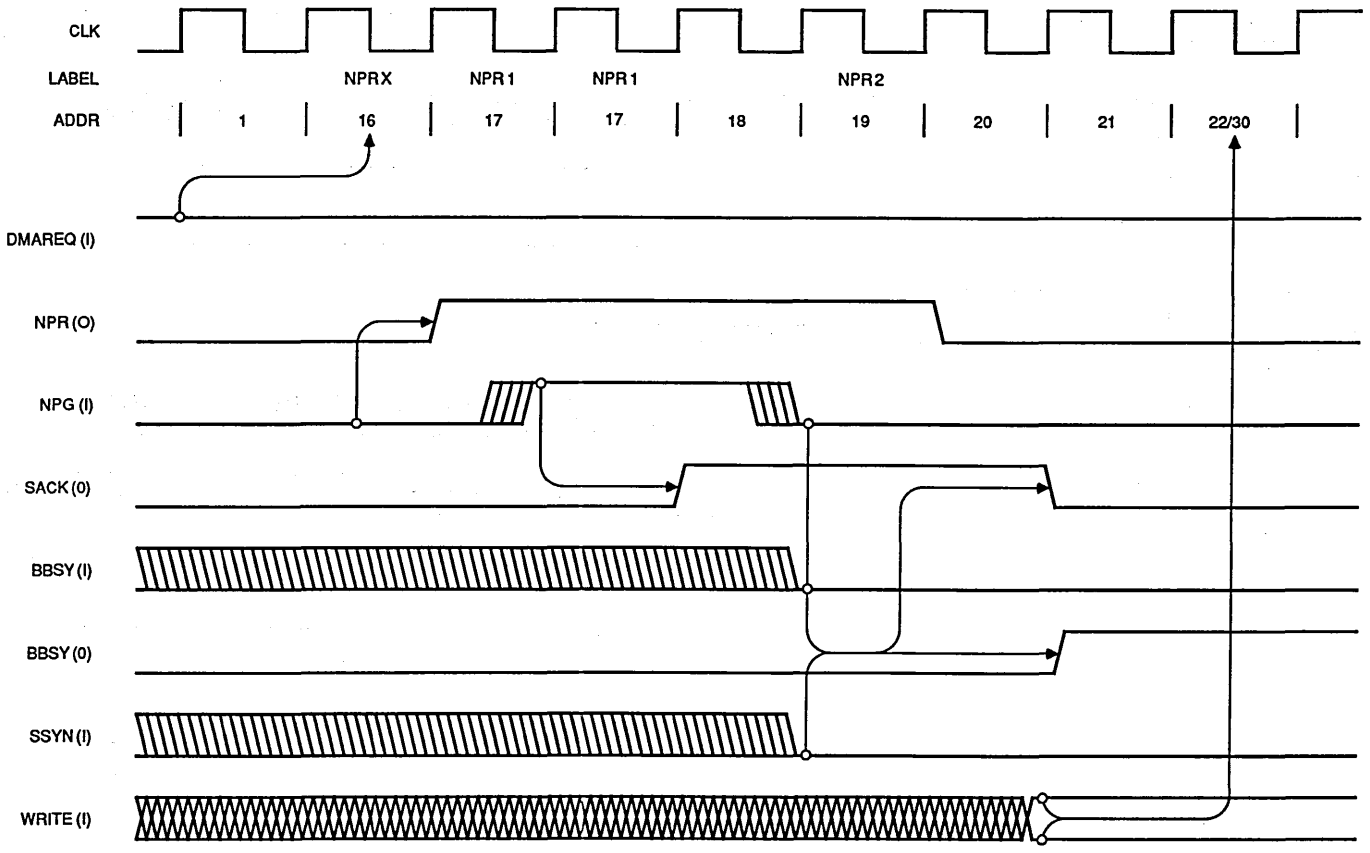


Figure 5-6. BR Timing Diagram



06591A 5-7

Figure 5-7. NPR Timing Diagram

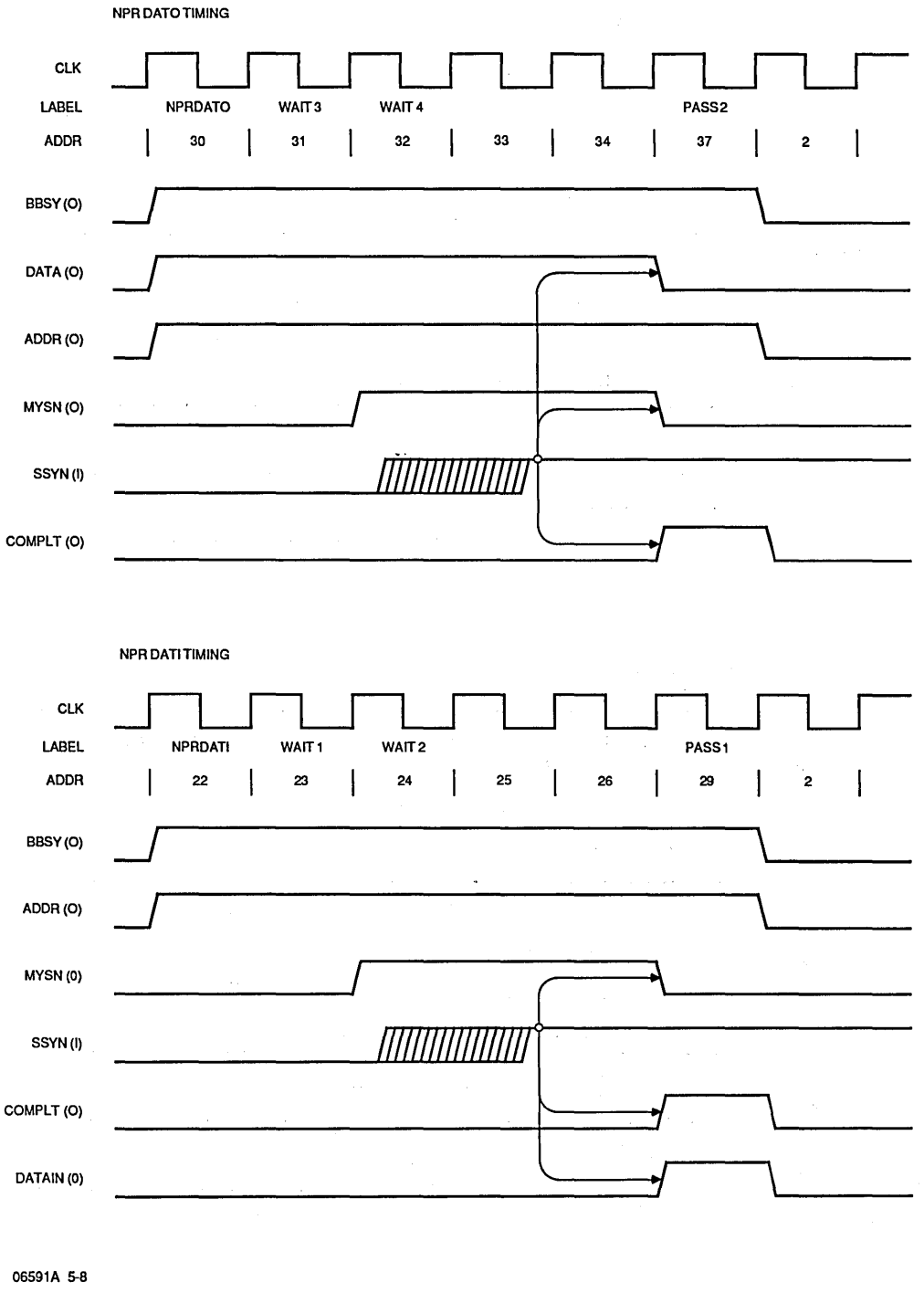


Figure 5-8. NPR DATI and DATO Timing Diagram

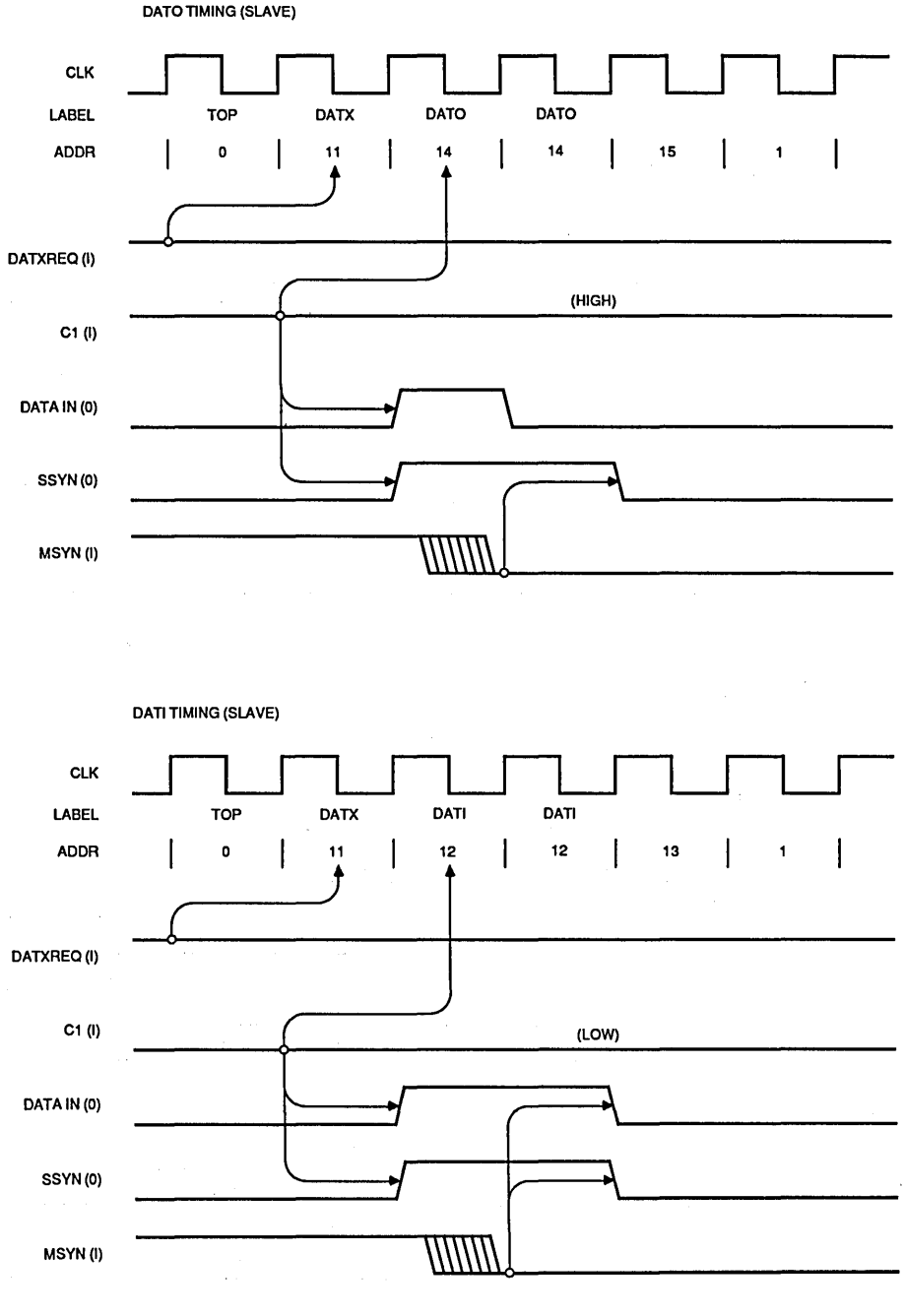


Figure 5-9. DATI and DATO (Slave) Timing Diagram

CHAPTER 6

Am29PL141 BASED DEC Q-BUS CONTROLLER

6.1 THE DESIGN PROBLEM

Designing an interface for the DEC Q-Bus has been approached using many techniques. One technique, microprogramming, has in the past been economically unattractive because it required use of a separate sequencer, control store, and pipeline registers. Now that Advanced Micro Devices has introduced the single chip Am29PI141 Fuse Programmable Controller (see Section 1), engineers can economically apply powerful microprogramming techniques to the design of medium complexity state machines like that required to control the Q-Bus.

The problem is to design an interface between the Q-Bus and a generic device to allow the following operations:

- DATI/DATO with device as slave
- Device interrupt request
- Device Direct Memory Access request
- DATI/DATO with device as master

The DEC Q-Bus is an asynchronous bus which supports Programmed I/O, prioritized Interrupts, and Direct Memory Access (DMA) operations. All bus transfers are between a bus master and bus slave, and are controlled by the master. An arbitrator grants bus mastership to requesting devices.

The nine basic types of transfers allowed are:

DATI	–	Word data transfer from slave to master
DATO	–	Word data transfer from master to slave
DATOB	–	Byte data transfer from master to slave
DATIO	–	Read-modify-write word transfer
DATIOB	–	Read-modify-write byte transfer
DATBI	–	Block data transfer from slave to master
DATBO	–	Block data transfer from master to slave
DMR	–	Direct Memory Access request to become bus master.
IRQi	–	Interrupt request at level i (4,5,6,or 7).

The following control signals are used during transfers:

SNYC	sync	– master timing control
DOUT	data out	– indicates master write
DIN	data in	– indicates master read
RPLY	reply	– slave acknowledge
WTBT	write/byte	– byte write cycle
BS7	I/O page select	
IRQi	interrupt request level i	
IAK	interrupt grant	
DMR	DMA request	
DMG	DMA grant	
SACK	select acknowledge	

6.2 Q-BUS CONTROLLER HARDWARE DESIGN

A block diagram of this interface is shown in Figure 6-1. It consists of three sections—Q-Bus buffering, address decoding, and control logic. The address decoder detects addressing of the device as a slave during DATI and DATO transfers.

The control logic is based on the Am29PL141 Fuse Programmable Controller (FPC). Its microprogram implements a state machine to control both device and Q-Bus handshaking. Test inputs are synchronized with the FPC clock using an AM29821A 10-bit register and a D flip-flop. Note the use of a multiplexer to expand the FPC test capability. The additional D flip-flop and AND gates are used to implement the interrupt and DMA request/grant handshaking.

6.3 MICROWORD FORMAT

The microword organization for this application of the FPC is shown in Figure 6-2. The 32-bit microword is subdivided into fields of various sizes and functions. The 16 most significant bits are used during next address generation within the FPC, while the lower 16 bits are application interface signals.

6.4 MICROCODE

The microcode of Figure 6-3 was written using the Am29PL141 assembler available from AMD (refer to Chapter 2). Mnemonic definitions are shown, followed by code to control the interface. Figure 6-4 shows the FPC PROM contents. A brief description of the code follows.

After reset to address 63, the program branches to label TOP and loops until one of the external

conditions DATXREQ, DMAREQ, or INTREQ is asserted.

DATXREQ true indicates a Q-Bus DATO or DATI operation addressing the device and causes a subroutine call to DATX. Q-Bus signal WTBT is tested, and DATO or DATI handshaking is completed beginning at label DATO or DATI.

INTREQ is asserted when the device wants to interrupt the CPU, causing execution to continue at INTR0. Interrupt request/grant processing occurs and then the vector is read by the CPU.

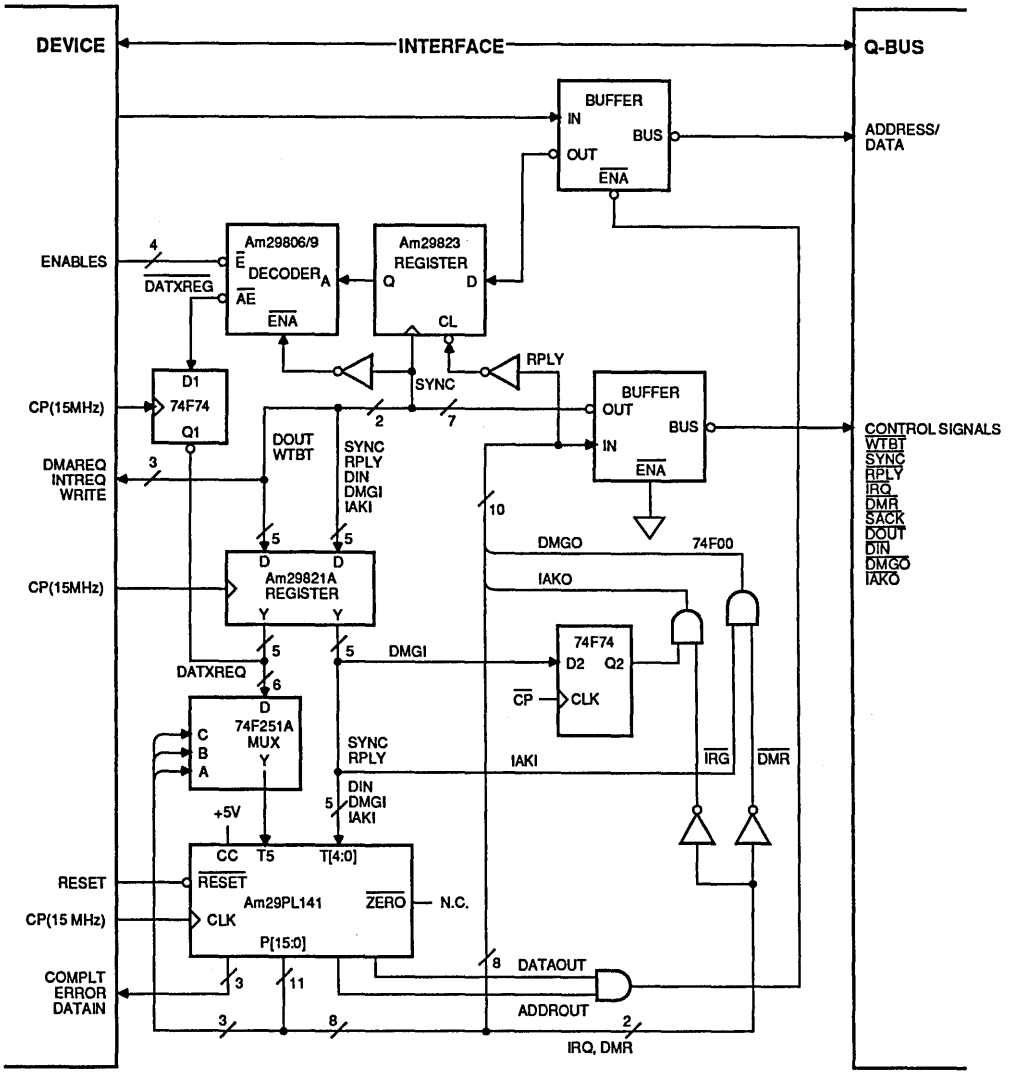
6.5 CONCLUSION

The problem statement for this interface does not require block, byte, or read-modify-write master handshaking. These features can be implemented by adding extra device request lines and microcoding the additional handshake algorithms. Another possible change is to implement the Q-Bus four-level interrupt configuration. These changes are left as an exercise for the interested reader!

References:

Microsystems Handbook, Digital Equipment Corporation, 1985.

Am29PL141 FPC Data Sheet, Advanced Micro Devices, 1985.



06591A 6-1

Figure 6-1. Q-Bus Controller Block Diagram

31	30 - 26	25	24,23,22	21 - 16	15	14,13,12	11 - 0
oe	opcode	pol	test	data	error	aux tst	command

oe: (31) — output enable

opcode: (30-26) — 29PL141 command

00 - RETPL	08 - LPPL	10 - CMP	18 - FORK
01 - RETPLN	09 - DEC	11 - CMP	19 - GOTOPL
02 - RET	0A - LPPLN	12 - CMP	1A - WAIT
03 - RETN	0B - GOTOPLZ	13 - CMP	1B - DECG0/C
04 - LDPL	0C - DECAL	14 - PSHPL	1C - CALPL
05 - LDPLN	0D - CONT	15 - PSH	1D - CALPLN
06 - LDTM	0E - CTTM	16 - PSHTM	1E - CALTM
07 - LDTMN	0F - GOTOTM	17 - PSHN	1F - CALTMN

pol: (25) — test polarity (1 = negate)

test: (24,23,22) — conditional test input select

0 - sync	4 - iak
1 - rply	5 - aux tests
2 - din	6 - pass
3 - dmg	7 - equal flag

data: (21-16) — branch address, test input mask, or counter load value

error: (15) — timeout error indication to device

aux test: (14,13,12) — additional test inputs when test = 5

0 - datxreq	4 - dout
1 - dmareq	5 - wtbt
2 - intreq	6 - spare
3 - write	7 - spare

command: (11-0)

11	10	9	8	7	6	5	4	3	2	1	0
com plt	data in	data out*	addr out*	rply	irq	dmr	sack	dout	din	sync	wtbt

* - indicates active low microcode bits

Figure 6-2. Q-Bus Controller Microword Format

```
"     Q-Bus Controller microcode using Am29PL141 assembler          "

"     Version 1.1                    R. Purvis, 3 January 86        "

device (pl141)
default = 1 ;

define
"      **********   DEFINITION OF TEST INPUTS  ********  "

       tsync = t0              " test Q-Bus signal SYNC "
       trply = t1              "                   RPLY "
       tdin  = t2              "                   DIN  "
       tdmgi = t3              "                   DMGI "
       tiaki = t4              "                   IAKI "
       aux   = t5              " auxiliary test conditions "
       pass  = cc              " unconditional pass "
       sync_rply  = 03#h       " test mask "

"      **********  DEFINITION OF OUTPUTS  **************    "

"      AUXILIARY TEST CONDITIONS                                "
       datxreq = 0300#h      " Q-Bus DATI or DATO request     "
       dmareq  = 1300#h      " device DMA request             "
       intreq  = 2300#h      " device Interrupt request       "
       write   = 3300#h      " device write request           "
       tdout   = 4300#h      " Q-Bus signal DOUT              "
       twtbt   = 5300#h      " Q-Bus signal WTBT              "
                             " aux tests 6 and 7 are spares   "

"      CONTROL SIGNALS                                          "
       off     = 0300#h       " no signals active             "
       error   = 8300#h       " error flag to device          "
       complt  = 0B00#h       " complete flag to device       "
       datain  = 0700#h       " strobe data in from Q-Bus     "
       dataout = FDFF#h       " gate data onto Q-Bus          "
       addrout = FEFF#h       " gate address onto Q-Bus       "

       rply    = 0380#h       " assert Q-Bus signal RPLY      "
       irq     = 0340#h       "                    IRQ        "
       dmr     = 0320#h       "                    DMR        "
       sack    = 0310#h       "                    SACK       "
       dout    = 0308#h       "                    DOUT       "
       din     = 0304#h       "                    DIN        "
       sync    = 0302#h       "                    SYNC       "
       wtbt    = 0301#h ;     "                    WTBT       "

test_condition = cc;  " default test condition    "
```

Figure 6-3. Q-Bus Controller Source Program Listing (Sheet 1 of 3)

```
" *****   assumptions   ******          "

"   -  no I/O page DMA                  "
"   -  single xfer DMA (not block mode) "
"   -  single level interrupts          "
"   -  no byte operations               "
"   -  no parity                        "

begin       "   ***************   Source Code   ***************          "
            "   Q-Bus Controller V1.0                                    "


"           ***********************************************************   "
"           *    MAIN LOOP  -  Loop at TOP until external condition     "
"           *    DATXREQ, DMAREQ, or INTREQ is true.                    "
"           ***********************************************************   "

top:        datxreq, if (aux) call pl(datx);
            dmareq, if (aux) call pl(dmax);
            intreq, if (not aux) goto pl(top);


"           ***********************************************************   "
"           *     INTERRUPT SERVICE ROUTINE  -  Device interrupt service  "
"           *     request.  Perform Q-Bus interrupt handshake.            "
"           ***********************************************************   "

intr0:      off, if (tdin) goto pl(intr0);      "  request/grant handshake "
intr1:      irq, if (not tdin) goto pl(intr1);
intr2:      irq, if (not tiaki) goto pl(intr2);
intr3:      rply * dataout, if (tdin) goto pl(intr3);     "  output vector "
intr4:      rply * dataout, if (tiaki) goto pl(intr4);
            complt, goto pl(top);


"           ***********************************************************   "
"           *     PROGRAMMED I/O ROUTINE  -  Q-Bus master accessing       "
"           *     device.  Perform Q-Bus DATO/DATI handshake.             "
"           ***********************************************************   "

datx:       twtbt, if (aux) goto pl(dato);

dati:       off, if (not tdin) goto pl(dati);                " slave DATI "
wait6:      rply * dataout, if (tdin) goto pl(wait6);
            off, ret;

dato:       tdout, if (not aux) goto pl(dato);               " slave DATO "
wait5:      rply + datain + tdout, if (aux) goto pl(wait5);
            off, ret;
```

Figure 6-3. Q-Bus Controller Source Program Listing (Sheet 2 of 3)

```
"          ************************************************************   "
"          *     DMA SERVICE ROUTINE  -  Device DMA service request.      "
"          *     Perform Q-Bus DMA handshake.                             "
"          ************************************************************   "

dmax:      off, if (tdmgi) goto pl(dmax);       "  request/grant handshake "
dma1:      dmr, if (not tdmgi) goto pl(dma1);
dma2:      dmr, cmp tm(sync_rply) to pl(0);
           dmr, if (not eq) goto pl(dma2);
           sack + write, if (aux) goto pl(dmadato);    "  bus master now "

"          DMA READ ROUTINE  (Q-Bus master DATI)                           "

dmadati:   sack * addrout, continue;                     "  addr setup    "
           sack * addrout, continue;
          (sack + sync) * addrout, continue;             "  addr hold     "
          (sack + sync) * addrout, load pl(2B#h);        " 10 us timeout "
wait1:     sack + sync + din, if (trply) goto pl(pass1);
           sack + sync + din, if (trply) goto pl(pass1);
           sack + sync + din, while (creg<>0) loop to pl(wait1);
error1:    sack + sync + error, ret;                     " timeout exit  "
pass1:     sack + sync + din, continue;                  " data deskew   "
           sack + sync + din, continue;
wait2:     sack + sync, if (trply) goto pl(wait2);       " clock data in "
           complt, ret;

"          DMA WRITE ROUTINE  (Q-Bus master DATO)                          "

dmadato:  (sack + wtbt) * addrout, continue;              "  addr setup   "
          (sack + wtbt) * addrout, continue;
          (sack + wtbt + sync) * addrout, continue;       "  addr hold    "
          (sack + wtbt + sync) * addrout, load pl(2b#h);
wait3:    (sack + sync + dout) * dataout, if (trply) goto pl(pass2);
          (sack + sync + dout) * dataout, if (trply) goto pl(pass2);
          (sack + sync + dout) * dataout, while(creg<>0)loop to pl(wait3);
error2:    sack + sync + error, ret;                     " timeout exit  "
pass2:    (sack + sync + dout) * dataout, continue;      " data deskew   "
          (sack + sync + dout) * dataout, continue;
          (sack + sync) * dataout, continue;             " data hold     "
          (sack + sync) * dataout, continue;
wait4:     sack + sync, if(trply) goto pl(wait4);
           complt, ret;
           .org 63#d
           off, goto pl(0);                        "  hardware reset here. "

end.
```

Figure 6-3. Q-Bus Controller Source Program Listing (Sheet 3 of 3)

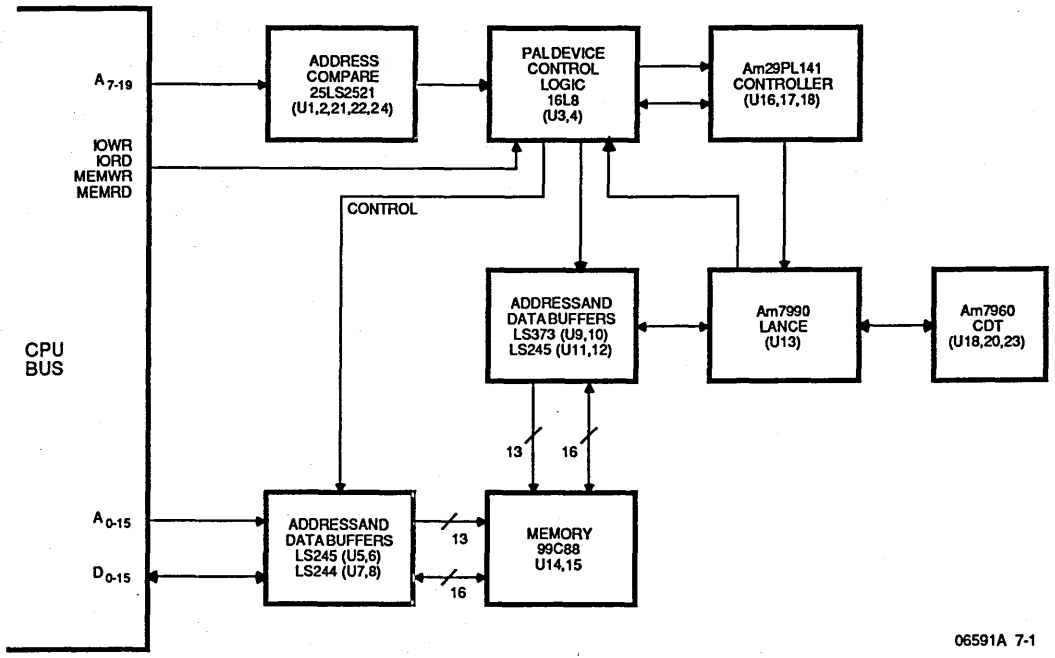


Figure 7-1. Starlan DMA Controller Block Diagram

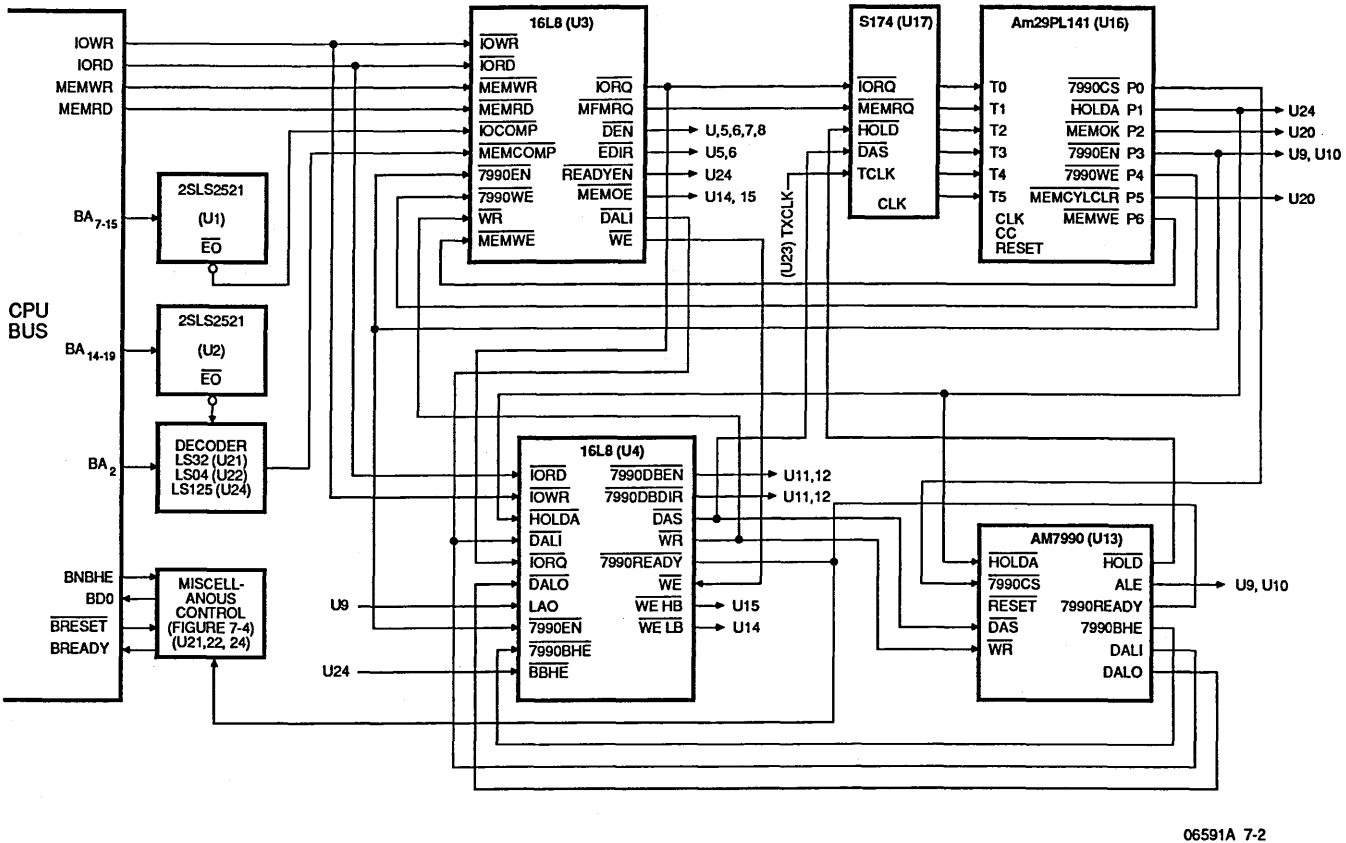


Figure 7-2. Starlan Controller Circuitry

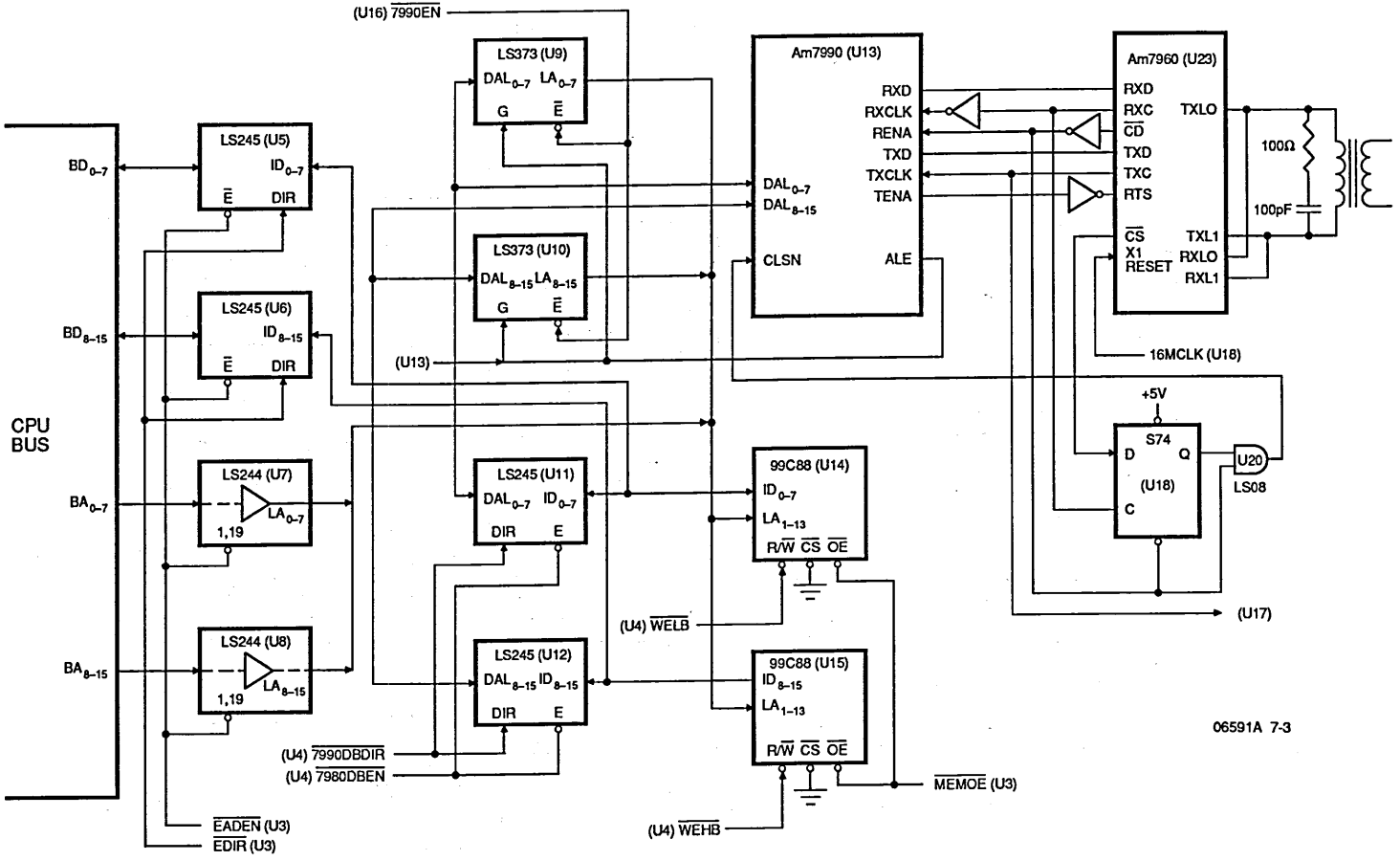


Figure 7-3. Starlan Address and Data Circuitry

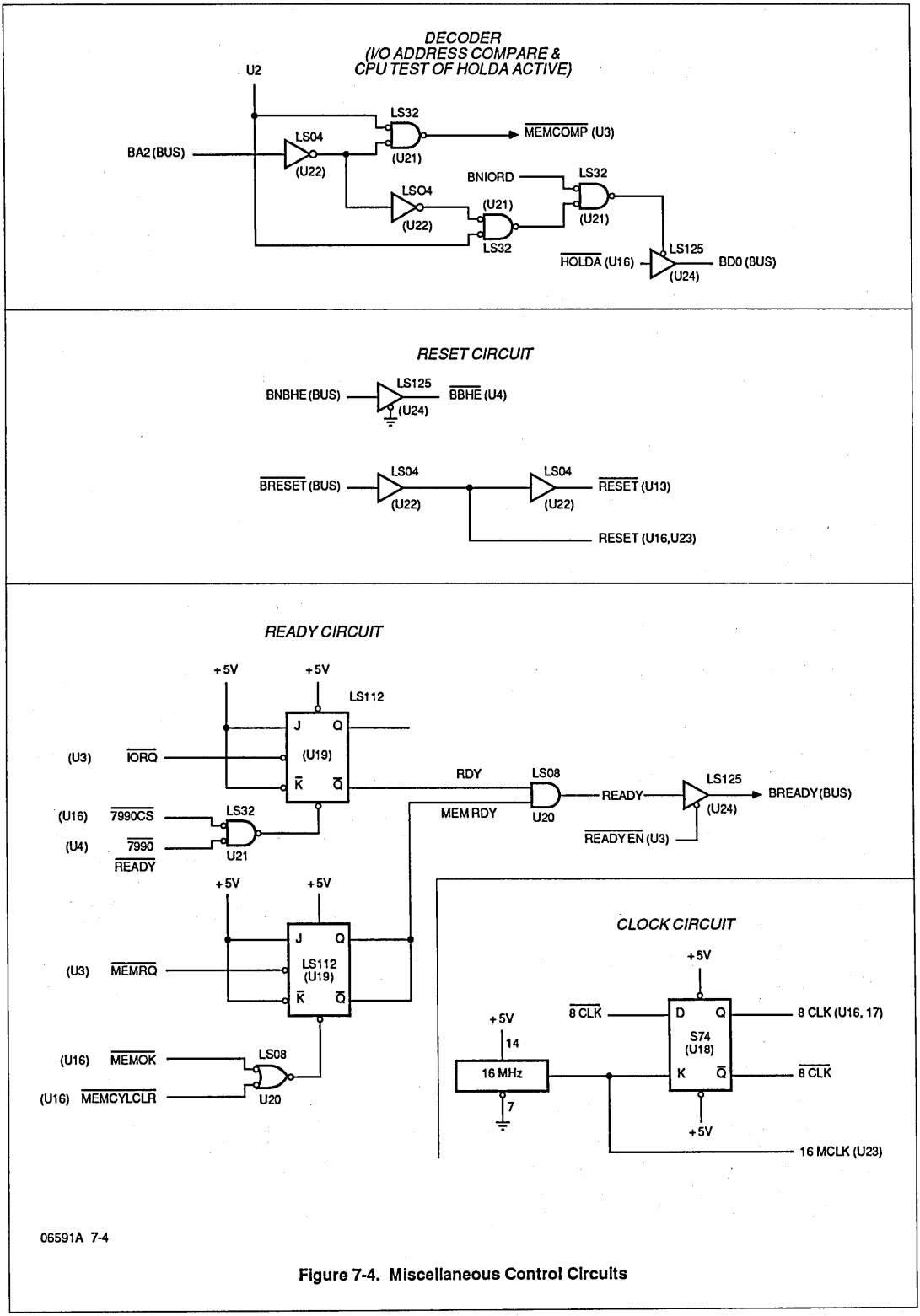


Figure 7-4. Miscellaneous Control Circuits

PAL Devices to control the memory access. See Figure 7-2 for the routing of its signals.

The Am29PL141 can accept seven (7) different test inputs and control 16 different events. This application uses six (6) input lines and eight (8) output lines to accomplish the handshaking and control.

U17 (S174) is used to provide metastability hardening of the Am29PL141.

In the following discussion, refer to Figure 7-3 for the address and data circuity blocks: U5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18,, and 23.

U5 and U6 (LS245) provide the Data Bus buffering.

U7and U8 (LS244) provide the address bus buffering.

U9 and U10 (LS373s) serve as address latches to demultiplex the 7990's DAL bus.

U11 and U12 (LS245s) are data buffers to isolate the 7990 for the dual porting.

U13 is the Am7990. It uses U23 (Am7960) as the Manchester encoder/decoder and media interface to the TXD and RXD lines. This circuitry is shown in Figure 7-3.

U14 and U15 (99C88) are the memories themselves. These may also be expanded very easily if required. The address and data lines are shown in Figure 7-3.

U19 (LS112), U20 (LS08), U21 (LS32), and U24 (LS125) provide the Ready line conditions appropriate to the Bus timing of valid data to the main CPU. This circuitry is shown in Figure 7-4. The clock circuit is also shown in Figure 7-4. The 16 MHz clock is a crystal oscilator. Its fundamental is used to drive the Am7960 (U23) directly. The oscillator frequency is divided by two to drive the prelatch (U17) and the Am29PL141 (U16). Figure 7-4 also shows the RESET circuitry which sends a CPU bus reset signal to the Am29PL141, Am7990, and the Am7960.

This design may also be used to not only provide isolation to the DMA but also to provide a bus translation service for an 8 bit CPU. The 16 bit I/O transfer needed by the Am7990 write and read can be accomplished if, on the data bus side, the D8-15 LS245s are replaced with LS373. In memory operation, the LS373s are made transparent but in I/O, the high byte is written first and then as the low byte is written, both are enabled into the 7990. On a read, the full 16 bit transfer takes place and the low byte is read immediately. The next operation reads location I/O + 2 for the D8-15 value.

In this application, memory is treated as memory and the 7990 is treated as I/O space. The 2 port memory is used by the CPU to set up ring descriptors as well as the rings themselves. Packet buffers can be assembled and disassembled in this area under the operating system at low level drivers. 16K space is enough for 8-512 byte transmit rings and 8-512 byte receive rings. At 1 MHz data rate, that is probably more than enough. However, a 10 MHz design may require 64K DRAM to provide sufficient high speed memory bandwidth.

7.3 MICROPROGRAM

The Am29PL141 controller's major function is to process a HOLD request by the Am7990. When the Am7990 is not active, it processes normal CPU memory read/write and normal I/O read/write (Figure 7-5 shows the microprogram flow diagram).

When the Am29PL141 receives a HOLD request, it sends a HOLDA signal to the Am7990 to activate the Am7990. The HOLDA signal also goes to the BD0 pin of the CPU so that the CPU can check to see if the Am7990 is using the DMA. Only in the HOLDA path (main path) is another task allowed besides the normal path. In the HOLDA path, the CPU is allowed access until T5 of the Am7990 state machine. At that point, the memory is diverted and remains until the completion of the 7990 DMA. The Am7990 dropping Hold Request (HOLD) is what finally clears the HOLDA cycle and returns control to the Am29PL141. Branch #1 is just a normal CPU I/O read/write and branch #2 is a normal CPU memory read/write when the HOLDA is not active. Figure 7-6 is the actual microcode of the 29PL141.

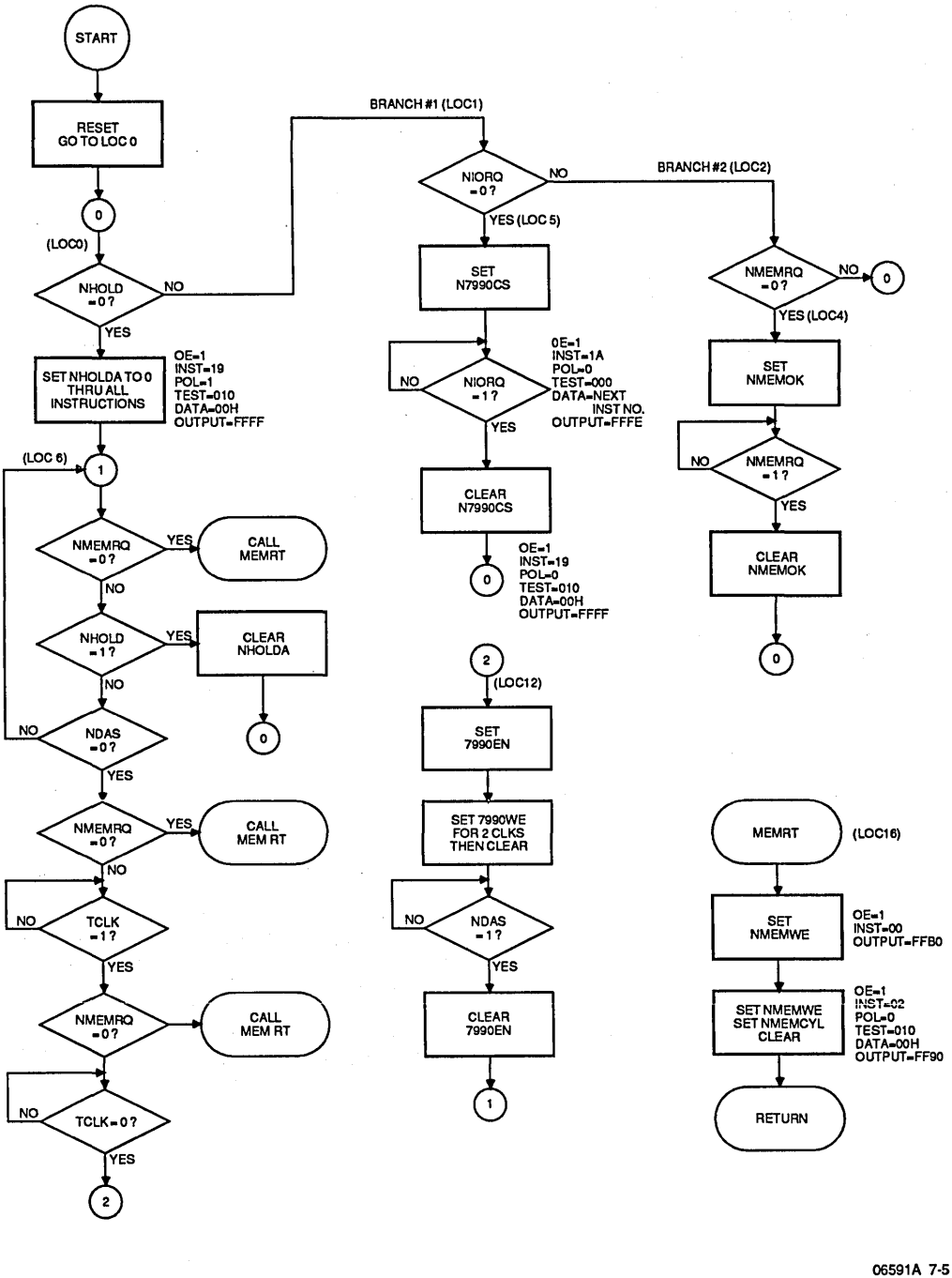
Note: The 7990 cannot be slave-accessed with HOLDA valid. Therefore, any I/O request is blocked in the controller during a DMA transfer. In order to prevent a possible 48 microsecond Ready/Wait signal, HOLDA can be sampled by the CPU at the data I/O pin BD0 and when logically false, the I/O request can then be made at I/O address of 7990 + 4.

7.4 PAL DEVICE EQUATIONS

```
PAL Device #1 (U3): CPU Bus Control
                    (AmPAL16L8)

PIN

  /IORD     = 1        /MEMWE    = 11

  /IOWR     = 2        /WE       = 12
  /MEMRD    = 3        /DALI     = 13
  /MEMWR    = 4        /MEMOE    = 14
  /MEMCOMP  = 5        /READYEN  = 15
  /IOCOMP   = 6        /EDIR     = 16
  /7990EN   = 7        /EADEN    = 17
  /7990WE   = 8        /IORQ     = 18
  /WR       = 9        /MEMRQ    = 19
;

BEGIN.

  MEMRQ = MEMRD * MEMCOMP + MEMWR *
    MEMCOMP ;

  IORQ    = IORD * IOCOMP + IOWR *
     IOCOMP ;

  EADEN = MEMRQ * /7990EN + IORQ *
     /7990EN ;

  EDIR  = MEMRD + IORD ;

  READYEN = MEMRQ + IORQ ;

  WE = 7990WE * WR + MEMWE * MEMWR +
   /7990EN * MEMRQ * MEMWR ;

  MEMOE = /7990EN * MEMRD + 7990EN *
   DALI ;

END.


PAL Device #2 (U4): 7996 control
                    equations

PIN

  /IORD     = 1        /BBHE     = 11

  /IOWR     = 2        /WELB     = 12
  /HOLDA    = 3        /WEHB     = 13
  /DALI     = 4        /WE       = 14
  /IORQ     = 5        /7990READY = 15
  /DALO     = 6        /WR       = 16
  /LA0      = 7        /DAS      = 17
  /7990EN   = 8        /7990DBDIR = 18
  /7990BHE  = 9        /7990JDBEB = 19
;

BEGIN.

  IF ( /HOLDA ) THE ENABLE (DAS , WR,
  7990READY ) ;

  DAS = OWWR + IORD ;

  WR = IOWR ;

  7990READY = HOLDA;

  7990DBEN = /HOLDA * IORQ + HOLDA *
  7990EN * ( DALI + DALO ) ;

  7990DBIR = /HOLDA * IORQ + HOLDA *
  DALI ;

  WELB = /LA0 * WE ;

  WEHB = WE *7990EN * 7990BHE + WE *
  /7990EN * BBHE ;

END.
```

7.5 SUMMARY

In summary, the design solves the system requirements of double buffering and DMA isolation using a minimum of parts yet retaining memory at bus bandwidth without a large number of wait states added. The 7990 is allowed full access as needed without ever seeing a slow down and the basic design has a large amount of frequency latitude for the LAN Speed.



06591A 7-5

Figure 7-5. Starlan Controller Program Flow Diagram

```
DEVICE ( PL141 )

DEFAULT = 1 ;

 DEFINE
          NIORQ = T0
          NMEMRQ = T1
          NHOLD = T2
          NDAS = T3
          TCLK = T4
          VCC = CC
          N7990CS = FFFE#H
          NHOLDA = FFFD#H
          NMEMOK = FFFB#H
          N7990EN = FFF7#H
          N7990WE = FFEF#H
          NMEMCYLCLR = FFDF#H
          NMEMWE = FFBF#H
          NEXEC = FF7F#H;

DEFAULT_OUTPUT = FFFF#H;

BEGIN

EXEC :    NEXEC , IF ( NOT NHOLD ) THEN GOTO PL ( HOLDA ) ;
          NEXEC , IF ( NOT NIORQ )THEN GOTO PL ( IORQ ) ;
          NEXEC , IF ( NOT NMEMRQ ) THEN GOTO PL ( MEMRQ ) ;
          NEXEC , IF ( VCC ) THEN GOTO PL ( EXEC ) ;

MEMRQ :   NMEMOK , IF ( NMEMRQ ) THEN GOTO PL (EXEC) ELSE WAIT;

IORQ :    N7990CS , IF ( NIORQ ) THEN GOTO PL (EXEC) ELSE WAIT;

HOLDA :   NHOLDA , IF ( NOT NMEMRQ ) THEN CALL PL ( MEM ) ;
          NHOLDA , IF ( NHOLD ) THEN GOTO PL ( EXEC ) ;
          NHOLDA , IF ( NDAS ) THEN GOTO PL ( HOLDA ) ;
          NHOLDA , IF ( NOT MEMRQ ) THEN CALL PL ( MEM ) ;
          NHOLDA , IF ( NOT TCLK ) THEN GOTO PL ( HOLDA1 ) ELSE WAIT;
HOLDA1 :  NHOLDA , IF ( NOT NMEMRQ ) THEN CALL PL ( MEM ) ;
          NHOLDA , IF ( TCLK ) THEN GOTO PL (HOLDA2) ELSE WAIT;
HOLDA2 :  FFF5#H , CONTINUE ;
          FFE5#H , CONTINUE ;
          FFE5#H , CONTINUE ;
          FFF5#H , IF ( NDAS ) THEN GOTO PL (HOLDA) ELSE WAIT;

MEM :     FFBB#H , CONTINUE ;
          FF9B#H , CONTINUE ;
          NHOLDA , IF (VCC ) THEN RET ;
          .ORG 63#D
          EXEC   , IF ( VCC ) THEN GOTO PL ( EXEC ) ;

END.
```

Figure 7-6. Starlan Controller Source Program Listing

CHAPTER 8

IBM PC-SSR INTERFACE USING an Am29PL141 CONTROLLER

8.1 THE DESIGN PROBLEM

This application note describes the use of an Am29PL141 controller and an IBM PC or other computer to run diagnostics tests on a device containing a Serial Shadow Register (SSR). The SSR is a special serial in, serial out register built into devices to facilitate diagnostic testing.

To test a complex state machine or a microcoded CPU engine in a manufacturing environment is a complex task. The conventional method has been to use a "Bed of Nails" consisting of probes making contact to the printed circuit board (PCB) in specially assigned places. A master program in the tester provides a stimulus and then checks the response. These Bed of Nails test fixtures are complex and costly and worst of all, are mechanically interlinked in such a manner that a simple movement of an IC on the PCB may cause a whole fixture to be scrapped or at least reworked. Each fixture may cost up to $10,000 and requires an expensive tester to control it.

8.2 SSR FUNCTIONAL DESCRIPTION

AMD in conjunction with MMI pioneered a concept called Serial Shadow Register (SSR). Typically in state machines or microcoded CPUs, data is latched into a register on one clock to drive the logic and on the next clock, the result is latched into a destination register. The SSR is an additional diagnostic register linked to the main device register. It can load new information into the device register and capture the response of the device. Various test inputs are entered into the SSR serially from a computer with the assistance of a controller (FPC). The device executes the input and returns the result into the SSR. The controller serially extracts the result from the SSR and transfers it to the computer. The computer then checks the response with the known correct response. Using serial input and output to the SSR keeps the pin count down.

SSRs can be used in all phases of the product testing because they are a part of the device and therefore available at all times. They can be used in engineering to debug the design, in manufacturing to test each device for compliance, and, in field service, to diagnose faulty operation either at the customer site or at the repair depot.

The controller's task is to convert the parallel IBM PC bus, or equivalent, to a serial data stream to be shifted into the SSRs. The SSR is driven from a relatively inexpensive Personal Computer (PC) that has a file of many stimulus patterns and the corresponding response patterns. In operation, the PC writes the first byte of the stimulus pattern to the SSR controller, in parallel (See Figure 8-1). The controller then shifts the pattern out to the SSR (stimulus chain, N1 bits long, in the device to be tested) and informs the PC through the "DONE" flag that it can accept more parallel data. This interchange goes on until the stimulus chain in the device being tested is full (N1 bits shifted).

Then the PC changes the state from "SHIFT OUT" to "EXECUTE" and the controller generates the necessary clocks to compute the response. The FPC then loads the first byte from the SSR response chain into the PC read register and informs the PC. The PC now examines, on a bit for bit basis, the response pattern just read with the known good response pattern in its file. Any errors can be flagged and output to the printer or displayed on the CRT screen, thereby helping pinpoint the exact area of fault. This byte compare goes on until the entire response chain of N2 bits has been examined. This whole sequence can be done as many times as necessary to fully check out the PCB at the bit level.

8.3 ARCHITECTURE

The heart of the operation is the AMD Am29PL141, Fuse Programmable Controller. It takes care of controlling the D clock, P clock, and Mode of the serial chain. It shifts the 8 bits out and then specifies "DONE". It monitors the "SHIFT OUT" and "Go" control bits for status change. Figure 8-2 gives pin level detail of the blocks or units shown in the the block diagram. Figure 8-3 shows the user interface circuitry.

U1 serves as an address decode PAL Device whose equations are given later. U2 is just a data bus buffer to keep the loading to 1 LS TTL load.

U3 and U4 form the handshake flip flops for the Am29PL141 controller to the PC interface. U3

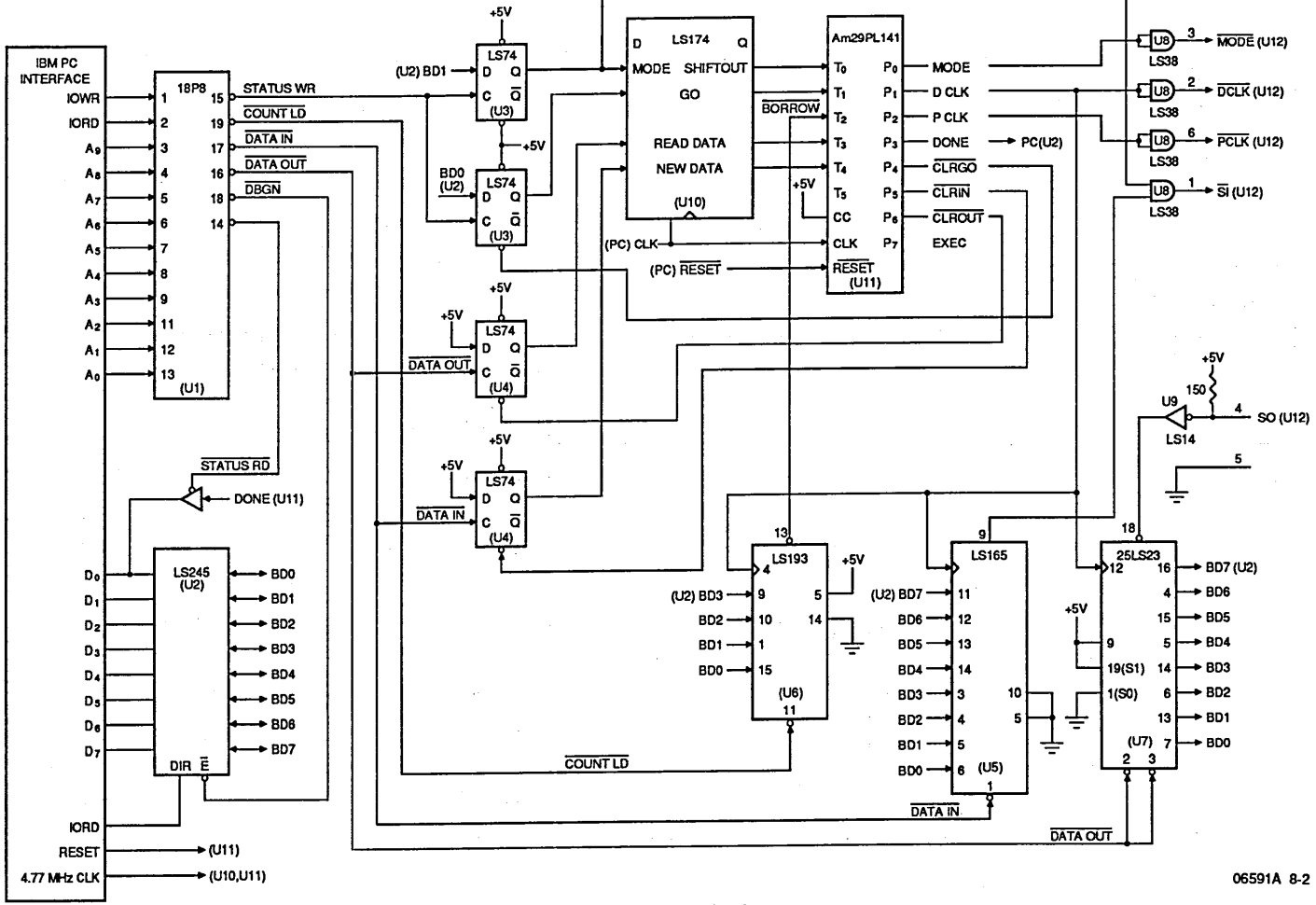
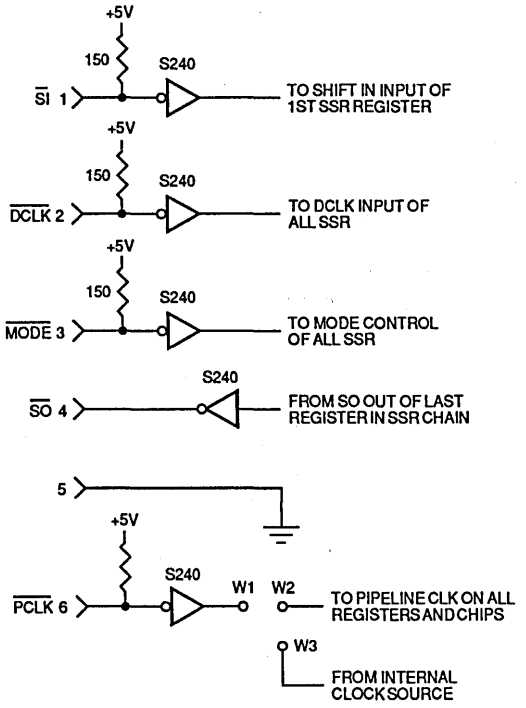


Figure 8-2. SSR Controller Circuitry



JUMPER W2–W3 FOR NORMAL OPERATION
W1–W2 FOR DIAGNOSTIC MODE

06591A 8-3

Figure 8-3. User Equipment Interface Circuitry

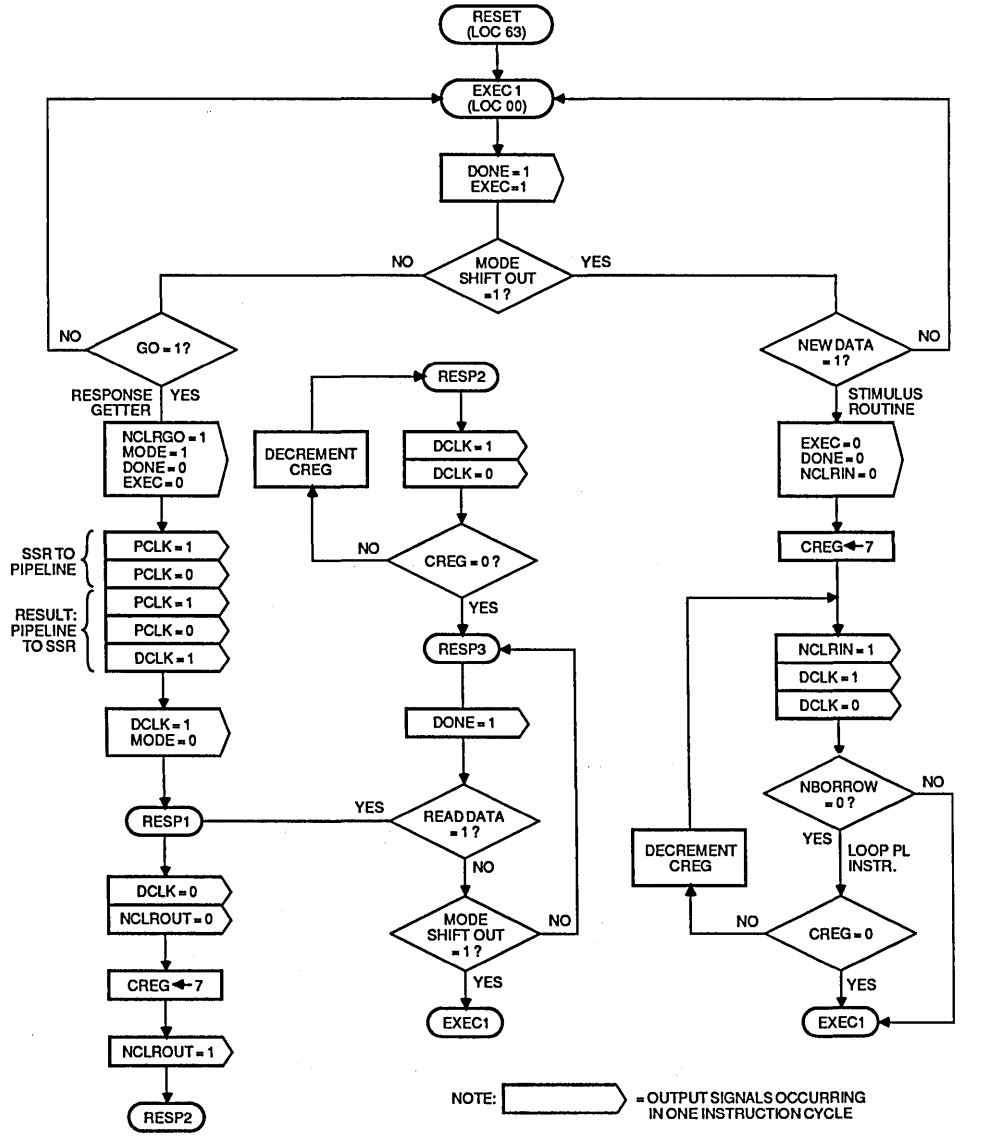


Figure 8-4. SSR Controller Program Flow Diagram

```
DEVICE ( PL141 )

DEFAULT = 1 ;

 DEFINE
         MODE_SHIFTOUT = T0
         GO = T1
         NBORROW = T2
         READDATA = T3
         NEWDATA = T4
         VCC = CC
         MODE = 0071#H
         DCLK = 0072#H
         PCLK = 0074#H
         DONE = 0078#H
         NCLRGO = 0060#H
         NCLRIN = 0050#H
         NCLROUT = 0030#H
         EXEC = 00F0#H ;

DEFAULT_OUTPUT = 0070#H ;

BEGIN

EXEC1 :   EXEC + DONE , IF ( MODE_SHIFTOUT ) THEN GOTO PL ( EXEC2 ) ;
         EXEC + DONE , IF ( GO ) THEN GOTO PL ( RESP ) ;
         EXEC + DONE , IF ( VCC )  THEN GOTO PL ( EXEC1 ) ;
EXEC2 :   EXEC + DONE ,IF ( NOT NEWDATA ) THEN GOTO PL ( EXEC1 ) ;


STIM :    NCLRIN , IF ( VCC ) THEN LOAD PL ( 07#H ) ;
STIM1 :   DCLK , CONTINUE ;
             , CONTINUE ;
             , IF ( NOT NBORROW ) THEN GOTO PL ( STIM2 ) ;
             , WHILE ( CREG < > 0 ) LOOP TO PL ( STIM1 ) ;
STIM2 :   EXEC + DONE , IF ( VCC ) THEN GOTO PL ( EXEC1 ) ;


RESP :    NCLRGO + MODE , CONTINUE ;
         MODE + PCLK , CONTINUE ;
         MODE , CONTINUE ;
         MODE + PCLK , CONTINUE ;
         MODE + DCLK , CONTINUE ;
RESP1 :   NCLROUT , IF ( VCC ) THEN LOAD PL ( 07#H ) ;
RESP2 :   DCLK , CONTINUE ;
               , CONTINUE ;
                , WHILE ( CREG < > 0 ) LOOP TO PL ( RESP2 ) ;
RESP3 :       , IF ( READDATA )  THEN GOTO PL ( RESP1 ) ;
             , IF ( NOT MODE_SHIFTOUT ) THEN GOTO PL ( RESP3 ) ;
         DONE + EXEC , IF ( VCC ) THEN GOTO PL ( EXEC1 ) ;
         .ORG 63#D
         DONE + EXEC , IF ( VCC ) THEN GOTO PL ( EXEC1 ) ;
END.
```

Figure 8-5. SSR Controller Source Program Listing

CHAPTER 9

QUARTER-INCH TAPE CARTRIDGE and SMALL COMPUTER SYSTEM INTERFACE CONTROLLER USING Am29PL141

9.1 OVERVIEW

This application note describes the use of the Am29PL141 Fuse Programmable Controller (FPC), to control both the Quarter Inch Tape Cartridges via the QIC-02 industry standard and the Small Computer Systems Interface (SCSI), also an industry standard as defined by ANSI X3T9.2 subcommittee. This controller functions as the "Host" to the QIC-02 interface and as an "Initiator" to a SCSI system. This design provides the capability to transfer data in both directions, between the SCSI bus and QIC-02.

A practical use is to back up data on a hard disk (SCSI) via Tape (QIC-02). The FPC functions as a high performance (50 ns instruction cycle time) I/O Controller which is slave to the system CPU (host). It supports the maximum data rates of both interfaces (1.5 Mbyte/Sec. asynchronous mode for SCSI). This design uses the 80188 microprocessor, but any host microprocessor could be interfaced to the FPC in a similar fashion. The QIC-02 standard interface is fully supported and the single initiator multiple target mode is supported for SCSI. Although this application does not include using all advanced features of SCSI, the section on "Advanced Features of SCSI" does provide insight into upgrading this design.

In the following discussions, it is assumed that the reader is somewhat familiar with the 80188, FPC, QIC-02, and SCSI. An overview of the QIC- 02 and SCSI is given below. A discussion of the QIC-02 and SCSI, including timing diagrams, has been included as an Appendix.

9.1.1 QIC-02 Overview

QIC-02 is an industry standard which defines the interface between a host system and Quarter Inch Cartridge Tape Drives. Read/write commands, status and, of course, data are transmitted over this interface, as depicted in Figure 9-1. The bus and control signals between QIC-02 and host are all standard TTL levels. Timing diagrams for this interface are given in Appendix C. This interface handshake timing is duplicated for the host side by the FPC and two AmPAL22V10s.

The interface lines are used as follows:

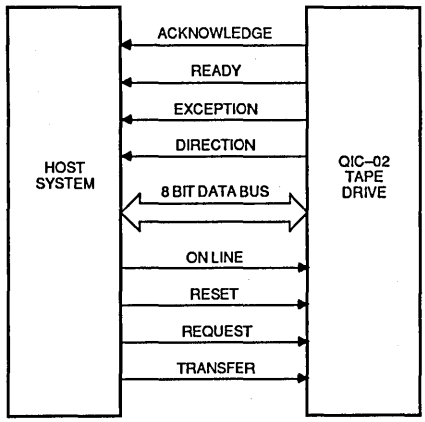
ACKNOWLEDGE (ACK) is used with Transfer to transfer data across the interface.

READY (RDY) indicates that the tape drive can accept a command. It is used to handshake the command across the interface. In the write mode, READY indicates that the drive's internal buffer is empty and ready to receive new data. In the read mode, READY indicates the drive buffer can now be accessed by the host.

EXCEPTION (EXP) alerts the host that the execution of a command has been terminated. This may be a normal completion or an interrupt due to a fault (hard errors, write protected, etc.). The response by the host must be READ STATUS.

DIRECTION (DIRC) indicates direction of data flow. This signal is used to enable/disable the data bus transceivers in the HOST.

ON-LINE signal is deasserted at the beginning of a read (from tape) or write (to tape) operation.



06591A 9-1

Figure 9-1. QIC-02 Interface

RESET initializes the tape drive. The drive repositions the heads to track zero.

REQUEST indicates that a command is on the data bus.

TRANSFER is used with ACKNOWLEDGE to handshake data over the bus, see timing diagram.

9.1.2 SCSI Overview

Small Computer Systems Interface (SCSI) is a disk controller standard developed by the ANSI X3T9.2 subcommittee. SCSI defines an 8-bit parallel bi-directional data bus with parity, plus nine control lines. The SCSI protocol allows single or multiple host computers (initiators) to share multiple peripherals (targets, i.e. hard disk, floppy disks, etc.). Up to eight daisy chained devices can reside on the SCSI bus, with data transfer rates of 4 Mbytes/sec. synchronous and 1.5 Mbyte/sec. asynchronous. The timing diagrams are given in Appendix C.

The following is a summary of the interface signals:

I/O is driven by a target to control the direction of data movement. True indicates input to the initiator.

MSG is driven by a target to indicate "Message Phase". When MSG is asserted, REQ (Request) is also asserted by the target for transfer of data byte indicating the end of the operational phase ("Message").

REQ is asserted by target to indicate that a data byte is to be transferred on the data bus. Data byte is transferred via handshake with ACK (Acknowledge).

ATN (Attention) is driven by an initiator to indicate to target an "attention" condition.

An initiator uses SEL along with asserting the appropriate data (address) bits (0-7) to select a target. Select line is deasserted after the target asserts BSY to acknowledge selection.

RST (Reset) is a pulse asserted by the initiator to stop the target's present operation and return same to idle condition.

Data bus and control signals require open collector drivers capable of sinking 48 mA each to support SCSI mode of multiple initiators with multiple targets. SCSI provides for either single ended (6 meter max. cable length) transmission or differential (up to 25 meters).

9.2 FUNCTIONAL DESCRIPTION

Figure 9-2 shows the block diagram of the Am29PL141 (FPC) QIC-02 and SCSI Controller. This controller functions as a "Host" to the QIC-02

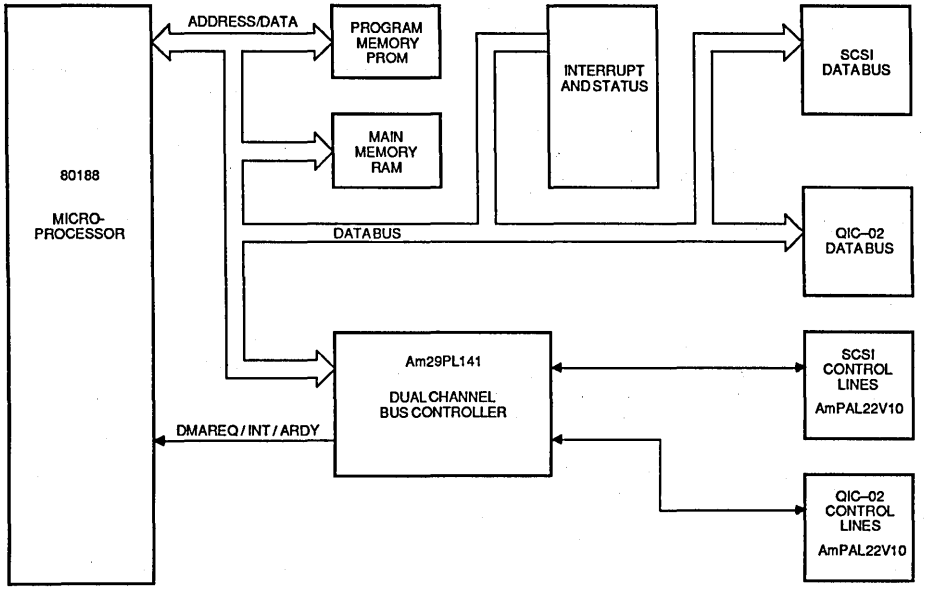


Figure 9-2. AmPL141 QIC-02 and SCSI Controller Block Diagram 06591A 9-2

interface and as an "Initiator" to a SCSI system. This design is composed of three main functional blocks: Microprocessing Unit, Dual Channel Bus Controller, I/O Bus Interface.

The Microprocessing Unit is a straightforward design centered around the 80188 microprocessor which provides system level control to the FPC through commands issued over its 8-bit data bus and with feedback from the FPC via DMA requests, interrupt, wait state insertion asynchronous ready (ARDY), Interrupt Register, and a Status Register.

The heart of the Dual Channel Bus Controller is the Fuse Programmable Controller (Am29PL141) which generates and monitors interface control signals for both I/O bus interfaces (QIC-02 and SCSI). The FPC is slave to the 80188, and controls the transfers of commands, status, and data to/from both I/O interfaces via single byte DMA transfers to/from Main Memory. Interleaved single byte transfer to/from both I/O devices is provided. This approach supports maximum rates for both I/O channels.

The I/O Bus Interface provides single-ended drive for both I/O channels (48 mA per line). Open collector drivers are required for all SCSI generated control signals; however, standard (Am29800 family) buffers and transceivers are satisfactory for the QIC-02 and SCSI data bus.

Each of these functional blocks are now described in detail.

9.2.1 80188 Microprocessing Unit

The Microprocessing Unit in this design performs all of the high level system control and application functions required when interfacing to tape and disk. These functions include system and application programs, direct memory access (DMA) controllers, timers, interrupt controllers and chip select decoders. The 80188 High Integration Microprocessor was chosen for this design because all of the above functions except the programs and associated memory are contained in a single chip. The 80188 provides two DMA channels, three programmable timers, a programmable interrupt controller and a programmable chip select decoder. In this design, both DMA channels, one timer, one external interrupt and four peripheral chip selects (PCS1-4) are dedicated to the SCSI and QIC-02 interfaces.

In order to configure the 80188 for this application, certain operations must be performed prior to executing any instructions which will access the SCSI or QIC-02 interfaces. After reset, only the Upper Memory Chip Select (UMCS) is active in order to allow the 80188 to begin execution at location FFFOH. At this time, UMCS is programmed for a block size of 1K bytes. To allow full use of the Am27512, 64KX8 EPROM, the UMCS register should be programmed with the value F03DH. This sets UCMS for a 64K byte block size, inserts one automatic wait state and ignores external RDY in the range FOOOOH to FFFFFH. Likewise, the Lower Memory Chip Select (LMCS) must be programmed via the LMCS register.

Programming this register with the value O1FCH selects an 8K byte block size, zero automatic wait states and ignores external RDY in order to take full advantage of the Am99C88-70, 70ns 8KX8 CMOS Static RAM. Finally, the Peripheral Chip Selects (PCSM) must be configured. Four of these PCSM are used to select the SCSI and QIC-02 interfaces. The PCSM are configured via MPCS and PACS control registers. The MPCS register is programmed with the value, 84B8H, which places the PCSM in I/O address space, enables all seven PCS lines, inserts no automatic wait state, and uses external RDY. This value also configures the Mid-Range Memory Selects (MCSM) for 8 Kbyte block size The PACS register is programmed with the value 0078H. This places the PCS block at I/O address OOOOH, inserts no automatic wait states, and uses external RDY.

With the hardware now configured, the 80188 is prepared to run applications utilizing the SCSI and QIC-02 interfaces. An example of a simple application is shown in Figure 9-3. This application selects DISKO on the SCSI and reads 2000 bytes into a data buffer. It then rewinds the tape on the QIC-02 and writes the data buffer onto the tape. As can be seen in Figure 9-3, there are several support routines which perform the actual communication with the SCSI/QIC-02 interface.

SOFTWARE SUPPORT ROUTINES:

FPC Control. This procedure outputs a function and a code to the FPC command register. It also reinitializes the watchdog timer via another procedure (WD.Init) not described here. The watchdog timer is used to reset the Am29PL141 in the event that a device on either the SCSI or QIC-02 fails to complete the proper handshake and locks up the bus of 80188.

SCSI–Init. This procedure uses the FPC Control routine to assert and deassert the SCSI RST signal in order to initialize the SCSI interface.

QIC2–Init. This procedure asserts and deasserts the QIC-02 RESET signal to initialize the interface.

```
PROGRAM MAIN;

/* THIS PROGRAM IS AN EXAMPLE OF THE ROUTINES NECESSARY TO
UTILIZE THE SCSI/QIC-02 INTERFACE.  EACH ROUTINE IS DESCRIBED IN
THE ACCOMPANYING TEXT.  THE MAIN PROGRAM PERFORMS THE SIMPLE
OPERATIONS OF READING A MULTI-SECTOR BUFFER, REWINDING THE TAPE
AND WRITING THAT BUFFER TO THE TAPE. */

    CONST
        DISKØ = 1; /* DISK ADDRESS ON THE SCSI BUS */

        DATN =
        DRST =
        INT1 =
        DTREQ =
        TPONL =
        TRINT =
        TPRST =
        DACK =

        SET =
        RESET = Ø; /* CONTROL CODE FOR RESET OPERATION */

        FPC_COMMAND = Ø; /* FPC COMMAND REGISTER ADDRESS */
        SCSI = 128; /* SCSI DATA PORT ADDRESS */
        TAPE = 256; /* QIC-02 DATA PORT ADDRESS */
        ISR = 384;  /* INTERRUPT STATUS REGISTER ADDRESS */
        STAT = 512; /* STATUS BUFFER ADDRESS */

        READ_COMMAND = BYTE [ 8, /* READ COMMAND CODE */
                              Ø, /* LUN Ø, HEAD Ø , TRACK Ø,
                                    SECTOR Ø */
                              Ø,
                              Ø,
                              4, /* FOUR BLOCKS OF 512 TO BE READ */
                              Ø] /* ENABLE RETRIES AND ERROR CORRECTION */

        CHANØ = Ø; /* DMA CHANNEL INDICATORS */
        CHAN1 = 1;

        EOI = 34 + 65280; /* EOI REGISTER OFFSET PLUS CONTROL
                             BLOCK BASE ADDRESS */

        INT1_IS = 13; /* INTERRUPT 1 IDENTIFIER TO RESET
                         IN-SERVICE BIT IN EOI REGISTER */
        DMAØ_IS = 1Ø; /* DMA CHANNEL Ø IDENTIFIER TO RESET
                         IN-SERVICE BIT IN EOI REGISTER */
        DMA1_IS = 11; /* DITTO FOR DMA CHANNEL 1 */

    VAR
        SCSI_FLAG, TAPE_FLAG, COUNT, I : INTEGER;
        DATA_BUFFER [2ØØØ] : BYTE;
        STATUS_BUFFER [2] : BYTE;

   PROCEDURE FPC_CONTROL (FUNC, CODE);
       CONST CMDMASK = BYTE 8;
       VAR CMDACK : BYTE;

       BEGIN
            WD_INIT; /* INITIALIZE WATCHDOG TIMER */
            CMDACK := 8;
            DO WHILE CMDACK <> Ø
                 CMDACK := CMDMASK AND INPUT(ISR);
            OUTPUT(FUNC*8+CODE, FPC_COMMAND);
       END;
```

Figure 9-3. SCSI/QIC-02 Driver Example (Sheet 1 of 3)

```
PROCEDURE SCSI_INIT;
    BEGIN
         FPC_CONTROL (SET, DRST); /* ASSERT SCSI RST */
         DELAY (100); /* WAIT 100 USECS */
         FPC_CONTROL (RESET, DRST); /* DEASSERT SCSI RST */
    END;

PROCEDURE QIC2_INIT;
    BEGIN
         FPC_CONTROL (SET, TPRST); /* ASSERT QIC-02 RESET */
         DELAY (100); /* WAIT 100 USECS */
         FPC_CONTROL (RESET, TPRST); /* DEASSERT RESET */
    END;

PROCEDURE D_SELECT (IDENT);
    BEGIN
         WD_INIT; /* INITIALIZE WATCHDOG TIMER */
         OUTPUT (IDENT, SCSI); /* OUTPUT THE IDENTIFIER TO THE
                                  SCSI PORT */
    END;

PROCEDURE T_CMD (COMMAND);
    BEGIN
         WD_INIT;
         OUTPUT (COMMAND, TAPE);
    END;

PROCEDURE D_XFER (FUNC, BUFFER, COUNT);
    BEGIN
         IF FUNC = READ THEN
              DMA_SETUP (SCSI, BUFFER, COUNT, CHAN0);
         ELSE
              DMA_SETUP (BUFFER, SCSI, COUNT, CHAN0);
         WD_INIT;
         DMA_START (CHAN0);
    END;

PROCEDURE T_READ (BUFFER, COUNT);
    BEGIN
         DMA_SETUP (TAPE, BUFFER, COUNT, CHAN1);
         WD_INIT;
         DMA_START (CHAN1);
    END;

PROCEDURE T_WRITE (BUFFER, COUNT);
    BEGIN
         DMA_SETUP (BUFFER, TAPE, COUNT, CHAN1);
         WD_INIT;
         DMA_START (CHAN1);
    END;

PROCEDURE FPC_ISR;
    VAR INTSTAT : BYTE;
    BEGIN
         INTSTAT := INPUT (ISR); /* GET THE INTERRUPT STATUS */
         IF INTSTAT AND TRDY_MASK THEN
              BEGIN
                   FPC_CONTROL (RESET, TRINT);
                   TAPE_FLAG := 0;
              END;
         IF INTSTAT AND SCSI_ERROR_MASK THEN
              SCSI_INIT;
         IF INTSTAT AND TAPE_ERROR_MASK THEN
              QIC2_INIT;
         FPC_CONTROL (RESET, INT1);
         OUTPUT (INT1_IS, EOI);
    END;
```

Figure 9-3. SCSI/QIC-02 Driver Example (Sheet 2 of 3)

```
    PROCEDURE DMA0_ISR;
        BEGIN
             SCSI_FLAG := -;
             OUTPUT (DMA0_IS, EOI);
        END;

    PROCEDURE DMA1_ISR;
        BEGIN
             TAPE_FLAG := 0;
             OUTPUT (DMA1_IS, EOI);
        END;

    BEGIN  /* MAIN PROGRAM BODY */
        SCSI_INIT;
        TAPE_INIT;
        D_DELECT (DISK0);
        SCSI_FLAG := 1; /* SHOW SCSI OPERATION IN PROGRESS */
        D_XFER (WRITE, READ_COMMAND, 6); /* SEND READ COMMAND TO DISK */
        DO WHILE SCSI_FLAG = 1
             I:= I+1; /* WASTE TIME WAITING FOR COMPLETION */
        SCSI_FLAG := 1; /* SHOW A NEW SCSI OPERATION IN PROGRESS */
        D_XFER (READ, DATA_BUFFER, 2000); /* READ 2000 BYTES */

        /* START AN OPERATION ON THE QIC-02 SIDE OF THE INTERFACE
           TO RUN IN PARALLEL WITH THE SCSI OPERATION */

        TAPE_FLAG := 1; /* SHOW QIC-02 OPERATION IN PROGRESS */
        T_CMD (REWIND); /* REWIND THE TAPE */
        FPC_CONTROL (SET, TRINT); /* ENABLE INTERRUPT ON TAPE RDY */
        DO WHILE TAPE_FLAG = OR SCSI_FLAG = 1
             I := I+1; /* WAIT FOR THE OPERATIONS TO COMPLETE */

        /* BOTH OPERATIONS ARE NOW COMPLETE */

        SCSI_FLAG := 1;
        D_XFER (READ, STATUS_BUFFER, 2); /* GET DISK STATUS */
        DO WHILE SCSI_FLAG = 1
             I := I+1;
        IF STATUS_BUFFER [1] = GOOD_STATUS THEN
             BEGIN
                  TAPE_FLAG := 1;
                  T_CMD (WRITE); /* PUT TAPE IN WRITE MODE */
                  T_WRITE (DATA_BUFFER, 2000); /* SEND OUT THE DATA */
                  DO WHILE TAPE_FLAG = 1
                       I := I+1;
             END;
    END;
END.
```

Figure 9-3. SCSI/QIC-02 Driver Example (Sheet 3 of 3)

D–Select. This procedure outputs an eight bit select code to the SCSI interface. This process is intercepted by the Am29PL141 which performs the SELECT handshake.

T–CMD. This procedure outputs an eight bit command to the QIC-02 interface. This process is intercepted by the Am29PL141 which performs the COMMAND handshake.

D–XFER. This procedure performs all data, command and status transfers to and from the SCSI interface.

T–Read. This procedure reads data from the QIC-02 and places it in a memory data buffer.

T–Write. This procedure writes data from a memory buffer to the QIC-02.

FPC–SR. This procedure is the interrupt service routine for the Am29PL141. Upon entry it obtains the interrupt status from the FPC Interrupt Status Register (ISR). This status is examined to detect the occurrence of any errors. If any are detected, the offending interface is reinitialized. This is a very rudimentary form of error handling and is used only for purposes of this example. More elaborate error handling is possible in actual applications. Prior to exiting this procedure, the interrupt source is reset and the in-service bit in the interrupt controller is cleared.

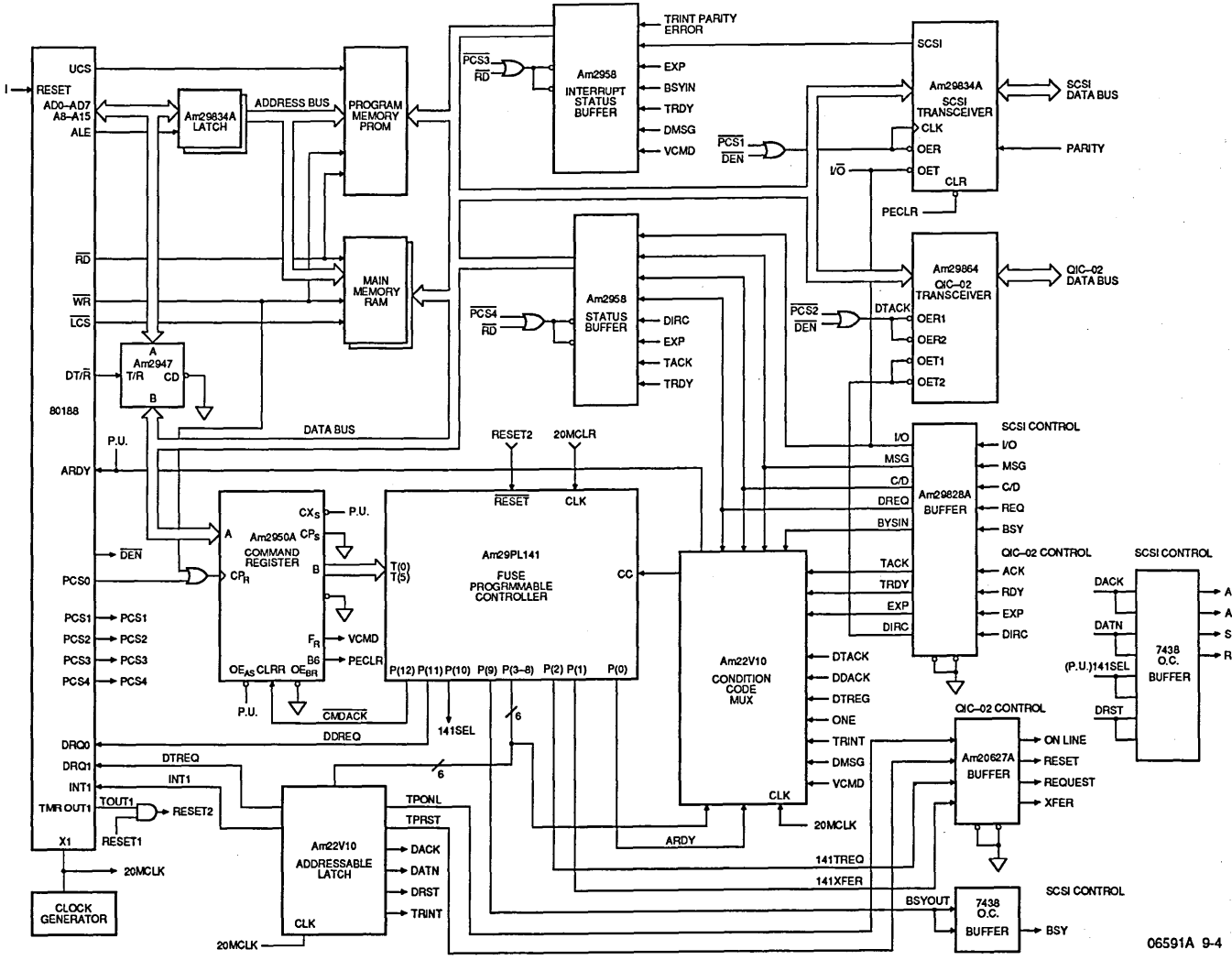


Figure 9-4. Am29PL141 QIC-02 and SCSI Controller Circuitry

DMA0–ISR, DMA1–ISR. These procedures signal the completion of data transfers to other modules by clearing the appropriate in-process flag (SCSI - FLAG, TAPE - FLAG).

9.2.2 Dual Channel Bus Controller Architecture

Refer to the complete schematic (Figure 9-4) for the Am29PL141 QIC-02 and SCSI Controller and two AmPAL22V10s. These three programmable devices provide the intelligence to control SCSI and QIC-02 interfaces, and required additional MSI control logic off-loading these tasks from the 80188 (any host CPU). In this application, the FPC can be thought of as a high speed microprocessor-like controller with twenty-nine fixed instructions, and sixteen programmable output control lines (thirteen of which are used in this application). Each instruction is executed during a single clock cycle of 50 ns. Although it can operate as a stand-

```
DEVICE  condition_code_mux  (AmPAL22V10) ;

"This device selects one of many input conditions to be tested
 by the Am29PL141 and registers it in order to meet the CC setup
 time requirement.  It also collects two pieces of miscellaneous
 logic necessary to produce the ARDY and DMSG signals."

PIN
        clk = 1          vcmd = 2         trint = 3
        dtreq = 4        ddack = 5        dtack = 6
        exp = 7          trdy = 8         tack = 9
        bsyin = 10       dreq = 11        c_d_bar = 13
        msg = 14         dmsg = 15        ardy = 16
        cc = 17          spare = 18       ardy_in = 19
        cc_mux_sel_3 = 20                 cc_mux_sel_2 = 21
        cc_mux_sel_1 = 22                 cc_mux_sel_0 = 23 ;

BEGIN
        ardy = ardy_in + /ddack * /ardy_in ;

        dmsg = c_d_bar * msg ;

        CASE (cc_mux_sel_3,cc_mux_sel_2,cc_mux_sel_1,cc_mux_sel_0)
        BEGIN
                0)        cc := vcmd ;
                1)        cc := ddack ;
                2)        cc := dreq ;
                3)        cc := tack ;
                4)        cc := dtack * dtreq ;
                5)        cc := dtack * /dtreq ;
                6)        cc := msg * c_d_bar + trint * trdy ;
                7)        cc := exp ;
                8)        cc := bsyin ;
                9)        cc := 1 ;
               10)        cc := dtack ;
               11)        cc := dreq * ddack ;
               12)        cc := trdy ;
        END;
END.

Test_vectors

IN
        clk cc_mux_sel_3 cc_mux_sel_2 cc_mux_sel_1 cc_mux_sel_0
        vcmd ddack dreq tack dtack dtreq
        msg c_d_bar trint trdy exp ardy_in bsyin ;
I_O ;
OUT
        cc dmsg ardy ;
```

Figure 9-5. Condition Code MUX PAL Device Description (Sheet 1 of 2)

```
BEGIN

  cccc
  cccc

  ____
  mmmm
  uuuu          c      a
  xxxx                 r
  ____  d   dd d t   db
  ssss vddt tt _ rt  ys     d a
c eeee cara armb ire _y     m r
l 1111 mcec cesa ndx ii   c s d
k 3210 dkqk kqgr typ nn   c g y
                                "
------------------------------  -----

0 XXXX XXXX XXXX XXX 1X   X X H; "ardy"
0 XXXX X1XX XXXX XXX 0X   X X L;
0 XXXX X0XX XXXX XXX 0X   X X H;

0 XXXX XXXX XX11 XXX XX   X H X; "dmsg"
0 XXXX XXXX XX10 XXX XX   X L X;
0 XXXX XXXX XX01 XXX XX   X L X;
0 XXXX XXXX XX00 XXX XX   X L X;

C 0000 0XXX XXXX XXX XX   L X X; "cc = vcmd"
C 0000 1XXX XXXX XXX XX   H X X;

C 0001 X0XX XXXX XXX XX   L X X; "cc = ddack"
C 0001 X1XX XXXX XXX XX   H X X;

C 0010 XX0X XXXX XXX XX   L X X; "cc = dreq"
C 0010 XX1X XXXX XXX XX   H X X;

C 0011 XXX0 XXXX XXX XX   L X X; "cc = tack"
C 0011 XXX1 XXXX XXX XX   H X X;

C 0100 XXXX 11XX XXX XX   H X X; "cc = dtack * dtreq"
C 0100 XXXX 01XX XXX XX   L X X;
C 0100 XXXX 10XX XXX XX   L X X;
C 0100 XXXX 00XX XXX XX   L X X;

C 0101 XXXX 11XX XXX XX   L X X; "cc = dtack * /dtreq"
C 0101 XXXX 10XX XXX XX   H X X;
C 0101 XXXX 01XX XXX XX   L X X;
C 0101 XXXX 00XX XXX XX   L X X;

C 0110 XXXX XX11 00X XX   H X X; "cc = msg * c_d_bar + trint * trdy"
C 0110 XXXX XX00 11X XX   H X X;
C 0110 XXXX XX00 00X XX   L X X;
C 0111 XXXX XXXX XX0 XX   L X X; "cc = exp"
C 0111 XXXX XXXX XX1 XX   H X X;

C 1000 XXXX XXXX XXX X0   L X X; "cc = bsyin"
C 1000 XXXX XXXX XXX X1   H X X;

C 1001 XXXX XXXX XXX XX   H X X; "cc = 1"

C 1010 XXXX 0XXX XXX XX   L X X; "cc = dtack"
C 1010 XXXX 1XXX XXX XX   H X X;

C 1011 X11X XXXX XXX XX   H X X; "cc = dreq * ddack"
C 1011 X01X XXXX XXX XX   L X X;
C 1011 X10X XXXX XXX XX   L X X;

C 1100 XXXX XXXX X0X XX   L X X; "cc = trdy"
C 1100 XXXX XXXX X1X XX   H X X;

END.
```

Figure 9-5. Condition Code MUX PAL Device Description (Sheet 2 of 2)

alone controller, the FPC has been made a slave to the 80188 uP, through the FPC test inputs (T0-T5) and the Command Register (Am2950A).

The processor (80188) writes to the Command Register which contains valid system commands (6 bits) to the FPC. During the IDLE loop of the FPC software, the FPC selects VCMD (by setting output lines P3-P6) as its CC (condition code) input through the condition code mux. If CC (VCMD) is a "pass" condition (asserted) meaning the Command Register has been updated, then the FPC branches to the instruction whose address is given by input T0-T5 (from command register). After the command has been processed, the FPC deasserts the VCMD bit (in the Command Register) and returns to the IDLE loop to check for either another command from the processor or a function required by either SCSI or QIC-02.

Checking for a VCMD and then branching to the processor's command address enables the FPC to operate asynchronous to the processor, whose bus T states (100 ns) are at one-half the FPC's clock rate and skewed in time. The seventh bit in the command register is used for the parity error latch in the SCSI transceiver, Am29834A, (upper right corner of schematic, Figure 9-4).

The Condition Code Mux (CCM) selects the appropriate input to "CC" of the FPC as defined by the FPC's output lines P3-P6. This multiplexing is not always a straight selection but does include logical combinations of input signals in some cases (see Figure 9-5, Condition Code Mux PAL Definition File).

The CCM provides two other outputs. ARDY (asynchronous ready) to the processor is asserted when instructed by the FPC and is used to lengthen the processor's bus cycle time (amount of time data remains valid on the 80188 bus) when QIC-02 or SCSI data transfer timing requires it.

The remaining output from the CCM is DMSG (Disk Message) which is an input to the Interrupt Status Buffer. This is asserted when SCSI asserts both MSG and C/D. Under this condition, the FPC generates an interrupt (INT1), through the Addressable Latch (AmPAL22V10), to the processor indicating that the Disk (SCSI) is requesting "Command" Data. The processor then reads the Interrupt Status Buffer to determine this condition (DMSG asserted). The following inputs are available to the CCM: VCMD, DTACK, and DDACK signals (generated by the processor); MSG, C/D, DREQ, and BSYIN (generated by the SCSI control bus); TACK, TRDY, and EXP (generated by the QIC-02 control bus) and TRINT from the Addressable Latch.

Since the outputs from the FPC are subject to change on an instruction by instruction basis (each clock cycle), certain signals must be latched. The AmPAL22V10 serves as an addressable latch, addressed by the FPC output lines P3-P8 (LADDR). Note that output lines P4-P6 are overlaid with the 3-bit field for the CCM. This technique frees up three spare output lines at the expense of instruction lines in the FPC. Lines P4-P6 select which of the eight latches is selected. P8 enables all latches. P7 determines set or clear of the latch, and P3 (ARESET) provides an asynchronous reset to all latches. The eight outputs from LADDR are: INT1 and DTREG to the processor; TPONL and TPRST to the QIC-02 control bus; DACK, DATN and DRST (control signals to SCSI); and TRINT (a feedback signal to the CCM). Figure 9-6 describes this PAL (LADDR).

9.2.3 Am29PL141 Microprogram

The Am29PL141 is a single-chip Fuse Programmable Controller. It is used in this application as a complex controller by programming the appropriate sequence of instructions. The available instruction set is quite rich. It includes jumps, loops, waits, and subroutine calls, which can be conditionally executed based on the test inputs (T0-T5) or CC input (all of these are used in this application). The FPC flowcharts provide the details of the FPC microprogramming used in this design.

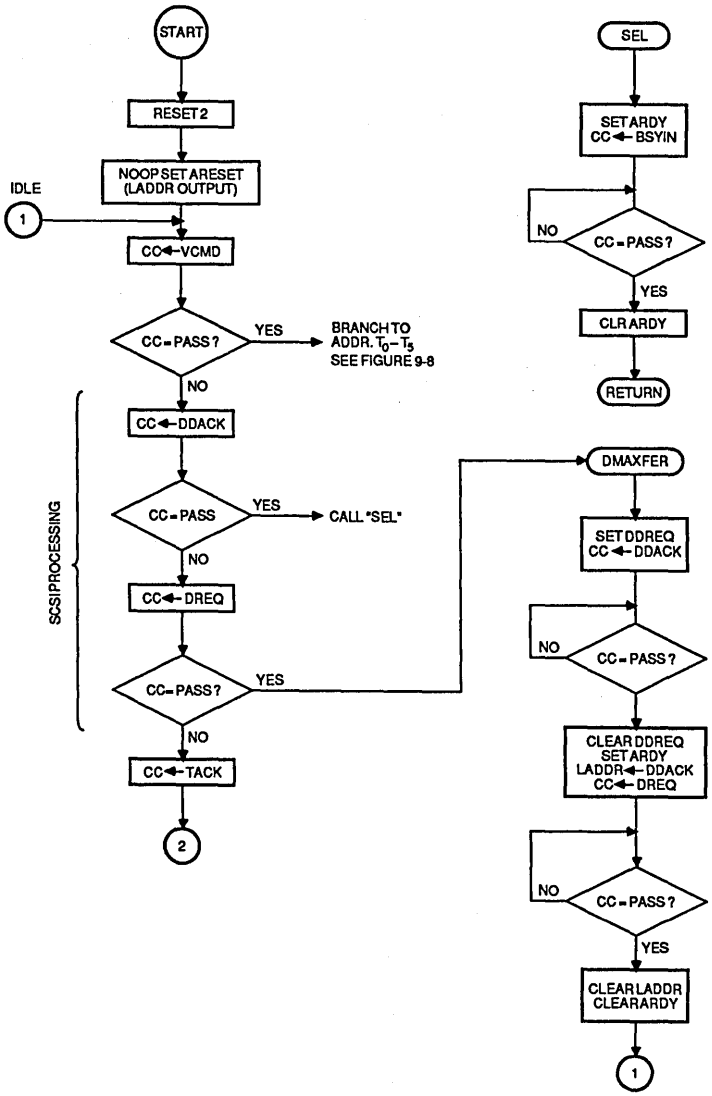
As shown in Figure 9-7, the IDLE LOOP flow diagram, the FPC continually cycles through this loop from initial power-on reset (RESET2), and jumps to one of nine routines depending on the task at hand. After completion of the task, control returns to the idle loop. RESET2 initializes the FPC to start at address sixty-three. RESET2 is generated on system power-up and when the processor's watchdog timer times out (TMROUT1). This timer is programmed to time out if the disk or tape accesses fail to complete the proper handshake in a reasonable time or the FPC locks up the bus of the 80188 because of some error condition.

The first instruction (at address 63) is a NOOP. It is used to assert ARESET (output line) to LADDR for deasserting of latches and to deassert all other output lines. The next instruction is the return/entry point into the idle loop. It selects the CCM to enable path for VCMD to CC input of FPC.

The next state is the first condition test. If CC is a PASS condition, there is a valid command (VCMD asserted). The FPC branches to the address given in Command Register (T0-T5). If VCMD is not asserted (CC = FALSE), it selects DDACK as an input for CC and continues to next incremental

```
DEVICE  addressable_latch (AmPAL22V10);
"This device is the addressable latch used by the Am29PL141 to expand
its I/O capabilities."

PIN
          clk     = 1     enable      = 2           a0        = 3
          al      = 4     a2          = 5           function  = 6
          reset   = 7     spare[0:4]  = 8:11,13     /datn     = 14
          /drst   = 15    intl        = 16          dtreq     = 17
          /tponl  = 18    /trint      = 19          /tprst    = 20
          /dack   = 21    spare_out[0:1] = 22:23 ;
DEFINE
          set = function
BEGIN
          IF (reset) THEN ARESET() ;
          case (A2,A1,A0)
          BEGIN
               0)   datn := datn * /enable + set * enable ;
               1)   drst := drst * /enable + set * enable ;
               2)   intl := intl * /enable + set * enable ;
               3)   dtreq := dtreq * /enable + set * enable ;
               4)   tponl := tponl * /enable + set * enable ;
               5)   trint := trint * /enable + set * enable ;
               6)   tprst := tprst * /enable + set * enable ;
               7)   dack := dack * /enable + set * enable ;
          END;
END.

Test_vectors
IN
          clk enable a2 a1 a0 function reset ;
I_O;
OUT
          /datn /drst intl dtreq /tponl /trint /tprst /dack;

BEGIN
                    f
                    u
                e   n       ///
                n   c r   // d ttt/
                a   t e   ddit prpd
          c b       i s   arnr oira
          l l aaa   o e   tste nnsc
          k e 210   n t   ntlq lttk
                                  "
          ------------------------
          X X XXX   X 1   HHLL HHHH;
          C 0 XXX   X 0   HHLL HHHH;
          C 1 000   1 0   LHLL HHHH;
          C 1 000   0 0   HHLL HHHH;
          C 1 001   1 0   HLLL HHHH;
          C 1 001   0 0   HHLL HHHH;
          C 1 010   1 0   HHHL HHHH;
          C 1 010   0 0   HHLL HHHH;
          C 1 011   1 0   HHLH HHHH;
          C 1 011   0 0   HHLL HHHH;
          C 1 100   1 0   HHLL LHHH;
          C 1 100   0 0   HHLL HHHH;
          C 1 101   1 0   HHLL HLHH;
          C 1 101   0 0   HHLL HHHH;
          C 1 110   1 0   HHLL HHLH;
          C 1 110   0 0   HHLL HHHH;
          C 1 111   1 0   HHLL HHHL;
          C 1 111   0 0   HHLL HHHH;
```

Figure 9-6. Addressable Latch PAL Device



06591A 9-7

Figure 9-7. QIC-02 Controller Program Flow Diagram (Sheet 1 of 2)

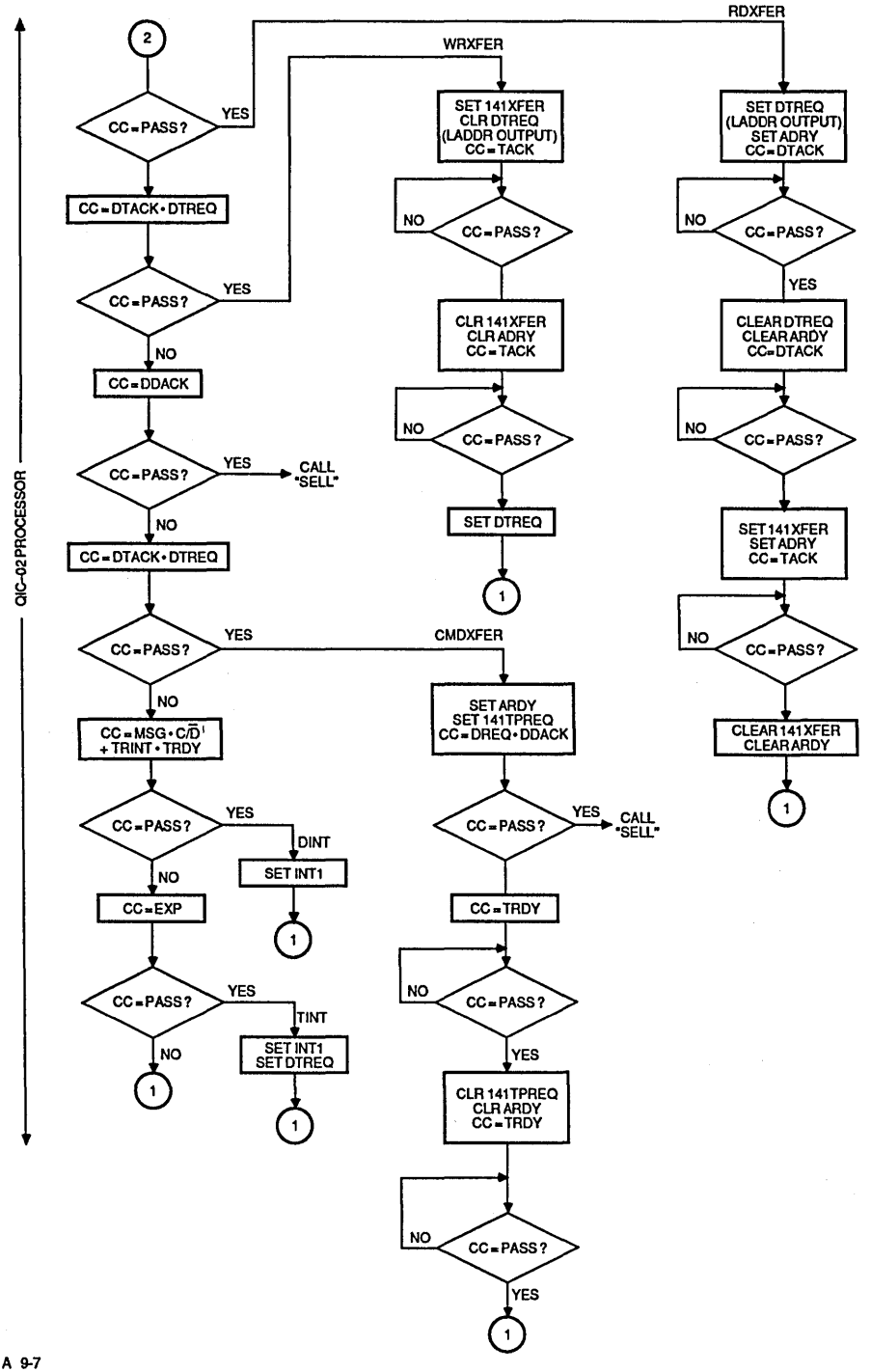
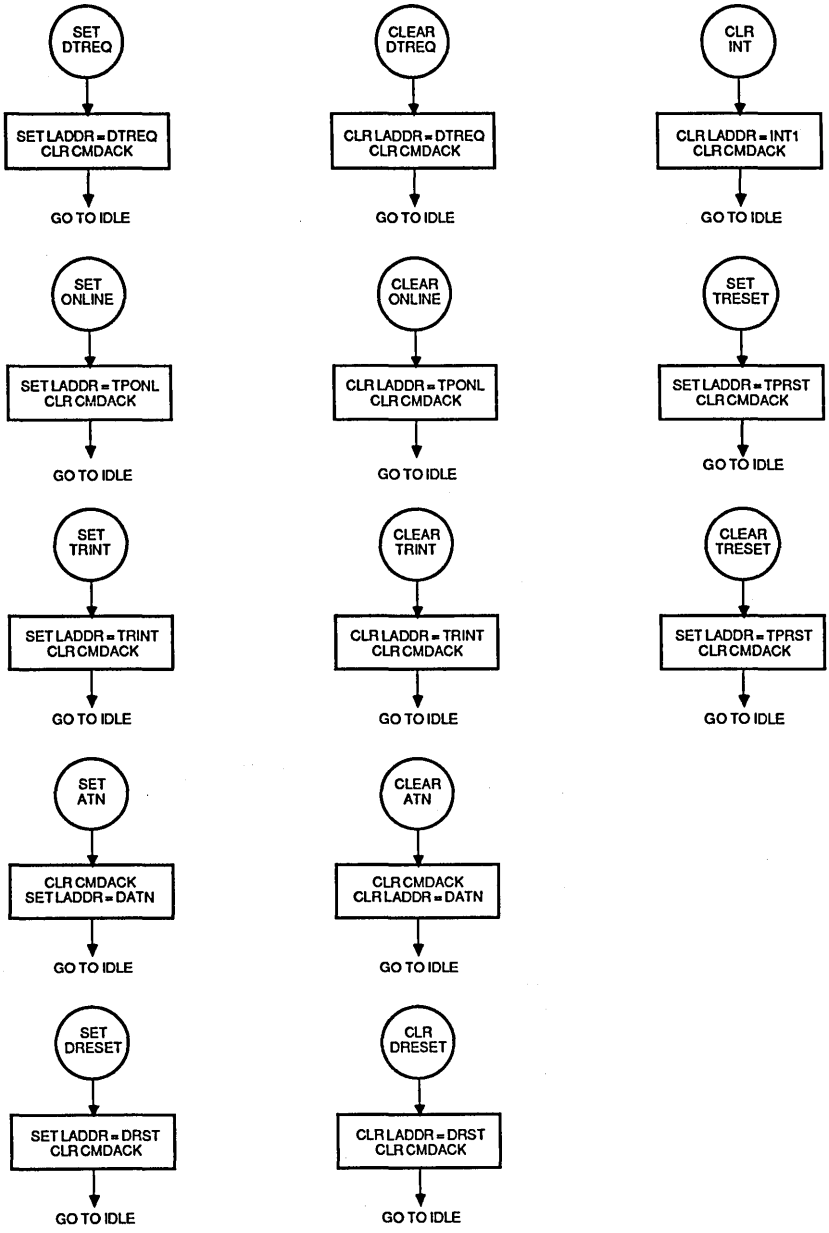


Figure 9-7. QIC-02 Controller Program Flow Diagram (Sheet 2 of 2)



06591A 9-8

Figure 9-8. Am29PL141 Valid Command Routines

```
device (pl141)
"Am29PL141 QIC-02 and SCSI controller"
default = 1;
define
        def = 1000#h
        vcmd = 1000#h      "condition code mux select lines"
        ddack = 1010#h
        dreq = 1020#h
        tack = 1030#h
        dtareq = 1040#h
        dtanreq = 1050#h
        mctirdy = 1060#h
        exp = 1070#h
        bsyin = 1080#h
        one = 1090#h
        dtack = 10a0#h
        drack = 10b0#h
        trdy = 10c0#h

        datn = 1000#h       "addressable latch lines"
        drst = 1010#h
        intl = 1020#h
        dtreq = 1030#h
        tponl = 1040#h
        trint = 1050#h
        tprst = 1060#h
        dack = 1070#h

        cmdack = 0111#h  "other output lines"
        ddreq = 1800#h
        sel = 1400#h
        bsyout = 1200#h
        lsrccms = 1080#h
        len = 1100#h
        ccmardy = 1001#h
        xfer = 1002#h
        tpreq = 1004#h
        lareset = 1008#h;

test_condition = cc;

begin
idle:   vcmd,    continue;
        vcmd,    goto tm(3f#h);
        ddack, if (cc) then call pl(nsel);
        dreq,    goto pl(dmaxfer);
        tack, goto  pl(rdxfer);
        dtareq, goto pl(wrxfer);
        ddack,  if (cc) then call pl(nsel);
        dtanreq, goto pl(cmdxfer);
        mctirdy, goto pl(dint);
        exp,     goto pl(tint);
        one,     goto pl(idle);
nsel:   ccmardy+bsyin, if (cc) then goto pl(next) else wait;
next:   one, goto pl(idle);

dmaxfer:ddreq+ddreq, if (cc) then goto pl(next1) else wait;
next1:  ccmardy+dack+lsrccms+len, continue;
        dreq, if (cc) then goto pl(next2) else wait;
next2:  dack+len, goto pl(idle);

rdxfer: ccmardy+dtreq+lsrccms+len, continue;
        ccmardy+dtack, if (cc) then goto pl(next3) else wait;
next3:  dtreq + len, continue;
        dtack, if (not cc) then goto pl(next4) else wait;
next4:  xfer+ccmardy+tack, if (not cc) then goto pl(next5) else wait;
```

Figure 9-9. QIC-02 Controller Source Program Listing (Sheet 1 of 2)

```
next5:  one, goto pl(idle);

wrxfer: xfer+dtreq+len,continue;
        tack+xfer, if (cc) then goto pl(next6) else wait;
next6:  tack, if (not cc) then goto pl(next7) else wait;
next7:  dtreq+len+lsrccms, continue;
        one,goto pl(idle);

cmdxfer: ccmardy+tpreq+drack, if (cc) then call pl(nsel);
        trdy,   if (cc) then goto pl(next8) else wait;
next8:  trdy, if (not cc) then goto pl(idle) else wait;

dint:   intl+len+lsrccms, continue;
        one,  goto pl(idle);
tint:   intl+len+lsrccms, continue;
        dtreq+len+lsrccms, continue;
        one, goto pl(idle);
setatn: datn+len+lsrccms, continue;
        one, goto pl(idle);

clratn: datn+len, continue;
        one, goto pl(idle);
setdrst: drst+len+lsrccms, continue;
        one, goto pl(idle);

clrdrst: drst+len, continue;
        one, goto pl(idle);

clrint: intl+len, continue;
        one, goto pl(idle);

sdtreq: dtreq+len+lsrccms, continue;
        one, goto pl(idle);

cdtreq: dtreq+len, continue;
        one, goto pl(idle);
stponl: tponl+len+lsrccms, continue;
        one, goto pl(idle);
ctponl: tponl+len, continue;
        one, goto pl(idle);
strint: trint+len+lsrccms, continue;
        one, goto pl(idle);
ctrint: trint+len, continue;
        one, goto pl(idle);
stprst: tprst+len+lsrccms, continue;
        one, goto pl(idle);
ctprst: tprst+len, continue;
        one, goto pl(idle);
        .ORG 63#d
        lareset,continue;
end.
```

Figure 9-9. QIC-02 Controller Source Program Listing (Sheet 2 of 2)

address (PC+1). The IDLE loop continues in this fashion to select and test CCM input conditions and branch accordingly.

Figure 9-8 shows the Valid Command (VCMD) routines. Each command, from the processor will branch to one of these thirteen valid routines. All of these routines are single instructions which set (assert) or clear (deassert) output control lines, which always includes resetting the VCMD signal in the Command Register and returning to idle.

Figure 9-9 is the FPC Microprogram source code listing.

SCSI Interface: The second conditional test in the idle loop is based on DDACK (disk DMA acknowledge). This subroutine is called after the FPC has generated DDREQ (Disk DMA Request) and the processor responded appropriately. The DDACK signal also enables the SCSI bus transceivers for transfer of data. Figure 9-7 shows this call routine (SEL). The FPC asserts ARDY

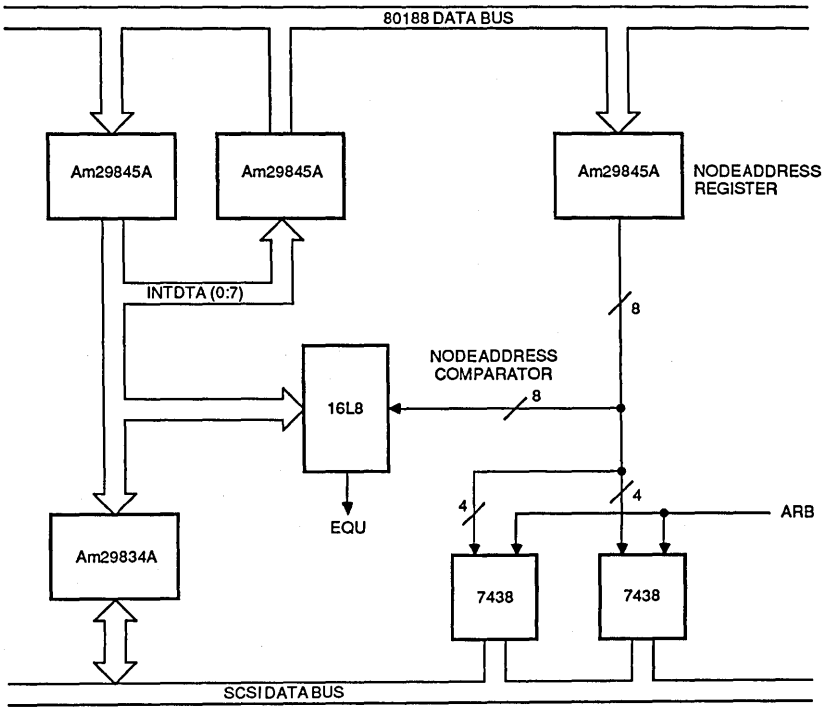
output, to insure processor bus is open long enough for transfer of SCSI data to main memory, and selects BSYIN as CC test input. The FPC waits for SCSI to assert BSYIN before proceeding. BSYIN indicates that the disk is using the SCSI bus. At this time, ARDY can be deasserted, since the data byte is in main memory, and FPC can return to idle at point of exit.

The IDLE Loop then conditionally tests the signal DREQ. If DREQ is asserted, then a jump to the DMAXFER routine takes place. DREQ stands for disk request for data. This signal is generated by SCSI during data transfer, write to or read from disk, as the handshake with acknowledge (ACK) from the FPC. Detecting DREQ being asserted causes the FPC to begin single byte DMA transfer to/from main memory.

First, the FPC asserts DDREQ (disk DMA request) on DMA Request Channel 0 (DRQ0) as an input to Processor (80188). The processor acknowledges this DMA request by asserting DDACK (disk DMA acknowledge) which is an input to the CCM. DDACK is the PCS1 (programmable chip select #1) from the processor. PCS1 is qualified (gated) with DEN, also from the processor, to enable the SCSI transceiver onto the internal 8-bit data bus. Direction of this transceiver is controlled by the signal "I/O" from the SCSI control bus.

After detecting DDACK asserted, FPC then deasserts DDREQ output, asserts output ARDY (to extend 80188 DMA bus cycle) and sets output to LADDR (addressable latch) which asserts DACK (disk acknowledge). DACK is asserted to SCSI (through LADDR) to continue the data byte transfer handshake (refer to SCSI timing diagram Figures in Appendix C). The CCM is selected for DREQ input. After DREQ is again asserted by SCSI, the transfer is complete. DACK and ARDY are deasserted by the FPC and flow returns to idle loop. This DMA transfer routine is used for both writes to and reads from SCSI since the only difference in timing signals is the I/O directional signal which is controlled by SCSI.

QIC-02 Interface. The next conditional jump instruction tests TACK (tape QIC-02 acknowledge). TACK from QIC-02 is the



06591A 9-10

Figure 9-10. SCSI Advanced Features Upgrade

handshake signal used with XFER from FPC to transfer data (see QIC-02 timing diagrams in Appendix C). With TACK asserted, a jump to RDXFER (read transfer from tape) takes place. All of the QIC-02 processing flow is shown in sheet 2 of Figure 9-7. In a similar fashion to SCSI data transfer, QIC-02 data is a DMA to/from main memory using DMA Request Channel 1 (DREQ1) of the processor. DTREQ is asserted by the FPC (through LADDR) and ARDY is asserted to the processor through CCM. Next is a conditional wait until the processor acknowledges this DMA REQ via DTACK (input to CCM and QIC- 02 Data Bus Transceiver enable). After CC = PASS (i.e. DTACK condition asserted), DTREQ and ARDY outputs are deasserted and the QIC-02 read timing handshake continues with a return to the idle loop.

The next conditional test in the idle loop is for a tape write cycle, indicated by both DTACK and DTREQ being asserted. The WRXFER routine shown in Figure 9-7 matches QIC-02 timing requirements as discussed in Appendix C. The flowcharts for FPC routines include the tape transfer commands and processor interrupts on tape exception conditions.

QIC-02 requires different timing during tape write, read, command, and for tape rewind, which has been divided into separate FPC routines which are interactive with the processor. It begins a tape access by issuing "set on line" (TPONL) valid command and ends tape access with "clear on line" (TPONL). The microprocessing unit section above discusses this interaction.

```
EQU = INTDTA0 * DEVADR0
    + INTDTA1 * DEVADR1
    + INTDTA2 * DEVADR2
    + INTDTA3 * DEVADR3
    + INTDTA4 * DEVADR4
    + INTDTA5 * DEVADR5
    + INTDTA6 * DEVADR6
    + INTDTA7 * DEVADR7
```

Figure 9-11. Node Address Comparator PAL Device Equation

9.3 ADVANCED FEATURES OF SCSI

This design can be upgraded to include SCSI bus arbitration, initiator reselection and operation as target as well as initiator. These features are required in a multiple initiator, multiple target environment.

The logic shown in Figure 9-10, when added to the original design, accomplishes the above. It also provides the means for transferring commands, status, messages, and target selection information via 80188 programmed I/O transfers. For support of target mode operation, it is necessary to provide SCSI bus drivers and addressable latches for the following SCSI signals: REQ, C/D, I/O, MSG, and SEL (not shown).

SCSI bus node addresses are one bit in length. That is, each node is assigned one of eight possible addresses corresponding to one of the eight SCSI bus data lines. During the SELECT phase of bus operation, a node must only test one bit of the data bus to determine if it is being selected. Similarly, during the ARBITRATION phase, the node that is asserting the highest bit on the data bus "wins" control of the bus.

Before allowing SELECTION or ARBITRATION, the 80188 must first load the SCSI "Node Address Register". This register is used as a mask register to determine which bit of the SCSI data bus will be tested during SELECT/RESELECT and which bit will be asserted by this node during the ARBITRATION phase.

9.3.1 Selection (Target reselecting Initiator / selection as Target)

The SCSI bus SEL must now be tested in the Am29PL141's idle loop. If asserted, the Am29PL141 tests the SCSI bus "address compare bit - EQU" (16L8 shown in Figure 9-11) and the SCSI bus BSY signal. If this SCSI node is being addressed and BSY is not asserted; then, the Am29PL141 branches to a routine that will monitor SCSI BSY; else, it returns to its idle loop. To monitor BSY, the Am29PL141 uses one of its internal counters to "time out" a 400 nsec bus free period and then retests SCSI BSY. If the bus is still free, this node is being SELECTED/ RESELECTED and the Am29PL141 will interrupt the 80188 which would then take the necessary action. If the bus is not free, the Am29PL141 returns to its idle loop. The 80188 interrupt handler should test the status of SEL and the "address compare bit" to determine that this is a SELECT/RESELECT interrupt.

9.3.2 Arbitration

To initate the ARBITRATION cycle, the 80188 issues a command to the Am29PL141 to set an "arbitration request flip-flop ARBRQ". This is another addressable latch bit controlled by the Am29PL141 and subsequently monitored in the Am29PL141's idle loop. If the ARBRQ bit is set, the Am29PL141 will then test SCSI BSY, and if asserted, the Am29PL141 returns to its idle loop. If ARBRQ is asserted and the SCSI bus is not busy, the Am29PL141 will interrupt the 80188, assert the address for this node onto the SCSI bus, assert BSY and begin monitoring SCSI SEL. The address for this node is asserted onto the SCSI bus via the 7438s and a new control bit "ARB". (See Figure 9-10.)

The Am29PL141 will now continuously monitor SCSI SEL and the ARBRQ signal. The asserting of SEL during the arbitration process indicates that another SCSI device has assumed control of the bus and this node should abort the arbitration process. The assertion of SEL causes an "arbitration failed flip-flop" to be set by the Am29PL141. This bit would be added to the status bits readable by the 80188. Also, the deassertion of ARBRQ indicates that the 80188 has terminated the arbitration process. In either case, the Am29PL141 will deassert BSY, remove this node's address from the bus, and return to its idle loop.

The 80188 interrupt processing routine is responsible for reading the SCSI data bus and determining whether this node is the highest currently requesting the bus. If this node has lost the arbitration process, ARBRQ should be deasserted to allow the Am29PL141 to return to its idle loop and then reasserted to begin the process again. If this node appears to have won the arbitration process, the interrupt handler should first check the "arbitration failed flip-flop" before entering the SELECTION phase. This final check is required to insure no other device issued a SEL while the 80188 was responding to the interrupt.

9.4 SUMMARY

This design solves the problem of interfacing older generation tape drives (QIC-02) to modern computer peripherals on the SCSI bus.

The use of the Fuse Programmable Controller and two programmable array logic devices (AmPAL22V10s), allows the implementation of this complex controller with minimum component count, off the shelf standard parts, (see Figure 9-12) and is reconfigurable/upgradable through reprogramming. This design should also give insight into the versatility of the FPC and ease of using this device for new designs.

PARTS LIST

DEVICE	DESCRIPTION	QUANTITY
Am29PL141	Fuse Programmable Controller	1
80188-1	10MHz, 8-bit Microprocessor	1
Am2947	Octal Bidirectional Transceiver	1
Am29843A	9-bit Latch, Non-Inverting	2
Am2958	Octal Buffer, Inverting	2
AmPAL22V10	24-pin Programmable Array Logic	2
Am2950A	8-bit I/O Port with Flags	1
Am29834A	Parity Bus Transceiver, Inverting	1
Am29864	9-bit Transceiver, Inverting	1
Am29828A	10-bit Buffer/Driver, Inverting	1
7438	Open-Collector Drive	2
Am29827A	10-bit Buffer/Drive, Non-Inverting	1
Am27512DC	512K-bit UV EPROM (250 ns)	1
*AmPAL16L8A	20-pin Programmable Array Logic	1

*Use for the five 2-input "OR" gates and for the one 2-input "AND" gate.

Figure 9-12. SCSI and QIC-02 Controller Parts List

CHAPTER 10

HIGH SPEED DMA CONTROLLER USING Am29PL141

10.1 SYSTEM OVERVIEW

In this application, the Am29PL141 Fuse Programmable Controller (FPC) is used to control two hardware blocks that are sequenced at a rate greater than 10 MHz. This application illustrates the power and flexibility of the Am29PL141 in distributed control applications.

The subsystem controlled by the FPC is just a small part of a large computer system. From the viewpoint of the main central processing unit (CPU), this subsystem is an asynchronous peripheral. The peripheral's function is to control a direct memory access (DMA) channel. This channel links the main CPU's memory to a digital signal processor's (DSP) memory. Figure10-1 shows the various hardware blocks which comprise the DMA channel interface. All operations are initiated by the main CPU. Once a command is passed to the subsystem, the main CPU is free to do other tasks. The DMA interface signals the completion of a task by generating an interrupt in the main CPU. A typical command consists of transferring data (totally under the control of the Am29PL141) and/or processing data (controlled by the DSP engine and the Am29PL141).

The overall system can be viewed as a digital signal processor (DSP). It performs high speed data acquisition, digitizing several incoming analog channels. The processor utilizes DSP techniques to modify and/or extract information from this data, and outputs results which are converted back to analog signals.

By their nature, many DSP algorithms operate on blocks of Data. In this particular application, the incoming channels consist of various speech signals. After digitalization, the speech bandwidth is compressed using linear predictive coding (LPC) techniques. A 64 kbit/sec channel is compressed to a 2.4 kbit/sec data stream using LPC. Six compressed input channels are multiplexed over one serial link. Simultaneously, the processor receives a multiplexed LPC data stream. It demultiplexes this data and expands the compressed data resulting in analog speech output channels.

Real time constraints mandate a high speed DMA controller to orchestrate the filling and emptying of the LPC data RAM. Incoming channels of raw speech data are stored in this RAM. Once available, the processor invokes an analysis routine that extracts the LPC parameters. This parametric information is multiplexed and transmitted over one serial link. In the other direction, received LPC parameters are demultiplexed. A synthesis routine is then invoked which reconstructs the speech signals. These reconstructed speech waveforms are stored in the data RAM. The Am29PL141 not only controls the DMA channel, but also performs a sequencing function assisting the subsystem's DSP engine.

The following sections describe the CPU-FPC interface, the FPC output lines, the use of 27S18 and Am2940 for address generation, and finally the microprogram for this application. A more complete discussion of the Am29PL141 FPC is given in Chapter 1 and Appendix F. Chapter 2 gives more detail about writing the microprogram source code.

10.2 CPU-FPC INTERFACE

Whenever the CPU desires service from the DSP subsystem, it issues a command by placing it in a 5-bit instruction register. This register's outputs are available to the FPC as T[4:0]. The CPU sets the valid instruction flip flop to indicate the presence of a new command. The flip flop output is connected to the FPC's CC test input. While idle, the FPC interrogates this flip flop. When a new command is detected, the FPC commences execution of the instruction. Upon completion, the valid instruction flip flop is cleared (using P[11]), and a status bit is output to the CPU. Data passes between the main CPU data bus and the DSP data bus via a specialized 16 bit bi-directional I/O port. In addition to buffering data during transfers, the I/O port is used to initialize the DSP data RAM.

There are actually 14 different instructions represented in bits T[3:0]. T[4] is used to tell the DSP engine to perform calculations with the DMA interface generating the addresses.

Three groups of CPU commands are defined:

1. Data Transfer In (to the DSP memory) – 6
2. Data Transfer Out (from the DSP memory) – 7
3. Data Memory Initialize – 1

The number following each group name denotes the number of instructions within that group.

Any instruction in the Data Transfer In group can additionally have T[4] as a qualifier. When T[4] is negated, the DMA interface only transfers data in to the DSP memory. When T[4] is asserted, the DMA interface serves as the address generator for the DSP engine for a particular task after the data transfer is complete. By reexamining the CPU command, the FPC determines how many addresses it needs to generate for the task.

Instruction decoding is a simple task in the Am29PL141 using its multiway branch instruction. In this application T[3:0] are masked and a branch to one of sixteen locations is taken as determined by the pattern present on T[3:0]. Subsequent paths taken are derived from this multiway branch.

10.3 Am29PL141 CONTROLLER

At the heart of the DMA interface is the Am29PL141. Once the CPU passes a command, the FPC takes over. All data transfer operations are under its control. When a new instruction is detected, the 29PL141 decodes it by reading it in on its T0–T4 test inputs. The DONE output of the Am2940 is connected to the FPC T5 test input for signaling the completion of an address sequence. When an input instruction is decoded, control branches to the appropriate control sequence.

A 64 x 32 bit PROM resides on the Am29PL141. The upper 16 bits of each word are used to control the on-board sequencer. The functions of these bits are defined by AMD and are not alterable by the user. The lower 16 bits of each word are brought out through a pipeline register as output lines and are user-defined (P15–P0). Appendix F, the Data Sheet, defines the microinstruction word in detail.

The control data that appears at the outputs (P[00:15]) of the FPC depends on the type of instruction. Five bits (P[00:04]) are used as an address to a 32 x 8 lookup PROM. Four bits (P[06:09]) provide instructions and control to an Am2940 high speed DMA address generator. Two bits (P[10], P[12]) control the specialized bidirectional I/O port between the two processor

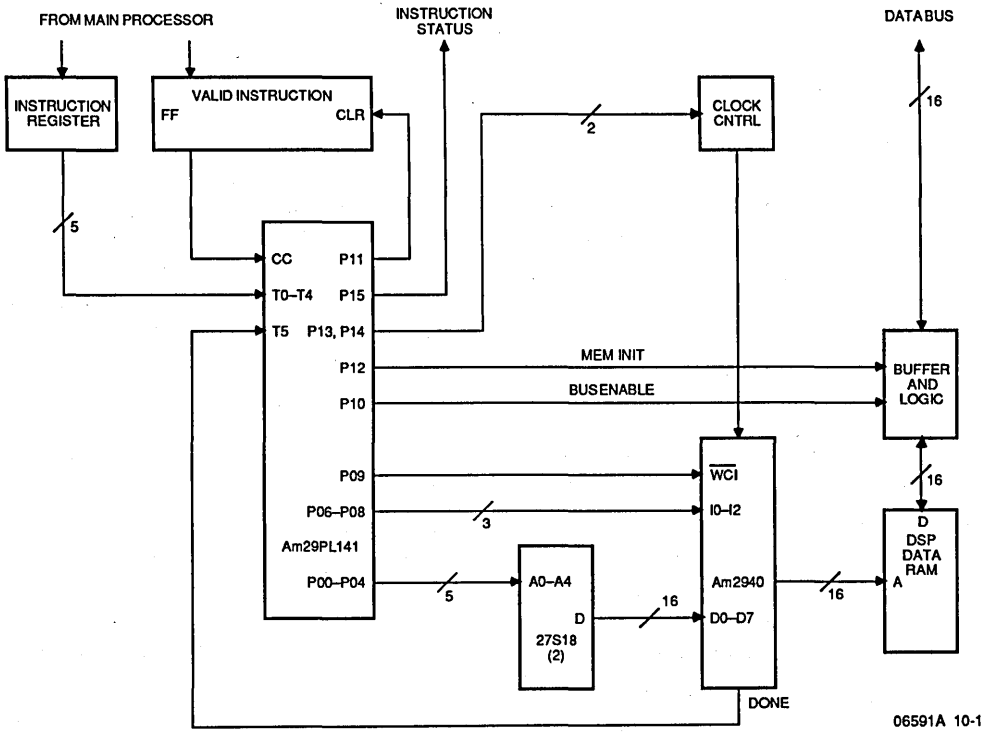


Figure 10-1. DMA Channel Interface

data buses. Finally, two bits (P[13], P[14]) are used to control the clock source to the Am2940 address generator. P[15] signals the main CPU when the execution of a command is complete.

Figure 10-2 illustrates the assignment of the 16 Am29PL141 output lines. These output lines are controlled by the FPC microprogram instructions. One-half of each microinstruction word is used to specify these outputs. All but one of these lines are used in this application. These 16 output lines are grouped into eight fields of varying widths. The specifics of each field, the field width, and the type of micro-operations performed, are as follows:

Prom Address Control

The 5 bit field formed by P[4:0] is named A[4:0]. After a CPU command is decoded, the FPC determines which block of data RAM is to be accessed and its length. The starting address of each block and its length are stored in the look-up PROMs. A[4:0] provide the addresses to the lookup PROMs for each new DMA operation.

DMA Address Generator Control

P[8:6] form a 3 bit field named I[2:0]. These bits are the instructions for the Am2940 address generator. Operations performed by the field include reading and writing various data and control registers on the Am2940.

DMA Count Control

P[9] is a one bit field named CNT wired to the ACI and WCI inputs of the Am2940. The signal enables the counting operation of the address generator. This effectively provides clock control in addition to the external clock circuitry.

Data Bus Interface Control

Bits P[10] and P[12] form two one-bit fields for this function. P[10] is named BEN and controls data transfers between the two CPU data buses. When it is asserted, transfers are allowed. P[12] is named ZEN (Zero Enable). When asserted, it overrides BEN for transfers into the DSP data memory and instead places zeroes on the data bus. This feature is useful for initialization in certain tasks. By having the DMA controller provide this function, the DSP is offloaded and subsequently has more time for performing calculations.

Instruction Status

P[11] and P[15] form two one-bit fields used in conjunction with the CPU instruction interface. P[11] is named CLR. This bit serves as the clear signal to the valid instruction flip flop. This flip flop can only be set by the main CPU and reset by the DMA controller. When an instruction is completed by the DMA controller, it resets this flip flop. The FPC idles until the main CPU sets this flip flop indicating the presence of a new instruction in the instruction register.

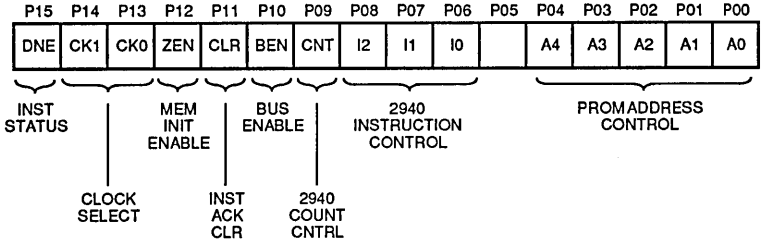
P[15] is named DNE and is sent back to the main CPU. When asserted by the FPC it indicates that the DMA subsystem has completed the execution of a command and is awaiting a new one.

Clock Control

P[14:13] form a two bit field named CK[1:0]. These bits control the source of the clock to the Am2940s. Three selections are possible: 1) System clock; 2) System clock shifted by 180°; and 3) clock inhibit.

10.4 ADDRESS GENERATION

Several channels of data are stored in the DSP data RAM. For each channel, the DMA controller must input to and/or output from the proper section of the memory. Generation of the appropriate addresses is handled by two Am27S18SA PROMs and two Am2940 address generators.



06591A 10-2

Figure 10-2. Format of User Output Portion of Am29PL141 Microcode

The FPC determines a starting address and a block length from a decoded instruction. The actual values of this data are stored in the Am27S18SA lookup PROMs. Two five-bit addresses, representing the starting address and block length are presented to the PROMs. The data outputs of the PROM are routed to the Am2940s on their data inputs (D0–D7) and loaded into the appropriate registers. Once initialized with these "seed" values, the Am2940s provide sequential addresses to the DSP data RAM until the word count expires. The DONE signal from the Am2940s alerts the FPC to this condition.

In addition to providing DMA addresses, this section of the hardware generates addresses for the DSP for certain processing steps that are time critical. Some sections of the LPC algorithm sequentially step through the memory block repeatedly. For these tasks, the FPC keeps track of how many passes are required and issues control data to the address generators. Basically it performs dummy DMA cycles where addresses are generated but no data passes through the data bus interface.

10.5 FPC MICROCODE

Figure 10-3 is the flowchart of the code implemented for this application. A total of 45 words are used. This leaves ample room for future modifications to the interface. Of the 45 locations used, 30 are used for instruction decoding. However, while the FPC is decoding an instruction, its control outputs are simultaneously loading values

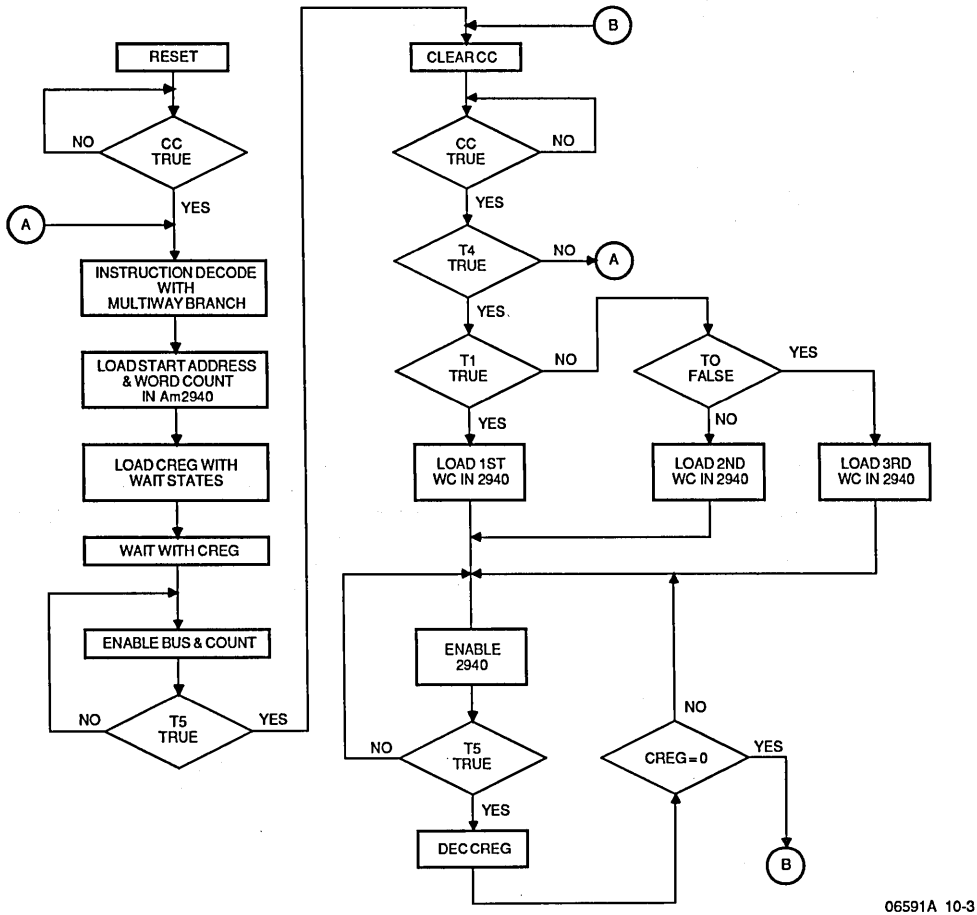


Figure 10-3. DMA Controller Program Flow Diagram

into the Am2940s. This parallel operation allows the data transfers to take place with a minimum of overhead. By the time the instruction is decoded, the Am2940 data and control registers are loaded and ready to start the transfer operation.

After some wait states are executed the data transfer commences. When finished, T[4] is tested. If asserted the FPC goes back and looks at T[3:0] to determine how many passes it must make through the data for the DSP engine. It then commands the Am2940s to start the dummy DMA cycles and runs until its pass count expires. A pass count is easily implemented using the C Register on board the Am29PL141. Between each pass the Am2940s are reinitialized to point at the start of a data block. When all passes are complete, the CPU is notified, and the FPC waits for the next instruction.

Figure 10-4 is a listing of the microcode described above. It is written using an assembler written specifically for the Am29PL141 by AMD. This software runs on an IBM PC/XT and is available gratis to any designer using the Am29PL141. Most of the code in this application was debugged using a companion simulator also available from AMD. Only real time timing aspects could not be evaluated. Having this software available makes the design engineer's job easier by minimizing the amount of time is spent translating concept to PROM data for the FPC.

```
"                    A HIGH SPEED DMA CONTROLLER                    "

device (pl141)
default = 1 ;
define

" The following mnemonics are the names assigned to the micro
  operations in the eight different fields defined for P(15:0)

  FIELD NAME = DNE
                                                                     "
        DONE     = 0000#H
        NDONE    = 8000#H

" FIELD NAME = CS(2:0)
                                                                     "
        CLK1     = 0000#H
        CLK2     = 2000#H
        NOCLK    = 6000#H

" FIELD NAME = ZEN
                                                                     "
        IMEM     = 0000#H
        NOIMEM   = 1000#H

" FIELD NAME = ICR
                                                                     "
        CLRINST  = 0000#H
        NOCLR    = 0800#H

" FIELD NAME = BEN
                                                                     "
        BUSON    = 0000#H
        BUSOFF   = 0400#H

" FIELD NAME = CNT
                                                                     "
        CNTON    = 0000#H
        CNTOFF   = 0200#H

" FIELD NAME = I(2:0)
                                                                     "
        WRCR     = 0000#H
        REIN     = 0100#H
        LDAD     = 0140#H
        LDWC     = 0180#H
        ENCT     = 01C0#H
```

Figure 10-4. DMA Controller Source Program Listing (Sheet 1 of 4)

```
" FIELD NAME = A(4:0)                                                    "

        ADD0    = 0000#H
        ADD1    = 0001#H
        ADD2    = 0002#H
        ADD3    = 0003#H
        ADD4    = 0004#H
        ADD5    = 0005#H
        ADD6    = 0006#H
        WC0     = 0008#H
        WC1     = 0009#H
        WC2     = 000A#H
        WC3     = 000B#H
        WC4     = 000C#H
        WC5     = 000D#H
        WC6     = 000E#H
        WC7     = 000F#H;

begin

" The first 16 locations form the branch table for decoding the
  instruction bits present on T(3:0)                                     "

ZERO:   NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD0,
          IF (CC) THEN GOTO PL(DTI1);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD1,
          IF (CC) THEN GOTO PL(DTI2);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD2,
          IF (CC) THEN GOTO PL(DTI3);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD3,
          IF (CC) THEN GOTO PL(DTI2);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD4,
          IF (CC) THEN GOTO PL(DTI3);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD5,
          IF (CC) THEN GOTO PL(DTI4);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD0,
          IF (CC) THEN GOTO PL(DTO1);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD1,
          IF (CC) THEN GOTO PL(DTO2);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD2,
          IF (CC) THEN GOTO PL(DTO3);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD3,
          IF (CC) THEN GOTO PL(DTO4);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD4,
          IF (CC) THEN GOTO PL(DTO1);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD5,
          IF (CC) THEN GOTO PL(DTO2);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD1,
          IF (CC) THEN GOTO PL(DTO3);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+REIN+001F#H,
          IF (CC) THEN GOTO PL(RESET);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+REIN+001F#H,
          IF (CC) THEN GOTO PL(RESET);
        NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+001F#H,
          IF (CC) THEN GOTO PL(MEMINIT);
```

Figure 10-4. DMA Controller Source Program Listing (Sheet 2 of 4)

```
" The next 4 instructions have identical internal control but different
  outputs on P(15:0).  They are used for instructions in the DATA  TRANS-
  FER IN (DTI) group.  They are also part of the instruction decoding."

DTI1:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC0,
           IF (CC) THEN GOTO PL(DTIWAIT);
DTI2:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC1,
           IF (CC) THEN GOTO PL(DTIWAIT);
DTI3:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC2,
           IF (CC) THEN GOTO PL(DTIWAIT);
DTI4:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC3,
           IF (CC) THEN GOTO PL(DTIWAIT);

" The next 4 instructions have identical internal control but different
  outputs on P(15:0).  They are used for instructions in the DATA TRANS-
  FER IN (DTI) group.  They are also part of the instruction decoding."

DTO1:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC4,
           IF (CC) THEN GOTO PL(DTOWAIT);
DTO2:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC5,
           IF (CC) THEN GOTO PL(DTOWAIT);
DTO3:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC6,
           IF (CC) THEN GOTO PL(DTOWAIT);
DTO4:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC7,
           IF (CC) THEN GOTO PL(DTOWAIT);

" This instruction is executed for the DATA MEMORY INITIALIZE (DMI) group"

MEMINIT: NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC7,
           IF (CC) THEN GOTO PL(ZWAIT);

" Program FPC for DTI wait states using the CREG                    "

DTIWAIT: NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN LOAD PL(4);
         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN GOTO PL(WAIT1);

" Program FPC for DTO wait states using the CREG                    "

DTOWAIT: NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN LOAD PL(6);
WAIT1:   NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           WHILE (CREG <> 0) LOOP TO PL (WAIT1);
         NDONE+CLK1+NOIMEM+NOCLR+BUSON+CNTON+ENCT+001F#H,
           IF (T5) THEN GOTO PL(CLEARCC) ELSE WAIT;

" Program FPC for MEMORY INITIALIZE function                        "

ZWAIT:   NDONE+CLK1+IMEM+NOCLR+BUSOFF+CNTON+ENCT+001F#H,
           IF (T5) THEN GOTO PL(CLEARCC) ELSE WAIT;

" Clear VALID INSTRUCTION flip flop (CC input to FPC)               "
```

Figure 10-4. DMA Controller Source Program Listing (Sheet 3 of 4)

```
CLEARCC: DONE+CLK1+NOIMEM+CLRINST+BUSOFF+CNTOFF+ENCT+001F#H,
           CONTINUE;

" Check for CC indicating the presence of an instruction to crack "

         DONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN GOTO PL(GENADD) ELSE WAIT;

" Check for T4 active.  If so, additional processing is required."

GENADD:  NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+REIN+001F#H,
           IF (NOT T4) THEN GOTO PL(62#D);

" If T4 is asserted, then the DMA controller assists the DSP engine by
  generating sequential addresses without passing data through the
  data bus interface.  Different pass counts are loaded depending on
  T1 and T0 values.                                                    "

         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDAD+ADD6,
           IF (T1) THEN GOTO PL(LPC1);
         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (NOT T0) THEN GOTO PL(LPC2);
LPC3:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC4,
           IF (CC) THEN LOAD PL(12);
         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN GOTO PL(WAIT2);
LPC1:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC0,
           IF (CC) THEN LOAD PL(10);
         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN GOTO PL(WAIT2);
LPC2:    NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+LDWC+WC2,
           IF (CC) THEN LOAD PL(8);
WAIT2:   NDONE+CLK2+NOIMEM+NOCLR+BUSOFF+CNTON+ENCT+001F#H,
           IF (T5) THEN GOTO PL(NXTPASS) ELSE WAIT;

" Decrement pass count                                                 "


NXTPASS: NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           WHILE (CREG <> 0) LOOP TO PL(WAIT2);

" When all passes are finished, clear CC and wait for next inst."

         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+REIN+001F#H,
           IF (CC) THEN GOTO PL(CLEARCC);

" This multiway branch is the first step of instruction cracking "
         .ORG       62#D

         NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+ENCT+001F#H,
           IF (CC) THEN GOTO TM(001111#B);
RESET:   NDONE+CLK1+NOIMEM+NOCLR+BUSOFF+CNTOFF+WRCR+001F#H,
           IF (CC) THEN GOTO PL(62#D);

END.
```

Figure 10-4. DMA Controller Source Program Listing (Sheet 4 of 4)

APPENDIX A

JEDEC STANDARD No.3

The fuse map generated by the Am29PL141 Assembler adheres to the JEDEC standard No. 3 (October 1983) which is a standard data transfer format between a data preparation system and a programmable logic device programmer.

The information to be sent to the device programmer is divided into the following categories:

1. The design specification identifier
2. The device to be programmed
3. Fuse links that must be blown to implement the design
4. Information to perform a structured functional test
5. Other information (e.g.,sumcheck)

A transmission must consist of the following legal characters. Any other characters present in the transmission file may result in invalid operation.

STX	02 hex	start of text
ETX	03 hex	end of text
LF	0A hex	line feed
CR	0D hex	carriage return
all printable characters	20 hex to 7E hex inclusive	

The Assembler forms the transmission file by putting the STX character at the beginning of the file, followed by the fuse link information, the fuse checksum, the ETX character, and the transmission sumcheck. An example Assembler transmission (fuse map) file is:

```
<STX>F1*
L0000 0 10010 1 111 111111 1111111111111000 *
C02EF*
<ETX>0A94
```

Fuse Link Information

Each device fuse link is assigned a decimal number. Each numbered fuse can have two possible states: a Zero specifying a low-resistance (unblown) link and a One specifying a high-resistance (blown) link.

Fuse information is presented in three fields: F, L and C.

F: This field specifies the state of the unspecified fuses in the logic device. This field corresponds to the DEFAULT section in the program source file. The default for this field is 'F0', all unspecified fuses unblown.

L: Each numbered link is addressed by the 'L' field. The L is immediately followed by a variable length decimal number indicating address of the first link in the following string of data. The string of data can be any convenient length terminated by an '*'. In the Assembler each string is 32 characters long.

C: This is the checksum field for the link information. It is computed by performing a 16-bit unsigned addition of 8-bit words formed from all the fuse link states specified in the file.

The 8-bit words are formed as in the following diagram:

Example: Checksum Computation

```
<STX>F1*
L0000 0 10010 1 111 111111 1111111111111000 *
C02EF*
<ETX>0A94
```

```
link   7        0
       1101 0010  →  D 2 hex
       1111 1111     F F hex
       1111 1111     F F hex
       0001 1111     1 F hex
       ---------------------
                     0 2 E F hex = checksum
```

Note:

If the number of fuse links is not a multiple of 8, then the last word will be formed by setting Zeroes for all the bit locations more significant than the last link. The 16-bit checksum is specified as a C followed by 4 hex characters and an '*'.

OTHER INFORMATION

The transmission format is ended by an ETX character followed by the sumcheck. The sumcheck is the 16-bit unsigned addition of the ASCII values of all the characters in the transmission file between and including STX and ETX. The parity bit is excluded in this calculation.

APPENDIX B

ASSEMBLER ERROR MESSAGES

This appendix lists the Am29PL141 Assembler diagnostic error messages alphabetically. Each message is also numbered as a part of the message. Each message is followed by an explanation of the message and suggested actions to remove the error from subsequent runs of the assembler.

Errors generated during assembly are prefaced by the error number and the source line where the error occurred.

The symbol *** is used to indicate a keyword or user-defined string which varies depending on the context of the error message.

There are two error types: warning and fatal. Warnings are displayed and assembly continues. Fatal errors terminate assembly.

When errors occur in sequential lines, it normally indicates a punctuation error on the line preceding the error. Punctuation should always be examined near the error that was detected.

ERR 1 29PL141 Assembler : cannot open ***
Fatal: The assembler cannot open the filename specified in the command line.
User Action: Make sure the file exists or that there is enough room on the disk

ERR 2 * database file is incorrect**
Fatal: The device database file *** may have been accidentally modified, or cannot be found on the same disk drive as the Assembler
User Action: Put the device database file on the same working disk

ERR 3 * in line *** has not been defined**
Warning: The label *** is not defined
User Action: Define the label

ERR 4 * is not supported by this Assembler**
Fatal: The device *** does not have a database file on the same drive as the Assembler
User Action: Put the database file on the same disk as the assembler

ERR 5 Assembler : illegal option ***
Fatal: Unrecognized option
User Action: Use recognized options such as '-O' for specifying an output file

ERR 6 assembler needs a valid device name
Fatal: This assembler recognizes only 'Am29PL141'
User Action: Specify the part name Am29PL141

ERR 7 assembler needs an opcode
Warning: Specify one of the opcodes or commands listed in Section 2.3.5 Statement Formats
User Action: Check for proper statement syntax

ERR 8 assign a number/name with the '=' sign
Warning: Use the '=' sign to separate identifiers and their values
User Action: Put a '=' sign

ERR 9 beyond addressing range of device
Warning: A memory reference has been made beyond the range of the device
User Action: Check statement label value

ERR 10 beyond the range of the machine
Warning: The number specified is too large
User Action: Check the value being used

ERR 11 cannot open the database
Fatal: The database file Am29PL141 is missing
User Action: Put the database file on the same disk with the assembler

ERR 12 check condition field
Warning: Check the condition field of the statement
User Action: Use a valid condition test expression

ERR 13 Check the database
Fatal: The database has been modified or is not in the correct format
User Action: Copy database file from master diskette

ERR 14 close the data field with ')'
Warning: Mismatched parentheses
User Action: Match the opening '(' with ')' in PL(data) or TM(data)

ERR 15 compare CREG with 0 only
Warning: CREG is compared with the value 0 only
User Action: Check CREG test condition

ERR 16 compare test condition with a binary number
Warning: Test conditions are compared with '0' or '1' only
User Action: Check test condition

ERR 17 compare TM to PL
Warning: TM must follow CMP in a compare statement
User Action: Check format of compare statement

ERR 18 default fuse map values should only be binary numbers
Warning: DEFAULT should be equated with a binary value of '0' or '1' only
User Action: Specify only '0' or '1'

ERR 19 default output value exceeds control output limits
Warning: The default output value defined exceeds device limits
User Action: Check the value of the default output

ERR 20 enclose the data field in '('
Warning: Use the opening parenthesis '(' as in PL(data) or TM(data)
User Action: Put a '('

ERR 21 enclose the device name in parenthesis
Warning: The device name must be enclosed in parenthesis
User Action: Use '(' and ')'

ERR 22 error in cleaning the STACK list
Warning: Invalid control output expression
User Action: Use '(', ')', '+' and '*' for logical expressions

ERR 23 error in cleaning the OPERATOR list
Warning: Invalid control output expression
User Action: Check that the logical operators '+' and '*' have operands

ERR 24 error in STACK PUSH
Warning: Will not accept this character as a logical operator
User Action: Check format of logical expression

ERR 25 error in STK_EVAL function
Warning: Invalid control output expression
User Action: Check format of logical expression

ERR 26 field limits exceeded for DATA field
Warning: The number specified is too large; it must be in the range of 0 to 63 decimal for the Am29PL141
User Action: Check value in DATA field

ERR 27 field limits exceeded for OUTPUT field
Warning: The number is too large; it must be in the range 0 to 216 for the Am29PL141
User Action: Check value in OUTPUT field

ERR 28 field limits have been exceeded for TEST field
Warning: The test condition specified does not exist; only 8 test conditions exist for the Am29PL141
User Action: Check the numerical value of the test condition

ERR 29 looking for '=' sign
Warning: Use a '=' in defining constants in the DEFINE section and in setting up test conditions
User Action: Put a '='

ERR 30 looking for ';'
Warning: End each program section (e.g., DEFAULT or SSR) or statement with a ';'
User Action: Put a ';' to terminate this program section

ERR 31 looking for ')'
Warning: Mismatched parentheses
User Action: Put a ')' to close the test condition or data field

ERR 32 looking for a ')' to close the condition
Warning:: Mismatched parentheses
User Action: Put a ')' to close the test condition

ERR 33 looking for a binary number
Warning: Specify a binary number for this section or test condition by putting the base '#b' after the number
User Action: Use the binary radix symbol '#b'

ERR 34 looking for a constant or number after OE/OD
Warning: OE or OD must be followed by a constant defined in the DEFINE section or a valid number
User Action: Put a predefined name or number after OE or OD

ERR 35 looking for TM
Warning: CMP must be followed by TM
User Action: Check format of CMP statement in your file

ERR 36 looking for the keyword BEGIN
Warning: The BEGIN-END block follows the DEFINE, DEFAULT_OUTPUT or TEST_CONDITION section
User Action: Use the keyword BEGIN

ERR 37 looking for the keyword END
Warning: Unexpected end of file
User Action: Put an 'END.' to terminate the program

ERR 38 looking for the keyword DEVICE
Warning: The first keyword must be the DEVICE
User Action: Start the source file with the keyword DEVICE

ERR 39 missing input filename
Fatal: The option '-I' must be followed by an input filename that already exists
User Action: Create an assembler source file

ERR 40 missing fusemap filename
Warning: The '-O' option needs a valid DOS filename after it to hold the fuse map file generated by the Assembler
User Action: Specify an output file

ERR 41 need a binary number
Warning: Put a binary number here
User Action: Specify the base '#b'

ERR 42 need a binary number (compare CREG with 0 only)
Warning: CREG must be compared with a binary number only
User Action: Check format of test condition

ERR 43 need a label, predefined constant, number or OE / OD
Warning: Incorrect statement format
User Action: Begin a statement with a label, constant, number or the output enable controls OE or OD

ERR 44 need a number or constant to enable
Warning: A variable or constant must follow OE or OD
User Action: Put a predefined name or number after OE or OD

ERR 45 need an opcode here
Warning: Put one of the opcodes or commands specified in Section 2.3.5
User Action: Check statement format

ERR 46 no default condition available
Warning: A default test condition was not specified in TEST_CONDITION
User Action: Specify a test condition

ERR 47 no error filename given
Warning: The option '-E' needs a valid DOS filename to contain the Assembler errors generated
User Action: Specify a valid DOS filename

ERR 48 no such command available for this part
Warning: The statement combination formed does not correspond to a valid opcode mnemonic in this device
User Action: Check the available statement forms

ERR 49 no such condition
Warning: Use existing test conditions for this device
User Action: Use one of the valid test conditions

ERR 50 not equal sign is '<>'
Warning: unknown operator
User Action: Use the '<>' as the 'not equal' sign for the CREG tests

ERR 51 Note : this input is used for diagnostics
Warning: This test condition is being used for SSR diagnostics
User Action: Use a different test condition

ERR 52 OPCODE field limits exceeded
Warning: The opcode specified is not valid for this device
User Action: Check the device database

ERR 53 OUTPUT field can accommodate 16 bits max
Warning: Only 16-bit numbers can be used for control outputs
User Action: Use only 16-bit numbers

ERR 54 OUTPUT field is 16 bits long only
Warning: Only 16-bit numbers can be used for control outputs
User Action: Use only 16-bit numbers

ERR 55 OUTPUT field limits exceeded
Warning: Only 16-bit numbers can be used for control outputs
User Action: Use only 16-bit numbers

ERR 56 OUTPUT limits exceeded
Warning: Only 16-bit numbers can be used for control outputs
User Action: Use only 16-bit numbers

ERR 57 PROM not large enough to hold microprogram
Warning: Too many statements have been defined
User Action: Check and remove redundant states

ERR 58 put a binary number for test comparisons
Warning: Only binary numbers allowed
User Action: Use the binary radix '#b'

ERR 59 put a number or a defined name here
Warning: Syntax error
User Action: Put a valid number or predefined name here

ERR 60 put a constant or a number here
Warning: Syntax error
User Action: Put a valid number or predefined constant here

ERR 61 put a '.' after END to terminate the assembler file
Warning: Unexpected end of file
User Action: Include a '.' after the keyword END

ERR 62 put a ':' for labels or ',' for output
Warning: The punctuation symbols ':' or ',' are necessary to separate sections in each statement
User Action: Use ':' or ','

ERR 63 put a ',' to separate the output section
Warning: The ',' symbol is required here
User Action: Put a ','

ERR 64 put a ';' here
Warning: The ';' symbol is necessary to separate program sections or statements
User Action: Put a ';' as a statement separator

ERR 65 put a name here
Warning: A valid predefined constant is necessary here
User Action: put a predefined name here

ERR 66 put a 'TO' here : loop TO PL
Warning: LOOP must be followed by the keyword 'TO'
User Action: put the keyword "TO"

ERR 67 put an operand between logical operators
Warning: Logical expression is incorrect
User Action: Put an operand between '*' and '+'

ERR 68 put an operand between nested operands
Warning: Logical expression is incorrect
User Action: Put an operand after the '('

ERR 69 put an operand here
Warning: Syntax error
User Action: Match an operand with this logical operator

ERR 70 put an operand or ')' to complete the expression
Warning: Unmatched parenthesis or missing operand
User Action: Check logical expression

ERR 71 put an operator between operands
Warning: Logical operators '*' and '+' cannot follow each other
User Action: Check the logical expression/ equation

ERR 72 put PL , TM , or SREG here
Warning: Incorrect statement syntax
User Action: Put GOTO PL, GOTO TM or GOTO (SREG)

ERR 73 redefinition of label
Warning: Label has been redefined
User Action: Check label names

ERR 74 separate the output section with a ','
Warning: Syntax error
User Action: Put the necessary ',' here

ERR 75 Severe warning : redefinition of PROM location * See source line *****
Warning: PROM location specified more than once
User Action: Check the flow of your microprogram; some statements may have overlapped due to the use of numbers as labels

ERR 76 SOFTWARE error ... see WRITE WORD module
Warning: The program cannot form the PROM word properly
User Action: None

ERR 77 specify the pipeline data field
Warning: Syntax error
User Action: Specify a data field in PL(data)

ERR 78 Statement * not supported in *****
Warning: This statement combination does not correspond to any device mnemonic
User Action: Check the list of available statements

ERR 79 this condition has not been defined
Warning: Undefined test condition
User Action: Pair this identifier with a test condition in the DEFINE section

ERR 80 this is a keyword
Warning: Cannot use this keyword in this context
User Action: Use a different variable name

ERR 81 this is not a binary number
Warning: Not a binary number
User Action: use '#b'

ERR 82 this is not a decimal number
Warning: Not a decimal number
User Action: Use 'd'

ERR 83 this is not a defined output value
Warning: Output value undefined
User Action: Check the DEFINE section

ERR 84 this is not a hexadecimal number
Warning: Not a hexadecimal number
User Action: Use '#h'

ERR 85 this is not an octal number
Warning: Not an octal number
User Action: use '#o'

ERR 86 this is not a valid test condition
Warning: Undefined test condition
User Action: Check the DEFINE section

ERR 87 this name has not been previously defined
Warning: Undefined constant
User Action: Define this name in the DEFINE section

ERR 88 this name is too long, more than 29 characters
Warning: Identifiers and constant can only be 29 characters long
User Action: limit the size of the variables

ERR 89 this variable name has not been defined
Warning: Undefined name
User Action: Define this name in the DEFINE section

ERR 90 too many operators
Warning: The logical equation contains too many operators
User Action: Check the control output logical expression

ERR 91 Unexpected end of file
Warning: Unexpected end of file
User Action: 'END.' was not encountered

ERR 92 Unexpected end of file (close comments in line *)**
Warning: Unexpected end of file
User Action: Check to make sure that all the comments have
matching ""

ERR 93 unmatched parenthesis or missing operand
Warning: Unmatched parenthesis
User Action: Match each parenthesis with its pair

ERR 94 use ';' to separate statements
Warning: No statement separator
User Action: Check the source file for ';' between different statements

ERR 95 use ')' to enclose SREG
Warning: Syntax error
User Action: Close SREG with ')'

ERR 96 use only predefined names or numbers
Warning: Undefined name
User Action: Check that the constant has been defined in the DEFINE section or that a valid number is being used

ERR 97 Warning : NOT has no effect on CREG condition Refer to source line ***
Warning: Any test condition using CREG is not affected by NOT
User Action: Modify the CREG test condition

APPENDIX C

QIC-02 AND SCSI INTERFACE SIGNALS AND TIMING DIAGRAMS

QIC-02 INTERFACE

QIC-02 is an industry standard which defines the interface between a host system and Quarter Inch Cartridge Tape Drives. Read/write commands, status and, of course, data are transmitted over this interface, as depicted in Figure C-1. The bus and control signals between QIC-02 and host are all standard TTL levels. Timing diagrams for this interface are shown in Figures C-2 through C-4. This interface handshake timing is duplicated for the host side by the FPC and two AmPAL22V10s.

ACKNOWLEDGE (ACK) is used with Transfer to transfer data across the interface.

READY (RDY) indicates that the tape drive can accept a command. It is used to handshake the command across the interface. In the write mode, READY indicates that the drive's internal buffer is empty and ready to receive new data. In the read mode, READY indicates the drive buffer can now be accessed by the host.

EXCEPTION (EXP) alerts the host that the execution of a command has been terminated. This may be a normal completion or an interrupt due to a fault (hard errors, write protected, etc.). The response by host must be READ STATUS.

DIRECTION (DIRC) indicates direction of data flow. Signal is used to enable/disable the data bus transceivers in the HOST.

ON-LINE signal is deasserted at the beginning of a read (from tape) or write (to tape) operation.

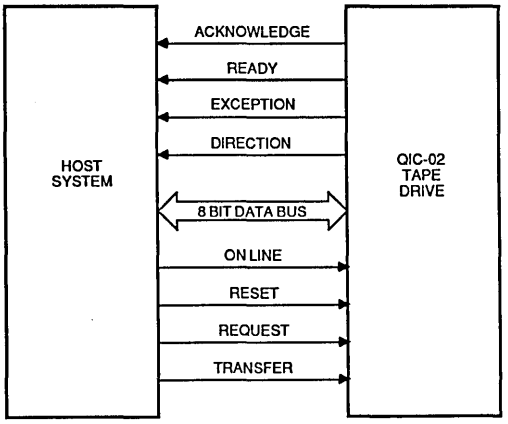
RESET initializes the tape drive. The drive recalibrates the heads to track zero.

REQUEST indicates that a command is on the data bus.

TRANSFER is used with ACKNOWLEDGE to handshake data over the bus, see timing diagram.

SCSI INTERFACE

Small Computer Systems Interface (SCSI) evolved from the disk controller standard developed by Shugart Associates (SASI) in the late 1970s. The SCSI standard was developed by ANSI X3T9.2 subcommittee starting in 1982. SCSI defines an 8-bit parallel bi-directional data bus with parity, plus nine control lines. SCSI protocol allows single or multiple host computers (initiators) to share multiple (expensive) peripherals (targets, i.e. hard disk, floppy disks, etc.), as depicted in Figure C-5. Up to eight Daisy Chained devices can reside on the SCSI bus, with data transfer rates of 4 Mbytes/sec. Synchronous and 1.5 Mbyte/sec. asynchronous. The timing diagrams for the interface are shown in Figures C-6 through C-8.



06591A C-1

Figure C-1. QIC-02 Interface

The interface signals are:

I/O is driven by a target to control the direction of data movement. True indicates input to the initiator.

MSG is drive by a target to indicate "Message Phase". When MSG is asserted, REQ (Request) is also asserted by the target for transfer of data byte indicating the end of the operational phase ("Message").

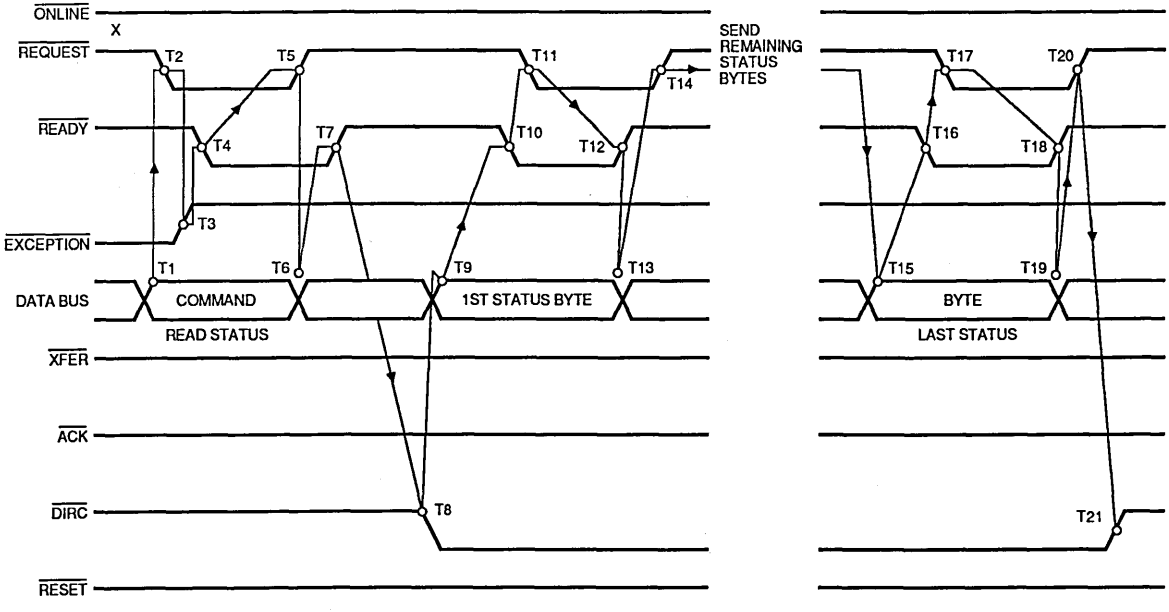
REQ asserted by target indicates that a data byte is to be transferred on the data bus. Data byte is transferred via handshake with ACK (Acknowledge).

ATN (Attention) is driven by an initiator to indicate to target an "attention" condition.

An initiator uses SEL along with appropriate data (address) bits (0-7) being asserted to select a target. Select line is deasserted after the target asserts BSY to acknowledge selection.

RST (Reset) is a pulse asserted by the initiator to stop target's present operation and return same to idle condition.

Data bus and control signals require open collector drivers capable of sinking 48 mA each to support SCSI mode of multiple initiators with multiple targets. SCSI provides for either single ended (6 meter Max. Cable Length) transmission or differential (if a distance up to 25 meters is required).

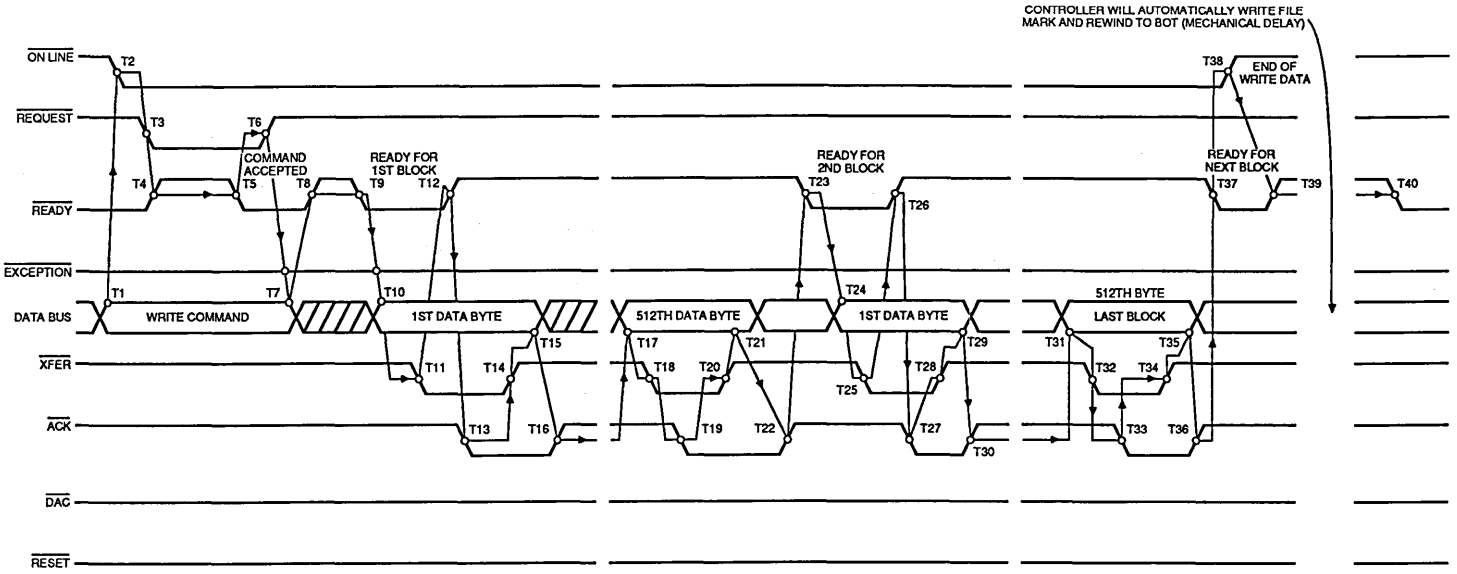


ACTIVITY	CRITICAL TIMING
T1 – HOST COMMAND TO BUS	N/A
T2 – HOST SETS REQUEST	T1–T2>0 µs
T3 – CONTROLLER RESETS EXCEPTION	T3–T4>10 µs
T4 – CONTROLLER SETS READY	20<T2–T4<500 µs *
T5 – HOST RESETS REQUEST	T4–T5>0 µs
T6 – BUS DATA INVALID	T4–T6>0 µs
T7 – CONTROLLER RESETS READY	20<T5–T7<100 µs
T8 – CONTROLLER CHANGES BUS DIRECTION	N/A
T9 – 1ST STATUS BYTE TO BUS	N/A
T10– CONTROLLER SETS READY	T7–T10>20 µ s
T11 – HOST SETS REQUEST	N/A
T12 – CONTROLLER RESETS READY	T11–T12<1 µs
T13 – BUS DATA INVALID	T11–T13>0 µs
T14 – HOST RESETS REQUEST	T11–T14>20 µs
T15 – LAST STATUS BYTE TO BUS	N/A
T16 – SAME AS T10	SAME AS T10
T17 – SAME AS T11	SAME AS T11
T18 – SAME AS T12	SAME AS T12
T19 – SAME AS T13	SAME AS T13
T20 – SAME AS T14	SAME AS T14
T21 – CONTROLLER CHANGES BUS DIRECTION	N/A
T22 – CONTROLLER SETS READY	T20–T21>0 µs
X – DON'T CARE	T21–T22>0 µsN/A
	T11–T12<1 µs

*NOTE: THIS MAY BE > 500 µs UNDER SOME CONDITIONS

06591A C-2

Figure C-2. QIC-02 Read Status Command Timing Diagram

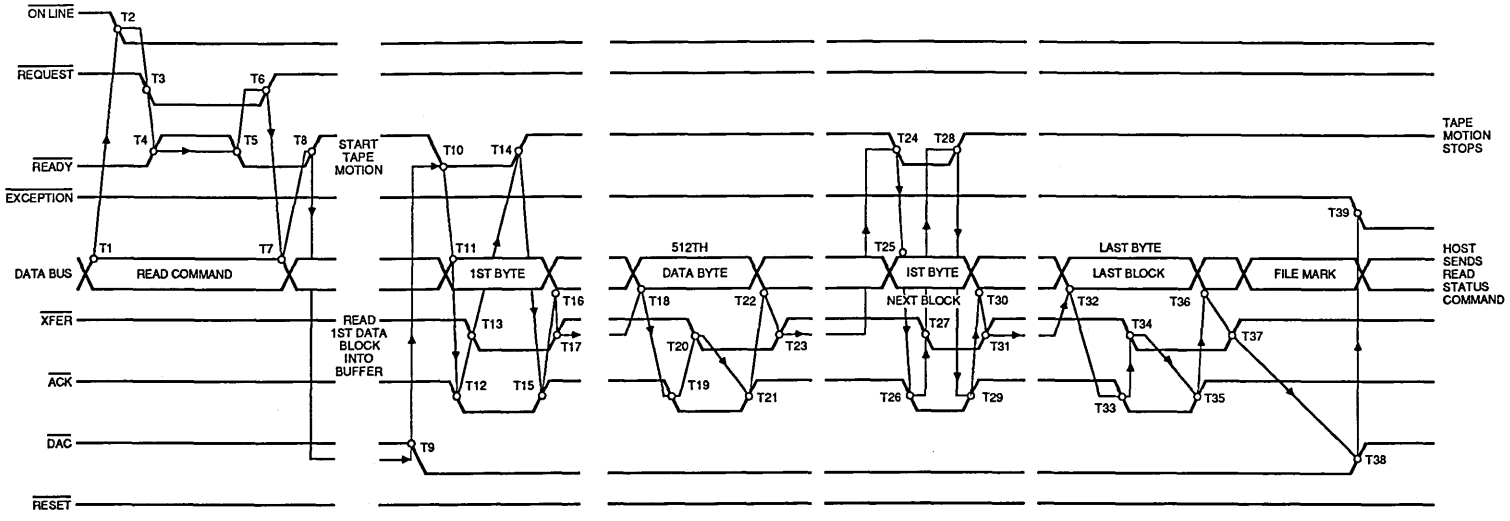


ACTIVITY	CRITICAL TIMING	ACTIVITY	CRITICAL TIMING	ACTIVITY	CRITICAL TIMING
T1 – HOST COMMAND TO BUS	N/A	T15 – BUS DATA INVALID	T13–T15>0 µs	T28 – HOST RESETS XFER	SAME AS T14
T2 – HOST SETS ONLINE	N/A	T16 – CONTROLLER RESETS ACK	0<T14–T16<3 µs	T29 – BUS DATA INVALID	SAME AS T15
T3 – HOST SETS REQUEST	T2–T3>0 µs	T17 – HOST DATA TO BUS	N/A	T30 – CONTROLLER RESETS ACK	SAME AS T16
T4 – CONTROLLER RESETS READY	T3–T4<1 µs	T18 – SAME AS T11	SAME AS T11	T31 – HOST DATA TO BUS	N/A
T5 – CONTROLLER SETS READY	20<T4–T5<500 µs	T19 – SAME AS T13	SAME AS T13	T32 – HOST SETS XFER	SAME AS T18
T6 – HOST RESETS REQUEST	T5–T6 >0 µs	T20 SAME AS T14	SAME AS T14	T33 – CONTROLLER SETS ACK	SAME AS T19
T7 – BUS DATA INVALID	T5–T7>0 µs	T21 – SAME AS T15	SAME AS T15	T34 – HOST RESETS XFER	SAME AS T20
T8 – CONTROLLER RESETS READY	20<T6–T8<100 µs	T22 – SAME AS T16	SAME AS T16	T35 – BUS DATA INVALID	N/A
T9 – CONTROLLER SETS READY	T8–T9>20 µs	T23 – CONTROLLER SETS READY	T22–T23>100 µs	T36 – CONTROLLER RESETS ACK	SAME AS T22
T10 – HOST DATA TO BUS	N/A	T24 – HOST DATA TO BUS	N/A	T37 – CONTROLLER SETS READY	SAME AS T23
T11 – HOST SETS XFER	T10–T11>–40 ns	T25 – HOST SETS XFER	SAME AS T11	T38 – HOST RESETS ONLINE	N/A
T12 – CONTROLLER RESETS READY	T11–T12<1 µs	T26 – CONTROLLER RESETS READY	SAME AS T12	T39 – CONTROLLER RESETS READY	N/A
T13 – CONTROLLER SETS ACK	0.5<T11–T13<100 µs	T27 – CONTROLLER SETS ACK	SAME AS T13	T40 – CONTROLLER SETS READY	N/A
T14 – HOST RESETS XFER	T13–T14>0 µs				

*NOTE: THIS TIME MAY BE > 500 µs UNDER SOME CONDITIONS

06591A C-3

Figure C-3. QIC-02 Write Data Command Timing Diagram



ACTIVITY	CRITICAL TIMING
T1 – HOST COMMAND TO BUS	N/A
T2 – HOST SETS ONLINE	N/A
T3 – HOST SETS REQUEST	T2–T3>0 µs
T4 – CONTROLLER RESETS READY	T3–T4<1 µs
T5 – CONTROLLER SETS READY	20<T4–T5<500 µs
T6 – HOST RESETS REQUEST	T5–T6>0 µs
T7 – BUS DATA INVALID	T5–T7>0 µs
T8 – CONTROLLER RESETS READY	20<T6–T8<100 µs
T9 – CONTROLLER CHANGES DIRC	N/A
T10– 1ST DATA BYTE TO BUS	N/A
T11– CONTROLLER SETS READY	N/A
T12– CONTROLLER SETS ACK	T11–T12>70 ns
T13– HOST SETS XFER	T12–T13>0 µs

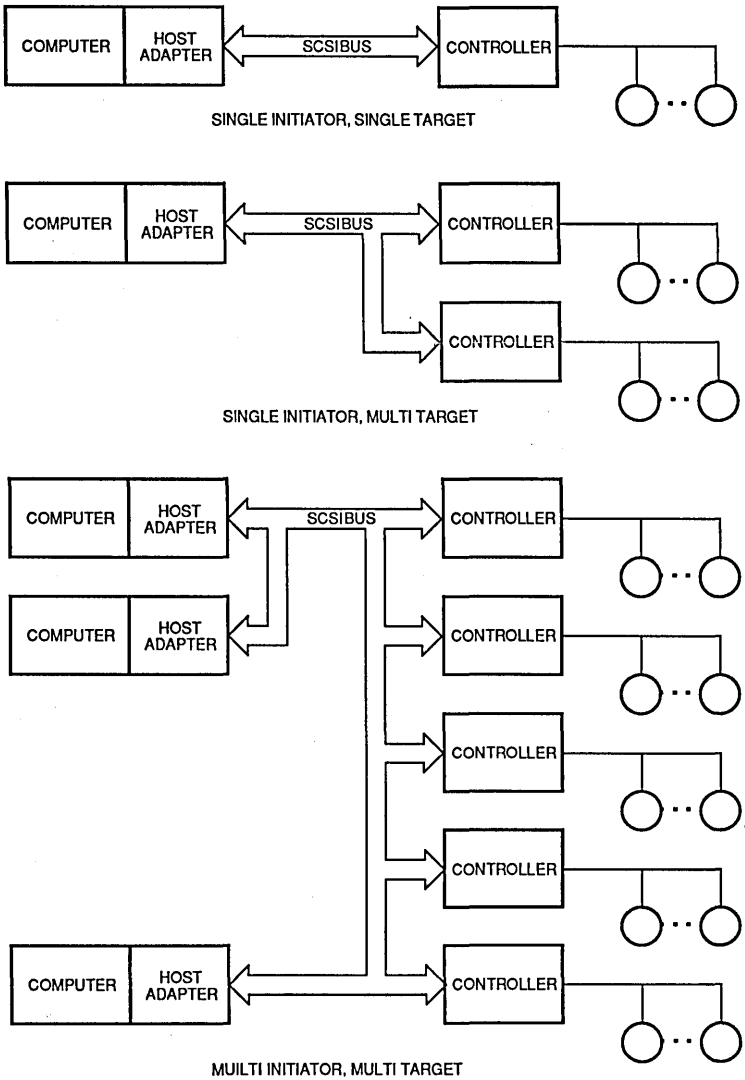
ACTIVITY	CRITICAL TIMING
T14 – CONTROLLER RESETS READY	T13–T14<1 µs
T15 – CONTROLLER RESETS ACK	0.5<T13–T15<3 < µs
T16 – BUS DATA INVALID	T13–T16>0 µs
T17 – HOST RESETS XFER	T15–T17>0 µs
T18 – BUS DATA VALID	
N/A	
T19 – CONCTROLLER SETS ACK	SAME AS T12
T20 – HOST SETS XFER	SAME AS T13
T21 – CONTROLLER RESETS ACK	SAME AS T15
T22 – BUS DATA INVALID	SAME AS T16
T23 – HOST RESETS XFER	SAME AS T17
T24 – CONTROLLER SETS READY	N/A
T25 – 1ST BYTE TO BUS	N/A
T26 – CONTROLLER SETS ACK	SAME AS T12

ACTIVITY	CRITICAL TIMING
T27 – HOST SETS XFER	SAME AS T18
T28 – CONTROLLER RESETS READY	SAME AS T14
T29 – CONTROLLER RESETS ACK	SAME AS T15
T30 – BUS DATA INVALID	SAME AS T16
T31 – HOST RESETS XFER	SAME AS T17
T32 – LAST BYTE TO BUS	N/A
T33 – CONTROLLER SETS ACK	SAME AS T12
T34 – HOST STES XFER	SAME AS T13
T35 – CONTROLLER RESETS ACK	SAME AS T15
T36 – BUS DATA INVALID	SAME AS T16
T37 – HOST RESETS XFER	SAME AS T17
T38 – CONTROLLER SETS EXCEPTION	N/A
T39 – CHANGE BUS DIRECTION	N/A

*NOTE: THIS TIME MAY BE > 500 µs UNDER SOME CONDITIONS

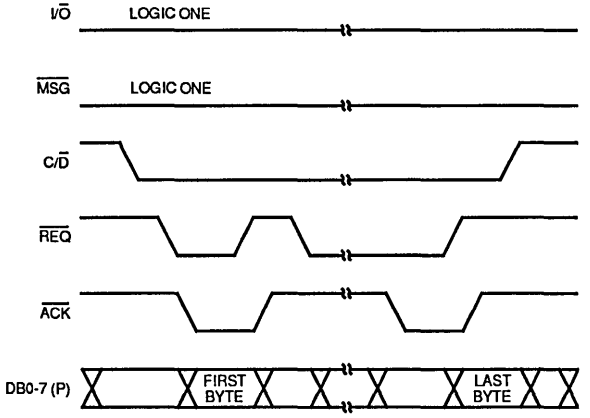
06591A C-4

Figure C-4. QIC-02 Read Data Command Timing Diagram



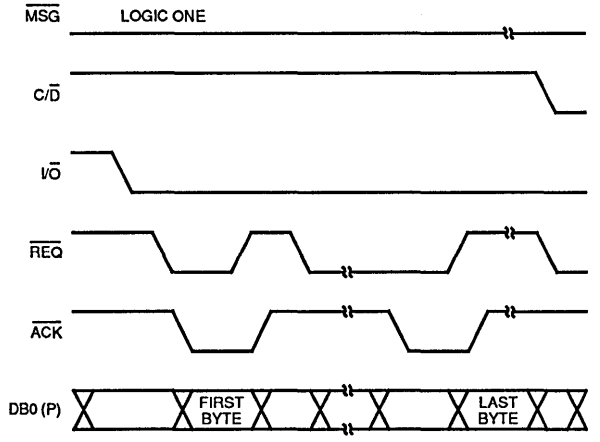
06591A C-5

Figure C-5. Possible Bus Configurations



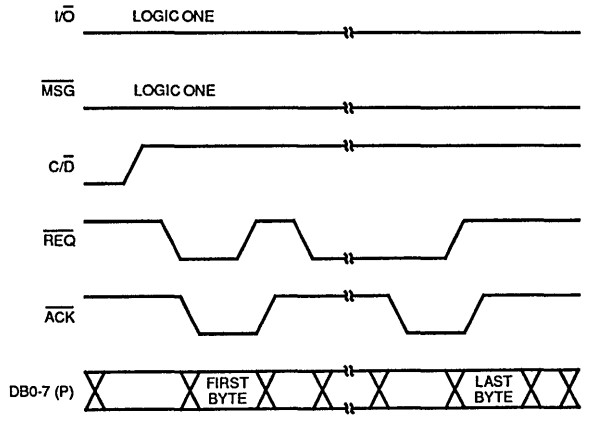
06591A C-6

C-6 SCSI Command Phase Timing



06591A C-7

C-7 SCSI Data Read (from disk) Timing



06591A C-8

C-8 SCSI Data Write (to disk) Timing

APPENDIX D

REFERENCES

1. *Advanced Micro Devices Bipolar Microprocessor Logic and Interface Data Book,* 1985.
2. *Advanced Micro Devices Programmable Array Logic Handbook,*1983.
3. *Advanced Micro Devices Bus Interface Product Specifications,* October 1985.
4. *Advanced Micro Devices Am29PL141 Data Sheet,* December 1985.
5. *Advanced Micro Devices Am29PL141 Assembler,* 1985.
6. *Advanced Micro Devices Am29PL141 User's Manual,* 1985.
7. *Advanced Micro Devices 80188 Data Sheet,* October 1985.
8. *Small Computer Systems Interface (SCSI) Specification* as defined by ANSI X3T9.2 Committee.
9. *Quarter Inch Cartridge (Tape Interface) (QIC-02) Specification.*
10. *PDP-11 Bus Handbook,* Digital Equipment Corporation, 1979.
11. *Microsystems Handbook,* Digital Equipment Corporation, 1985.

APPENDIX E

GLOSSARY OF ABBREVIATIONS/MNEMONICS

141SEL	Am29PL141 Selection (SCSI)
141TPREQ	Am29PL141 Tape Request (QIC-02) Signal
141XFER	Am29PL141 Transfer (QIC-02) Signal
ACK	Acknowledge
ARDY	Asynchronous Ready Line
ARESET	Asynchronous Reset
ATN	Attention
BSYIN	Busy Input (SCSI to FPC)
C/D	Control or Data, SCSI Interface Signal
CC	Condition Code (Input to FPC)
CCMUX	Condition Code MUX to Am29PL141
CMDXFER	Command Transfer Routine
CREG C	Register in Am29PL141 (Count Register)
DACK	Disk Acknowledge (SCSI)
DATN	Disk Attention (SCSI)
DCLK	Diagnostics Clock
DDACK	Disk (SCSI) DMA Acknowledge (to Am29PL141 from 80188)
DDREQ	Disk DMA Request
DIRC	Direction (QIC-02)
DMA	Direct Memory Access
DMAXFER	DMA Transfer Routine
DMSG	Disk (SCSI) Message = MSG C/D (to Int. Status Buffer from Am29PL141)
DREQ	Disk (Data Transfer) Request (to Am29PL141 from SCSI)
DRST	Disk Reset (SCSI)
DSP	Digital Signal Processor
DTACK	DMA Tape Acknowledge (to Am29PL141 from 80188)
DTREQ	DMA Tape Request (to 80188 from Addressable Latch)
EXP	Exception, QIC-02 Interface Signal
FPC	Fuse Programmable Controller
I/O	Input or Output
INT1	Interrupt Number One
ISR	Interrupt Status Register
JEDEC	Joint Electronic Device Engineering Council
LADDR	Addressable Latch
LAN	Local Area Network
LMCS	Lower Memory Chip Select
LPC	Linear Predictive Coding
MCSM	Mid-range Chip Select
MSG	Message SCSI Interface Signal (to Am29PL141 from SCSI)
MSI	Medium Scale Integration
NPR	Non-processor Request
PCS	Peripheral Chip Select
PL	Pipeline
POL	Polarity
RDXFER	Read Transfer Routine
RDY	Ready
RST	Reset
SCSI	Small Computer System Interface
SDI	Serial Data In
SDO	Serial Data Out
SIC-02	Quarter-inch Tape Cartridge Interface
SSR	Serial Shadow Register
TACK	Tape Acknowledge (to Am29PL141 from QIC-02)
TEST41	Am29PL141 Test Vector Generator Program
TOUT	Time Out
TPONL	Tape On Line (QIC-02)
TPRST	Tape Reset (QIC-02)
TRDY	Tape Ready (QIC-02)
TRINT	Tape Ready Interrupt (Addressable Latch to Condition Code MUX)
UMCS	Upper memory Chip Select
VCMD	Valid Command (to Am29PL141 from 80188)
WRXFER	Write Transfer Routine

APPENDIX F

Am29PL141 Data Sheet

For your reference, the first five pages of the 29PL141 data sheet are reprinted in this section. A complete copy of this 31 page document is available from the AMD sales offices listed on the last page. Copies are also available from authorized representatives.

Am29PL141

Fuse Programmable Controller (FPC)

DISTINCTIVE CHARACTERISTICS

- Implements complex fuse programmable state machines
- 7 conditional inputs, 16 outputs
- 64 words of 32-bit-wide microprogram memory
- Serial Shadow Register (SSR™) diagnostics on chip (programmable option)
- 29 high-level microinstructions
  - Conditional branching
  - Conditional looping
  - Conditional subroutine call
  - Multiway branch
- 20 MHz clock rate, 28-pin DIP

GENERAL DESCRIPTION

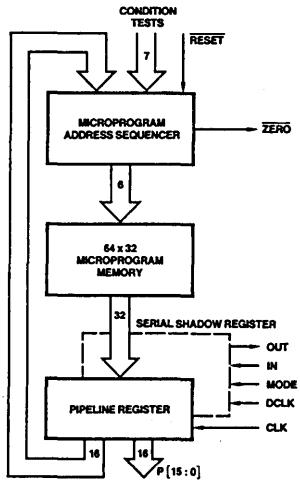
The Am29PL141 is a single-chip Fuse Programmable Controller (FPC) which allows implementation of complex state machines and controllers by programming the appropriate sequence of microinstructions. A repertoire of jumps, loops, and subroutine calls, which can be conditionally executed based on the test inputs, provides the designer with powerful control flow primitives.

The Am29PL141 FPC also allows distribution of intelligent control throughout the system. It off-loads the central controller by distributing FPCs as the control for various self-contained functional units, such as register file/ALU, I/O, interrupt, diagnostic, and bus control units.

A microprogram address sequencer is the heart of the FPC. It provides the microprogram address to an internal 64-word by 32-bit PROM. The fuse programming algorithm is almost identical to that used for AMD's Programmable Array Logic family.

As an option, the Am29PL141 may be programmed to have on chip SSR diagnostics capability. Microinstructions can be serially shifted in, executed, and the results shifted out to facilitate system diagnostics.

BLOCK DIAGRAM



BDR02340

RELATED PRODUCTS

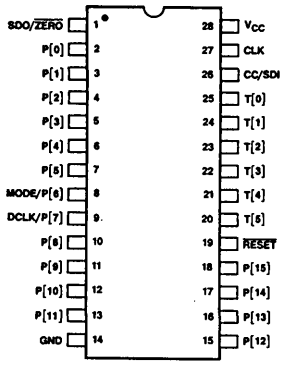
Part No.	Description
Am2914	Vectored Priority Interrupt Controller
Am29100	Controller Family Products

SSR is a trademark of Advanced Micro Devices, Inc.

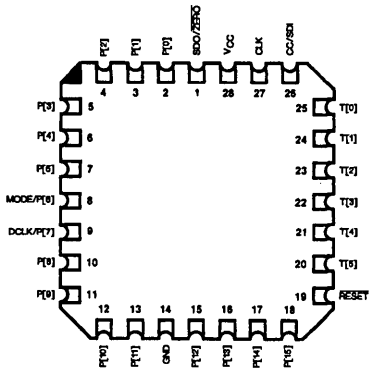
Publication # 04179 Rev. E Amendment /0
Issue Date: October 1986

Advanced Micro Devices

CONNECTION DIAGRAMS
Top View



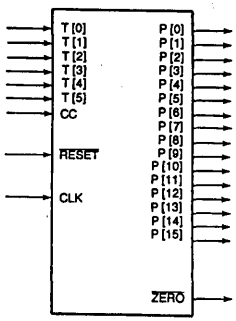
CDR04480



CD009110

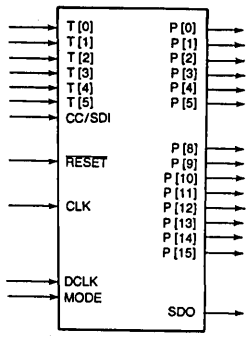
Note: Pin 1 is marked for orientation.

LOGIC SYMBOLS



LS002131

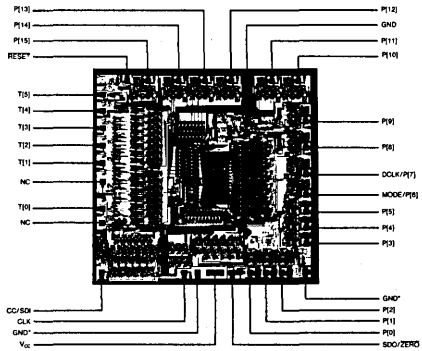
Normal Configuration



LS002140

SSR™ Diagnostics Configuration

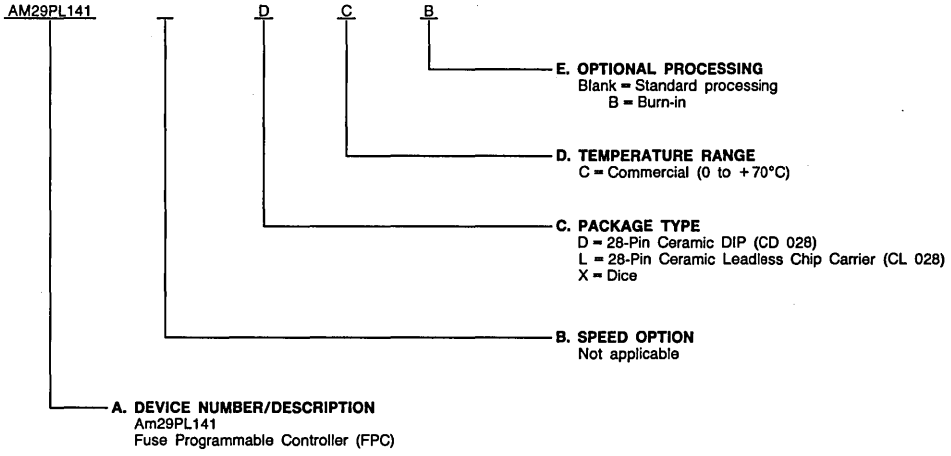
METALLIZATION AND PAD LAYOUT



Die Size: 0.211" x 0.202"
Gate Count: 600 Equivalent Gates and 2K of PROM

ORDERING INFORMATION
Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of: **A. Device Number**
B. Speed Option (if applicable)
C. Package Type
D. Temperature Range
E. Optional Processing



Valid Combinations	
AM29PL141	DC, DCB, LC, XC

Valid Combinations

Valid Combinations list cnfigurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

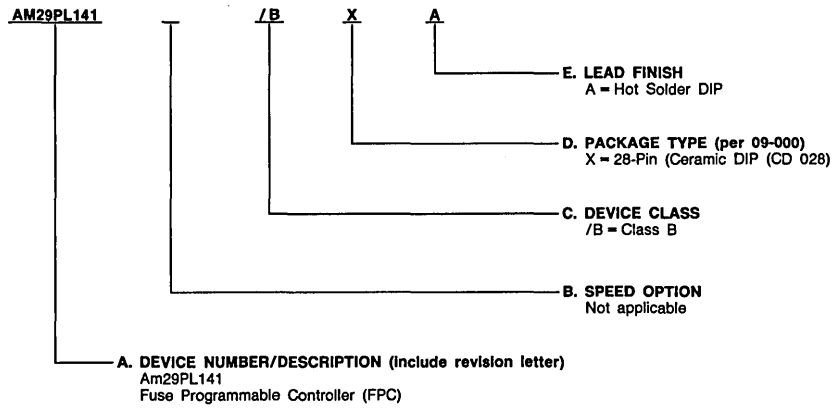
ORDERING INFORMATION
APL and CPL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. CPL (Controlled Products List) products are processed in accordance with MIL-STD-883C, but are inherently non-compliant because of package, solderability, or surface treatment exceptions to those specifications. The order number (Valid Combination) is formed by a combination of:

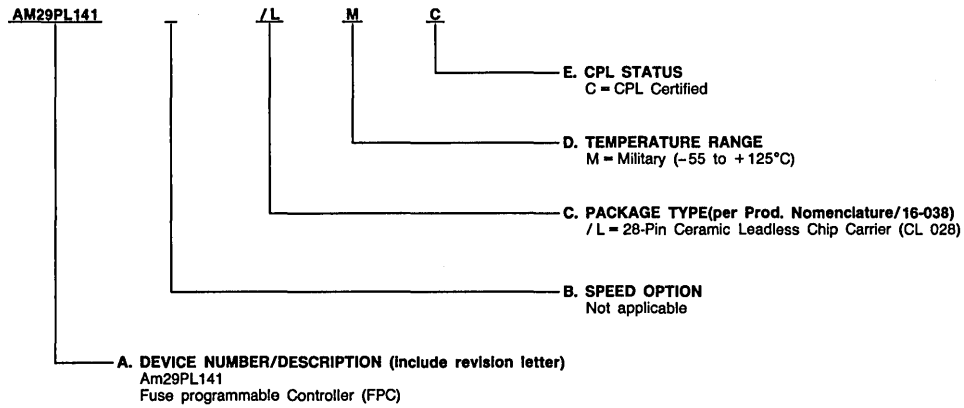
APL Products:
A. Device Number
B. Speed Option (if applicable)
C. Device Class
D. Package Type
E. Lead Finish

CPL Products:
A. Device Number
B. Speed Option (if applicable)
C. Package Type
D. Temperature Range
E. CPL Status

APL Products



CPL Products



Valid Combinations		
APL	Am29PL141	/BXA
CPL	Am29PL141	/LMC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consists of Subgroups:
1, 2, 3, 7, 8, 9, 10, 11

PIN DESCRIPTION

CC[SDI] Condition Code ((TEST) Input)
When the TEST (P[24:22]) field of the executing microinstruction is set to 6 (binary 110), CC is selected to be the conditional input. (Note: In SSR diagnostic configuration, CC is also the Serial Data Input SDI.)

CLK Clock (Input)
The rising edge clocks the microprogram counter, count register, subroutine register, pipeline register, and EQ flag.

P[15:8] (Outputs)
Upper eight, general-purpose microprogram control outputs. They are enabled by the OE signal from the microprogram pipeline register. When OE is HIGH, P[15:8] are enabled, and when LOW, P[15:8] are three-stated.

P[7:0] [DLCK, MODE] (Outputs)
Lower
Lower eight, general-purpose microprogram control outputs. They are permanently enabled. (Note: in the SSR diagnostic configuration, P[7] becomes the diagnostic clock input DCLK and P[6] becomes the diagnostic control input MODE.)

$\overline{\text{RESET}}$
Synchronous reset input. When it is low, the output of the PC MUX is forced to the uppermost microprogram address (63). On the next rising clock edge, this address (63) is loaded into the microprogram counter, the microinstruction at location 63 is loaded into the pipeline register and the EQ flag is cleared. The CREG and SREG values are indeterminate on reset.

T[5:0]
Test inputs. In conditional microinstructions, the inputs can be used as individual condition codes selected by the TEST field in the pipeline register. The T[5:0] inputs can also be used as a branch address when performing a microprogram branch, or as a count value.

$\overline{\text{ZERO}}$ [SDO]
Zero output. A Low state indicates that the CREG value is zero. (Note: In the SSR diagnostic configuration, $\overline{\text{ZERO}}$ becomes the Serial Data output SDO. This change is only on the output pin; internally, the zero detect functions is unchanged.)

ADVANCED MICRO DEVICES
U.S. SALES OFFICES

ALABAMA ..... (205) 882-9122
ARIZONA,
Tempe ..... (602) 242-4400
Tucson ..... (602) 792-1200
CALIFORNIA,
El Segundo ..... (213) 640-3210
Newport Beach ..... (714) 752-6262
San Diego ..... (619) 560-7030
Sunnyvale ..... (408) 720-8811
Woodland Hills ..... (818) 992-4155
COLORADO ..... (303) 741-2900
CONNECTICUT ..... (203) 264-7800
FLORIDA,
Clearwater ..... (813) 530-9971
Ft Lauderdale ..... (305) 484-8600
Melbourne ..... (305) 729-0496
Orlando ..... (305) 859-0831
GEORGIA ..... (404) 449-7920
ILLINOIS ..... (312) 773-4422
INDIANA ..... (317) 244-7207
KANSAS ..... (913) 451-3115
MARYLAND ..... (301) 796-9310
MASSACHUSETTS ..... (617) 273-3970
MINNESOTA ..... (612) 938-0001
MISSOURI ..... (314) 275-4415
NEW JERSEY ..... (201) 299-0002
NEW YORK,
Liverpool ..... (315) 457-5400
Poughkeepsie ..... (914) 471-8180
Woodbury ..... (516) 364-8020
NORTH CAROLINA ..... (919) 847-8471
OREGON ..... (503) 245-0080
OHIO,
Columbus ..... (614) 891-6455
PENNSYLVANIA,
Allentown ..... (215) 398-8006
Willow Grove ..... (215) 657-3101
TEXAS,
Austin ..... (512) 346-7830
Dallas ..... (214) 934-9099
Houston ..... (713) 785-9001
WASHINGTON ..... (206) 455-3600
WISCONSIN ..... (414) 782-7748

INTERNATIONAL SALES OFFICES

BELGIUM,
Bruxelles ..... TEL: . (02) 771 99 93
FAX: .. (02) 762-3716
TLX: 61028
CANADA, Ontario,
Kanata ..... TEL: . (613) 592-0090
Willowdale ..... TEL: . (416) 224-5193
FAX: . (416) 224-0056
FRANCE,
Paris ..... TEL: (01) 45 60 00 55
FAX: (01) 46 86 21 85
TLX: 202053F
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,
Kowloon ..... TEL: 3-695377
FAX: 1234276
TLX: 50426
ITALY, Milano ..... TEL: ..... (02) 3390541
FAX: ..... (02) 3498000
TLX: 315286
JAPAN, Tokyo ..... TEL: ..... (03) 345-8241
FAX: 3425196
TLX: J24064AMDTKOJ
LATIN AMERICA,
Ft. Lauderdale ..... TEL: ..... (305) 484-8600
FAX: ..... (305) 485-9736
TLX: .. 5109554261 AMDFTL
SWEDEN, Stockholm ..... TEL: ..... (08) 733 03 50
FAX: ..... (08) 733 22 85
TLX: 11602
UNITED KINGDOM,
Manchester area ..... TEL: ..... (0925) 828008
FAX: ..... (0925) 827693
TLX: 628524
London area ..... TEL: ..... (04862) 22121
FAX: ..... (04862) 22179
TLX: 859103

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) 5388450

Printed in U.S.A. CBM-B-10M-7/86-1

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

Order # 06591A

IH-MU-5M-11/86-0