

**Programmable
Logic**

Handbook/Data Book
1986-1987

PROGRAMMABLE

PAL™ Devices

Fuse Programmable Controller

PROMs

Programmable Event Generator

Sequencers

LOGIC



ADVANCED MICRO DEVICES





Advanced Micro Devices

Programmable Logic Handbook/Data Book

The International Standard of
Quality guarantees a 0.05% AQL on all
electrical parameters, AC and DC,
over the entire operating range.

INT-STD-500

© 1986 Advanced Micro Devices, Inc.

Advanced Micro Devices reserves the right to make changes in its products without notice in order to improve design or performance characteristics. The performance characteristics listed in this document are guaranteed by specific tests, correlated testing, guard banding, design and other practices common to the industry.

For specific testing details contact your local AMD sales representative.

The company assumes no responsibility for the use of any circuits described herein.

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

CREDITS

Prepared by the Fuse Programmable Logic (FPL) staff at Advanced Micro Devices, Inc. (listed in alphabetical order)

Editing:

Jeri Burdick
Naushik Desai

Technical:

Om Agrawal
Peter Alfke
Matthew Bonn
Sal Cagnina
Rick Calle
William Chen
Paul Coggeshall
Naushik Desai
Vineet Dujari
Steve Grossman
Alexandra Huff
Alfred Jey
Doug Kern
Arthur Khu
Brent McKay
Teresa Ordez
John Potterbom
Mitch Richman
Kapil Shankar
Bill Sievers
Jenny Yee

TRADEMARKS

ABEL, DASH/ABEL, FINGERPRINT, PLDtest, and PROMlink are trademarks of Data I/O Corporation.

CUPL is a trademark of Personal CAD Systems, Inc.

IMOX-III is a trademark of Advanced Micro Devices, Inc.

MULTIBUS is a trademark of Intel Corporation.

PALASAM is a registered trademark of Monolithic Memories, Inc.

Z8000 is a trademark of Zilog, Inc.

PREFACE

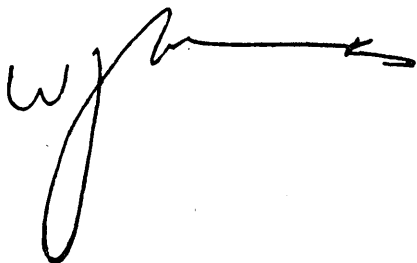
At Advanced Micro Devices we are committed to helping you solve design problems with innovative products that give you maximum flexibility to meet your particular needs. In the past, you often have had to choose between costly semi-custom products and standard devices that limit your design options. Now there's another choice.

Programmable Array Logic (PAL) devices from AMD give you the best of two worlds: the architectural flexibility of custom design and immediate availability, multiple-sourcing, and low cost of standard products.

PAL devices are user-programmable logic building blocks that give you the freedom to structure components for a specific application, often using a single PAL package to create functions that once would have required the use of hundreds of conventional TTL gates.

This handbook is your guide to AMD's growing list of PAL devices. It is intended to be both an introduction to field programmable logic devices, as well as a resource manual for experienced designers. In the following pages you will find data and descriptions of new, proprietary devices from AMD, as well as information on standard products. Whether you need CMOS, ECL, or TTL, AMD offers solutions to your design problems that give you a choice.

Best of all, AMD offers you the industry's leading commitment to quality, reliability, innovation, and customer satisfaction. If you have a question about any of the products described in this book or simply want to know more about the use of programmable logic devices, call your local Advanced Micro Devices' Sales Office.

A handwritten signature in black ink, appearing to read 'W.J. Sanders III', with a long horizontal stroke extending to the right.

W.J. Sanders III
Chairman, and Chief Executive Officer

TABLE OF CONTENTS

SECTION 1 — INTRODUCTION TO PROGRAMMABLE LOGIC

1.1	The Programmable Logic Design Alternative	1-1
1.2	An Introduction to Programmable Logic Architecture	1-8
1.3	How to Design with Programmable Logic Devices	1-25

SECTION 2 — SOFTWARE, PROGRAMMING, TESTING, RELIABILITY, AND TECHNOLOGY INFORMATION

2.1	Software Support for AMD's Programmable Logic Devices	2-1
2.1.1	Design-Aid Software for Programmable Logic	2-1
2.1.2	ABEL	2-3
2.1.3	CUPL, AmCUPL	2-12
2.1.4	PLPL	2-35
2.2	Programming Hardware	2-40
2.3	Testing Information	2-43
2.3.1	Factory Testing of PAL Devices	2-43
2.3.2	How Testability is Designed into AMD's Programmable Logic Devices	2-43
2.3.3	Specifications for Switching-Delay Minimums	2-51
2.4	AMD Programmable Array Logic Reliability	2-52
2.5	Programmable Logic Technology	2-59
2.5.1	IMOX-III — Advanced Bipolar Technology for PAL Devices	2-59
2.5.2	Advanced CMOS Technology for PAL Devices	2-61

SECTION 3 — APPLICATIONS

3.1	Overview	3-1
3.2	Combinatorial Logic	3-3
3.2.1	Multiplexers	3-3
3.2.2	Demultiplexers	3-3
3.2.3	Encoders/Decoders	3-9
3.2.4	Comparators	3-22
3.2.5	Address Decoding and Chip Select Generation Simplified with Combinatorial PAL Devices	3-25
3.3	Sequential Logic	3-29
3.3.1	Counters	3-29
3.3.2	Shifters	3-47
3.4	Microprocessor Interface Logic	3-55
3.4.1	The Interface Problem	3-55
3.4.2	Interfacing to the 8086/80186/80286	3-60
3.4.3	Interfacing to the 68000/68020	3-81
3.4.4	Interfacing to the 8088/80188	3-137
3.4.5	Interfacing to the Z80/Z8000	3-171
3.5	Bus Interface Logic	3-187
3.5.1	MULTIBUS to Am9516 Interface	3-187
3.5.2	Z-Bus and 8088/8086 Interface	3-191
3.5.3	An AMD PAL MULTIBUS Arbiter	3-201
3.5.4	VME Bus Control Simplified with PLDs	3-211

3.6	Miscellaneous Logic Functions	3-221
3.6.1	8088 to Am2968 Interface	3-221
3.6.2	A General-Purpose Interface for the Am2968	3-229
3.6.3	MC68000 to Am2968 Interface	3-237
3.6.4	General-Purpose Dual-Port Arbiter	3-243
3.6.5	Customize a Flexible DRAM Controller Using Second-Generation PAL Devices	3-252
3.6.6	Dynamic Memory Control State Sequencer	3-281
3.6.7	82284 and 82288 Emulation in an IBM PC/AT Computer Using Two AmPAL16RB Devices	3-290
3.6.8	Interfacing the 80186 Microprocessor to the 8087 with the AmPAL22V10A Device	3-301
3.6.9	A MULTIBUS Arbiter Design for 10 MHz Processors	3-315
3.7	Programmable Logic Devices — Article Reprints	
3.7.1	"Logical Alternatives in Supermini Design," reprinted by permission of Computer Design, November 1983. All rights reserved	3-320
3.7.2	"PLDs as Semicustom Substitutes," reprinted by permission of VLSI Design, June 1985. All rights reserved	3-328
3.7.3	"Programmable logic chip rivals gate arrays in flexibility," reprinted by permission of Electronic Design, December 1983. All rights reserved	3-336
3.7.4	"PAL device buries state registers, brings state machines to life," reprinted by permission of Electronic Design, July 1986. All rights reserved	3-342
3.7.5	"Mixing Data Paths Expands Options in System Design," reprinted by permission of Computer Design, January 1985. All rights reserved	3-347
3.7.6	"High Performance DMA for the VMEbus," reprinted by permission of Digital Design, November 1985. All rights reserved	3-357
3.7.7	"A Demultiplexed Analog Subsystem," reprinted by permission of Digital Design, November 1985. All rights reserved	3-362
3.7.8	"Advanced Programming Language for Programmable Logic Designs," from Southcon 1985	3-366
3.7.9	"Test Methods for Programmable Logic," from Southcon 1984	3-382
3.7.10	"Fuse-programmable chip takes command of distributed systems," reprinted by permission of Electronic Design, October 17, 1985. All rights reserved	3-390
3.7.11	"Programmable event generator conquers timing restraints," reprinted by permission of Electronic Products, July 1, 1986. All rights reserved	3-396
3.7.12	"PLDs implement encoder/decoder for disk drives," reprinted by permission of Electronic Design News, September 18, 1986. All rights reserved	3-401
3.8	Bibliography	3-41C

SECTION 4 — PRODUCT SPECIFICATIONS

4.1	Field Programmable Logic Selector Guide	4-
4.2	Glossary	4-
4.3	AmPAL16XX Family 20-Pin IMOX Programmable Array Logic (PAL) Elements	4-
4.4	AmPAL16XXD Family 20-Pin IMOX Ultra High-Speed Programmable Array Logic (PAL) Elements	4-3
4.5	AmPAL18P8 20-Pin IMOX Programmable Array Logic	4-3
4.6	AmPAL20EG8 IMOX-III ECL Programmable Array Logic	4-4
4.7	AmPAL20EV8 IMOX-III ECL Programmable Array Logic	4-4
4.8	AmPAL21VT8 24-Pin Dual-Clock Programmable Array Logic	4-4
4.9	AmPAL22V10 24-Pin IMOX Programmable Array Logic (PAL)	4-4
4.10	AmPAL22V10B 24-Pin IMOX Programmable Array Logic (PAL)	4-4
4.11	AmPAL23S8 20-Pin IMOX PAL-Based Sequencer	4-1C
4.12	AMD 24-Pin Standard PAL Family	4-1C
4.13	AMD 24-Pin Enhanced PAL Family	4-1C
4.14	AMD 24-Pin XOR PAL Family	4-1C
4.15	AmPALHC29M16/AmPALHCT29M16 24-Pin E ² -Based CMOS Programmable Array Logic	4-1
4.16	AmPALHC29MA16/AmPALHCT29MA16 24-Pin E ² -Based CMOS Programmable Array Logic	4-2
4.17	Bipolar PROMs as Programmable Logic Products	4-2

4.18	Am29PL141 Fuse Programmable Controller (FPC)	4-256
4.19	Am2971 Programmable Event Generator (PEG)	4-286

SECTION 5 — GENERAL INFORMATION

5.1	Cross Reference Guide	5-1
5.2	Package Outlines	5-5
5.3	Technical Report — Package Thermal Characteristics	5-12
5.4	AMD Sales Offices	5-16

SECTION 6 — INDEX

SECTION 1 — INTRODUCTION TO PROGRAMMABLE LOGIC



**1.1 THE PROGRAMMABLE LOGIC DESIGN
ALTERNATIVE**

**1.2 AN INTRODUCTION TO PROGRAMMABLE LOGIC
ARCHITECTURE**

**1.3 HOW TO DESIGN WITH PROGRAMMABLE LOGIC
DEVICES**

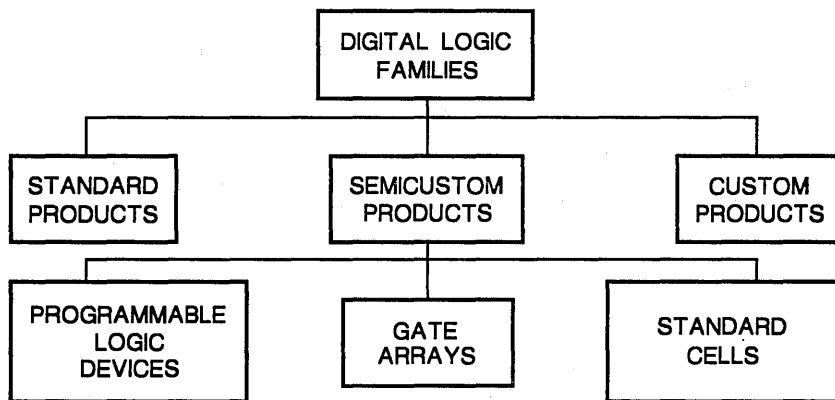
1.1. THE PROGRAMMABLE LOGIC DESIGN ALTERNATIVE

1

Today's logic designer can choose from a wide variety of circuit alternatives. One way to categorize these alternatives is by the extent to which they are customized for a specific application. The designer's options include general-purpose standard products, programmable logic, gate arrays, standard cells, and full custom integrated circuits (ICs) (Figure 1-1).

Standard products are defined by the IC manufacturer for a wide market and cannot be altered by the user. Examples of standard products are TTL and CMOS SSI/MSI devices, fixed

instruction set MOS microprocessors, and microprogrammable building blocks. Custom logic, on the other hand, is defined by the user for a specific application. Programmable logic devices have attributes of both standard products and custom logic. The IC manufacturer defines an architecture that a user can program by programming (or blowing) appropriate fuses to fit the application. Programmable Array Logic (PAL) devices, Programmable Logic Arrays (PLAs) and small Programmable Read-Only Memories (PROMs) are examples of programmable logic.



PF002411

Figure 1-1. Basic Categories of Digital Logic

Each of these design alternatives offers distinct advantages and disadvantages in terms of cost, availability, and architectural flexibility. Many system designs today, such as the controller board in Figure 1-2, incorporate each of the design

approaches to some degree. The following analysis will help the system designer to select the best logic type for a particular function.

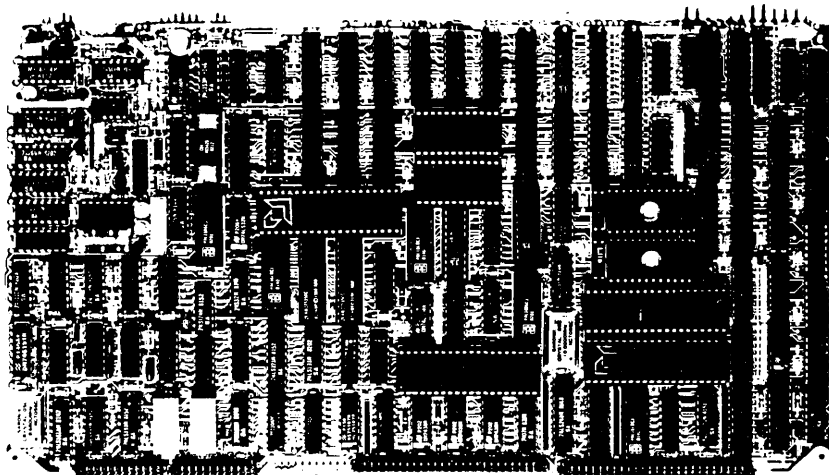


Figure 1-2. A System Design Using PAL Devices

Photo courtesy of Data Systems Design, Inc.

DEDICATED GENERAL-PURPOSE DEVICES — STANDARD PRODUCTS

There are five main advantages of standard products. They require little IC engineering expertise by the user, provide lowest cost for an individual device, usually have the best application support, provide the maximum logic density per device and are available off-the-shelf with no development lead time.

Standard products require little IC-level engineering effort by the user. The responsibility for design, test, and debugging is born by the IC manufacturer. Since the IC manufacturer is doing this on a large scale, the process is very efficient.

Standard products are very cost-effective per logic function. They are high volume products, and this volume results in lower manufacturing cost and thus lower price per unit. The increased competition encouraged by alternate sourcing also results in lower cost.

The design support available for standard products is generally far greater than that for semicustom and custom devices. Application software (assemblers, simulators), hardware (emulators) and literature (manuals, books, application notes) make them easier to use.

Since standard products reach a much larger market, the engineering effort necessary to provide this support can be spread over a large number of units, reducing the cost. When a custom logic device is used, this support must be developed by the engineer doing the design.

Standard products are optimized for high-volume production. The density of logic functions is therefore generally much greater than on semicustom logic. A fixed instruction set microprocessor or microprogrammable building block duplicated with gate arrays or programmable logic devices might take several packages compared to the single dedicated device.

The last main advantage of standard products is their off-the-shelf availability. There is no development lead time in the use of these devices, so system design can proceed rapidly. By using standard products the system designer can introduce a

system to the market more quickly and exploit the value of market leadership.

The three potential disadvantages of standard products are potentially poor fit to specific applications, higher system cost, and the lack of competitive features and advantages. A standard product, by the very nature of its generality, is not ideal for anyone. It includes too much functionality for some applications and not enough for others. The architecture is seldom ideal for a particular application. Standard products also offer a limited performance selection. IC manufacturers pick a specific performance level aiming at as large a market as possible.

Due to the general-purpose nature of standard products, it is difficult to achieve the lowest package-count solution. Additional components are required to tailor the function to fit a specific need.

Even though individual devices may be lower in price, more of them must be used, raising the cost for the total system when considering the additional PC boards, testing, power supplies, fans, etc.

Another disadvantage of standard products is the lack of competitive features and advantages. Anyone can buy them so it is difficult to differentiate one system supplier's hardware from another's. It is also very easy for a competitor to copy a design based on standard products.

CUSTOM AND SEMICUSTOM LOGIC DEVICES

The custom and semicustom logic alternatives offer the systems designer important advantages over standard products. Reduced package count, compared to SSI/MSI implementations, is of paramount importance. Custom and semicustom logic also provide the designer additional freedom in architecture. This freedom to develop innovative solutions to an application problem can add a significant competitive advantage to a product.

There are four main types of custom and semicustom logic today: fully custom logic, gate arrays, standard cell designs, and programmable logic. Fully "handcrafted" custom-IC logic

designs give the user the benefits of low system IC count and potentially low variable manufacturing cost per device, but the cost to develop a custom IC can be very high. This alternative makes sense only when production volumes will be very high and the system design will be very stable. Semicustom approaches such as programmable logic, gate arrays, and standard cell designs reduce the IC development cost of the full custom solution by trading off the chip-layout efficiency.

A gate-array design requires the customization of only a few interconnection layers in the semiconductor process. Standard cell designs require completely custom fabrication, but the design and layout are simplified by the use of a standard building-block library. A gate array has lower engineering cost and faster development time than a standard cell-based device. Standard cell devices, on the other hand, allow more logic variety and more efficient utilization of silicon than a gate array.

Gate arrays and the other custom/semicustom alternatives have four main disadvantages when compared to standard products. They are increased user engineering time and effort, higher cost per individual device, inferior high-level support tools, and lower density.

Engineering effort for a gate array can significantly increase the cost of a system design. Not only must the system be designed, but the gate arrays themselves must be designed, debugged and put into production. Both design tasks, chip as well as system, take similar amounts of engineering resources, possibly doubling the design effort and investment. Because of constraints on second-sourcing alternatives, semicustom logic devices can end up being substantially more expensive. Only if the complete system solution can be optimized will the

total cost be reduced. Another factor to be considered is the chance of design problems with a gate array, standard cell, or fully custom device. If extra iterations are necessary — or even worse, a bug is discovered after a product has been released, correcting the problem can take several months or longer. These potential costs are difficult to estimate and have virtually no upper limit.

The third disadvantage of custom and semicustom logic is the inadequacy of high-level support. Semiconductor manufacturers cannot provide significant support in the form of software, development systems, application notes, or books for a custom logic design because each implementation is different. The existing gate array and standard-cell design tools require extensive training and a large investment in time and money for the user. The designer must document the design fully and provide enough support for the system engineer to use the customized device correctly.

Finally, a key disadvantage of gate arrays is the reduced density and therefore high silicon cost compared to a dedicated general-purpose device. They are designed by repeating a common loosely packed structure, leaving wide channels for the metal interconnect. For a given set of design rules, a gate array will typically require two to five times the silicon area for the same gate count.

PROGRAMMABLE LOGIC DEVICES

Programmable logic devices combine the best characteristics of standard and custom products. They offer the flexible architecture of a custom design as well as the off-the-shelf availability and reduced investment, both in engineering time and device cost, of a standard product (see Table 1-1).

TABLE 1-1. IC SELECTION CRITERIA FOR DIFFERENT ARCHITECTURES

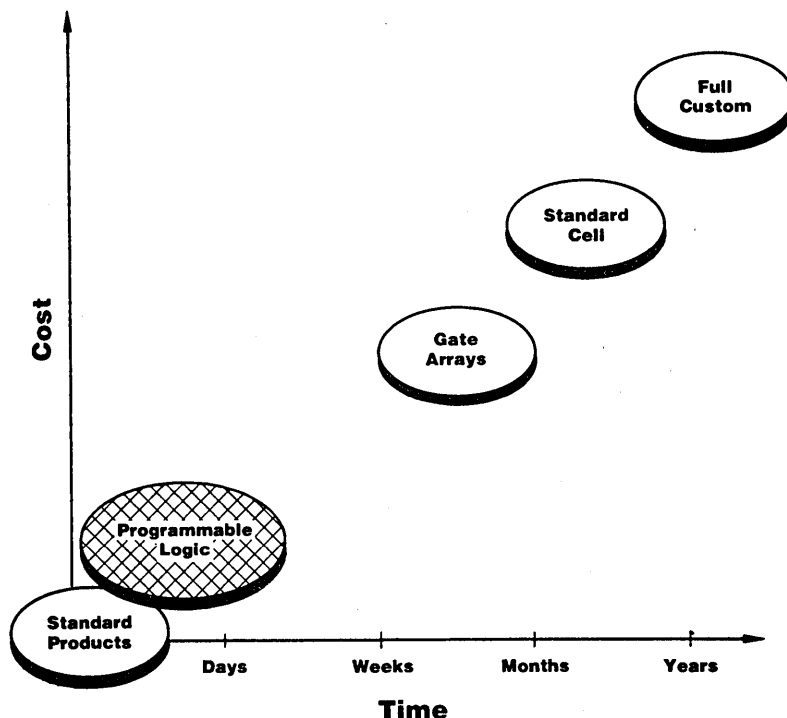
Criteria	Standard Products		Gate Arrays/ Standard Cells	Programmable Logic
	TTL SSI/MSI	LSI		
Development Lead Time	Immediate	Immediate	Weeks/Months	Hours
Development Cost	None	None	≈ \$20K	Low
Second Sources	Many	Several	Few	Many
Architectural Flexibility	Medium	Low	High	Medium
Logic Density	Low	High	Medium/High	Medium
IC Design Expertise Required	None	None	Some/Much	None

Programmable logic has the fastest design cycle time of any form of customizable logic. Instead of months or years, as with other semicustom or full custom designs, a programmable logic-based design can be created in just hours. This fast turnaround time allows a revolutionary interactive approach to system design. The engineer can try out a new architectural approach and evaluate it very quickly. If it does not work, a new idea can be quickly defined, programmed and ready to evaluate. The speed with which a new design approach can be explored and evaluated creates a design environment that enhances innovation.

Programmable logic devices enjoy the same high-volume production economics as standard products. Producing identical blank elements by the millions of units per year, the

manufacturer can achieve low cost. A volume market attracts multiple vendors and encourages price competition, as well as provides alternate source security. The cost advantages of a standard product are retained with programmable logic devices, but since the parts are then customized, system designs may be differentiated from the competition. In fact, truly innovative system designs implemented using programmable logic are even patentable, further protecting a design from the competition.

The engineering effort and time needed to design, test, debug and put into production a programmable logic-based system device is larger than the effort necessary when using standard products, but substantially less than when using custom elements (Figure 1-3).

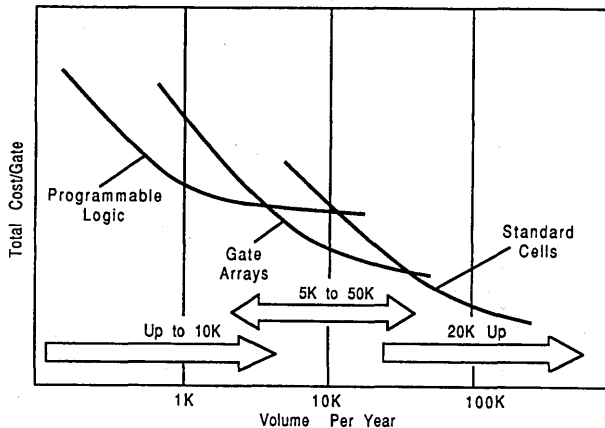


OP002220

Figure 1-3. Relative Development Time vs. Cost for Alternative Logic Implementations

Software tools are provided to reduce this overhead considerably. These permit designs to be specified in terms of Boolean equations, state transition equations, or gate array-like schematics. The software input specification format serves as a "data sheet" for the particular application and generates the essential documentation information. Simulation and test vector generation programs also exist to reduce the engineering effort associated with debugging and testing, both in prototyping and production environments.

Compared to other forms of semicustom logic, the maximum available logic density of a programmable logic device is smaller, and the manufacturing costs per equivalent gate of logic are greater. Since the fixed costs associated with using programmable logic are lower, however, this semicustom alternative has an advantage in lower volume designs. In fact, at the annual procurement volume of the majority of industrial electronic systems in production today—a few thousand systems per year—programmable logic has the lowest cost per gate of any form of custom logic available (see Figure 1-4).



OP002210

Figure 1-4. Programmable Logic Provides the Lowest Cost Per Gate of the Three Semicustom Techniques at Volumes up to About 5K

Programmable logic devices also offer significant user benefits over non-customizable standard product solutions. These include design optimization, design security, improved performance, cost reductions through board space and inventory cost savings, and reliability improvements through reductions in parts count and interconnections.

Flexibility

With the availability of programmable logic the designer is not constrained by the available selection of fixed-function TTL SSI/MSI parts. If a desired TTL function did not exist, the designer previously had to use several packages to generate it. With programmable logic devices the designer can easily create a customized part for a specific application.

Design modifications are also easier to implement in programmable logic-based systems. Changes may be made by reprogramming a device, rather than re-laying out a board. The time required for prototyping changes can be reduced by several weeks, and the cost can be substantially lower.

Design Security

Programmable logic devices can be used to enhance the security of a logic design. By programming a special "security fuse," the user can disable the fuse verify logic circuitry. This prevents unauthorized duplication of the device, while not interfering with the part's logic functionality. Programmable logic is ideal for any application where design security is essential.

Performance

System performance can be increased through the use of programmable logic. The designer has the freedom to optimize the system architecture by tailoring programmable devices to implement it precisely. Thus a design may be implemented in the most efficient manner, frequently improving performance. In addition, when a logic function is implemented in multiple SSI/MSI packages, the total delay incurred includes the time required for several on- and off-chip buffers. When the same function is implemented in a single programmable logic device the average delay per logic gate is reduced because there is only one pair of I/O buffers.

Cost Reduction

Programmable logic devices can provide complexity equivalent to hundreds of TTL gates. Implementing a design in programmable logic can therefore significantly reduce the board space or the number of boards necessary to implement a given function. This results in lower system cost, or alternatively, the ability to provide more function in the same enclosure.

Reliability

Compared to standard TTL SSI/MSI, programmable logic reduces the number of packages necessary to implement a given function. In some cases an entire PC board can be eliminated. This reduction in parts count will result in increased system reliability. Reducing the number of packages also reduces the number of external connections between devices in the system. Since these connections are often the least reliable portions of a digital system, the use of programmable logic can improve system reliability in this manner, too.

PROGRAMMABLE LOGIC FROM ADVANCED MICRO DEVICES

The remainder of this Handbook will focus on the programmable logic devices offered by Advanced Micro Devices. Included will be discussions of their architecture and features, how to design with them, hardware and software tools, testability, technology, reliability, applications, and detailed product specifications.

Advanced Micro Devices offers a broad family of PAL-type programmable array logic devices. These devices feature a programmable-AND array which feeds into a set of fixed-OR gates. As any logic function can be expressed in an AND/OR sum-of-products (SOP) form, these basic elements can be programmed to satisfy a wide variety of complex custom logic requirements.

Tables 1-2 and 1-3 show AMD's PAL product family and the features incorporated in AMD's products: on-chip registers, feedback paths, output enable control, user-programmable output logic macrocells, programmable output polarity, variable product term distribution, buried registers, special test

functions, etc. Today's devices range from 200 to over 1000 gates of functional complexity. Performance is high, with propagation delays lower than 15 ns and clock rates greater than 40 MHz. ECL PAL devices featuring 125-MHz operation are available.

AMD is not restricting itself to just the PAL architecture. Also available is a PROM-based fuse-programmable sequencer, the Am29PL141. In the future, AMD will be offering programmable logic devices based on other architectures that may be optimized for specific types of applications.

TABLE 1-2. INDUSTRY-STANDARD 20-PIN MEDIUM COMPLEXITY PAL FAMILY

Part Number	Array Inputs	Logic	OE	Outputs
16L8	Ten Dedicated, Six Bidirectional	Eight 7-Wide AND-OR-INVERT	Programmable	Six Bidirectional, Two Dedicated
16R4	Eight Dedicated, Four Feedback, Four Bidirectional	Four 8-Wide AND-OR	Dedicated	Registered Inverting
		Four 7-Wide AND-OR-INVERT	Programmable	Bidirectional
16R6	Eight Dedicated, Six Feedback, Two Bidirectional	Six 8-Wide AND-OR	Dedicated	Registered Inverting
		Two 7-Wide AND-OR-INVERT	Programmable	Bidirectional
16R8	Eight Dedicated, Eight Feedback	Eight 8-Wide AND-OR	Dedicated	Registered Inverting
16H8	Ten Dedicated, Six Bidirectional	Eight 7-Wide AND-OR	Programmable	Six Bidirectional, Two Dedicated
16LD8	Ten Dedicated, Six Bidirectional	Eight 8-Wide AND-OR-INVERT	—	Dedicated
16HD8	Ten Dedicated, Six Bidirectional	Eight 8-Wide AND-OR	—	Dedicated

TABLE 1-3. ADVANCED PAL DEVICES FROM ADVANCED MICRO DEVICES

Part Number	Technology	Propagation Delay (t_{PD})	Description
18P8	Bipolar TTL	15 ns	20-Pin, 18-Input Combinatorial PAL with Programmable Polarity
20L8 20R4 20R6 20R8	Bipolar TTL	15 ns	24-Pin PAL Family with 8 Registered/Combinatorial Outputs
20L10	Bipolar TTL	15 ns	24-Pin, 20-Input, 10-Output Combinatorial PAL
22P10	Bipolar TTL	15 ns	24-Pin, 22-Input, 10-Output Combinatorial PAL with Programmable Polarity
20RP4 20RP6 20RP8 20RP10	Bipolar TTL	15 ns	24-Pin PAL Family with 10 Registered/Combinatorial Outputs and Programmable Polarity
22XP10 20XRP4 20XRP6 20XRP8 20XRP10	Bipolar TTL	20 ns	24-Pin PAL Family with EXCLUSIVE-OR Capability, 10 Registered/Combinatorial Outputs, and Programmable Polarity
22V10	Bipolar TTL	25 ns	24-Pin "Family of One" with 10 Programmable Output Logic Macrocells (OLMs)
23S8	Bipolar TTL	20 ns	20-Pin PAL Sequencer with 4 OLMs, 4 Output Registers, and 6 Buried State Registers
20EV8	Bipolar ECL	6 ns	24-Pin ECL PAL with 8 Registered OLMs
20EG8	Bipolar ECL	6 ns	24-Pin ECL PAL with 8 Latched OLMs
29M16	E ² CMOS	35 ns	24-Pin High Complexity CMOS PAL with 16 OLMs and 2 Clock Inputs
29MA16	E ² CMOS	35 ns	24-Pin High Complexity Asynchronous CMOS PAL with 16 OLMs and Product-Term Driven Clocks

APPLICATIONS

PAL devices are used in a broad base of applications. They are used frequently in minicomputers, workstations, personal computers, and peripherals. They show up in military as well as commercial applications. They are used both as glue-logic replacement and as building block ICs for high-level functions in the control and data paths of computer systems. Detailed application examples are provided in this Handbook.

CONCLUSION

Programmable logic combines the strengths of the dedicated general purpose and the custom logic design approaches. It offers user-customizability with immediate turn-around time. This revolutionary design approach results in innovative, low cost designs, maximizing the competitive advantage of the systems in which they are used. AMD offers a broad family of programmable logic devices. Their architecture and features will be described in more detail in the next chapter.

1.2 AN INTRODUCTION TO PROGRAMMABLE LOGIC ARCHITECTURE

PAL (Programmable Array Logic), PROM (Programmable Read-Only Memory), and PLA (Programmable Logic Array) devices are the three most popular programmable logic devices (PLDs). All three share the same basic, two-level, internal AND-OR structure shown in Figure 1-5. The AND array is the first level, it accepts all the inputs (both true and complement), performs the desired AND functions on these inputs and drives the next level. The second-level OR array combines various AND functions together producing the desired (AND-OR) outputs. This basic AND-OR structure makes PLDs ideal for implementing logic equations in Boolean sum-of-products (SOP) form.

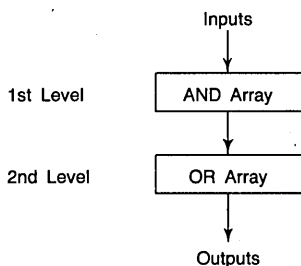


Figure 1-5. Basic Programmable Logic Device Architecture

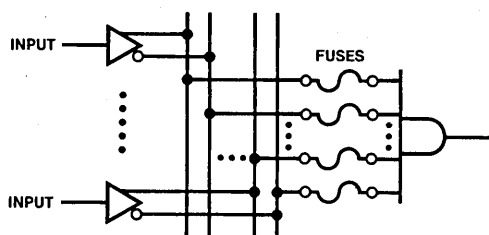
The major differences between these PLDs are in their programmability and their capability for supporting various logic features (Figure 1-6).

	PROM	PAL	PLA
AND	Fixed	Programmable	Programmable
OR	Programmable	Fixed	Programmable

Figure 1-6. Variation of AND-OR Programmability of PLDs

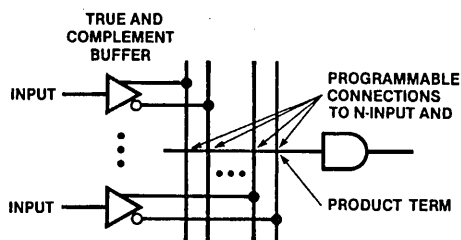
PLD NOTATION

For ease of use and better understanding, a simple convention has been adopted for programmable logic devices. Figures 1-7 and 1-8 depict these rules. Figure 1-7.a shows the logic equivalent of a programmable AND array and Figure 1-7.b shows the simplified conventional representation for an AND array. Figure 1-8 shows the equivalent technique for describing an OR array.



LD000710

Figure 1-7.a Programmable-AND Array Logic Equivalent



LD000701

Figure 1-7.b Programmable-AND Array Logic Diagram Notation

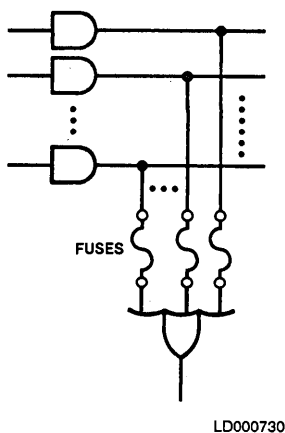


Figure 1-8.a Programmable-OR Array Logic Equivalent

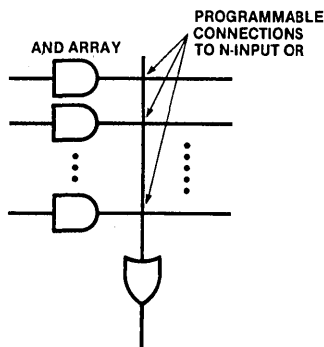


Figure 1-8.b Programmable-OR Array Logic Diagram Notation

The AND-gate inputs are represented by a single line, commonly described as the product term. All array inputs (true and complement of each device input) are shown connecting to a single input AND gate. In reality each input will have both of its true and complement routed to the AND array. Thus, an n input device will have AND gates with $2n$ inputs. For example, the AmpAL16R8 has sixteen inputs, but all these inputs and their complements (i.e., thirty-two lines) are routed

to each AND gate. In a programmable-AND array, each row and column intersection, as shown in Figure 1-9, represents a fusible input connection to the AND gate. The fuse state, either left intact or blown, determines the customized function of the device. An intact fuse connects the corresponding input to the product term, a blown fuse disconnects that input line from the product term.

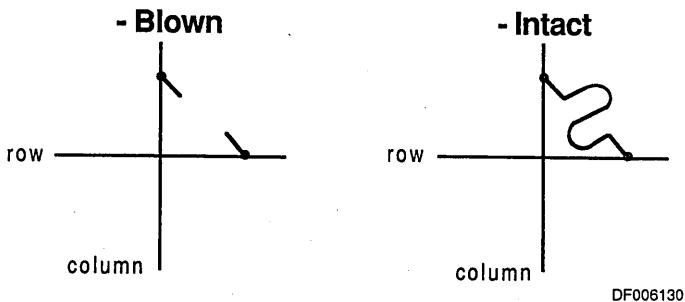
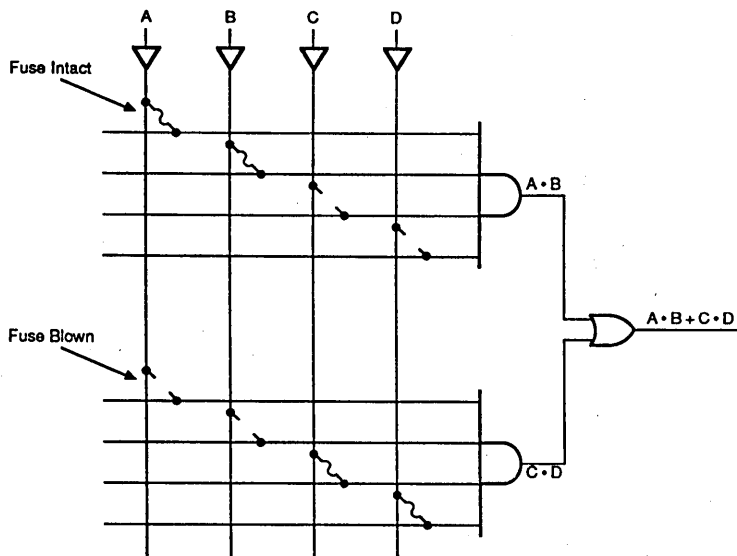


Figure 1-9. Fusible Arrays — Customizable Logic Functions Determined by Fuse State

Figure 1-10.a shows the fuse implementation of the logic AND-OR function $(A \cdot B + C \cdot D)^*$. To get the $A \cdot B$ function, the fuses connecting input lines A and B to the first product term are left intact, while the fuses connecting input lines C and D to the 1st product term are blown. To get the $C \cdot D$ function, the fuses connecting input lines C and D to the second product term are left intact, while the fuses connecting

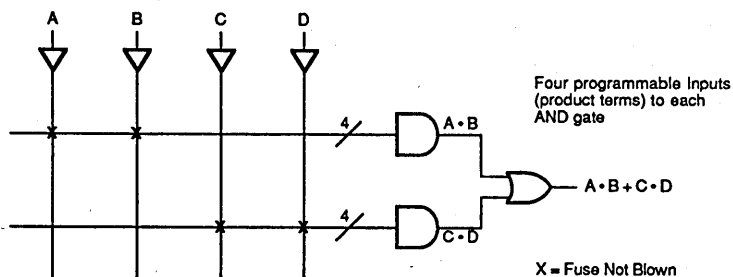
input lines A and B to the second product term are blown. In the example here, the inputs are shown in their true forms only, without their complements. Figure 1-10.b shows the simplified conventional representation of this function. An X at the intersection of input line and product line represents an intact fuse; a missing X represents a blown fuse.

*The symbol $\cdot$ or $\cdot$ represents the logical-AND function while $+$ represents the logical-OR function.



LD000680

Figure 1-10.a Fuse Implementation of AND-OR



LD000690

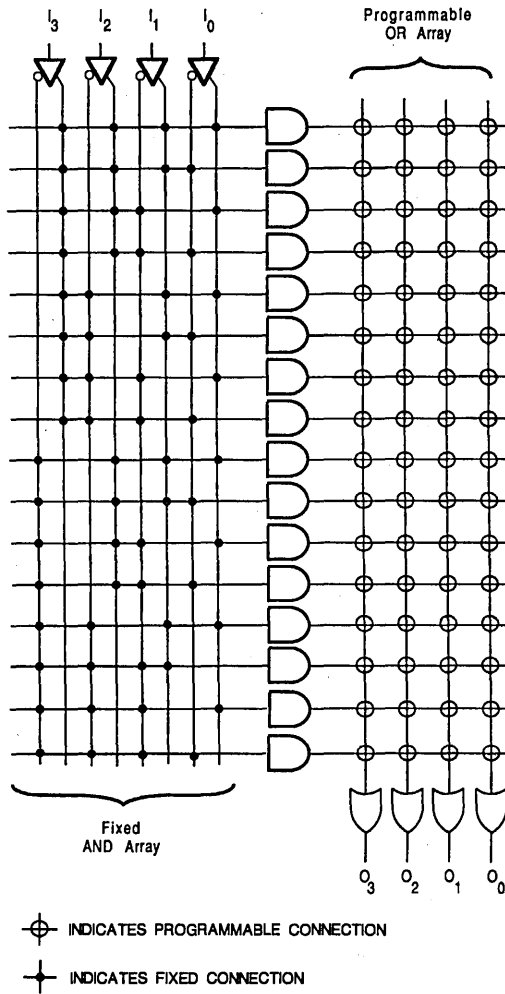
Figure 1-10.b Conventional Representation

Initially, all the AND gates of the programmable-AND array are connected, via fuses, to both the true and complement of every input. By selective programming of fuses, the AND gates may be "connected" to only the true input (by blowing the complement fuse), to only the complement input (by blowing the true fuse), or to neither type of input (by blowing both fuses) establishing a logical don't care.

An AND gate with all fuses blown assumes the logical-true (1) state. When all the true and complement fuses are left intact, an unconditional logical-false (0) results on the output of the AND gate. An AND gate with all of its input fuses intact is represented by an "X" within the AND gate.

ANATOMY OF A PROM DEVICE

Figure 1-11 shows the basic architecture of a very simple combinatorial PROM device using the notation of Figure 1-7, 1-8, and 1-9. The PROM shown has four inputs (with corresponding buffers), sixteen AND gates, and four OR outputs. The most important feature of the PROM architecture is that an array of fixed-AND gates feeds programmable-OR gates. The PROM inputs are fully decoded by the fixed-AND array. This means that every combination of inputs is represented by a separate AND gate, 2^n AND gates in a PROM with n inputs. For example, the PROM of Figure 1-7 has four inputs and has sixteen AND gates.



LD000671

Figure 1-11. PROM Array Structure

Notion of Min Terms and Product Terms

PROM devices incorporate the concept of Min term. Before we define a Min term, let us define first a Literal.

A Literal is either a variable or its complement. A Min term of n variables is a product of n literals containing all the variables in either true or complement form, but not both. For n variables, there are a maximum of 2^n Min terms. For example the variables A, B, C, have the following eight Min terms:

$A \cdot B \cdot C$
 $A \cdot B \cdot \bar{C}$
 $A \cdot \bar{B} \cdot C$
 $A \cdot \bar{B} \cdot \bar{C}$
 $\bar{A} \cdot B \cdot C$
 $\bar{A} \cdot B \cdot \bar{C}$
 $\bar{A} \cdot \bar{B} \cdot C$
 $\bar{A} \cdot \bar{B} \cdot \bar{C}$

While PROMs incorporate the concept of Min terms (since all the inputs are fully decoded), the PAL or PLA devices incorporate the concept of product terms. A product term of n variables is defined to be a product of 1 to n literals containing either one, two or many variables (up to n), in either true or complement form, but not both forms of the same variable. For example, for three input variables such as A, B, C there can be twenty-six different product terms:

- (6) $A, \bar{A}, B, \bar{B}, C, \bar{C}$
- (12) $A \cdot B, A \cdot C, B \cdot C, \bar{A} \cdot B, \bar{A} \cdot C, \bar{B} \cdot C$
 $A \cdot \bar{B}, A \cdot \bar{C}, B \cdot \bar{C}, \bar{A} \cdot \bar{B}, \bar{A} \cdot \bar{C}, \bar{B} \cdot \bar{C}$
- (8) $A \cdot B \cdot C, \bar{A} \cdot B \cdot C, A \cdot \bar{B} \cdot C, \bar{A} \cdot \bar{B} \cdot C$
 $A \cdot B \cdot \bar{C}, \bar{A} \cdot B \cdot \bar{C}, A \cdot \bar{B} \cdot \bar{C}, \bar{A} \cdot \bar{B} \cdot \bar{C}$

A PROM can be used for simple logic functions. Since the OR array is programmable, the outputs can be programmed individually from every possible input combination. This allows a PROM device to implement a separate and independent logic function on each of its outputs. Thus, each PROM output can implement any logic function limited only by the number of inputs available.

Since all the PROM inputs are fully decoded, applications such as look-up tables, character generators, code converters, and various function generators which require every input combination to be programmable are good candidates for PROMs.

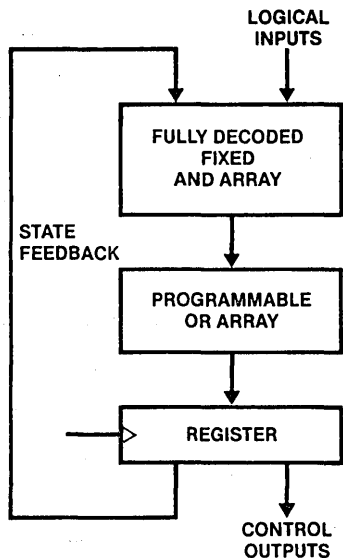
However, PROMs have a fixed number of inputs and a fixed number of outputs. For example, a $4K \times 8$ PROM has twelve inputs and eight outputs. Thus it needs 4,024 fixed-AND gates to fully decode its twelve inputs. This fixed input/output structure is a major limitation of a PROM device, especially for logic functions. A logic function requiring a different mix of inputs and outputs — even though the total number may be less than that offered by the structure of a PROM — will not be able to use that PROM device. Thus, a function requiring thirteen inputs and six outputs would not fit into the previously mentioned $4K \times 8$ device, even though it requires a smaller total number of inputs and outputs ($13 + 6$) than the device offers ($12 + 8$).

Another limitation of the PROM devices is that it is difficult to accommodate a large number of inputs. Each additional input, doubles the size of fuse matrix. For example, a ten-input, eight-output function requires a PROM with 8K fuses. Increasing the inputs to eleven increases the fuse array size to 16K fuses. Because of this, the largest PROM presently available is limited to fourteen inputs ($16K \times 8$).

Typical logic functions can easily have up to sixteen inputs which would require a PROM with 64K locations. For four outputs, this would require 256K locations. However, few

applications, especially for logic functions, would require all possible input combinations. A large number of fuses would therefore not be used. Also, typical output functions don't always come in data granularity matching the PROM width. For example, data path functions tend to be wider than the path itself because of additional functions such as parity bits, ripple carry, and serial inputs and outputs etc. Thus, these would not be well served by fixed width PROM sizes.

Various control path functions, such as state machines, can also quickly use up both inputs and outputs. A PROM with a register on the outputs as a state machine would require both logical inputs and state feedback inputs, while generating control outputs and state feedback outputs. Note that for each bit of state information, the feedback inputs and outputs are tied together, using up an input and output pin (Figure 1-12). Thus, when a large number of states are required, very few input and output pins are left over to do something else.



BD006700

Figure 1-12. Registered PROM State Machine

ANATOMY OF A PAL DEVICE

The array architecture of a PAL device is shown in Figure 1-13. The basic PAL structure is exactly opposite that of a PROM: the AND array is programmable and the OR array is fixed. Unlike PROMs, the inputs are not fully decoded. There are six inputs to the PAL array of Figure 1-13, but only sixteen AND gates (not 2^6 gates). Since the AND array of a PAL device is programmable, all the inputs need not be fully decoded. This helps to remove one of the key inefficiencies of a PROM (2^n gates for n inputs), allowing PAL devices to have considerably more inputs.

In other words: increasing the number of inputs does not result in a dramatic increase in the number of fuses. For example, increasing the number of inputs (from six to ten) for the example of Figure 1-13 increases the fuse array only from 12×16 , to 20×16 .

The fixed-OR array of a PAL device dictates to which OR gate any particular AND gate connects. In Figure 1-13, four AND gates are dedicated to each OR gate in the array. Since, for a PAL device, the output provides the sum of a limited number

of AND gates, the number of AND gates required by an equation must not exceed this number.

Besides their larger number of inputs, PAL devices contain many additional architectural features which make them ideal

for implementing logic functions. These features include programmable I/O pins, registered or combinatorial outputs with internal feedback to the array, outputs with programmable polarities, etc. (A detailed description of further features is provided later in this chapter)

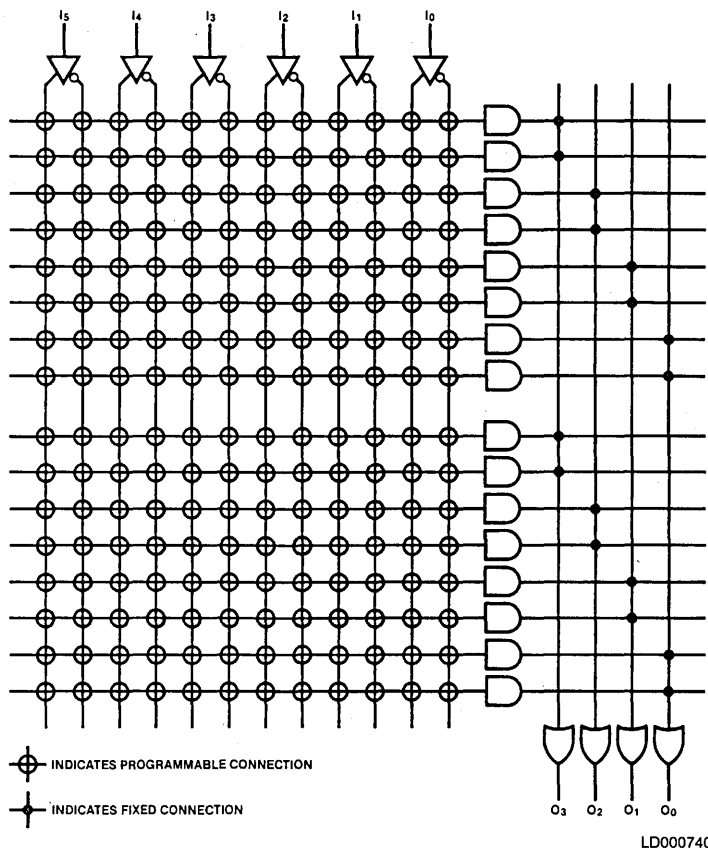


Figure 1-13. PAL Array Architecture

ANATOMY OF A PLA DEVICE

The array architecture of a PLA device is shown in Figure 1-14. The PLA architecture has both the AND array and the OR array user-programmable. This gives additional logic capability over PROMs and PAL devices. PLAs combine the advantages of PAL devices over PROMs (the programmable-AND plane), with the advantages of PROMs over PAL devices (the programmable-OR plane). PLA devices can include the same logic features which overcome the limitations of too few inputs, the allocation of inputs versus outputs, registered feedback, or output polarity, although few commercially available devices actually implement them. The programmable-OR array allows AND gates to be tied to OR gates, as desired, by blowing appropriate fuses. Here logic functions are limited by the total number of AND gates allocated to all outputs, instead of by the AND gates allocated to a particular OR gate in a PAL device. Thus, logic functions requiring a larger number of AND gates may be allocated to a particular OR gate appropriately.

Additionally, a PLA structure allows true sharing of AND gates for an output; the same AND gate may drive multiple outputs. This allows more efficient utilization of AND gates in a PLA than in a PAL device.

The disadvantages of PLAs are not quite so obvious. Because of the extra programmable-OR plane, a given signal has to pass through two programmable arrays; as a result, PLAs are inherently slower than PAL devices and PROMs. This can make a PLA unsuitable for many high-performance applications. Also, in practice the user can seldom take advantage of allocating a large number of AND gates to a particular OR gate. The number of AND gates required for a particular equation is related to the number of inputs to the equation. Equations using a large number of AND gates, with a large number of inputs pins, can be very cumbersome. Traditional logic design techniques such as Karnaugh maps cannot handle much more than five or six inputs, and computer aid for this task is not generally available. Also, because of the added

silicon real estate required for the programmable-OR array, most of the commercial PLAs have fewer AND gates than comparable PAL devices.

In data-path applications such as barrel shifters, the sharing of AND gates between outputs is almost impossible. Here individual equations are dependent upon their individual data

line (i.e., the equation for output Q0 is dependent on D0; Q1 is dependent on D1, etc). Here, PAL devices which do not have to provide AND-gate sharing for output, tend to fit better. Since the critical path of most systems is in the data path, PAL devices tend to be better suited for these applications, since they are faster than PLAs.

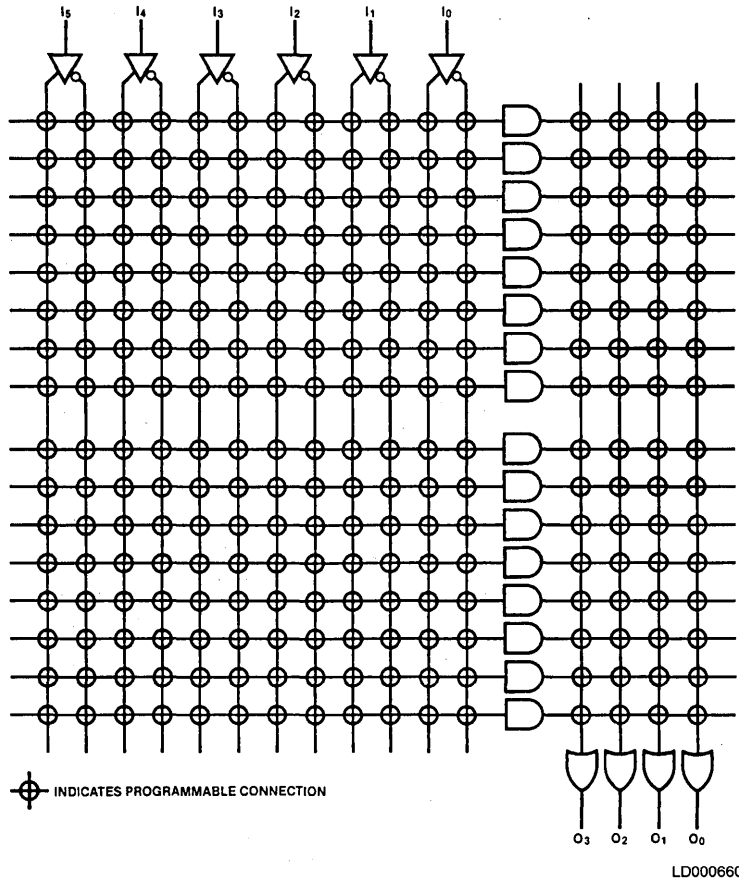


Figure 1-14. PLA Array Architecture

SUMMARY

PAL, PROM, and PLA devices represent three programmable logic architectures. Although very similar in basic array architecture, they differ significantly in their ability to implement logic functions, in their applications and in their programmability aspects.

Each device type implements an AND-OR two-level logic array which allows implementation of logic equations in SOP form. The PROM is the most limited of the three device types. While it is able to implement any logic function it has very few inputs to work with. In a PROM device, all inputs are fully decoded, all the Min terms are generated, and each output provides the sum of Min terms of all the input variables. The PROM has a fixed number of inputs and outputs and does not provide any architectural features to enhance logic design capability.

PLA devices, on the other hand, provide the most flexible architecture of the three for implementation of logic equations by using a programmable-AND array and a programmable-OR array. However, in practice the added flexibility of the PLA can seldom be effectively used. Further, the PLAs' inherent lower speed is unacceptable in high-performance designs.

The PAL device, sits in between. It provides significant capability to implement logic functions. Its programmable-AND array allows equations with many inputs. Its other architectural features such as programmable I/O, internal register feedback, and choice of output polarity allow optimization of pin allocation and logic equations.

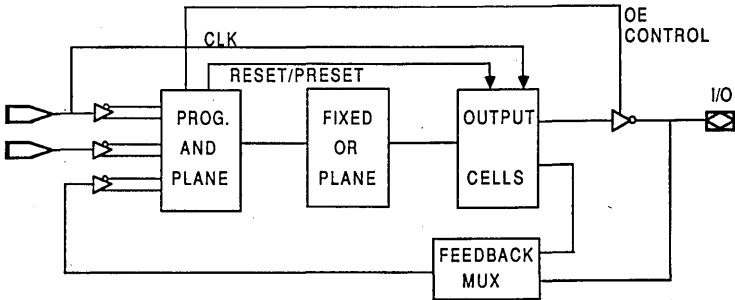
Table 1-4 summarizes the advantages and disadvantages of the three programmable logic architectures.

TABLE 1-4. COMPARISON OF THREE DIFFERENT PROGRAMMABLE LOGIC ARCHITECTURES		
PROM	PAL	PLA
Full decoding of all input variables	Partial decoding of all input variables	Partial decoding of all input variables
Generate all Min terms	Generate limited number of Product Terms	Generate limited number of Product Terms
Each output provides sum of Min terms	Each output provides sum of limited number of Product Terms	Each output can provide sum of all the Product Terms
Limited Architecture features	Additional Architecture features	Additional Architecture features
Speed—Faster	Speed—Fastest	Speed—Slowest
Flexibility		
Limited	Better than PROM	Maximum
Easy to understand and use	Fairly easy to understand and use	Hard to understand and use

DETAILED ARCHITECTURE OF PAL DEVICES — CAPABILITIES

Figure 1-15 shows the detailed architecture of a typical PAL device, and its important architecture variables. Four most important components of PAL architecture are:

- AND plane
- OR plane
- Storage elements
- I/O pins



BD006690

Figure 1-15. PAL Architecture

PAL Architectural Variables

- | | | | |
|-----------------------|------------------------|--------------------------|---------------------|
| • Number of Inputs | • Number of OR Outputs | • Nature of Output Cells | • Dedicated I/O |
| • Number of AND Gates | • Number of PTs/Output | • Nature of Feedback | • Bidirectional I/O |
| | • Distribution of PTs | • Prog. Polarity | |
| | | • Number of Banks | |
| | | • Clock Control | |
| | | • RESET/PRESET Control | |
| | | • PRELOAD | |

Architecture of the AND Plane

This plane provides the interconnection of inputs (both true and complement) to the AND gates to form both logical and control product terms. Logical product terms are used for logic functions and control product terms are used for control

functions such as Output-Enable control, RESET, PRESET, PRELOAD, and observability.

The total number of inputs and product terms determine the size of the AND plane. While the first-generation PAL devices had up to 2K fuses and 64 product terms (each with 32 inputs), the trend is towards larger AND planes.

Architecture of the OR Plane

This plane determines the connectivity of the "logical ORing" of the AND gates to the outputs, and defines three things: the number of OR outputs, the number of product terms (PTs) per output, and the distribution of PTs per output whether same or variable.

In a typical PAL architecture, the outputs of the AND gates are connected to fixed-OR gates. The limitation of the PAL devices' AND-OR plane is the number of inputs to the AND gates, the number of AND gates, and the number of "OR" outputs. While first-generation devices had seven-eight logical product terms per output and a maximum of eight OR outputs, the trend is toward a larger number of outputs with more product terms per output.

Architecture of Output Cells

The architecture of the output cells specifies at least the following:

- Nature of output
- Nature of flip-flop used as storage element
- Organization of outputs
- Flexibility of feedback path

The output(s) can be configured to have either sequential (registered) or combinatorial capability, as well as polarity control for active HIGH and active LOW.

The flip-flops used for output cells can be one of the following: edge-triggered D-type flip-flops, J-K, S-R, or T flip-flops, or latches. Flip-flops are good for clocked synchronous system designs, while latches are good for asynchronous logic applications.

D-type flip-flops are the easiest to design with, while J-K flip-flops are probably the most flexible. However, there is a trade-off especially on the OR array for implementing either D or J-K type flip-flop. A disadvantage of the J-K flip-flop is that both the inputs have to be driven by the OR array. This increases the OR-array size. The disadvantage of the D flip-flop is that it requires a HOLD term to hold a particular state; J-K flip-flops do not need HOLD terms. It is conceivable that a given logic function could be implemented with a smaller number of product terms with J-K rather than D flip-flops. However, this is application dependent. For state-machine designs involving a large number of frequent state transitions, the benefits of J-K over D becomes less important.

Traditionally, for PAL devices, speed and architecture simplicity have been the most important criteria and most PAL devices use D-type flip-flops only. Lately even other types of flip-flops are being offered on PAL devices.

The organization of the output cells determine whether all the cells are organized as a single bank (with a common clock), or configured into multiple banks with separate clocks.

The output cell structure also determines the flexibility of the feedback path. The feedback may be from the combinatorial output, registered output or from the I/O pins. The feedback paths can be either a single or multiple lines for increased flexibility.

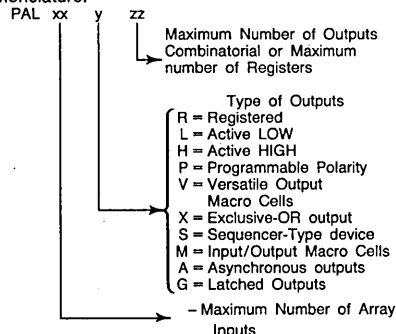
Architecture of I/O Pins

The I/O pin architecture determines whether a pin may be defined as a dedicated input pin, dedicated output pin, or a dynamically-controllable I/O pin.

While the first-generation devices had a limited number of dedicated input pins, few output pins, and few programmable I/O pins, the newer PAL-type devices have more programmable I/O pins.

PAL Nomenclature

PAL devices are known in the industry by the following nomenclature:



For example, the AmPALHCT29M16 (which is a 24-pin CMOS PAL device) has a maximum of twenty-nine inputs to its AND array, and sixteen input/output macrocells. Since the maximum number of inputs (twenty-nine) plus the outputs (sixteen) exceed the total number of pins in the package, it implies that the device has feedback and/or bidirectional I/O pins.

PAL ARCHITECTURAL FEATURES AND BENEFITS

PAL devices contain many architectural features which make them ideal for implementing logic functions. These features include:

- Programmable I/O pins
- Flexible Output-Enable control and Bidirectional I/O
- Dedicated versus programmable output structure
- Programmable polarity
- Flexible clocking scheme
- Buried state registers
- Miscellaneous features
 - Accessibility
 - Controllability
 - Testability
 - Observability

Programmable I/O Pins

Programmable input/output pins are one of the most important resources of a PAL device. They allow the PAL device to be tailored to fit the required allocation of inputs and outputs. PAL devices can thus implement far more complex and different logic functions than a PROM — even one with more pins.

Flexible Output Enable Control & Bidirectional I/O

Logic diagrams for the bidirectional output structures of the PAL devices are shown in Figures 1-16.a and 1-16.b. One important feature of the PAL devices' bidirectional output is the flexibility in controlling the Output Enable.

The Output Enable can be either dedicated (controlled by a pin) or programmable (controlled by a product term from the AND array).

The output buffer associated with the output pin may be programmed in one of three ways: as a dedicated output, a dedicated input, or a dynamically-controllable input/output.

When programmed as a dedicated output, the output buffer is always enabled and the logic function is fed back to the AND array. This feedback path allows more complex logic functions

to be implemented by using two or more levels of AND-OR gating.

When programmed as a dedicated input, the AND-OR gate associated with that pin is unused. This ability to trade off outputs for inputs is one of the big advantages of PAL devices over other programmable logic devices, especially PROMs. The designer is no longer limited to a fixed number of input and output pins. The ratio may be programmed to fit the intended application.

Finally, when programmed as a dynamically controllable input/output buffer (i.e., enabled/disabled by a logical combination of one or more inputs) this pin may be used as an input, while retaining the full logical capability of the AND-OR gate. This is especially useful in control applications (microprocessor handshaking protocols) and bus-oriented data operations (data steering and data storage/manipulation). A serial input/output pin is a common example. When shifting left the pin is a serial input, but when shifting right the pin is a serial output. This mode provides maximum utilization of the PAL architectural resources.

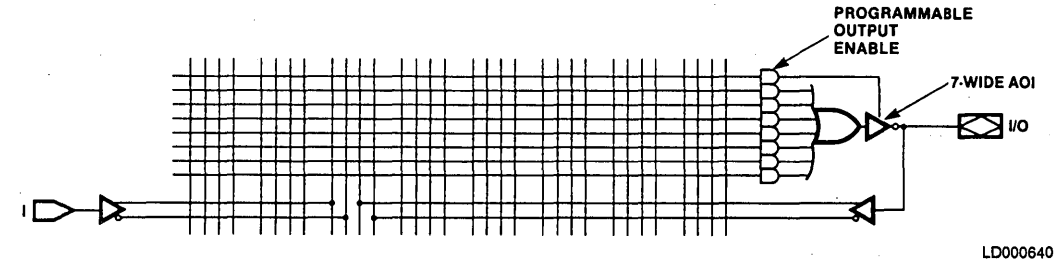


Figure 1-16.a Active-LOW Bidirectional Output

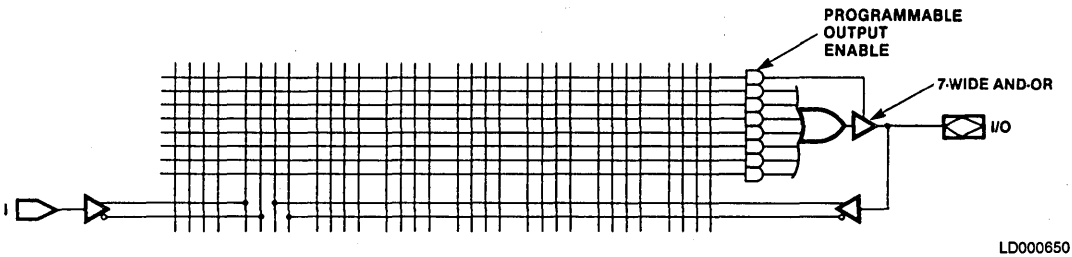


Figure 1-16.b Active-HIGH Bidirectional Output

Figures 1-17.a and 1-17.b show the active-LOW and active-HIGH versions of PAL devices with dedicated outputs. Here the outputs are always enabled and the AND gate previously used for the Output-Enable function can be used for an extra logic product term. The feedback path from output to input is

still provided, allowing for implementation of multi-level logic. This extra AND gate makes these outputs ideal for non-bus-oriented logic replacement, especially complex control-signal generation, encoding, and decoding.

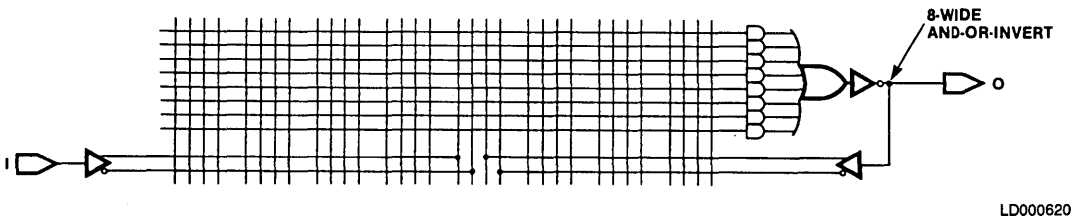
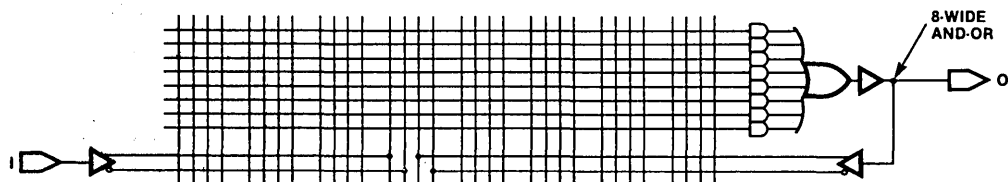


Figure 1-17.a Active-LOW Dedicated Output

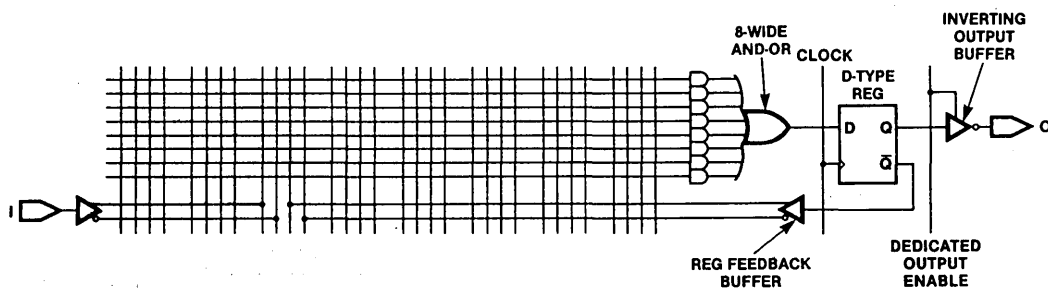


LD000630

Figure 1-17.b Active-HIGH Dedicated Output

Individual product-term control for each Output-Enable function gives the designer the ability to configure each output on an individual basis. On the other hand, a common, dedicated Output Enable (Figure 1-18) makes registered PAL devices ideal for bus-oriented systems. The registered PAL device can

be programmed to provide data storage, operation, or steering functions, the result of which is placed on the data bus by enabling the output buffer. Since most PAL devices have 24 mA current sinking capability, they can drive most on-board buses and many backplane buses.

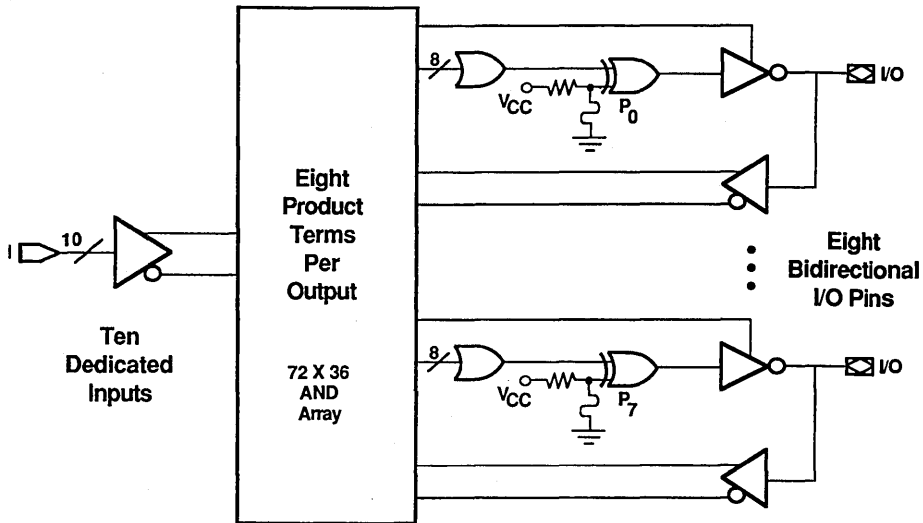


LD000610

Figure 1-18 Registered Output

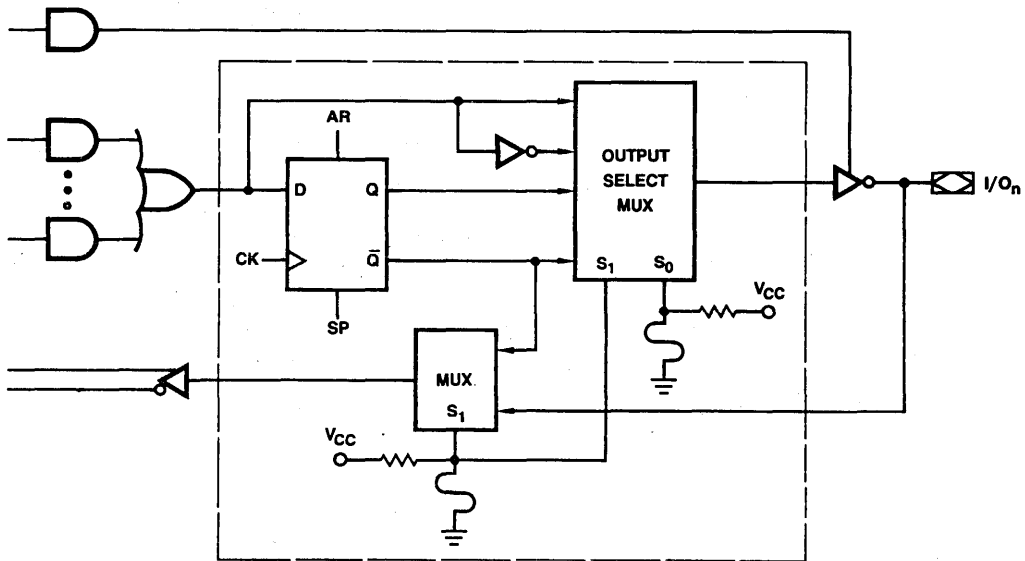
Figures 1-19 and 1-20 show the output structure of second-generation devices such as the AmPAL18P8 and the AmPAL22V10, where each output is controlled by a separate Output Enable product term. Figures 1-21.a and 1-21.b show the output structure of the AmPAL23S8. Here, besides a

separate product-term control for each output, there is a polarity fuse. This polarity fuse allows the designer to control the Output Enable as a combination of various signals (DeMorganized equations) rather than a single AND term. This is especially useful in bus-control applications.



BD006680

Figure 1-19. Block Diagram of AmPAL18P8

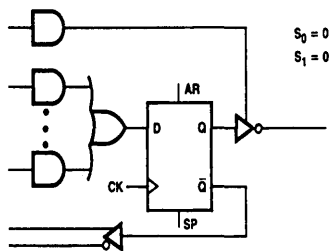


LD000411

Figure 1-20.a AmPAL22V10 Output Logic Macrocell Diagram

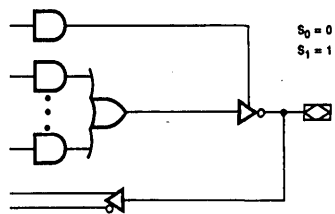
S_1	S_0	Output Configuration
0	0	Register/Active LOW
0	1	Register/Active HIGH
1	0	Combinatorial/Active LOW
1	1	Combinatorial/Active HIGH

0 = Unblown Fuse
1 = Blown Fuse



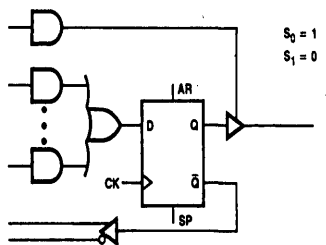
LD000420

Registered/Active LOW



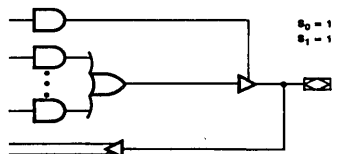
LD000430

Combinatorial/Active LOW



LD000440

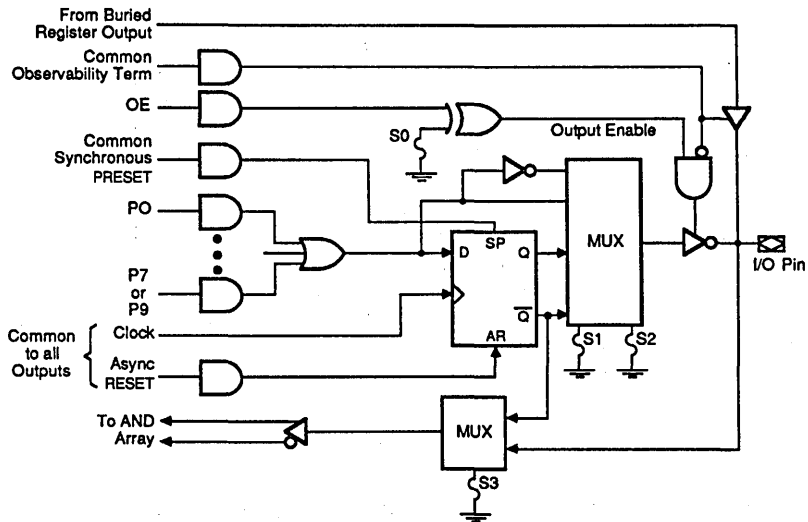
Registered/Active HIGH



LD000450

Combinatorial/Active HIGH

Figure 1-20.b Output Configurations of AmpAL22V10



LD000581

Figure 1-21.a AmpAL23S8 Output Logic Macrocell

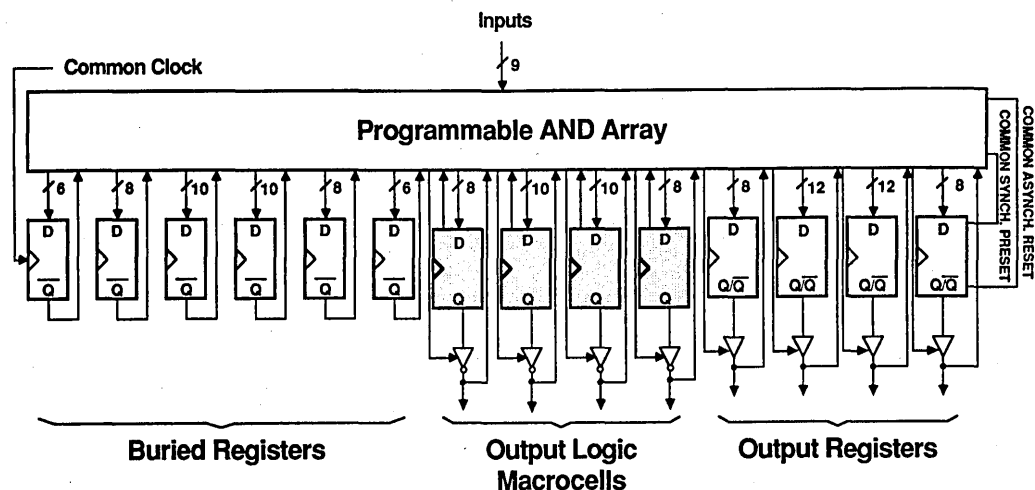
Buried State Registers

The output storage elements of the PAL device are used either as outputs or as state bits. Typically these output storage elements are associated with I/O pins, and are driven by the AND-OR array. The number of output/state registers, I/O pins, and array size (product terms) are the three most important resources of programmable logic devices. These three resources are always in short supply. For optimum state-machine design, system designers always strive to achieve an optimum balance of these resources.

Traditional first-generation devices such as the 16R4/16R6/16R8 have been used as SSI/MSI logic replacement and for doing state-machine designs. However, because of a limited number of registers, only simple state machines can be

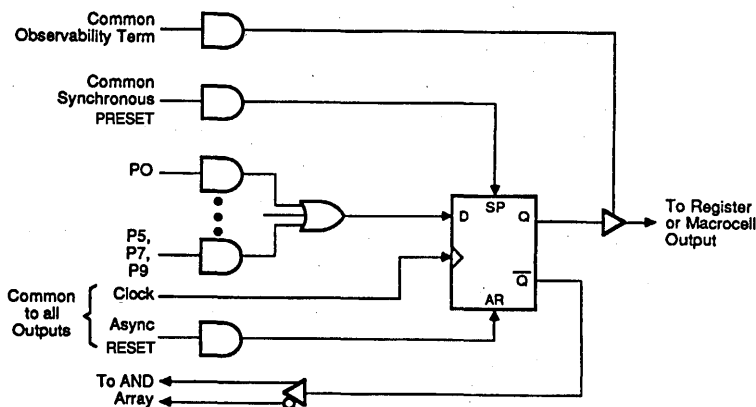
designed with these devices. For more complex state-machine designs, designers have asked for dedicated buried state registers. These registers are driven from the same AND-OR array, but, they are not tied to the output pins. Therefore, they are called "buried registers." However, they are accessible and controllable by the AND-OR array, just like the output registers. These buried registers provide extra functionality. They could be used for keeping track of various "internal flags" for generating timing and various other internal control information, without tying up the valuable I/O pins.

Figure 1-22 shows the architecture of the AmPAL23S8, the industry's first bipolar PAL-based sequencer device. As seen from the block diagram, this device offers six dedicated buried registers in addition to eight output registers. Figure 1-23 shows the architecture of its buried registers.



BD006670

Figure 1-22. Block Diagram of AmPAL28S8



LD000571

Figure 1-23. AmPAL23S8 Buried State Register

MISCELLANEOUS FEATURES — ACCESSIBILITY, CONTROLLABILITY, TESTABILITY AND OBSERVABILITY

PAL- and PLA-based sequencer devices need a large number of internal registers that are easily accessible, controllable, testable, and observable.

Power-up RESET

Power-up RESET resets all internal registers during system power-up. All the registered devices in the AMD PAL Family have been designed to reset automatically during system power-up. This feature is especially valuable in simplifying state-machine initialization.

Due to the asynchronous operation of the power-up RESET and the wide range of possible V_{CC} rise time, certain conditions are necessary to insure a valid power-up reset. For AMD's PAL devices, these conditions are:

- The V_{CC} rise must be monotonic
- Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.

RESET/PRESET

The ability to RESET and PRESET registers increases the system functionality.

These RESET/PRESET functions can be asynchronous or synchronous, and can be controlled either by a dedicated pin or by a product term driven from the AND array.

If these functions are driven by product term(s), these could be either a common product term or individual product terms. When the synchronous product term is asserted (HIGH), the output registers will be loaded with a HIGH on the next LOW-TO-HIGH clock transition. When the asynchronous RESET product term is asserted (HIGH), the output registers will be immediately loaded with a LOW (independent of the clock). These functions are particularly useful for applications such as system power-on and reset.

PRELOAD

PRELOAD allows any arbitrary value to be loaded into the PAL device's output registers. AMD's registered PAL devices are designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality.

PRELOAD is the only way to allow full logic verification of programmed registered PAL devices and thus guarantee correct logical functionality. Without PRELOAD, many device

failures cannot be discovered until the device is tested as a part of the finished system.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is clocked into the "next state," which is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to test an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state registers "interferes" with the inputs, forcing the machine to go through many transitions before it can reach a particular state value. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

In addition, complete logic verification may become impossible when states that need to be tested are never entered with normal state transitions. "Forbidden" or don't-care states that are not normally entered need to be tested to ensure that the state machine returns to a valid state.

PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper system operation.

Observability of Buried State Registers

The AmPAL23S8 is the first PAL device to offer product-term controlled, observable buried state registers. The observability product term, driven by a common AND-OR array, allows the system designers to observe buried registers under pin control or product-term control (by a combination of various input signals). The contents of the buried registers can be monitored for system debugging purposes.

An observability product term controls a set of six inverting buffers, which serve both the six buried registers and six output registers (four output registers and two output logic macrocells). The observability product term causes the six buffers to disable signal flow from the six output registers to the I/O pins, and enables signal flow from the six buried registers to the respective I/O pins (Figures 1-21.a and 1-21.b). This feature is especially useful for system-level testing and debugging.

SUMMARY

Table 1-5 summarizes different architectural features and their benefits for the system designers.

TABLE 1-5. PAL ARCHITECTURAL FEATURES AND THEIR BENEFITS

Features	Benefits
Programmable I/O Pins	• Removes limitation of fixed I/Os • Offers variable number of I/Os • Allows allocation of I/Os based on application requirements
Bidirectional I/O Capability	• Dedicated inputs • Dedicated outputs • Dynamically controllable I/Os
Programmable Polarity	• Simplifies deMorganization of equations
Registered Outputs with Feedback	• Simplifies state-machine design
Combinatorial Outputs with Feedback	• Allows multiple levels of logic
Output Logic Macrocell	• Variable number of registered/combinatorial outputs • Simplifies Mealy-Moore-type state-machine design
Variable Distribution of Product Terms	• Better application fit
Flexible Clocking Scheme	• Better synchronous design • Eases pipelined system design
Product-Term-Driven Clocks	• Offers multiple clocks • Good for glue-logic applications
Buried State Registers	• Frees up I/O Pins • Offers more complex state-machine design capability
Asynchronous RESET/PRESET	• Better system initialization
PRELOAD	• Better testability
Power-Up RESET	• System initialization
Observability of Buried State Registers	• Simplifies system testing and debugging

1.3 HOW TO DESIGN WITH PROGRAMMABLE LOGIC DEVICES

1

INTRODUCTION

Programmable Logic Devices (PLDs) with their programmable-AND-OR array structures are ideal for implementing Boolean logic functions expressed in sum-of-products (SOP) form. Any logic function can be implemented in a PAL device as long as the number of inputs, outputs, and the number of product terms required do not exceed what is available on the device.

The SOP form can be derived from truth/function tables, state or timing diagrams, and Karnaugh maps. High-level software packages are available to ease this derivation.

Logic Equations

Digital systems are based on the Boolean logic system, which processes only two types of values: 0 and 1. These two values are processed through electronic circuits known as gates to produce an output. Combinations of such gates can be used to implement a logical equation that generates outputs based on combinations of inputs. These outputs can be interpreted as data or used as inputs to other logic networks.

All digital logic can be expressed in terms of three fundamental logic gates: AND (*), OR (+), and NOT (/). These logic gates or functions are manipulated using Boolean algebraic

theorems and laws to simplify and reduce complex logic expressions.

Implementing Boolean Equations in PAL Devices

The programmable-AND and fixed-OR structure of a PAL device require that logic expressions be expressed in the SOP form. In SOP form, the maximum number of logic levels is two: an AND and an OR operation. Any NOT or invert operation is assumed to occur before the AND plane of the PLD and therefore does not contribute to the propagation delay through the PLD AND-OR plane.

For example: $A \cdot (B + C \cdot (D + E))$ in Figure 1-24 is a four-level logic expression. The subexpression $D + E$ must be evaluated before ANDing with C . This is then ORed with B before finally ANDing with A . The logically equivalent SOP form is: $A \cdot B + A \cdot C \cdot D + A \cdot C \cdot E$.

The SOP equation maps directly into the PAL device structure: each product term ($A \cdot B$, $A \cdot C \cdot D$, and $A \cdot C \cdot E$) is programmed into the AND array of the output dedicated to this logic expression. This output must have at least three product terms to express this logic equation (Figure 1-25).

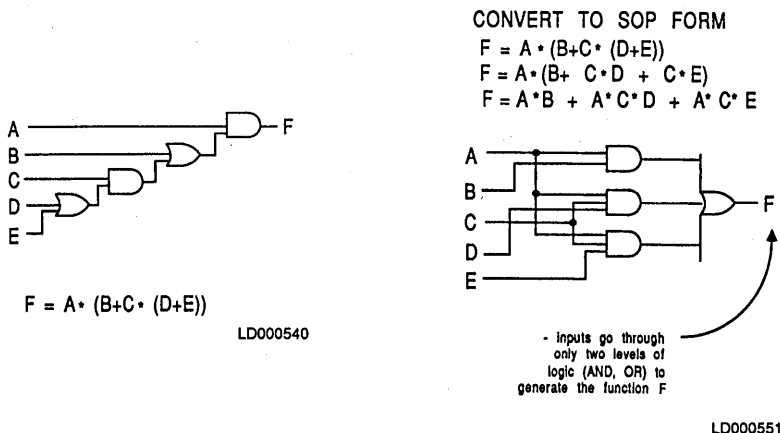
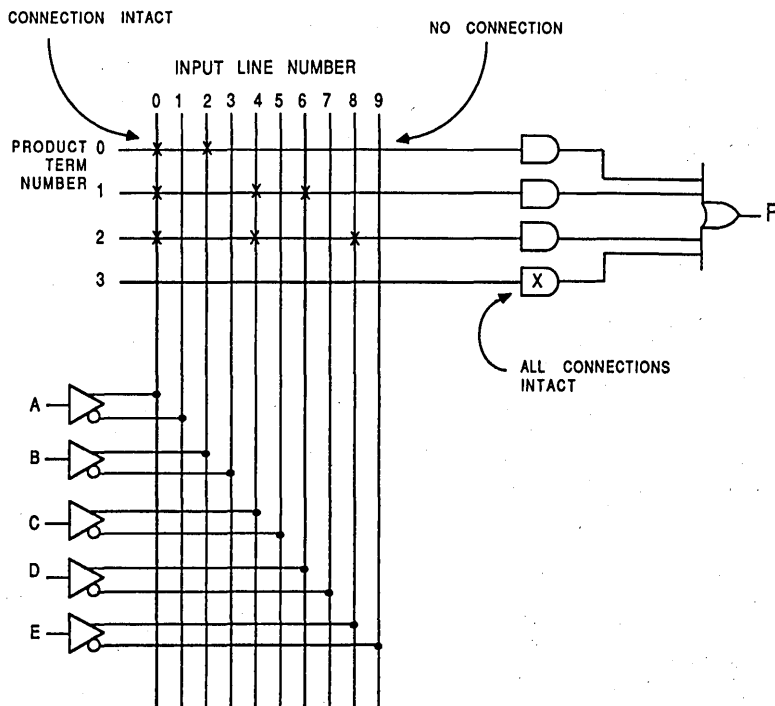


Figure 1-24 A Simple Combinatorial Logic Expression



LD000560

Figure 1-25 Logic Expression Mapped Into a PAL Device

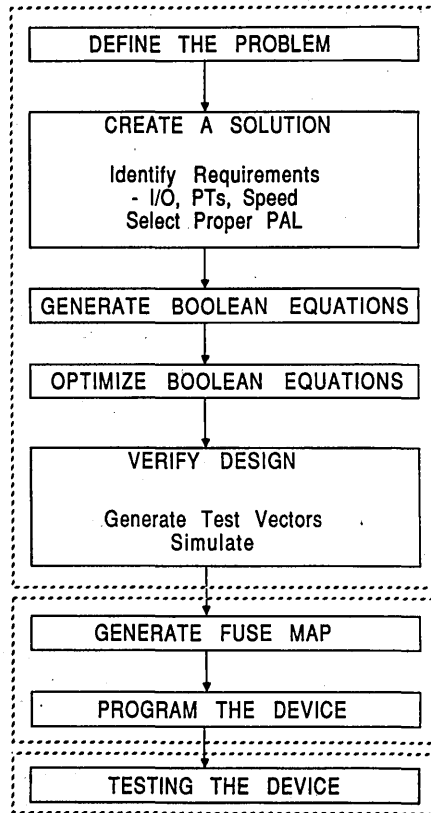
From Concept to Implementation

Designing with programmable logic usually involves three phases: design, programming, and testing. In the design phase, you specify Boolean functions, i.e., logic equations for solving the problem. In the programming phase, the logic equations are converted into SOP expressions, which are then

translated into a device fuse map used to program the device. The test phase ensures that the programmed logic device performs the functions specified.

DESIGN PHASE

The design phase can be further partitioned into a number of basic steps (Figure 1-26).



BD006790

Figure 1-26. PAL Design Process

Define Problem

Defining the particular logic function to solve the problem is the first step in the design phase. Here you identify the nature of the problem to be solved: whether a combinatorial function (such as address decoding, priority encoding, data multiplexing/demultiplexing, or generating control signals), or a sequential function (such as counting, data shifting, or implementing a particular state machine).

Identify Design Requirements

After you have defined the problem, choose a device based on the design requirements: number of input/output pins, number of product terms, registered or combinatorial outputs, polarity, power consumption and speed.

Generate Boolean-Logic Equations

Identification of inputs, outputs, and signed polarity may be easy. However, the number of product terms used to solve the design problem is determined only after the necessary Boolean logic equations have been generated.

Generating Boolean-logic equations for combinatorial logic is relatively straightforward. You start with the inputs, outputs, and the truth table. You can derive logic equations from the truth table by grouping the "1s" (for active-HIGH outputs) or "0s" (active-LOW outputs). Grouping the "0s" instead of

"1s" has the effect of inverting the equations. This is a convenient and common technique for generating inverted logic for use in active-LOW PAL devices.

Generating logic equations for sequential circuits is more complex. You have to define the inputs, outputs, and states. You then draw a state diagram, specifying all the appropriate state transitions to generate the state table. Regular digital design methods are then used to assign states such that redundant states are eliminated to obtain the minimal number of states.

After the states have been determined, assign state numbers and use logic minimization techniques to arrive at the minimum amount of logic necessary to implement the sequential function.

The logic equations derived from truth tables, state diagrams and Karnaugh maps are converted into the SOP form to fit into a PAL device. You can use computer-aided design (CAD) tools to greatly speed up the logic design process. These CAD packages allow you to express the design solution in a high-level syntax such as logic schematics or logic language descriptions. They also take the truth tables, state diagrams, flowcharts, and other high-level descriptions of the solution and automatically generate logic equations in SOP form. The equations can be minimized if they exceed the number of

product terms available on the device. These CAD packages are relatively inexpensive and available from a number of vendors including DATA I/O (ABEL*), Personal CAD systems (CUPL*), and AMD (AmCUPL, and PLPL).

We will demonstrate designing with a PAL device with a simple design example. We want to implement the following combinatorial logic function (Figure 1-27):

$$\begin{aligned} O1 &= /I1 \\ O2 &= /I1 \cdot I2 \\ O3 &= I1 + I3 \\ O4 &= /(I3 \cdot I4) \\ O5 &= /(I3 \cdot I5 \cdot I6 + I7 + I8 \cdot I9) \\ O6 &= /(I8 \cdot I9 + /I3 \cdot /I7 \cdot I9 \cdot I10) \end{aligned}$$

Figure 1-27. Logic Design Example

For this example, the optimized Boolean equations can be written in the SOP form:

$$\begin{aligned} O1 &= /I1 \\ O2 &= /I1 \cdot I2 \\ O3 &= I1 + I3 \\ O4 &= I3 + /I4 \\ O5 &= (I3 + /I5 + /I6) \cdot /I7 \cdot (/I8 + /I9) \\ &= (I3 \cdot /I7 + /I5 \cdot /I7 + /I6 \cdot /I7) \cdot (/I8 + /I9) \\ &= I3 \cdot /I7 \cdot I8 + /I5 \cdot /I7 \cdot I8 + /I6 \cdot /I7 \cdot I8 \\ &\quad + I3 \cdot /I7 \cdot I9 + /I5 \cdot /I7 \cdot I9 + /I6 \cdot /I7 \cdot I9 \\ O6 &= (/I8 + /I9) \cdot (I3 + I7 + /I9 + /I10) \\ &= /I8 \cdot I3 + /I8 \cdot I7 + /I8 \cdot I9 + /I8 \cdot /I10 \\ &\quad + /I9 \cdot I3 + /I9 \cdot I7 + /I9 \cdot I9 + /I9 \cdot /I10 \end{aligned}$$

Outputs O5 and O6 have six and eight PTs, respectively. Using logic minimization (available with most PLD design packages), O6 can be reduced to the following equation with only four PTs.

$$O6 = /I9 + /I8 \cdot I3 + /I8 \cdot I7 + /I8 \cdot /I10$$

Logic minimization reduces the number of PTs necessary to represent a function such that the logic expression can fit into an output on a PLD. If the number of PTs after minimization still exceeds the number of PTs for the output pin, then you will have to use a larger PLD with more PTs per output.

Inverted Logic Expressions

In the above example, we have specified active-HIGH outputs. However, if you wanted to use an active-LOW PLD such as the AmPAL16L8, the equations in Figure 1-27 would have to be converted from active HIGH to active LOW by applying DeMorgan's theorem as follows:

$$\begin{aligned} /O1 &= I1 \\ /O2 &= I1 + /I2 \\ /O3 &= /I1 \cdot /I3 \\ /O4 &= /I3 \cdot I4 \\ /O5 &= /I3 \cdot I5 \cdot I6 + I7 + I8 \cdot I9 \\ /O6 &= I8 \cdot I9 + /I3 \cdot /I7 \cdot I9 \cdot I10 \end{aligned}$$

The equations are in the SOP form and can map directly into a PLD.

Note that more advanced PLDs such as the AmPAL22V10 have output macrocells which give the user many output options. These outputs can be programmed with HIGH or LOW polarity. Since the output pin is programmable, you can choose between the active-HIGH or active-LOW implementation, whichever, gives the lowest number of PTs per output.

Verify Design

After you have generated the logic equations, you can perform logic simulation on the resulting equations to test for correct functionality before actually programming a part. This is an inexpensive and fast way to catch mistakes. The simulation procedure uses test conditions or vectors which specify the inputs to and expected outputs from a PLD. Some CAD systems can generate the test vectors automatically, or you can generate them manually.

If manual test vector generation is performed, it is usually done by the logic designer who knows the design best. The designer creates a function table showing the inputs and expected outputs into and from the device. A CAD package can then convert the data in the function/truth table into a valid test vector format acceptable by the simulator and the PLD programmer/tester to be used.

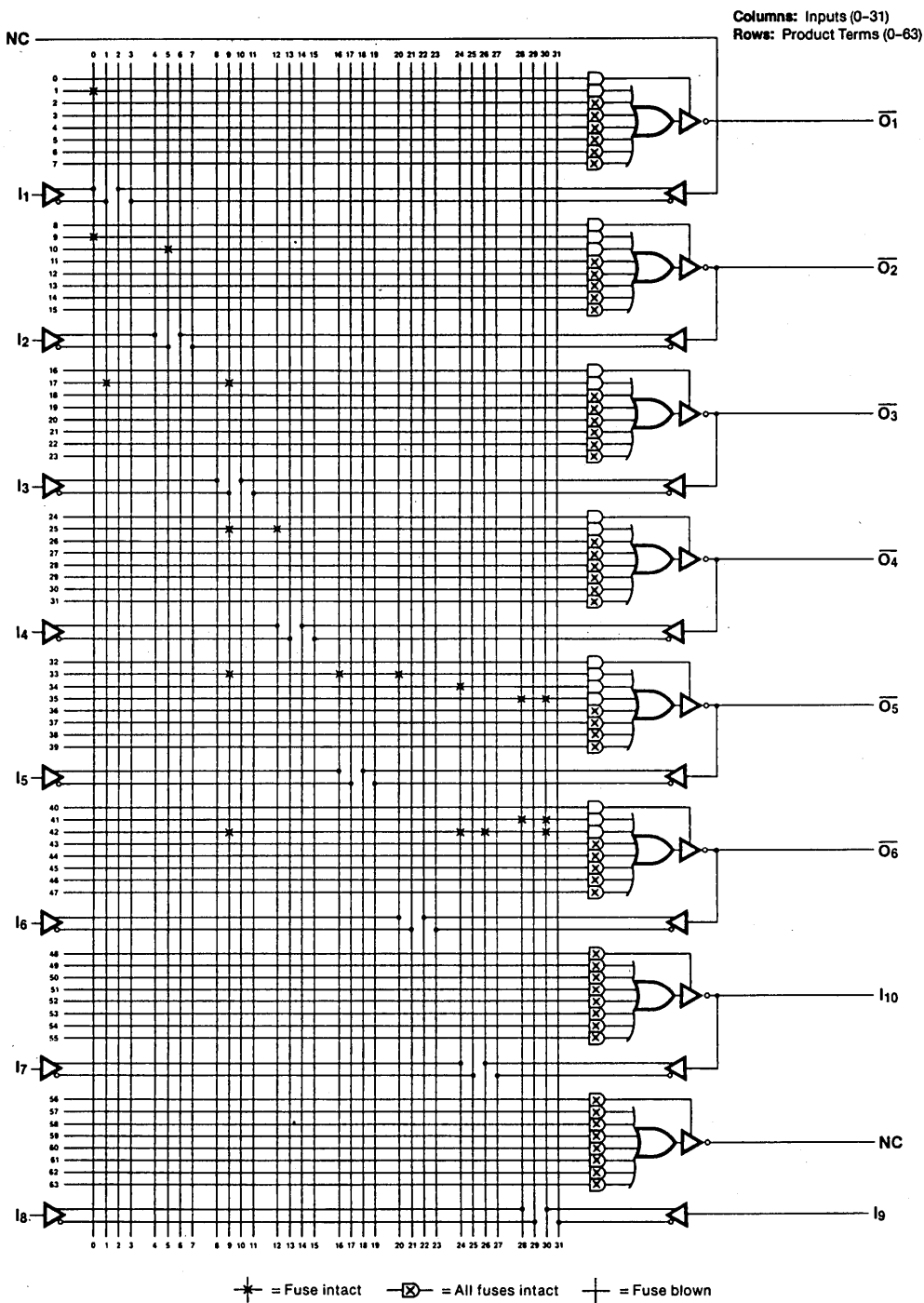
PROGRAMMING PHASE

Once you are satisfied with the results of logic simulation, a device map can be generated from the equations. Figure 1-28 shows the logic diagram of the AmPAL16L8 for implementing above equations. We have arbitrarily assigned outputs O1-O6 to pins 14-19, and the inputs I1-I10 to pins 2-9, 11, and 13.

O1 is assigned to pin 19. To make this output the inverse of I1, connect input line 0 to product term 1 and disconnect all the remaining links for this product term. For fuse-based PLDs, you have to selectively make disconnections by blowing fuses. Connections are indicated by an X at the intersection of input line 0 and product term 1 in Figure 1-28.

Since the other PTs for the "O1" output OR gate are unused, their outputs are forced to zero by connecting all the links. When all the links are connected, both the true and complement value of an input are fed to a PT, resulting in a logic zero. In Figure 1-28 unused PTs (with all connections intact) are indicated by Xs in the AND gates at the OR gate inputs.

The final consideration for "O1" output is the output enable. The logic diagram and logic equation for the function "O1" do not show output enable function. This means that "O1" should be always enabled. In Figure 1-28, PT 0 controls the Output Enable function for "O1". PT 0 is always TRUE/HIGH when all the links in a PT are disconnected.



LD000530

Figure 1-28. Logic Diagram for Example of Using the AmpPAL16L8

The next output /O2 is the OR function of I1 and /I2. The equation $/O2 = I1 + /I2$ is represented in the PLD by leaving input line 0 (I1) connected to PT 9 and disconnecting the rest of the links on this PT. Then since an OR function is needed, we move to the next PT and leave input line 5 (/I2) connected while disconnecting the other links in PT 10. Since PT 11 through 15 will be unused, we indicate this by putting an X in the AND gates at the input of the NOR gate. Function O2 is also always enabled, hence no X is put in the AND gate representing the enable product term PT 8, indicating that all the links in PT 8 have been disconnected.

Output /O3 is the AND of /I1 and /I3. To implement this, input line 1 (/I1) is connected to PT 17. Since we want an AND function, input line 9 (/I3) is also connected to PT 17. These connections are represented by Xs. The other links in PT 17 are disconnected. Since the rest of the PTs are unused, an X is placed in the AND gates for PT 18 through 23. The Output-Enable PT for /O3 is also left blank, which will always enable the output /O3.

Output /O4 is similar to /O3. To generate the AND function, input line 9 (/I3) and input 12 (/I4) are connected to PT 25, while the rest of the links in PT 25 are disconnected. The other PTs are unused, and the output is always enabled.

Output /O5 is generated by ANDing /I3, I5, and I6 on PT 33, connecting I7 to PT 34, ANDing I8 and I9 on PT 35, and leaving PTs 36-39 unused.

/O6 is generated by ANDing I8 and I9 on PT 41 and ANDing /I3, /I7, I9, and I10 on PT 42. Product terms 43 through 47 are left unused. For both outputs /O5 and /O6, the output enable PTs 32 and 40 are left blank to always enable the respective outputs.

Since pins 12 and 13 are not being used as outputs, Xs are put in the AND gates for all these product terms.

As you can see, any function can be put into the SOP form and then a device map generated for it. However, it is very time consuming to generate these maps by hand. Therefore, PLD CAD packages have been developed which automatically generate the PLD map from the Boolean equations. The map can then be loaded into a PLD programmer to program the device.

One such first-generation Assembler based CAD package, called AmPALASM20*, was developed by AMD. It allows you to enter Boolean equations using regular logical operators (AND, OR) and produces a device map. Figures 1-29 and 1-30 show the example in Figure 1-27 written in AmPALASM20 and the resulting device map.

```
PAL16L8
PAT001
DESIGN EXAMPLE
ADVANCED MICRO DEVICES
NC I1 I2 I3 I4 I5 I6 I7 I8 GND
I9 NC I10 06 05 04 03 02 01 VCC
;
;
/O1 = I1
/O2 = I1 + /I2
/O3 = /I1 * /I3
/O4 = /I3*I4
/O5 = /I3*I5*I6 + /I7 + I8*I9
/O6 = I8*I9 + /I3*/I7*I9*I10
```

Figure 1-29. Abbreviated AmPALASM20 Input

*We have used AmPALASM20 for illustration purpose only. The trend is towards advanced Compiler-based CAD packages. See "Design Aid Software for Programmable Logic" section.

DESIGN EXAMPLE

11 1111 1111 2222 2222 2233
0123 4567 8901 2345 6789 0123 4567 8901

```

0 ----
1 X----- I1
2 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
4 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
5 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
6 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
7 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

8 ----
9 X----- I1
10 ----- /I2
11 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
12 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
13 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
14 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
15 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

16 ----
17 -X----- -X----- /I1*/I3
18 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
19 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
20 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
21 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
22 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
23 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

24 ----
25 ----- -X- X----- /I3*I4
26 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
27 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
28 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
29 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
30 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

32 ----
33 ----- -X- ----- X----- /I3*I5*I6
34 ----- X----- I7
35 ----- X-X- I8*I9
36 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
37 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

40 ----
41 ----- X-X- I8*I9
42 ----- -X- ----- -XX- --X- /I3*/I7*I9*I10
43 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
44 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
45 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
46 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
47 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

48 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
49 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
50 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

56 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
57 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
58 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX
63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX

```

LEGEND: X : FUSE NOT BLOWN (L,N,O) - : FUSE BLOWN (H,P,1)

NUMBER OF FUSES BLOWN = 493

TB000250

Figure 1-30. AmPALASM20 Output Fuse Map

TESTING PHASE

Two tests can be performed after the part is programmed. First, the PLD programmer can verify that only the links/connections specified in the device map have been programmed in the part. Second, the programmed part can be tested for correct functionality; the same test vectors used for logic simulation can be used to test that the device generates the correct outputs for a given combination of inputs. The test vectors in the device map will be in a JEDEC-standard format (see Device Map).

If the device does not function as expected or differs from the logic simulation results, then the device is malfunctioning. The test phase can be performed on PLD programmers with testing capability.

THE DEVICE MAP

Fuse-Based and Erasable PLDs

Fuse-based PLDs have fuses that form the links or connections in the device. Electrically-erasable PLDs or UV-erasable PLDs have erasable cell arrays that form the connections. Erasable PLDs can be considered as having "erasable links" in this discussion.

Fuse-based PLDs are shipped by the manufacturer with all fuse links intact. A function is programmed into the part by selectively blowing or programming the fuse links that are not needed. On the other hand, an erased PLD has no connections at all. The designer will have to selectively make connections in order to implement a function. These fuses or links are specified in a data file called a fuse map or a device map.

This device map can be expressed in many formats, but one of the most common formats is the Joint Electronic Devices Engineering Council (JEDEC) standard which is supported by most PLD programmers and semiconductor manufacturers.

The JEDEC Standard (No. 3-A May 1986)

The standard put forth by the JEDEC committee contains many options which are used for transferring data between a PLD development system and a PLD programmer/tester. A minimum configuration JEDEC-device map file contains at least the following fields: F, L, and C.

This is a sample JEDEC-device map file:

```
*F0*
L0000 0110 0100 1000 1010*
L0032 1000 0000 0000 0000*
C0078*
```

Figure 1-31. Minimum Configuration Device Map File

The device map file starts with a design-specification field which may contain any ASCII character (except "****") to describe the design. This field is terminated by an "****".

The F field specifies the default link state of any unspecified fuses in the PLD. In Figure 1-31, "F0" will set all unspecified fuses in the PLD to the 0 state (i.e., a low resistance link specifying a connection between two points) A "1" would have specified a high-resistance link or no logical connection between two points. In fuse-based PLDs, a "1" would have instructed the PLD programmer to burn/blow the fuse connecting two points. The programmer will leave the fuse connected when a "0" is received.

The L field specifies the address of the fuse link that is to be programmed. A sequence of binary numbers follows which specifies the fuse states of the fuses starting at the address

specified by the L field. Referring to Figure 1-31, the first fuse to be programmed is at location "0000" decimal and the fuse state is "0".

The binary digits following the first "0" refer to the fuses following the fuse at location 0. When an "****" is detected, the fuse link information field terminates unless another Lxxxx is specified. In Figure 1-31, sixteen fuse states are transmitted beginning at fuse location 0. More fuse information will be transferred to the PLD programmer, but will now start at fuse location 32 decimal because of the new L field L0032.

The C field specifies the fuse checksum that is used by the PLD programmer to detect transmitting and receiving errors. This checksum is for all the fuses in the PLD, not only for the fuses specified in the file. The fuse checksum is a 4-digit hexadecimal value representing the unsigned 16-bit sum of 8-bit bytes formed with the fuse states. The 8-bit bytes are formed as follows:

word 0 msb	7	6	5	4	3	2	1	0	lsb	
	7	6	5	4	3	2	1	0		fuse #

word 1 msb	15	14	13	12	11	10	9	8	lsb	
	15	14	13	12	11	10	9	8		fuse #

word 4 msb	39	38	37	36	35	34	33	32	lsb	
	39	38	37	36	35	34	33	32		fuse #

word 12 msb	x	x	x	x	99	98	97	96	lsb	
	x	x	x	x	99	98	97	96		fuse #

8-bit bytes formed for a PLD with 100 fuses

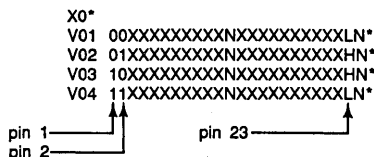
Unused bits in the last byte are filled with "0s". For the fuse map example, the fuse checksum is calculated as follows:

```
Word 0  0010 0110
Word 8  0101 0001
Word 16 0000 0000
Word 24 0000 0000
Word 32 0000 0001
Word 40 0000 0000
      :
Word 96 0000 0000
-----
Total  0111 1000  →  0078H
```

A PLD data sheet will usually show the JEDEC fuse numbers for every fuse link in the device. PLD CAD packages usually have a JEDEC fuse-map generator which relieves the designer from having to individually program the fuses in a PLD to implement a logic circuit.

TESTING THE PLD

If the PLD programmer has functional testing capability, test vectors can be inserted into the fuse map file. These test vectors begin with a "V" and a decimal vector number followed by a sequence of characters symbolizing the inputs and expected outputs to every pin on the device. Each vector is terminated by an "****". For example, the AmPAL22V10 has twenty-four pins. If the signals A,B, and C are assigned to pins 1, 2, and 23, then the following test vectors can be used to check that a two-input XOR function programmed into the AmPAL22V10 is functioning properly.



The X field defines the don't care condition in the test vectors. The N field represents power pins and pins that are not tested. In this case the two Ns in the four vectors represent the V_{CC} and ground pins.

There are many test-vector specification options available, but this is the minimum configuration for including test vectors in a fuse map file. This test-vector file is normally appended to the end of the fuse-map file to improve readability and documentation, but it can actually be placed anywhere in the fuse-map file if the test engineer so desires. These vectors were manually generated, but some CAD packages support automatic test vector generation.

PLD PROGRAMMER TRANSMISSION PROTOCOL

The data transfer protocol used to transfer the fuse map to a PLD programmer is simple. The transmission consists of the start-of-text (STX) character, the fuse-map information (fuse-link states, test-vector information, and fuse checksum), the end-of-text (ETX) character, and the transmission checksum. The transmission checksum is the unsigned 16-bit sum (modulo 65,535) of all the ASCII characters transmitted between and including the STX and ETX.

XA XB XC XD XOR_Y (XA,XB,XC,XD)

0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$\begin{aligned} \text{XOR_Y (XA,XB,XC,XD)} = & /XA \cdot /XB \cdot /XC \cdot XD \\ & + /XA \cdot /XB \cdot XC \cdot /XD \\ & + /XA \cdot XB \cdot /XC \cdot /XD \\ & + /XA \cdot XB \cdot XC \cdot XD \\ & + XA \cdot /XB \cdot /XC \cdot /XD \\ & + XA \cdot /XB \cdot XC \cdot XD \\ & + XA \cdot XB \cdot /XC \cdot XD \\ & + XA \cdot XB \cdot XC \cdot /XD \end{aligned}$$

Figure 1-32.a. 4-Bit XOR Truth Table and Logic Equations

Some computer operating systems do not allow users to control what characters are sent. If this is the case with your PLD development system, then the transmission checksum must be disabled by always sending the dummy value "0000". Any PLD programmer complying with this JEDEC standard will always accept this as a valid transmission checksum.

DESIGN EXAMPLES

We will illustrate the design examples of two combinatorial functions and two sequential functions using PAL devices. For the combinatorial examples we will show the implementation of a 4-bit "exclusive-OR generator" and an 8:1 MUX in a PAL device. Instead of using standard SSI/MSI devices, both functions can be performed in the same PAL device.

Identify Device Requirements

The 4-bit XOR requires four inputs and one output pin; the 8:1 MUX requires eight data lines, three select lines, and one MUX output. The PLD selected must therefore have at least fifteen inputs and two outputs. In this example, we are also assuming active-HIGH outputs. The AmPAL18P8 fits all the I/O requirements. However, the number of product terms (PTs) can only be determined from the function table.

Figures 1-32.a and 1-32.b show the truth tables for both these functions. Figure 1-32.c shows logic diagrams for both 4-input XOR and 8:1 MUX functions. The XOR and MUX functions can be described in a high-level language-type syntax (Figure 1-33) directly from these truth tables. (For detailed discussion of PLPL syntax, see "Design-Aid Software for Programmable Logic" section).

S2	S1	S0	D7	D6	D5	D4	D3	D2	D1	D0	MUX_Y
0	0	0	x	x	x	x	x	x	x	D0	D0
0	0	1	x	x	x	x	x	x	D1	x	D1
0	1	0	x	x	x	x	x	D2	x	x	D2
0	1	1	x	x	x	x	D3	x	x	x	D3
1	0	0	x	x	x	D4	x	x	x	x	D4
1	0	1	x	x	D5	x	x	x	x	x	D5
1	1	0	x	D6	x	x	x	x	x	x	D6
1	1	1	D7	x	x	x	x	x	x	x	D7

$$\text{MUX_Y} = /S2 \cdot /S1 \cdot /S0 \cdot D0 + /S2 \cdot /S1 \cdot S0 \cdot D1 + /S2 \cdot S1 \cdot /S0 \cdot D2 + /S2 \cdot S1 \cdot S0 \cdot D3 + S2 \cdot /S1 \cdot /S0 \cdot D4 + S2 \cdot /S1 \cdot S0 \cdot D5 + S2 \cdot S1 \cdot /S0 \cdot D6 + S2 \cdot S1 \cdot S0 \cdot D7$$

Figure 1-32.b. 8:1 MUX Truth Table and Logic Equations

Based on the function table, we see that the XOR and MUX functions require eight PTs each. Figure 1-32.c shows the logic diagram for both of these functions. The AmPAL18P8

has eight PTs per output, and hence satisfies all the I/O and PT requirements.

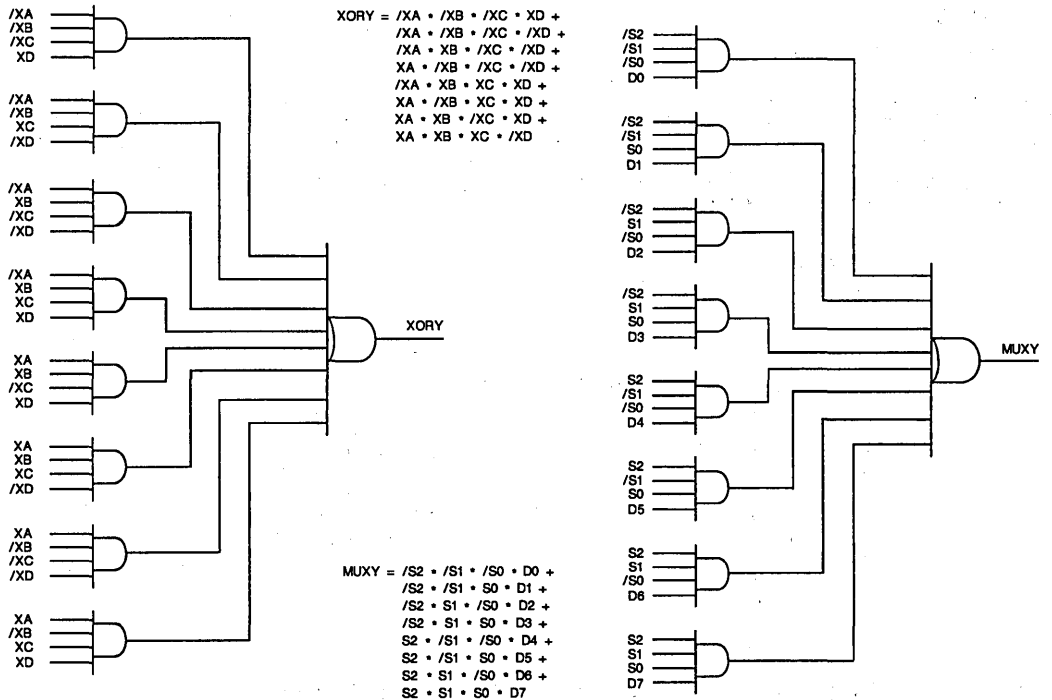


Figure 1-32.c. Logic Diagrams for a 4-Input XOR and 8:1 MUX

LD000520

```

DEVICE XORMUX (AMPAL18P8)
"This logic description file written in PLPL defines
an AmPAL18P8 performing a 4-bit EXOR and an 8:1
multiplexing function"

PIN  XA = 1  XB = 2  XC = 3  XD = 11
    D0 = 4  D1 = 5  D2 = 6  D3 = 7  D4 = 8 D5 = 9 D6 = 13 D7 = 14
    S0 = 15  S1 = 16 S2 = 17
    MUX_Y = 18  XOR_Y = 19;

```

```

BEGIN
XOR_Y = XA XOR XB XOR XC XOR XD; "4-bit EXOR function"
CASE (S2,S1,S0)
  BEGIN
    0) MUX_Y = D0; "if select lines are all 0, then select D0"
    1) MUX_Y = D1;
    2) MUX_Y = D2;
    3) MUX_Y = D3;
    4) MUX_Y = D4;
    5) MUX_Y = D5;
    6) MUX_Y = D6;
    7) MUX_Y = D7;
  END; "end of CASE statement"
END.

```

```

TEST_VECTORS "test vectors to be used for simulation"
IN  S2 S1 S0 XA XB XC XD D7 D6 D5 D4 D3 D2 D1 D0;
OUT XOR_Y MUX_Y;
BEGIN
"XOR_Y output test"
"S2 S1 S0 XA XB XC XD D7 D6 D5 D4 D3 D2 D1 D0 | XOR_Y MUX_Y"
X X X 0 0 0 0 X X X X X X X X L X;
X X X 0 0 0 1 X X X X X X X X H X;
X X X 0 0 1 0 X X X X X X X X H X;
X X X 0 0 1 1 X X X X X X X X L X;
X X X 0 1 0 0 X X X X X X X X H X;
X X X 0 1 0 1 X X X X X X X X L X;
X X X 0 1 1 0 X X X X X X X X L X;
X X X 0 1 1 1 X X X X X X X X H X;
X X X 1 0 0 0 X X X X X X X X H X;
X X X 1 0 0 1 X X X X X X X X L X;
X X X 1 0 1 0 X X X X X X X X L X;
X X X 1 0 1 1 X X X X X X X X H X;
X X X 1 1 0 0 X X X X X X X X L X;
X X X 1 1 0 1 X X X X X X X X H X;
X X X 1 1 1 0 X X X X X X X X H X;
X X X 1 1 1 1 X X X X X X X X L X;

"MUX_Y output test"
0 0 0 X X X X X X X X X X X 0 X L;
0 0 0 X X X X X X X X X X X 1 X H;
0 0 1 X X X X X X X X X X X 0 X L;
0 0 1 X X X X X X X X X X X 1 X H;
0 1 0 X X X X X X X X X X 0 X L;
0 1 0 X X X X X X X X X X 1 X H;
0 1 1 X X X X X X X X X 0 X L;
0 1 1 X X X X X X X X X 1 X H;
1 0 0 X X X X X X X 0 X X X X L;
1 0 0 X X X X X X X 1 X X X X H;
1 0 1 X X X X X X X 0 X X X X L;
1 0 1 X X X X X X X 1 X X X X H;
1 1 0 X X X X X X 0 X X X X L;
1 1 0 X X X X X X 1 X X X X H;
1 1 1 X X X X X 0 X X X X L;
1 1 1 X X X X X 1 X X X X H;

```

END.

TB000370

Figure 1-33. Logic Description File for the XOR and MUX Functions

Test vectors can be generated (Figure 1-34) from the logic description file (Figure 1-33) and sent to a simulator to test the logic equations. In this example, the vectors are generated from the test vectors specified manually. After simulation (Figure 1-35), a device map (Figure 1-36) is generated from these equations and the PLD programmed (Figure 1-37) with the map.

The inputs XA, XB, XC, XD, D0 through D7, and select lines S0 through S2 may be assigned to any input or I/O pin as shown in the AmpAL18P8 logic diagram. The outputs XOR_Y and

MUX_Y have been assigned to pins 18 and 19. Since we have not specified any Output-Enable product term, PLPL assumes that the outputs will always be enabled. In the completed device map (Figure 1-37), all the connections for the Output- Enable PTs for the two outputs XOR_Y and MUX_Y have been disconnected which will always enable the outputs.

The input and output signals can be reassigned to different pins on the PLD package to fit your PC board routing requirements.

```
V0001 000XXXXXXN0XXXXXXLN *
V0002 000XXXXXXN1XXXXXXHN *
V0003 001XXXXXXN0XXXXXXHN *
V0004 001XXXXXXN1XXXXXXLN *
V0005 010XXXXXXN0XXXXXXHN *
V0006 010XXXXXXN1XXXXXXLN *
V0007 011XXXXXXN0XXXXXXLN *
V0008 011XXXXXXN1XXXXXXHN *
V0009 100XXXXXXN0XXXXXXHN *
V0010 100XXXXXXN1XXXXXXLN *
V0011 101XXXXXXN0XXXXXXLN *
V0012 101XXXXXXN1XXXXXXHN *
V0013 110XXXXXXN0XXXXXXLN *
V0014 110XXXXXXN1XXXXXXHN *
V0015 111XXXXXXN0XXXXXXHN *
V0016 111XXXXXXN1XXXXXXLN *
V0017 XXX0XXXXXXNXXX000LXN *
V0018 XXX1XXXXXXNXXX000HXN *
V0019 XXXX0XXXXXXNXXX100LXN *
V0020 XXXX1XXXXXXNXXX100HXN *
V0021 XXXXX0XXXXXXNXXX010LXN *
V0022 XXXXX1XXXXXXNXXX010HXN *
V0023 XXXXX0XXXXXXNXXX110LXN *
V0024 XXXXX1XXXXXXNXXX110HXN *
V0025 XXXXX0XXXXXXNXXX001LXN *
V0026 XXXXX1XXXXXXNXXX001HXN *
V0027 XXXXX0XXXXXXNXXX101LXN *
V0028 XXXXX1XXXXXXNXXX101HXN *
V0029 XXXXXXXXXNXXX0X011LXN *
V0030 XXXXXXXXXNXXX1X011HXN *
V0031 XXXXXXXXXNXXX0111LXN *
V0032 XXXXXXXXXNXXX1111HXN *
```

TB000360

Figure 1-34. Test Vectors for EXOR and 8:1 MUX PLD Design

V0001	INPUT	OUTPUT	V0008	INPUT	OUTPUT	V0015	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 000XXXXXX0XXXXXX ==>	XL		Expected: 011XXXXXX1XXXXXX ==>	XH		Expected: 111XXXXXX0XXXXXX ==>	XH	
Computed:	LL		Computed:	LH		Computed:	LH	
-----			-----			-----		
V0002	INPUT	OUTPUT	V0009	INPUT	OUTPUT	V0016	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 000XXXXXX1XXXXXX ==>	XH		Expected: 100XXXXXX0XXXXXX ==>	XH		Expected: 111XXXXXX1XXXXXX ==>	XL	
Computed:	LH		Computed:	LH		Computed:	LL	
-----			-----			-----		
V0003	INPUT	OUTPUT	V0010	INPUT	OUTPUT	V0017	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 001XXXXXX0XXXXXX ==>	XH		Expected: 100XXXXXX1XXXXXX ==>	XL		Expected: XXXX0XXXXXX0XXXX0 ==>	LX	
Computed:	LH		Computed:	LL		Computed:	LL	
-----			-----			-----		
V0004	INPUT	OUTPUT	V0011	INPUT	OUTPUT	V0018	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 001XXXXXX1XXXXXX ==>	XL		Expected: 101XXXXXX0XXXXXX ==>	XL		Expected: XXX1XXXXXX0XXXX0 ==>	HX	
Computed:	LL		Computed:	LL		Computed:	HL	
-----			-----			-----		
V0005	INPUT	OUTPUT	V0012	INPUT	OUTPUT	V0019	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 010XXXXXX0XXXXXX ==>	XH		Expected: 101XXXXXX1XXXXXX ==>	XH		Expected: XXXX0XXXXXX0XXXX100 ==>	LX	
Computed:	LH		Computed:	LH		Computed:	LL	
-----			-----			-----		
V0006	INPUT	OUTPUT	V0013	INPUT	OUTPUT	V0020	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 010XXXXXX1XXXXXX ==>	XL		Expected: 110XXXXXX0XXXXXX ==>	XL		Expected: XXXX1XXXXXX0XXXX100 ==>	HX	
Computed:	LL		Computed:	LL		Computed:	HL	
-----			-----			-----		
V0007	INPUT	OUTPUT	V0014	INPUT	OUTPUT	V0021	INPUT	OUTPUT
	11111111	11		11111111	11		11111111	11
Pin # : 1234567891234567		89	Pin # : 1234567891234567		89	Pin # : 1234567891234567		89
Expected: 011XXXXXX0XXXXXX ==>	XL		Expected: 110XXXXXX1XXXXXX ==>	XH		Expected: XXXXX0XXXXXX010 ==>	LX	
Computed:	LL		Computed:	LH		Computed:	LL	

TB000350

Figure 1-35. Simulation Run for EXOR and MUX PLD Design

V0022 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXX1XXXXXX010 ==> HX
 Computed: HL

V0023 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXX0XXXXXX110 ==> LX
 Computed: LL

V0024 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXX1XXXXXX110 ==> HX
 Computed: HL

V0025 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXX0XXXXX001 ==> LX
 Computed: LL

V0026 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXX1XXXXX001 ==> HX
 Computed: HL

V0027 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXX0XXXX101 ==> LX
 Computed: LL

V0028 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXX1XXXX101 ==> HX
 Computed: HL

V0029 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXXXX0X011 ==> LX
 Computed: LL

V0030 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXXXX1X011 ==> HX
 Computed: HL

V0031 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXXXX0X0111 ==> LX
 Computed: LL

V0032 INPUT OUTPUT
 11111111 11
 Pin # : 1234567891234567 89
 Expected: XXXXXXXXXX1111 ==> HX
 Computed: HL

Simulation completed, Errors detected = 0

Listing sum-of-products equations for XORMUX

XOR_Y = /XA*XB*/XD*/XC
 + /XA*XB*XD*XC
 + /XA*/XB*XD*/XC
 + /XA*/XB*/XD*XC
 + XA*/XB*/XD*/XC
 + XA*/XB*XD*XC
 + XA*XB*XD*/XC
 + XA*XB*/XD*XC;

MUX_Y = /S2*/S1*/S0*D0
 + /S2*/S1*S0*D1
 + /S2*S1*/S0*D2
 + /S2*S1*S0*D3
 + S2*/S1*/S0*D4
 + S2*/S1*S0*D5
 + S2*S1*/S0*D6
 + S2*S1*S0*D7;

Figure 1-35. (Cont'd.)

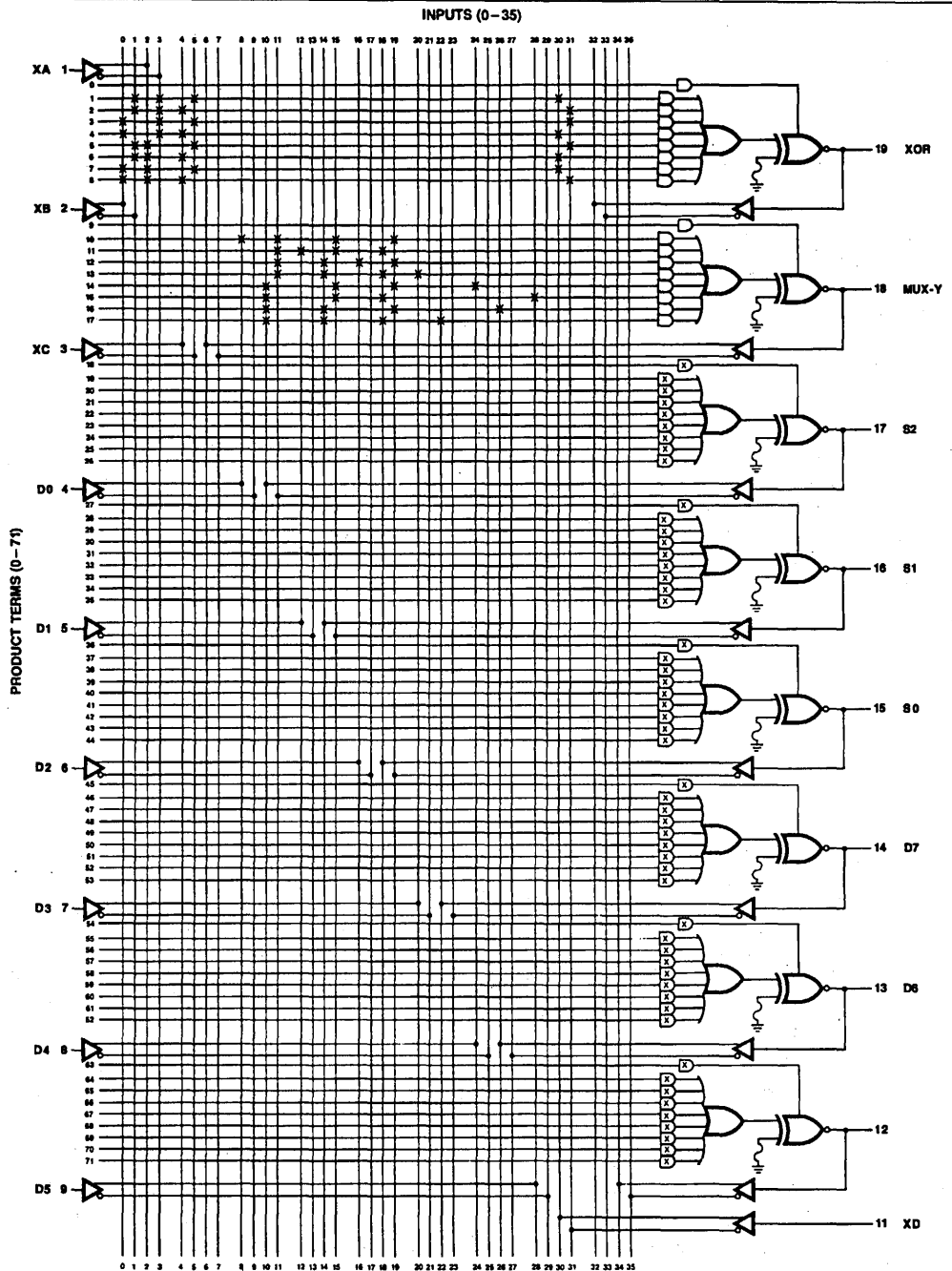
Title: XORMUX
 Part Type: PAL18P8*
 DEVICE: PAL18P8*
 MFG: AMD*
 F0*

L0000 1111 1111 1111 1111 1111 1111 1111 1111 1111*
 L0036 1010 1011 1111 1111 1111 1111 1111 1101 1111*
 L0072 1010 0111 1111 1111 1111 1111 1111 1110 1111*
 L0108 0110 1011 1111 1111 1111 1111 1111 1110 1111*
 L0144 0110 0111 1111 1111 1111 1111 1111 1101 1111*
 L0180 1001 1011 1111 1111 1111 1111 1111 1110 1111*
 L0216 1001 0111 1111 1111 1111 1111 1111 1101 1111*
 L0252 0101 1011 1111 1111 1111 1111 1111 1101 1111*
 L0288 0101 0111 1111 1111 1111 1111 1111 1110 1111*
 L0324 1111 1111 1111 1111 1111 1111 1111 1111 1111*
 L0360 1111 1111 0110 1110 1110 1111 1111 1111 1111*
 L0396 1111 1111 1110 0110 1101 1111 1111 1111 1111*
 L0432 1111 1111 1110 1101 0110 1111 1111 1111 1111*
 L0468 1111 1111 1110 1101 1101 0111 1111 1111 1111*
 L0504 1111 1111 1101 1110 1110 1111 0111 1111 1111*
 L0540 1111 1111 1101 1110 1101 1111 1111 0111 1111*
 L0576 1111 1111 1101 1101 1110 1111 1101 1111 1111*
 L0612 1111 1111 1101 1101 1101 1101 1111 1111 1111*
 L0648 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0684 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0720 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0756 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0792 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0828 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0864 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0900 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0936 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L0972 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1008 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1044 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1080 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1116 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1152 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1188 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1224 0000 0000 0000 0000 0000 0000 0000 0000 0000*

L1260 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1296 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1332 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1368 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1404 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1440 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1476 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1512 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1548 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1584 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1620 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1656 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1692 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1728 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1764 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1800 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1836 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1872 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1908 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1944 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L1980 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2016 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2052 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2088 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2124 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2160 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2196 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2232 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2268 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2304 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2340 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2376 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2412 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2448 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2484 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2520 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2556 0000 0000 0000 0000 0000 0000 0000 0000 0000*
 L2592 11000000*
 C4876*
 B97A

TB000380

Figure 1-36. Device/Fuse Map for EXOR and MUX PLD Design



LD000042

Eighteen Array Inputs

- 10 dedicated
- 8 bidirectional I/O

Eight 8-Wide AND-OR Structures

- Combinatorial outputs
- Programmable output enable for each output
- Programmable polarity on each output

Figure 1-37. AmPAL18P8 Programmed with EXOR and MUX Design

SEQUENTIAL DESIGN EXAMPLE

Next we will show implementation of a 4-bit binary up-counter and a decade counter in a single PAL device.

Design of a Synchronous Binary Counter

Counters are one of the simplest types of sequential networks. A synchronous counter is usually built from a number of flip-flops which change state in a prescribed sequence when input pulses are received. The operation of the flip-flops is synchronized to a common input pulse (a common clock).

Synchronous counters are used for state sequencing, delay timing and event counting. The key to designing a counter is knowing when a bit should be toggled. For an up-counter, a bit is toggled whenever every bit of lesser significance is "1" (see the counting sequence of Figure 1-38).

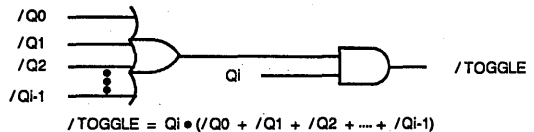
Conversely, for a down-counter, a bit is toggled whenever every bit of lesser significance is "0". In both cases, the LSB is always toggled. By ANDing all bits of lesser significance along with the complement of the current data in the register, the problem of when this bit is to be toggled is solved. However, to complete the design, the bit must remain unchanged under all other conditions. This can be accomplished by ORing the complements of the lesser significant bits together and then ANDing the result with the current data in the register (Figure 1-39). The equation in Figure 1-39 can be changed into the SOP form (Figure 1-40) for direct implementation in a PAL device. Thus, if a bit is to be toggled, the complement of the current data will be clocked in; if not, the data remains unchanged by clocking in the current data.

A 4-bit binary up-counter example illustrates this approach. We will build this 4-bit binary counter using D flip-flops. The state of the counter is determined by the state of the individual flip-flops. For example, if flip-flop A is in state 0, B in state 1, C in state 1 and D in state 0, the state of the counter is 0110. Initially all the flip-flops are set to the zero state. When a clock pulse is received, the counter will change to state 0001; when a second clock pulse is received, the state will change to 0010, etc. The state-counting sequence is shown in Figure 1-38.

Figure 1-41 shows the state diagram for a 4-bit binary up-counter. Typical counter functions are loading data, counting, and "holding" data. The function table is shown in Figure 1-42 and the logic diagram in Figure 1-43.

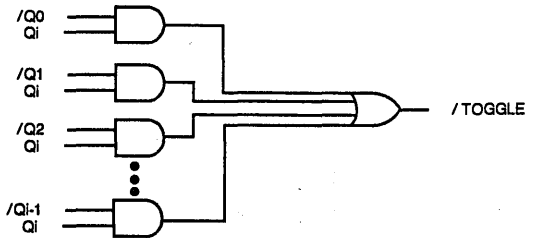
CURRENT STATE	NEXT STATE
0 0 0 0	0 0 0 1
0 0 0 1	0 0 1 0
0 0 1 0	0 0 1 1
0 0 1 1	0 1 0 0
0 1 0 0	0 1 0 1
0 1 0 1	0 1 1 0
0 1 1 0	0 1 1 1
0 1 1 1	1 0 0 0
1 0 0 0	1 0 0 1
1 0 0 1	1 0 1 0
1 0 1 0	1 0 1 1
1 0 1 1	1 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	1 1 1 0
1 1 1 0	1 1 1 1
1 1 1 1	0 0 0 0

Figure 1-38. Counting Sequence



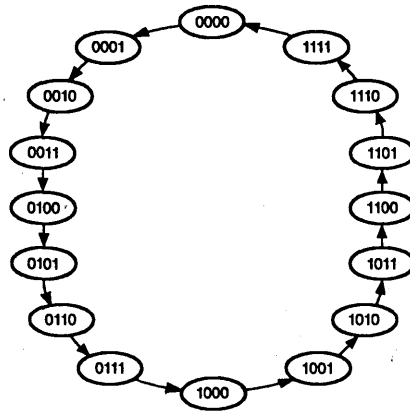
LD000500

Figure 1-39. Logic for Not Toggling Bit i



LD000510

Figure 1-40. Equivalent Form of Figure 1-39



DF006140

Figure 1-41. State Diagram of a 4-Bit Binary Up-Counter (16 States)

INPUTS		OUTPUTS							
S ₁	S ₀	Q ₃	Q ₂	Q ₁	Q ₀	Q ₃	Q ₂	Q ₁	Q ₀
CLEAR LOAD COUNT	0	0	X	X	X	X	0	0	0
	0	1	X	X	X	X	D ₃	D ₂	D ₁
	1	0	0	0	0	0	0	0	1
	1	0	0	0	0	1	0	0	0
	1	0	0	0	1	0	0	1	1
	1	0	0	0	1	1	0	1	0
	1	0	0	1	0	0	1	0	1
	1	0	0	1	0	1	0	1	0
	1	0	0	1	1	0	1	1	1
	1	0	0	1	1	1	0	0	0
	1	0	1	0	0	0	1	0	1
	1	0	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	1	1
	1	0	1	0	1	1	1	0	0
	1	0	1	1	0	0	1	1	0
	1	0	1	1	0	1	1	1	0
	1	0	1	1	1	0	1	1	1
	1	0	1	1	1	1	0	0	0
HOLD	1	1	X	X	X	X	Q ₃	Q ₂	Q ₁
	1	1	X	X	X	X	Q ₃	Q ₂	Q ₀

TB000320

Figure 1-42. Function Table for 4-Bit Binary Up-Counter

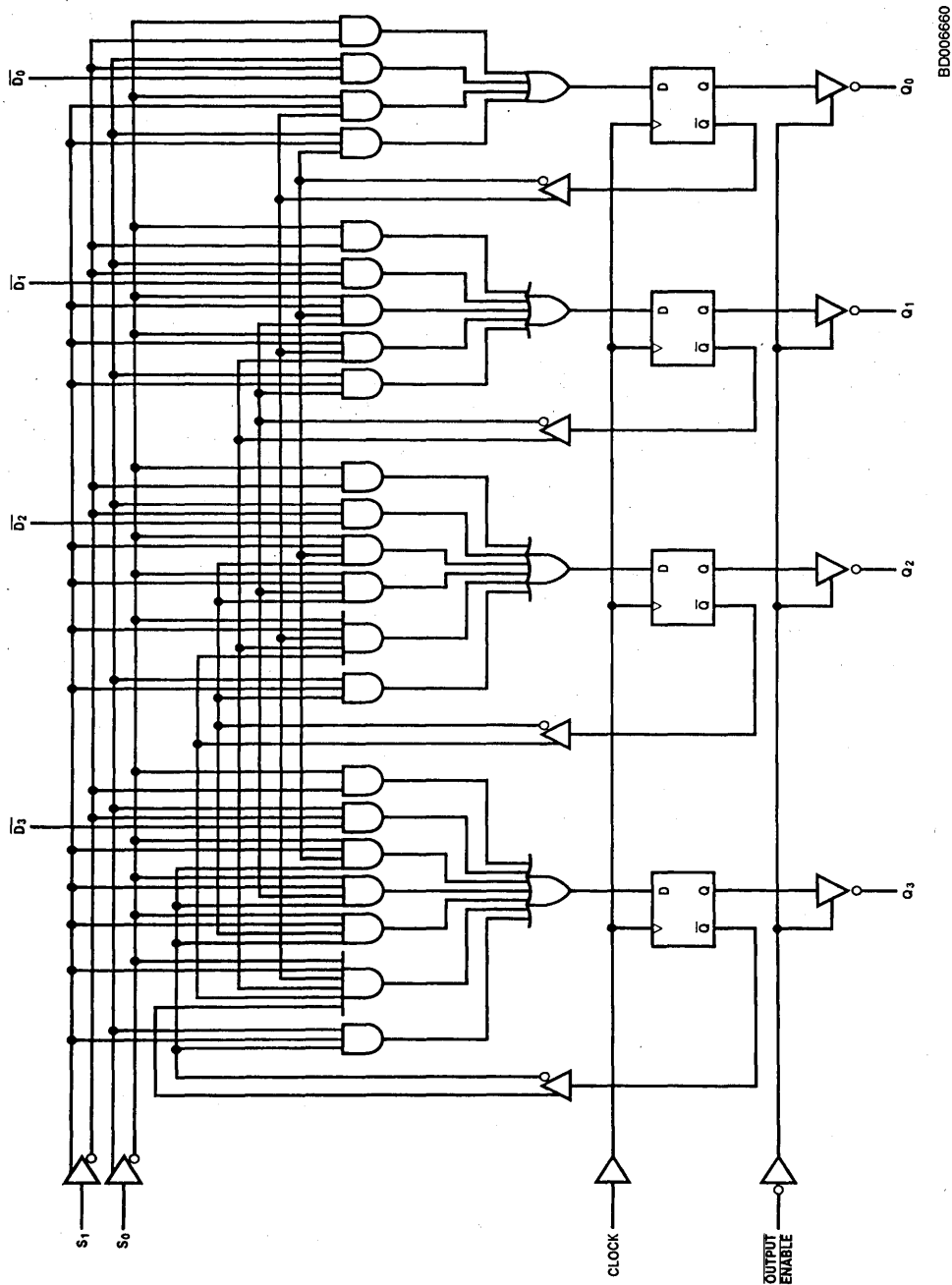
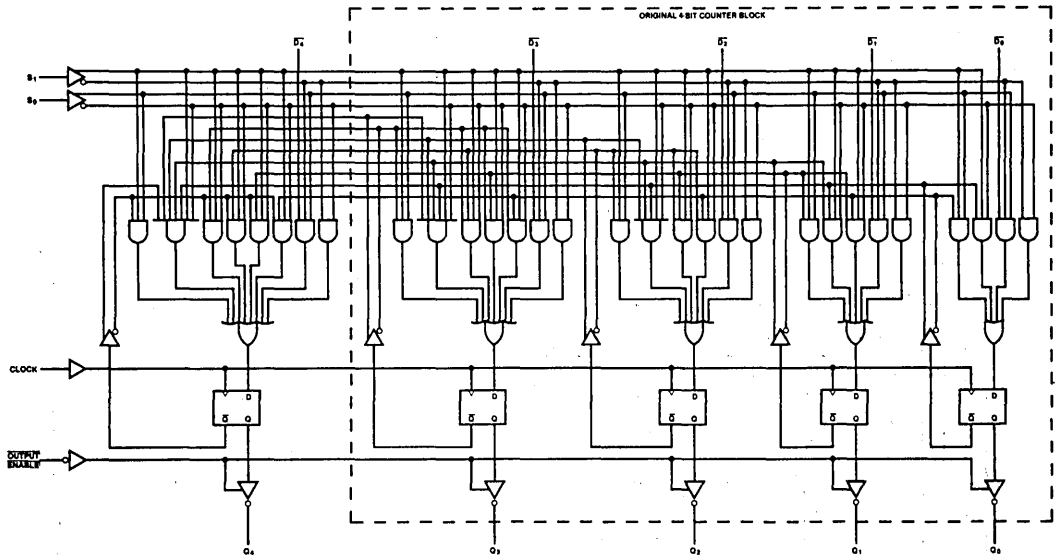


Figure 1-43. 4-Bit Binary Up-Counter

Expanding the number of bits in the counter is done by expanding the function table to incorporate the additional bits. Karnaugh maps, although not essential, can be used to find the required equations in SOP form for a PAL-device implementation. In general, besides any fixed overhead for control functions (CLEAR, LOAD, and HOLD in this example) bit n will require additional n product terms. Therefore, if this example 4-bit counter is to be expanded to 5 bits, the fifth bit will require five product terms plus three additional product terms for clearing, loading, and counting (see Figure 1-44). Note that the original 4-bit block is unaffected by the addition of the fifth bit.

This basic counter is easily expandable to perform more complex functions.

A high-level language design specification for the 4-bit binary counter is shown in Figure 1-45. Corresponding test vectors are shown in Figure 1-46 and are used as inputs by the simulator to test the logic equations. Figure 1-47 shows the PLPL Optimizer's output; Figure 1-48, the equations list; Figure 1-49, the simulation run; Figure 1-50, the device map; and Figure 1-51, the implementation of this function in an AmPAL16R8 device (Note that Figures 1-46 through 1-51 also implement a decade-counter function, the description for which follows).



BD006650

Figure 1-44. 5-Bit Binary Up-Counter

```

DEVICE BIN_DCD_CNTR (AMPAL16R8)
"An AmpAL16R8 programmed as a dual base counter: binary and decimal"
PIN CLK = 1  S1 = 2  S0 = 3  D[3:0] = 5,6,7,8 "data"
    /BIN[3:0] = 19,18,17,16 "binary counter active LOW"
    /DCD[3:0] = 15,14,13,12; "decimal counter active LOW"

BEGIN
IF (/S1*/S0) THEN "clear"
    BEGIN
    BIN[3:0] := 0; "output is active LOW"
    DCD[3:0] := 0;
    END;
IF (/S1*S0) THEN "load"
    BEGIN
    BIN[3:0] := D[3:0];
    DCD[3:0] := D[3:0];
    END;
IF (S1*/S0) THEN "hold"
    BEGIN
    BIN[3:0] := BIN[3:0]; "hold BIN and DCD active LOW"
    DCD[3:0] := DCD[3:0];
    END;
IF (S1*S0) THEN "count"
    BEGIN
    IF (/DCD[3]*/DCD[2]*/DCD[1]*/DCD[0]) THEN DCD[3:0] := 1;
    IF (/DCD[3]*/DCD[2]*/DCD[1]* DCD[0]) THEN DCD[3:0] := 2;
    IF (/DCD[3]*/DCD[2]* DCD[1]*/DCD[0]) THEN DCD[3:0] := 3;
    IF (/DCD[3]*/DCD[2]* DCD[1]* DCD[0]) THEN DCD[3:0] := 4;
    IF (/DCD[3]* DCD[2]*/DCD[1]*/DCD[0]) THEN DCD[3:0] := 5;
    IF (/DCD[3]* DCD[2]*/DCD[1]* DCD[0]) THEN DCD[3:0] := 6;
    IF (/DCD[3]* DCD[2]* DCD[1]*/DCD[0]) THEN DCD[3:0] := 7;
    IF (/DCD[3]* DCD[2]* DCD[1]* DCD[0]) THEN DCD[3:0] := 8;
    IF ( DCD[3]*/DCD[2]*/DCD[1]*/DCD[0]) THEN DCD[3:0] := 9;
    IF ( DCD[3]*/DCD[2]*/DCD[1]* DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]*/DCD[2]* DCD[1]*/DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]*/DCD[2]* DCD[1]* DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]* DCD[2]*/DCD[1]*/DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]* DCD[2]*/DCD[1]* DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]* DCD[2]* DCD[1]*/DCD[0]) THEN DCD[3:0] := 0;
    IF ( DCD[3]* DCD[2]* DCD[1]* DCD[0]) THEN DCD[3:0] := 0;

    IF (/BIN[3]*/BIN[2]*/BIN[1]*/BIN[0]) THEN BIN[3:0] := 1;

```

```

    IF (/BIN[3]*/BIN[2]*/BIN[1]* BIN[0]) THEN BIN[3:0] := 2;
    IF (/BIN[3]*/BIN[2]* BIN[1]*/BIN[0]) THEN BIN[3:0] := 3;
    IF (/BIN[3]*/BIN[2]* BIN[1]* BIN[0]) THEN BIN[3:0] := 4;
    IF (/BIN[3]* BIN[2]*/BIN[1]*/BIN[0]) THEN BIN[3:0] := 5;
    IF (/BIN[3]* BIN[2]*/BIN[1]* BIN[0]) THEN BIN[3:0] := 6;
    IF (/BIN[3]* BIN[2]* BIN[1]*/BIN[0]) THEN BIN[3:0] := 7;
    IF (/BIN[3]* BIN[2]* BIN[1]* BIN[0]) THEN BIN[3:0] := 8;
    IF ( BIN[3]*/BIN[2]*/BIN[1]*/BIN[0]) THEN BIN[3:0] := 9;
    IF ( BIN[3]*/BIN[2]*/BIN[1]* BIN[0]) THEN BIN[3:0] := 10;
    IF ( BIN[3]*/BIN[2]* BIN[1]*/BIN[0]) THEN BIN[3:0] := 11;
    IF ( BIN[3]*/BIN[2]* BIN[1]* BIN[0]) THEN BIN[3:0] := 12;
    IF ( BIN[3]* BIN[2]*/BIN[1]*/BIN[0]) THEN BIN[3:0] := 13;
    IF ( BIN[3]* BIN[2]*/BIN[1]* BIN[0]) THEN BIN[3:0] := 14;
    IF ( BIN[3]* BIN[2]* BIN[1]*/BIN[0]) THEN BIN[3:0] := 15;
    IF ( BIN[3]* BIN[2]* BIN[1]* BIN[0]) THEN BIN[3:0] := 0;
    END;
END.

TEST_VECTORS
IN CLK,S1,S0,D[3:0];      I_O BIN[3:0],DCD[3:0];
BEGIN
"CLK S1 S0 DATA | BIN[3:0] DCD[3:0]"
C 0 1 0110 LHL LHL ; "load"
C 1 0 1111 LHL LHL ; "hold"
C 0 0 0110 LLLL LLLL ; "clear, start count from 0"
C 1 1 0110 LLLH LLLH ; "count"
C 1 1 0110 LLHL LLHL ; "count"
C 1 1 0110 LLHH LLHH ; "count"
C 1 1 0110 LHL LHL ; "count"
C 1 1 0110 LHLH LHLH ; "count"
C 1 1 0110 LHL LHL ; "count"
C 1 1 0110 LHHH LHHH ; "count"
C 1 1 0110 HLLL LLLL ; "count"
C 1 1 0110 HLLH HLLH ; "count"
C 1 1 0110 HLHL LLLL ; "count"
C 1 1 0110 HLHH LLLH ; "count"
C 1 1 0110 HLLL LLHL ; "count"
C 1 1 0110 HHLH LLHH ; "count"
C 1 1 0110 HHLL LHL ; "count"
C 1 1 0110 HHHH LHLH ; "count"
C 1 1 0110 LLLL LHL ; "count"
END.

```

TB000310

Figure 1-45. Logic Description File for the 4-Bit Binary Counter and a Decade Counter, Written in PLPL Format

```

V0001 C01X0110XNXHLLHLLHN *
V0002 C10X1111XNXHLLHLLHN *
V0003 C00X0110XNXHHHHHHHN *
V0004 C11X0110XNXLHHHLLHN *
V0005 C11X0110XNXHLLHLLHN *
V0006 C11X0110XNXLLHLLHLLHN *
V0007 C11X0110XNXHLLHLLHLLHN *
V0008 C11X0110XNXLHLLHLLHLLHN *
V0009 C11X0110XNXHLLHLLHLLHN *
V0010 C11X0110XNXLLLHLLHLLHN *
V0011 C11X0110XNXHHHLLHLLHLLHN *
V0012 C11X0110XNXLHLLHLLHLLHN *
V0013 C11X0110XNXHHHLLHLLHLLHN *
V0014 C11X0110XNXLHHHLLHLLHLLHN *
V0015 C11X0110XNXHLLHHHLLHLLHN *
V0016 C11X0110XNXLLHHHLLHLLHLLHN *
V0017 C11X0110XNXHHHLLHLLHLLHLLHN *
V0018 C11X0110XNXLHLLHLLHLLHLLHN *
V0019 C11X0110XNXHLLHHHLLHLLHLLHN *

```

TB000300

Figure 1-46. Test Vectors Specification for 4-Bit Binary Counter and the Decade Counter

```

BIN_DCD_CNTR
PAL16R8
CLK 1 INPUT
S1 2 INPUT
S0 3 INPUT
D[3] 5 INPUT
D[2] 6 INPUT
D[1] 7 INPUT
D[0] 8 INPUT
/DCD[0] 12 OUTPUT REGISTERED INVERTED
/DCD[1] 13 OUTPUT REGISTERED INVERTED
/DCD[2] 14 OUTPUT REGISTERED INVERTED
/DCD[3] 15 OUTPUT REGISTERED INVERTED
/BIN[0] 16 OUTPUT REGISTERED INVERTED
/BIN[1] 17 OUTPUT REGISTERED INVERTED
/BIN[2] 18 OUTPUT REGISTERED INVERTED
/BIN[3] 19 OUTPUT REGISTERED INVERTED
*
19 = 29/19*18 +
      29/19*17 +
      29/19*16 +
      /2*3*5 +
      29/3*/19 +
      2*3*19*/18*/17*/16 ;
18 = 28/18*17 +
      28/18*16 +
      2*3*18*/17*/16 +
      /2*3*6 +
      28/3*/18 ;
17 = 27/17*16 +
      2*3*17*/16 +
      /2*3*7 +
      28/3*/17 ;
16 = 2*3*16 +
      /2*3*8 +
      28/3*/16 ;
15 = 26/15*14*13*12 +
      /2*3*5 +
      28/3*/15 +
      2*3*15*/14*/13*/12 ;
14 = 25/14*15*13 +
      28/14*15*12 +
      /2*3*6 +
      28/3*/14 +
      2*3*14*15*/13*/12 ;
13 = 24/13*15*12 +
      2*3*13*15*/12 +
      /2*3*7 +
      28/3*/13 ;
12 = 2*3*12*15 +
      2*3*12*14*13 +
      /2*3*8 +
      28/3*/12 ;

```

TB000290

Figure 1-47. PLPL Optimizer Output for the 4-Bit Binary Counter and the Decade Counter

Listing sum-of-products equations for BIN_DCD_CNTR

```

BIN[3] := S1*BIN[3]/BIN[2]
        + S1*BIN[3]/BIN[1]
        + S1*BIN[3]/BIN[0]
        + /S1*S0*D[3]
        + S1*/S0*BIN[3]
        + S1*S0*/BIN[3]*BIN[2]*BIN[1]*BIN[0];

BIN[2] := S1*BIN[2]/BIN[1]
        + S1*BIN[2]/BIN[0]
        + S1*S0*/BIN[2]*BIN[1]*BIN[0]
        + /S1*S0*D[2]
        + S1*/S0*BIN[2];

BIN[1] := S1*BIN[1]/BIN[0]
        + S1*S0*/BIN[1]*BIN[0]
        + /S1*S0*D[1]
        + S1*/S0*BIN[1];

BIN[0] := S1*S0*/BIN[0]
        + /S1*S0*D[0]
        + S1*/S0*BIN[0];

DCD[3] := S1*DCD[3]/DCD[2]*DCD[1]*DCD[0]
        + /S1*S0*D[3]
        + S1*/S0*DCD[3]
        + S1*S0*/DCD[3]*DCD[2]*DCD[1]*DCD[0];

DCD[2] := S1*DCD[2]/DCD[3]*DCD[1]
        + S1*DCD[2]/DCD[3]*DCD[0]
        + /S1*S0*D[2]
        + S1*/S0*DCD[2]
        + S1*S0*/DCD[2]*DCD[3]*DCD[1]*DCD[0];

DCD[1] := S1*DCD[1]/DCD[3]*DCD[0]
        + S1*S0*/DCD[1]*DCD[3]*DCD[0]
        + /S1*S0*D[1]
        + S1*/S0*DCD[1];

DCD[0] := S1*S0*/DCD[0]*DCD[3]
        + S1*S0*/DCD[0]*DCD[2]*DCD[1]
        + /S1*S0*D[0]
        + S1*/S0*DCD[0];

```

TB000280

1

Figure 1-48. PLPL Listing of Equations for the 4-Bit Binary Counter and the Decade Counter

V0001	INPUT	OUTPUT	V0008	INPUT	OUTPUT	V0015	INPUT	OUTPUT
	1	11111111		1	11111111		1	11111111
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789	Pin # : 1234567891		23456789
Expected: C01X0110XX ==>		HLLHLLH	Expected: C11X0110XX ==>		LHLHLHLH	Expected: C11X0110XX ==>		HLHHHLL
Computed:		HLLHLLH	Computed:		LHLHLHLH	Computed:		HLHHHLL
-----			-----			-----		
V0002	INPUT	OUTPUT	V0009	INPUT	OUTPUT	V0016	INPUT	OUTPUT
	1	11111111		1	11111111		1	11111111
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789	Pin # : 1234567891		23456789
Expected: C10X1111XX ==>		HLLHLLH	Expected: C11X0110XX ==>		HLLHLLH	Expected: C11X0110XX ==>		LLHLLHLL
Computed:		HLLHLLH	Computed:		HLLHLLH	Computed:		LLHLLHLL
-----			-----			-----		
V0003	INPUT	OUTPUT	V0010	INPUT	OUTPUT	V0017	INPUT	OUTPUT
	1	11111111		1	11111111		1	11111111
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789	Pin # : 1234567891		23456789
Expected: C00X0110XX ==>		HHHHHHH	Expected: C11X0110XX ==>		LLLHLLH	Expected: C11X0110XX ==>		HHLHLLH
Computed:		HHHHHHH	Computed:		LLLHLLH	Computed:		HHLHLLH
-----			-----			-----		
V0004	INPUT	OUTPUT	V0011	INPUT	OUTPUT	V0018	INPUT	OUTPUT
	1	11111111		1	11111111		1	11111111
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789	Pin # : 1234567891		23456789
Expected: C11X0110XX ==>		LHHHLHH	Expected: C11X0110XX ==>		HHHLHHH	Expected: C11X0110XX ==>		LHLHLLH
Computed:		LHHHLHH	Computed:		HHHLHHH	Computed:		LHLHLLH
-----			-----			-----		
V0005	INPUT	OUTPUT	V0012	INPUT	OUTPUT	V0019	INPUT	OUTPUT
	1	11111111		1	11111111		1	11111111
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789	Pin # : 1234567891		23456789
Expected: C11X0110XX ==>		HLHHHLH	Expected: C11X0110XX ==>		LHLLHHL	Expected: C11X0110XX ==>		HLLHHHH
Computed:		HLHHHLH	Computed:		LHLLHHL	Computed:		HLLHHHH
-----			-----			-----		
V0006	INPUT	OUTPUT	V0013	INPUT	OUTPUT	Simulation completed, Errors detected = 0		
	1	11111111		1	11111111			
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789			
Expected: C11X0110XX ==>		LLHLLHH	Expected: C11X0110XX ==>		HHHHHLH			
Computed:		LLHLLHH	Computed:		HHHHHLH			
-----			-----					
V0007	INPUT	OUTPUT	V0014	INPUT	OUTPUT			
	1	11111111		1	11111111			
Pin # : 1234567891		23456789	Pin # : 1234567891		23456789			
Expected: C11X0110XX ==>		HLLHHHLH	Expected: C11X0110XX ==>		LHHLLHL			
Computed:		HLLHHHLH	Computed:		LHHLLHL			

Figure 1-49. Simulation Run for the 4-Bit Binary Counter and the Decade Counter

Title: BIN_DCD_CNTR
 Part Type: PAL16R8*
 DEVICE: PAL16R8*
 MFG: AMD*
 F0*

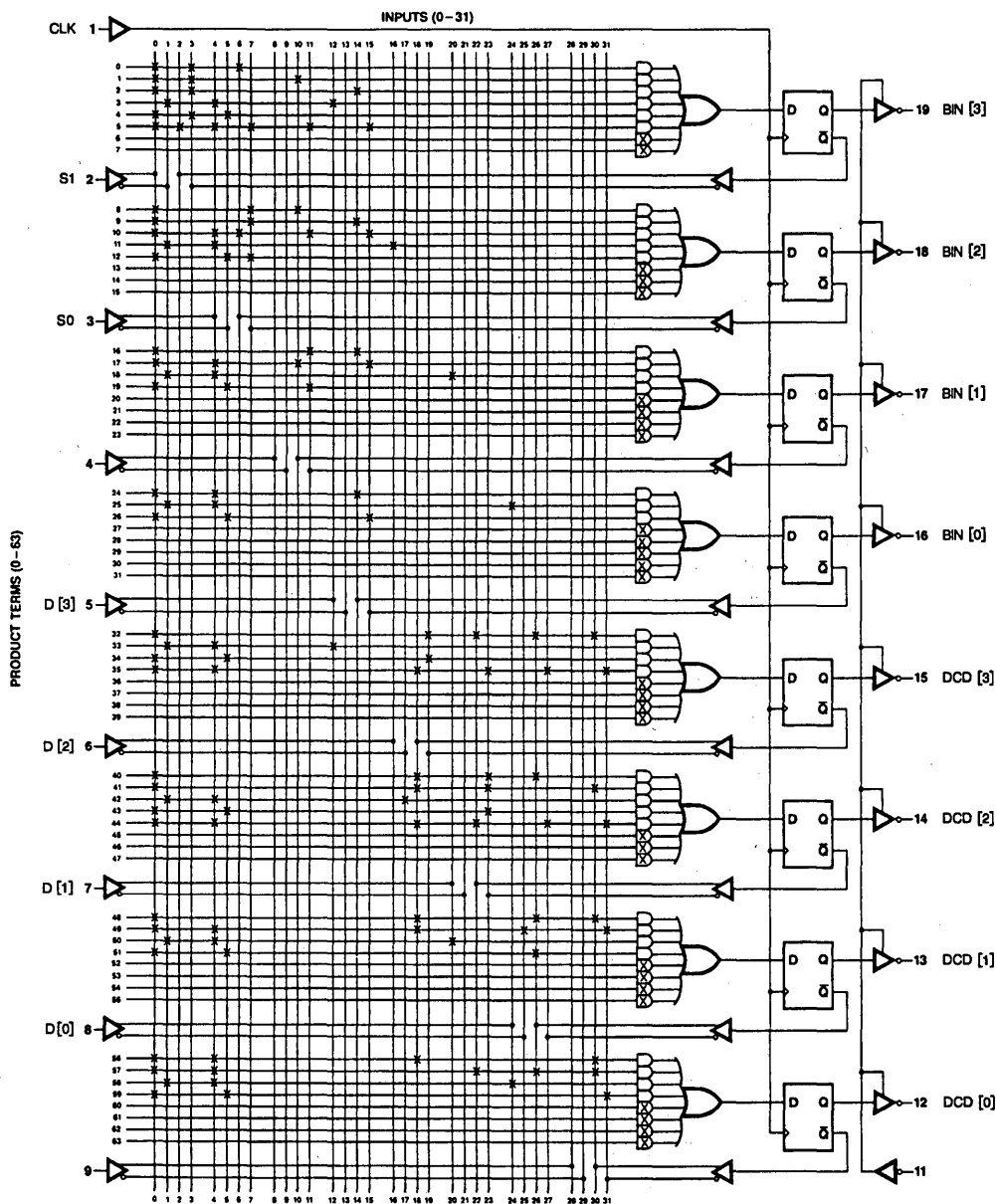
```

L0000 0110 1101 1111 1111 1111 1111 1111 1111*
L0032 0110 1111 1101 1111 1111 1111 1111 1111*
L0064 0110 1111 1111 1101 1111 1111 1111 1111*
L0096 1011 0111 1111 0111 1111 1111 1111 1111*
L0128 0110 1011 1111 1111 1111 1111 1111 1111*
L0160 0101 0110 1110 1110 1111 1111 1111 1111*
L0256 0111 1110 1101 1111 1111 1111 1111 1111*
L0288 0111 1110 1111 1101 1111 1111 1111 1111*
L0320 0111 0101 1110 1110 1111 1111 1111 1111*
L0352 1011 0111 1111 1111 0111 1111 1111 1111*
L0384 0111 1010 1111 1111 1111 1111 1111 1111*
L0512 0111 1111 1110 1101 1111 1111 1111 1111*
L0544 0111 0111 1101 1110 1111 1111 1111 1111*
L0576 1011 0111 1111 1111 1111 0111 1111 1111*
L0608 0111 1011 1110 1111 1111 1111 1111 1111*
L0768 0111 0111 1111 1101 1111 1111 1111 1111*
L0800 1011 0111 1111 1111 1111 1111 0111 1111*
L0832 0111 1011 1111 1110 1111 1111 1111 1111*
L1024 0111 1111 1111 1111 1110 1101 1101 1101*
L1056 1011 0111 1111 0111 1111 1111 1111 1111*
L1088 0111 1011 1111 1111 1110 1111 1111 1111*
L1120 0111 0111 1111 1111 1101 1110 1110 1110*
L1280 0111 1111 1111 1111 1101 1110 1101 1111*
L1312 0111 1111 1111 1111 1101 1110 1111 1101*
L1344 1011 0111 1111 1111 0111 1111 1111 1111*
L1376 0111 1011 1111 1111 1111 1110 1111 1111*
L1408 0111 0111 1111 1111 1101 1101 1110 1110*
L1536 0111 1111 1111 1111 1101 1111 1110 1101*
L1568 0111 0111 1111 1111 1101 1111 1101 1110*
L1600 1011 0111 1111 1111 1111 0111 1111 1111*
L1632 0111 1011 1111 1111 1111 1111 1110 1111*
L1792 0111 0111 1111 1111 1101 1111 1111 1101*
L1824 0111 0111 1111 1111 1111 1101 1101 1101*
L1856 1011 0111 1111 1111 1111 1111 0111 1111*
L1888 0111 1011 1111 1111 1111 1111 1111 1110*
C7CC9*
40F8

```

TB000260

Figure 1-50. Device Map for the 4-Bit Binary Counter and the Decade Counter



BD002002

Figure 1-51. AmPAL16R8 Programmed with 4-Bit Binary Counter and the Decade Counter Functions

4-Bit Decade Counter with D Flip-Flops

Table 1-6 shows the current and next state description of a 4-bit decade counter. Figure 1-52 shows the state-counting sequence.

A decade counter can be implemented with four flip-flops and some additional logic. The four flip-flops are represented by four variables A, B, C, and D. A represents the most significant bit and D represents the least significant bit. Note that the next states are unspecified for present states 1010, 1011, 1100, 1101, 1110 and 1111.

Logic for each flip-flop's output can be derived easily either from the state table or by the Karnaugh-map method. Figure 1-53 shows a four-variable Karnaugh-map format. A separate Karnaugh map is required for each flip-flop. States 1010, 1011, 1100, 1101, 1110 and 1111 are indicated as don't-care states by placing X in the Karnaugh maps.

Figures 1-54, 1-55, 1-56, and 1-57 show the next state maps for A*, B*, C* and D* as functions of A, B, C, and D. These next state maps are easily generated from Table 1-6.

Examination of the next state table indicates that the first seven transitions, states 0 to 6, do not produce a "1" output at the most significant flip-flop (A). Therefore, the map entry for this flip-flop in these states is set to "0" (Figure 1-54). When current state is either 7 or 8, the next state for this flip-flop becomes "1" and for current-state 9, the next state is "0".

Karnaugh maps for B*, C*, and D* can be derived similarly.

Having created the Karnaugh maps, we can attempt to minimize the Boolean expression for each flip-flop. Figure 1-58.a shows the unsimplified Min term expressions for each flip-flop derived from the state table. In Karnaugh-map minimization, Min term expressions can be simplified by grouping them with don't-care terms. Figure 1-58.b shows the Karnaugh-map minimization of the different functions.

Once the minimized SOP logic equations are generated the translation to a corresponding PAL device is relatively straightforward. Figure 1-51 shows the implementation of both the 4-bit binary counter and the decade counter in a single AmPAL16R8 PAL device.

TABLE 1-6. FUNCTION TABLE FOR A 4-BIT DECADE COUNTER

	INPUTS				OUTPUTS					
	S1	S2	A	B	C	D	A*	B*	C*	D*
CLEAR LOAD COUNT	0	0	x	x	x	x	0	0	0	0
	0	1	x	x	x	x	D3	D2	D1	D0
	1	0	0	0	0	0	0	0	0	1
	1	0	0	0	0	1	0	0	1	0
	1	0	0	0	1	0	0	0	1	1
	1	0	0	0	1	1	0	1	0	0
	1	0	0	1	0	0	0	1	0	1
	1	0	0	1	0	1	0	1	1	0
	1	0	0	1	1	0	0	1	1	1
	1	0	0	1	1	1	1	0	0	0
HOLD	1	0	1	0	0	0	1	0	0	1
	1	0	1	0	0	1	0	0	0	0
	1	1	x	x	x	x	Q3	Q2	Q1	Q0

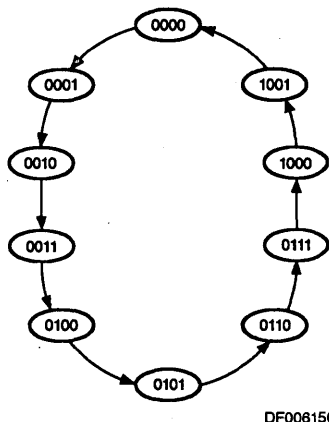


Figure 1-52. State Diagram of a 4-Bit Decade Counter (10 States)

AB	CD			
	00	01	11	10
00				
01				
11				
10				

Figure 1-53. Four-Variable Karnaugh-Map Format

		CD			
AB		00	01	11	10
	00	0	0	0	0
	01	0	0	1	0
	11	X	X	X	X
	10	1	0	X	X

$A^* = BCD + A^*/D$

Figure 1-54. Next State Map for A^*

		CD			
AB		00	01	11	10
	00	0	0	1	0
	01	1	1	0	1
	11	X	X	X	X
	10	0	0	X	X

$B^* = /B^*C^*D + B^*/C + B^*/D$

Figure 1-55. Next State Map for B^*

		CD			
AB		00	01	11	10
	00	0	1	0	1
	01	0	1	0	1
	11	X	X	X	X
	10	0	0	X	X

$C^* = C^*/D + /A^*/C^*D$

Figure 1-56. Next State Map for C^*

		CD			
AB		00	01	11	10
	00	1	0	0	1
	01	1	0	0	1
	11	X	X	X	X
	10	1	0	X	X

$D^* = /D$

Figure 1-57. Next State Map for D^*

$$\begin{aligned}
 A^* &= /A \cdot B \cdot C \cdot D + A \cdot /B \cdot /C \cdot /D \\
 B^* &= /A \cdot /B \cdot C \cdot D + /A \cdot B \cdot /C \cdot /D + /A \cdot B \cdot C \cdot D + /A \cdot B \cdot C \cdot /D \\
 C^* &= /A \cdot /B \cdot /C \cdot D + /A \cdot /B \cdot C \cdot /D + /A \cdot B \cdot /C \cdot /D + /A \cdot B \cdot C \cdot /D \\
 D^* &= /A \cdot /B \cdot /C \cdot /D + /A \cdot /B \cdot C \cdot /D + /A \cdot B \cdot /C \cdot /D + /A \cdot B \cdot C \cdot /D + A \cdot /B \cdot /C \cdot /D
 \end{aligned}$$

Figure 1-58.a. Reduced Equations from Function Table

$$\begin{aligned}
 A^* &= B \cdot C \cdot D + A \cdot /D \\
 B^* &= /B \cdot C \cdot D + B \cdot /C + B \cdot /D \\
 C^* &= C \cdot /D + /A \cdot /C \cdot D \\
 D^* &= /D
 \end{aligned}$$

Figure 1-58.b. Reduced Equations from Karnaugh Map

A high-level language design specification for the 4-bit decade counter is shown in Figure 1-45. Corresponding test vectors are shown in Figure 1-46 and are used as inputs for the simulator to test the logic equations. Figure 1-47 shows the PLPL Optimizer's output; Figure 1-48, the equation list; Figure 1-49, the simulation run; Figure 1-50, the device map; and Figure 1-51, the implementation of this function in an AmPAL16R8 device (Note that Figures 1-45 through 1-51 also implement a 4-bit binary counter function).

This simple example shows the advantages of using PLDs:

- User-defined functions can be programmed quickly,
- Two or more logic circuits can be integrated into a single PLD,

- PC board space is saved and power consumption reduced.

Numerous PLD CAD packages are available that simplify the system designer's tasks. You can express the circuit at a higher level instead of being restricted to the early design methodology of truth-table/Karnaugh-map/logic equation. Designs can now be expressed through a schematic-entry system, or through a high-level language syntax which concurrently improves design documentation.

In this chapter we have shown detailed design steps necessary for designing with PAL devices. A later section in this handbook contains more examples of logic design with PAL devices.

SECTION 2 — SOFTWARE, PROGRAMMING, TESTING, RELIABILITY and TECHNOLOGY INFORMATION



2.1 SOFTWARE SUPPORT FOR AMD's PROGRAMMABLE LOGIC DEVICES

2.1.1 Design-Aid Software for Programmable Logic

2.1.2 ABEL

2.1.3 CUPL, AmCUPL

2.1.4 PLPL

2.2 PROGRAMMING HARDWARE

2.3 TESTING INFORMATION

2.3.1 Factory Testing of PAL Devices

**2.3.2 How Testability is Designed into AMD's Programmable Logic
Devices**

2.3.3 Specifications for Switching-Delay Minimums

2.4 AMD PROGRAMMABLE LOGIC RELIABILITY

2.5 PROGRAMMABLE LOGIC TECHNOLOGY

2.5.1 IMOX-III — Advanced Bipolar Technology for PAL Devices

2.5.2 Advanced CMOS Technology for PAL Devices

2.1 SOFTWARE SUPPORT FOR AMD'S PROGRAMMABLE LOGIC DEVICES

Several design-aid software tools are available to the system designers to assist them in designing-in programmable logic devices (PLDs). Section 2.1.1 provides an introduction to these tools, while Sections 2.1.2 through 2.1.4 describe in detail the capabilities and features of some of the software packages currently available.

2.1.1 DESIGN-AID SOFTWARE FOR PROGRAMMABLE LOGIC

The main function of programmable logic design-aid software is to translate a custom logic design specification into a format which can be accepted by a programmer (Figure 2-1).

Programmable logic software is also an excellent tool for design simulation and documentation. Simulation aids in debugging an initial design and helps to assure that a device will operate as intended the first time instead of requiring multiple design iterations. Documentation capability is essential for someone other than the original designer to understand a custom programmable logic-specification.

2

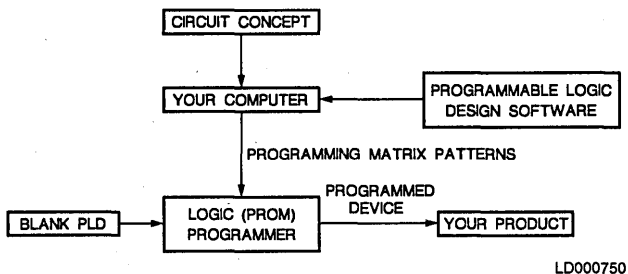


Figure 2-1. The Programmable Logic Development Cycle

Design Software for Programmable Logic

Available design software may be classified into two major categories: Assemblers and Compilers.

Boolean Assemblers

These programs allow you to use symbolic names for signals appearing on input and output pins. Equations must, however, be written at the fuse level, creating more work for the designer and producing a logic description which may be more difficult to understand.

Commonly available program of this type is PALASM.

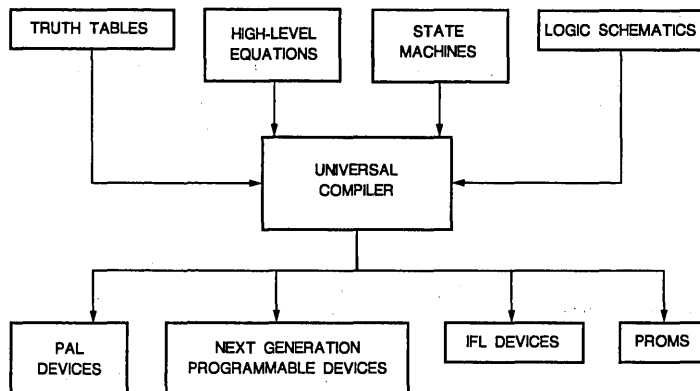
Logic Compilers

In contrast to the lower-level Boolean assemblers, the compiler lets the designer write logic descriptions at a higher level:

i.e., at a level that more accurately reflects the design concept. This type of software increases productivity while producing designs that are more thoroughly documented.

Available programs of this type are CUPL, ABEL, AmCUPL and PLPL.

When applied to programmable logic-design software, the term "universal" compiler (Figure 2-2) refers to the ability of the compiler to support all programmable-logic device types, all popular logic (PROM) programmers, and a large number of popular development computers. In addition, universal compilers offer a variety of input design formats such as state machines, high-level Boolean equations, truth tables, or logic schematics.



LD000760

Figure 2-2. The Universal Compiler

A universal compiler's syntax offers a general and easy description of the desired configuration of the chosen programmable logic device. This means that the functionality embodied in a particular logic description can be programmed into the same PLD from different manufacturers without altering the description at all.

In addition, the high-level description of the design provides flexibility in changing the design if so desired. A designer might use a particular type of PLD. Later, when fixes or enhancements require more product terms or an architectural configuration that the chosen PLD cannot support, the function can easily be placed in an alternate device. In many cases this will allow design modifications without altering printed circuit boards which may have already been manufactured.

Logic Simulation

Most of the programmable-logic software design-aid tools also offer logic-simulation capability to the designers. Logic simulation is typically performed to verify the logical design (logic equations) prior to programming an actual device. This may save some of the time spent trouble-shooting a programmable logic design using conventional techniques (scope and logic analyzer).

A simulation file consists of a table of stimulus patterns applied to inputs and response patterns expected at outputs. The simulator compares each stimulus/response pattern (vector) with the logic equations to verify that the expected response agrees with that produced according to the equations.

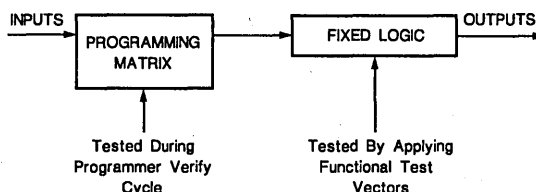
Not simulating may be of little consequence for simple designs, but for complex designs, especially complex sequential logic, it is well worth the time.

Testing Programmable Logic

Programmable-logic software design-aid tools also assist the designer in testing the PLDs after they have been programmed.

Before shipping a PLD, programmability may be verified by the manufacturer by exercising the device's address and programming circuitry on redundant test sites.

After the device has been received and programmed by the user, the logic programmer will read the states of all the fuses in the device and compare them with the data stored in the programmer's memory to check the status of the programming matrix in its verify cycle (Figure 2-3). If any mismatches are detected, the device is rejected.



LD000770

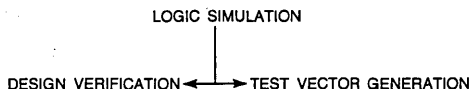
Figure 2-3. Programmable Logic Device Testing

However, a correct fuse verify does not guarantee that the device will work properly, since the fixed logic of the device has not been fully tested. To ensure proper operation, the device must be functionally tested.

Functional testing of PLDs involves applying stimulus patterns to a device while looking for the expected response. The test sequence consists of a table of stimulus/response patterns similar to those used to perform a simulation. Programmable-logic software design-aid programs offer the capability of generating these test patterns.

These patterns (test vectors) are produced by creating a simulation input file containing stimulus/response patterns. After running the simulator to verify the integrity of the vectors, they are appended to the JEDEC down-loadable file which already contains the programming patterns for the particular target device.

We can now see that there are two distinct benefits of logic simulation in working with PLDs.



Third-Party Software

Many different programmable-logic design-aid software programs and software programs resident on programmable logic programmers are available. Table 2-1 lists some current suppliers of these design tools. Contact the indicated companies for the status of their particular product.

The next three chapters of this section describe in detail the features of four high-level software design aids which support AMD's programmable logic devices. These software programs are ABEL, CUPL, AmCUPL, and PLPL.

TABLE 2-1. THIRD-PARTY SOFTWARE DESIGN-AID TOOLS

Vendor	Software	Hardware Platform
Data I/O Corp. 10525 Willows Road N.E. Redmond, WA 98073 (206) 881-6444	ABEL	IBM PC or compatible DEC VAX (VMS, UNIX) Apollo (AEGIS) Sun Microsystems (UNIX)
	DASH/CADAT DASH/ABEL	IBM PC or compatible
ISDATA Haid-und-Neu-Str. 7 D-7500 Karlsruhe West Germany (0721) 693092	LOGIC	IBM PC or compatible DEC VAX (VMS, UNIX) Apollo (AEGIS)
JMC PROMAC Division 2999 Monterey/Salinas Highway Monterey, CA 93940 (408) 373-3607	PALASM	PROMAC P3
Personal CAD Systems Inc. 1290 Parkmoor Avenue San Jose, CA 95126 (408) 971-1300	CUPL	IBM PC or compatible DEC VAX (VMS, UNIX)
	CAE	IBM PC or compatible
MMI (Public Domain)	PALASM	IBM PC or compatible PC (CPM-80) DEC VAX (VMS, UNIX)
AMD (Public Domain)	PLPL	IBM PC or compatible DEC VAX (UNIX)
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043 (415) 968-2900	PERFECT VISTA	IBM PC or compatible

2.1.2 ABEL

ABEL is a complete logic-design tool that lets you easily describe and implement programmable logic designs in PAL devices, IFLs, and PROMs. ABEL consists of a special-purpose, high-level language that is used to describe logic designs, and a language processor that converts logic descriptions to programmer-load files. Programmer-load files contain the information necessary to program and test programmable logic devices. Figure 2-4 shows ABEL used with Data I/O's Model 29 Logic Programmer in programmable logic development system.

ABEL may be used with several other Data I/O design development tools such as:

- PLDtest — an automatic test-vector generator that allows 100% testing of programmed logic parts

- DASH/ABEL — a schematic-diagram interface that converts schematic designs to ABEL source files
- PROMlink — a program that permits control of and communication with Data I/O programmers by means of a personal computer

Features of the ABEL design language are:

- Universal syntax for all programmable logic types
- High-level, structured design language
- Flexible forms for describing logic:
 - Boolean Equations
 - Truth Tables
 - State Diagrams
- Test Vectors for simulation and testing
- Time-Saving Macros and Directives

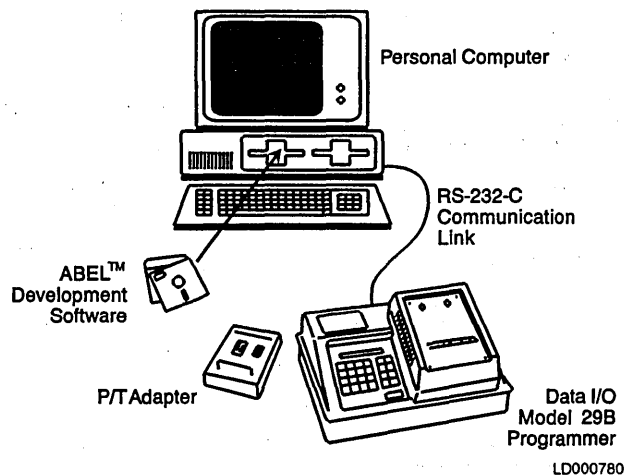


Figure 2-4. ABEL — A Logic Design Tool

The ABEL language processor also has many powerful features:

- Syntax checking
- Verification that a design can be implemented with a chosen part
- Logic reduction
- Design simulation
- Automatic design documentation
- Creation of programmer-load files in JEDEC and PROM format

Together, the ABEL design language and language processor make it easy to design and test logic functions that are to be implemented with programmable logic devices. For example, you can design a three-input AND function with the inputs A, B, and C and the output Y using a truth table like this:

```
truth_table "3-input AND gate"
  (A,B,C) → Y
  [0,.X,.X.] → 0;
  [.X., 0,.X.] → 0;
  [.X.,.X., 0] → 0;
  [1, 1, 1] → 1;
```

The ".X."s in the table indicate "don't-care" conditions, and the output Y is set to 1 only when all three inputs equal 1. You also could have specified the output Y in terms of simple Boolean operators and have achieved the same result. This is done here, where "&" is the logical AND operator:

$Y = A \& B \& C;$

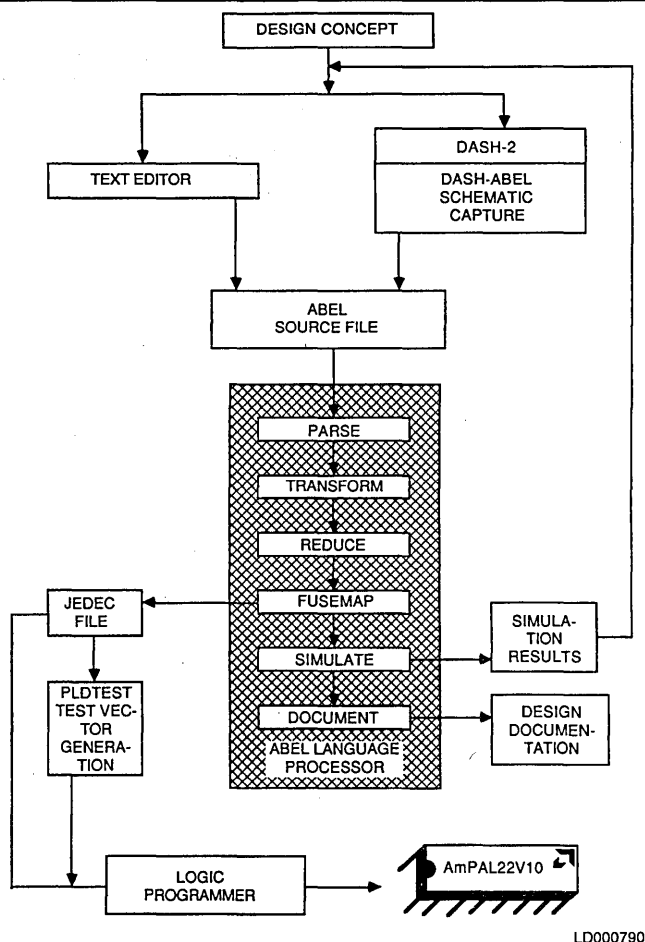
Design Process with ABEL

ABEL lets you choose the type of description that is best suited to the logic being described, or the type of description

you feel most comfortable with. And, in most cases, the same description can be used for many different devices simply by changing the device specification. ABEL enters the design process in a way that reduces errors and saves time. You can think about designs in a logical, functional way, describe them in that fashion, and then test your design to see that it operates as expected, all without worrying about which fuses should be blown or left intact.

Figure 2-5 shows the logic-design process and the role ABEL takes in it. Beginning with the design concept, the designer creates the ABEL source file required by the language processor in order for it to generate the programmer-load file. The source file is written by you and contains a complete description of your logic design. You can create the source file manually by means of a text editor (or word processor) that generates ASCII files, or you can use DASH/ABEL to convert a DASH-generated schematic of the design to an ABEL source file.

The source file is presented to the ABEL language processor which performs several functions to produce a programmer-load file (in JEDEC format) and design documentation. The first ABEL function, Parse, checks the syntax of the source file and flags any errors. Transform converts the logic description to an intermediate form. Reduce performs logic reduction, and Fusemap creates the programmer-load file. The programmer-load file can then be downloaded to the logic programmer to program parts, or can be first transmitted to PLDtest, an automatic test-vector generator. The Simulate function tests the design of the part against your test vectors contained in the source file and reports any functional failure of the design. The Document function generates a listing of the source file, a drawing of the logic device pin assignments, and a listing of the programmer-load file.



LD000790

Figure 2-5. Logic Design Steps with ABEL

More About ABEL Features

Design Checking

The language processor checks your logic design for correct language syntax and explicitly tells you where an error occurs and what the error is. The language processor also checks your design to see if it can be implemented on the chosen device. For example, if a device input pin is used as an output in an equation, the language processor detects and reports the error.

Logic Reduction

The language processor reduces your logic design to a near minimal form, so that you do not have to perform the tedious task of logic reduction by traditional methods such as Karnaugh maps. You may choose different levels of reduction based on the design and the device.

Simulation of a design is performed after a logic design has been reduced and converted to a programmer-load file. The simulation facility uses device characteristics, a fuse map, and test vectors to simulate the actual operation of the device. The

fuse map and test vectors used for simulation are the same as those that will be used to program and test the real device.

Functional Device Testing

If test vectors are specified in a source file, the programmer-load file created by the language processor contains these vectors in a form that can be used to test a programmed device with a logic programmer.

Standard JEDEC-Format Programmer-Load File

The standard programmer-load file created by the language processor conforms with the JEDEC Standard, No. 3, for data transfer to logic programmers. JEDEC-format files are used to transfer PAL and IFL designs to the logic programmer. Other formats for PROM programmers are supported.

System Requirements

ABEL presently runs on the following computers and operating systems. Versions for additional systems are under development.

- IBM/AT/XT and MS-DOS compatibles
- VAX/VMS
- VAX/Unix

- Sun
- Valid
- Apollo/Mentor

The configuration information and installation instructions for ABEL differ for each type of system. To install ABEL in your particular system, refer to the installation guide supplied with your ABEL package. In addition to ABEL, you will need an editor or word processor with which to create ABEL source files. This may be any editor of your choice as long as it produces a standard ASCII file.

Note: Some word processors, such as WordStar, operate in more than one mode, and may create non-standard ASCII files that cannot be used with ABEL. If you are using such a word processor, choose the mode of operation that creates printable ASCII files.

For downloading programmer-load files to a logic programmer, you will need:

- An RS-232C port and a cable to connect to the programmer.

2.1.2 ABEL

DESIGN TOOLS FOR PROGRAMMABLE LOGIC

Michael Holley
Project Engineering Manager
FutureNet Corporation
10525 Willows Road N.E.
Redmond, WA 98073-9746

Programmable logic devices consist of an array of logic gates whose interconnections may be programmed to implement specific logic designs. Gate interconnections are programmed by opening selected fuses while leaving others intact. The logic gates are generally arranged so that the device inputs are connected to AND gates which are in turn connected to OR gates. Two common types of structures are the PALtm structure, where the AND array is programmable, and the FPLA structure where both the AND and OR arrays are programmable.

Figure 1 shows the fuses opened and left intact for a basic PALtm in order to provide the function expressed in the equations. This PALtm application is quite simple and the fuses to be blown could be determined with a low-level design tool, such as PALASMtm, or even manually. However, many next-generation programmable logic devices contain complex macro cells that require powerful design tools to efficiently program them. In addition to the AND and OR gate interconnections, many newer devices contain programmable registers and programmable feedback paths from the outputs back to the AND gate inputs.

To program a programmable logic device, a conceptual design must be converted to a binary bit pattern that is loaded into a logic programmer. Since the gates are arranged in a sums of products form (AND-OR), early design tools required that the user transform the design to the sums-of-products form. The next generation programmable logic devices have multi-level logic so you don't have convert the design to the sum-of-products form, just sum-of-products equations.

PERSONAL SILICON FOUNDRY

One solution to the programming of complex programmable logic devices is the Personal Silicon Foundry from Data I/O Corporation. This desk-top system contains all the design tools and hardware to take a design from conception to programmed devices in just a few hours. PSF allows the engineer to express the design in the form of Boolean equations, truth tables, state diagrams, or even schematic diagrams. Design tools within PSF transform the design description, after logic reduction, into a sums of products form and then into the required bit patterns to program the logic device. This paper provides a brief description of the Personal Silicon Foundry and how it enhances the PC engineering workstation as a complete design system.

PSF includes an IBM-AT or -XT and a logic programmer, plus the following:

ABEL (Advanced Boolean Expression Language) - a high-level logic design language used to describe and implement programmable logic designs and a multi-program language processor to process logic descriptions (in ASCII format) to JEDEC-format logic programmer load files. Programmer load files contain the information necessary to program and test programmable logic devices.

DASH - a full-featured schematic design system that provides graphics output to printers/plotters and design file output to post-processes of CAD/CAE systems.

2.1.2 ABEL (Cont'd.)

```
!X = A & B & C & D
# !A & !B & C & D
# E & !F & !G;
```

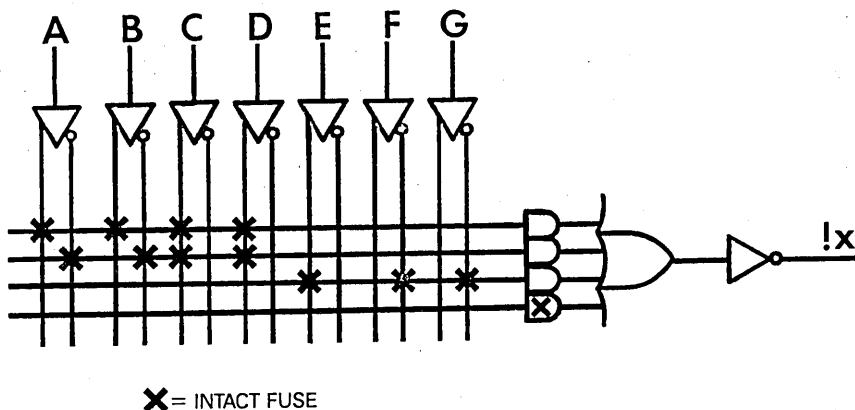


Figure 1. Intact PAL Fuses and the Logic Equations

DASH-ABEL - a schematic capture interface tool that allows logic designs to be expressed as a schematic diagram. DASH-ABEL also allows designs within schematic diagrams to be processed by ABEL.

PLDtest - a programmable logic device fault analysis tool that insures complete testing of devices after they are programmed.

To proceed from design concept through to programmed devices, several basic steps are followed, with optional choices to accommodate a variety of design situations. The flowchart presented in figure 2 shows the general flow of the design, simulation, testing, and programming operations provided by the system.

USING ABEL

The heart of the PSF is the ABEL Language Processor. The input to the language processor is a source file that contains all elements of the design, declarations for the target logic device, test vectors, etc. The ABEL source file is created in either of two ways: 1) with a text editor that generates an ASCII file, or 2) with the DASH schematic editor.

If a text editor is used, the designer keys-in the design using the desired method, i.e., equations, truth tables, or state diagrams (plus logic device pin information, test vectors, etc.) to produce a design description similar to those shown in figures 3 and 4. The partial source file in figure 3 describes a seven-segment display decoder by means of a truth table. The partial source file in figure 4 shows how a design (a traffic signal controller) is expressed by means of a state diagram.

2.1.2 ABEL (Cont'd.)

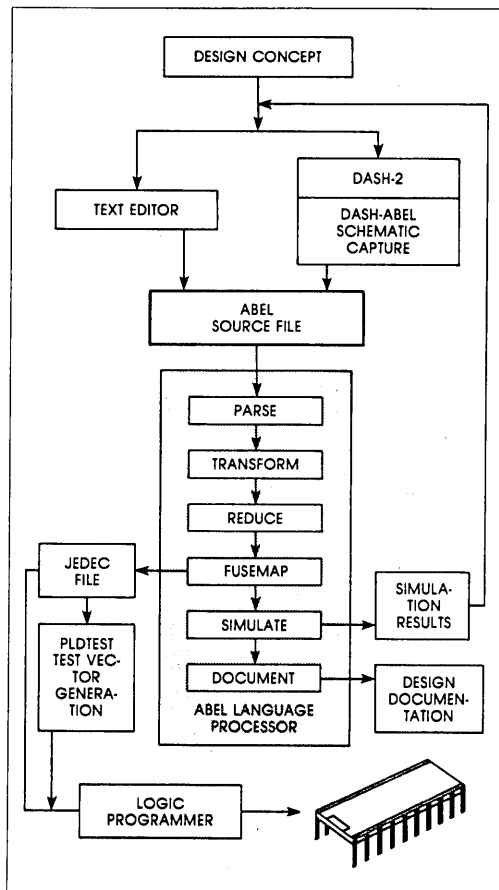


Figure 2. Logic Design Steps with PSF

If the DASH schematic editor was used to express the design, it would be translated by the DASH-ABEL Schematic Capture program to produce source file with Boolean equations substituted for the logic circuit. (Refer to Session 8 for more information on the schematic capture feature of PSF.)

Reduction of Logic Terms

As indicated in figure 2, the completed source file is input to the ABEL Language Processor. The first step is the Parse-Transform process. These programs read the source file and convert truth table or state diagrams to Boolean equations in a non-minimized form.

The subsequent Reduce program applies DeMorgan's theorem to convert the equations to the sum-of-products form minimize the logic description so that fewer product terms are used in the programmable logic device.

Fusemap and Simulation

The Fusemap program processes the output file from the Reduce program and creates a programmer load file. The output of the Fusemap program is passed either to the Simulate program or (in JEDEC format) to the logic programmer or to PLDtest.

```

module bcd7rom flag "-d82"
title "BCD to seven segment display decoder
Data I/O Corp Redmond WA 27 Feb 1986"
"
"      a      BCD-to-seven-segment decoder
"      f | g | b
"      e | d | c      segment identification
"
U6      device "RAS7P8";

D3,D2,D1,D0 pin 10,11,12,13;
a,b,c,d,e,f,g pin 1,2,3,4,5,6,7;
ena pin 15;

bcd = [D3,D2,D1,D0];
led = [a,b,c,d,e,f,g];

ON,OFF = 0,1; " for common anode LEDs
L,H,X,Z = 0,1,.X,.Z.;

truth_table
(bcd -> [ a , b , c , d , e , f , g ])
0 -> [ ON , ON , ON , ON , ON , ON , OFF ];
1 -> [ OFF , ON , ON , ON , OFF , OFF , OFF ];
2 -> [ ON , ON , OFF , ON , ON , OFF , OFF ];
3 -> [ ON , ON , ON , ON , OFF , OFF , ON ];
4 -> [ OFF , ON , ON , ON , OFF , OFF , ON ];
5 -> [ ON , OFF , ON , ON , ON , OFF , ON ];
6 -> [ ON , OFF , ON , ON , ON , ON , ON ];
7 -> [ ON , ON , ON , OFF , OFF , OFF , OFF ];
8 -> [ ON , ON , ON , ON , ON , ON , ON ];
9 -> [ ON , ON , ON , ON , OFF , ON , ON ];
  
```

Figure 3. An ABEL Source File Using a Truth Table to Describe the Design

2.1.2 ABEL (Cont'd.)

state_diagram Count

```
State 0:      case      SenseA & !SenseB : 0;
                !SenseA & SenseB : 4;
                (SenseA == SenseB) : 1;
            endcase;

State 1:      goto 2;          "Delay three clocks
State 2:      goto 3;
State 3:      goto 4;

State 4:      GreenA := Off;
                YellowA := On ;
                goto 5;

State 5:      YellowA := Off;
                RedA := On ;
                RedB := Off;
                GreenB := On ;
                goto 8;

State 6:      goto 15;         "Unused states
State 7:      goto 15;

State 8:      case      !SenseA & SenseB : 8;
                SenseA & !SenseB : 12;
                (SenseA == SenseB) : 9;
            endcase;

State 9:      goto 10;         "Delay three clocks
State 10:     goto 11;
State 11:     goto 12;

State 12:     GreenB := Off;
                YellowB := On ;
                goto 13;

State 13:     YellowB := Off;
                RedB := On ;
                RedA := Off;
                GreenA := On ;
                goto 0;

State 14:     goto 15;         "Unused state
State 15:     GreenA := On ;   "Power on initialize state
                YellowA := Off;
                RedA := Off;
                GreenB := Off;
                YellowB := Off;
                RedB := On ;
                goto 0;
```

@page

Figure 4. Using a State Diagram to Describe a Logic Design in an ABEL Source File

2.1.2 ABEL (Cont'd.)

The Simulate program uses the design and device information to simulate operation of the design in a programmable logic device. That is, it simulates operation of the device as if it were already programmed with the information contained in Fusemap output file. If the design fails to operate in accordance with the test vectors, errors are listed that indicate any failed functions. The flagged errors allow the design to be corrected early in the development stage, before any programmable logic devices are actually programmed.

The Programmer Load File and PLDtest

After successful simulation, the JEDEC programmer load file is passed on to the logic programmer so that parts can then be programmed with the logic design. Before device programming, the JEDEC file can be passed through PLDtest, a fault analysis tool that insures complete testing of programmable logic devices. PLDtest generates a series of test vectors that are added to the JEDEC file so that the logic programmer can perform additional testing on each device after it is programmed. This lowers the device failure rate by identifying marginal devices before they can be placed in inventory or installed in the end-product. Since programmable

logic devices cannot be adequately tested before they are programmed with the logic design, manufacturers cannot provide this type of testing and fault grading before device leave the factory.

Design Documentation

The final step of the ABEL Language processor is that of providing design documentation. This feature is of real benefit to engineers take no delight in producing thorough documentation. The Document program generates printed reports. Included in the documentation is signal name/pin information for the device, a list of reduced equations that show the design, a pinout diagram of the device, a representation of the fusemap, and the test vectors.

CONCLUSION

Personal Silicon Foundry is a complete design, simulate, and testing system for programmable logic. It is intended to reduce the time between concept and programmed parts to a matter of hours. All operating software is tried and proven with more than a year of service in the customer base.

Designer's Guide to: Programmable logic—Part 1

Compiler-based software and PLDs improve logic design

Programmable logic devices allow you to complete a design faster than you can using SSI devices or custom ICs, and PLD implementations take up less space than do SSI-based circuits. Moreover, easy-to-use compiler-based languages that don't require you to understand PLD architectures make PLDs increasingly attractive for logic designs.

Bob Osann, *Assisted Technology*

Circuits that incorporate programmable logic devices (PLDs) take up less board space than do SSI-based implementations and require less design time than do custom-IC or SSI-based versions. But until recently, the PLDs' unusual architecture and lack of software support made designers hesitant to use the devices, despite the advantages they offer. Compiler-based software, however, is simplifying PLD use; this high-level software makes it unnecessary for you to be concerned with the PLDs' internal details when implementing logic functions with the devices.

This first article in this 3-part series, which is aimed at first-time PLD users, discusses basic PLD architecture and shows you how to replace two simple logic

designs with PLDs using a compiler-based PLD design language. Part 2 will show you how to replace more complicated combinatorial and registered-TTL designs with PLDs. Part 3 will introduce the state-machine concept and show you how to implement a logic design directly, without ever developing a gate-level description of the system.

Although the PLD approach lets you go from logic function to PLD circuit without conceiving a gate-level description, when designers decide to use PLDs, they usually have either completed TTL designs that they want to shrink or else gate-level descriptions of circuits they don't want to implement in discrete logic. Therefore, the first two articles in this series target converting existing designs.

Why use a PLD?

For one-of-a-kind designs, prototypes, or small production runs, designers have traditionally taken the discrete approach. Discrete designs are easy to modify and inexpensive to manufacture in small quantities, and you can complete them more quickly than you can complete custom or semicustom designs. For production runs over 500, designers have typically chosen the semicustom and custom routes and sacrificed short design cycles and ease of modification to reduce manufacturing costs.

PLDs bridge the gap between bulky discrete designs and long custom-IC design cycles. On the one hand, PLD designs are easier to modify than SSI-based ones

2.1.3 CUPL, AmCUPL (Cont'd.)

and use much less space. Moreover, depending on the application, they can cost less than SSI-based implementations for even small production runs. And on the other hand, although custom ICs can prove more economical than PLDs for large production runs, PLD design cycles are much shorter. So, if you need to get a small, inexpensive design to market quickly and can't wait for a completed custom design, PLDs can provide you with a quick stand-in until your custom design is completed.

In general, the PLD architecture contains a fixed logic array made of AND gates—whose outputs feed

A PLD approach allows designers to go from a logic function to a PLD-based circuit without conceiving a gate-level description.

OR gates—and a programming matrix. The programming matrix is made up of fuses that you blow with a programming device. By blowing the appropriate fuses, you can achieve any AND/OR product or combination. Fig 1 shows the PAL-type and FPLA-type architectures. The total number of terms that you can generate is limited only by the size of the matrix.

Because you can represent any logical function as the logical sum of product terms, you can realize any logical function using a PLD. A product term consists of any combination of input variables or their complements ANDed together. A logical sum is any combination of

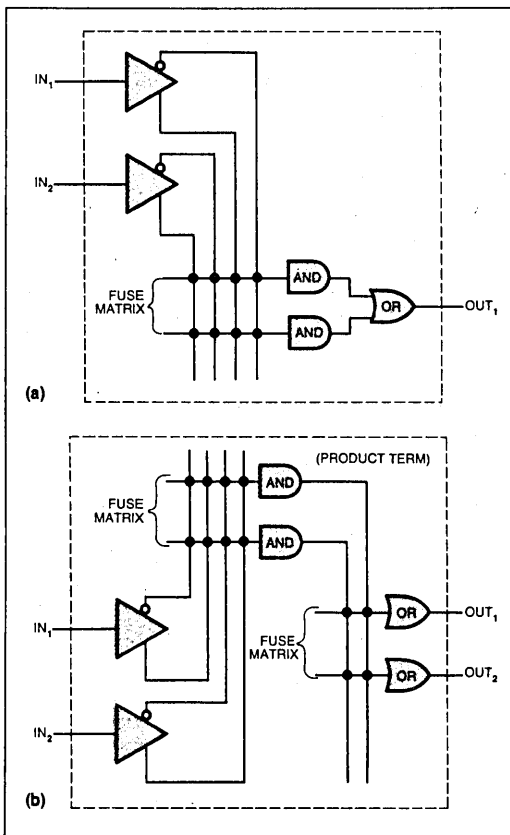


Fig 1—Typical PLDs use one of two general architectures to permit implementation of a wide range of logic functions. PAL-type devices (a) prove easier to use, but FPLAs (b) provide more flexibility by allowing two levels of programmability.

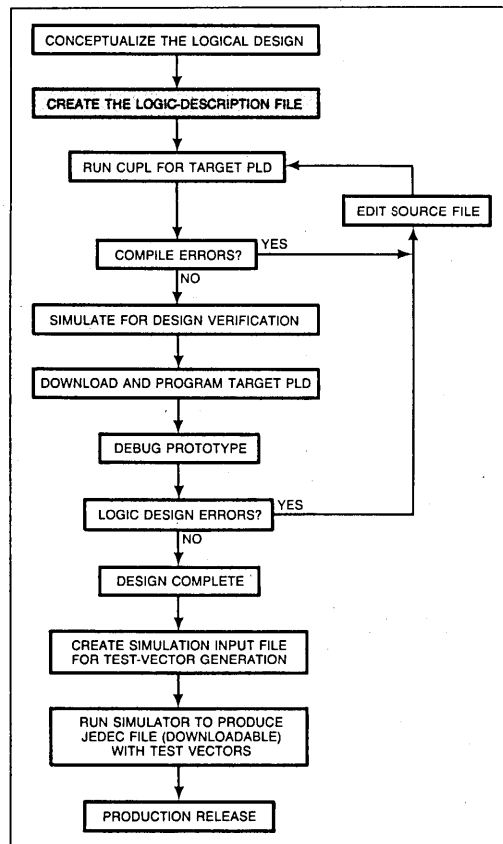


Fig 2—PLDs greatly simplify logic design. After you complete the logic-description file, the PLD software automatically compiles the data for downloading to a programming device.

2.1.3 CUPL, AmCUPL (Cont'd.)

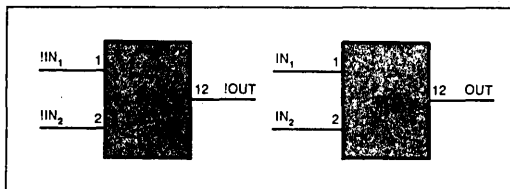


Fig 3—When using CUPL, you can always write your logic equations in positive logic, regardless of the actual polarity of the signals entering the device. For example, the two cases illustrated above both yield the same logic equation: $OUT = IN_1 \& IN_2$.

product terms ORed together. Using De Morgan's theorems,

$$\overline{(AB)} = \overline{A} + \overline{B}, \text{ and}$$

$$\overline{(A+B)} = \overline{A} \cdot \overline{B}.$$

Then, using the distributive property,

$$A(B+C) = \overline{A} + \overline{B}, \text{ and}$$

$$(A+B)(C+D) = AC + AD + BC + BD.$$

The PLD software determines the best form of the equation that will fit into a PLD, which uses a general architecture to permit implementation of a wide range of functions. The software should allow you to think in terms of logical functions rather than gates. The better the software, the more you can abstract from the details of discrete design and attend to system concerns.

Once you've decided to use a PLD approach, you'll need to choose the software development support for that device. You can use two basic types of software: assembler-based software and compiler-based software (Ref 1). Assembler-based software is supplied by the PLD manufacturer; it typically supports only that manufacturer's devices. If you buy PLDs in large quantity, you can usually get the software for well under \$100. An alternative to assembler-based software is the compiler-based software sold by Data I/O and Assisted Technology. Compiler-based software supports almost all PLD devices and programmers; typical prices range from \$750 for a version that runs on CP/M-based systems to \$2695 for a version that runs on VAX/VMS systems.

Although compiler-based software is more expensive, it will make your PLD design task easier. Capabilities such as symbolic signal representation and macro

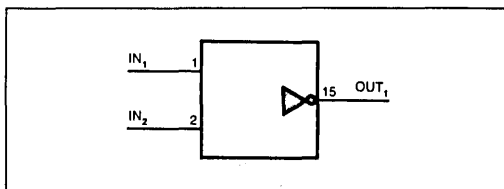


Fig 4—Some PLD devices use an inverting output buffer. As a result, to accommodate applications that demand an active-high output signal, the compiler often must generate extra product terms that might make the design too big for the target PLD.

substitution make it easier for you to formulate and enter your logic equations. These improvements allow you to formulate your design at a higher conceptual level; that is, you can think in terms of systems instead of individual circuits.

Fig 2 illustrates the PLD design process using Assisted Technology's CUPL language. (The Abel language, developed by Data I/O, could also be used to demonstrate the techniques involved.)

The CUPL syntax

Before you can design with CUPL, you have to learn the syntax. CUPL's operators, which were chosen largely from the C programming language, are as follows:

&=logical AND

#=logical OR

\$=logical exclusive-OR

!=logical negation.

You can place comments anywhere within a CUPL logic specification by using the symbol /* for "start comment" and the symbol */ for "end comment." You can also nest parentheses to any level, as in this example: $OUT = !((A \& B) \& (C \# (D \& E)))$.

To facilitate clear documentation, CUPL allows you to use symbolic names of arbitrary length (the first 31 characters must be unique). Symbolic names can represent pin variable names, internal device nodes, intermediate variables, bit-field representations, and symbolic constants. To further improve clarity, you can use the underscore character—

RAMPARITY_INTEN.

When you're converting an existing design, CUPL allows you to give symbolic names to internal nodes within your design. For example, for flip-flops connected to the pin PIN_VAR, you would name the node as follows:

- D-type flip-flop—PIN_VAR.D=Expression

2.1.3 CUPL, AmCUPL (Cont'd.)

- JK-type flip-flop— $\text{PIN_VAR.J} = \text{Expression}$, $\text{PIN_VAR.K} = \text{Expression}$
- RS-type flip-flop— $\text{PIN_VAR.R} = \text{Expression}$, $\text{PIN_VAR.S} = \text{Expression}$.

For 3-state-device enable signals connected to a pin, you would write:

- $\text{PIN_VAR.OE} = \text{Expression}$
- $[\text{PIN_VAR LIST}].\text{OE} = \text{Expression}$,

as in $[\text{DATA7..0}].\text{OE} = \text{Expression}$. If you're leaving the 3-state device enabled, you don't have to write an equation for it.

Handling signal polarities

One issue that often confuses first-time PLD users is the representation of signal polarities. In CUPL, you can always write equations in positive logic, regardless of the polarity of the signals entering the device. Because all signals entering the PLD are buffered, you have access to both the true and complement versions of the input signal for your logic equations. Fig 3 illustrates two simple cases. For each case—if you were using the PLD as an AND gate—you would write the same logic equation: $\text{OUT} = \text{IN}_1 \& \text{IN}_2$.

The specification of signal polarities is complicated by the inverting-output architecture of, for instance, 20-pin PAL devices (Fig 4). If you need an active-low output polarity, this doesn't create a problem. In this case, the compiler has to implement only one P (product) term. However, if you need an active-high output signal, the compiler must apply De Morgan's theorem,

The PLD architecture contains a fixed logic array made of a programming matrix and AND gates whose outputs feed OR gates.

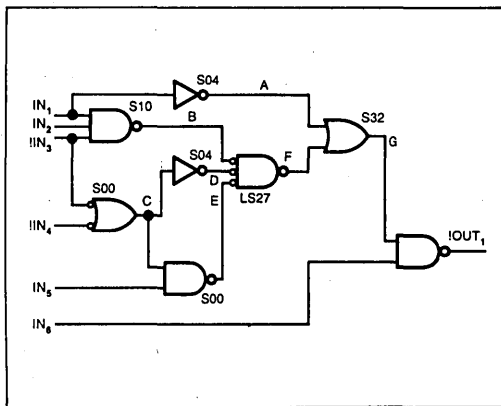


Fig 6—Reduced propagation delays are one of the benefits of using PLDs. A PLD implementation of the circuit shown here has, on the average, half the propagation delay of the discrete implementation.

and $!\text{OUT}_1 = !(\text{IN}_1 \& \text{IN}_2)$ becomes $!\text{IN}_1 \# !\text{IN}_2$. Note that this equation contains two product terms. The additional space the compiler requires reduces the probability that the compiler will be able to fit the logic function into the target PLD.

CUPL can eliminate this problem for PLD devices that have programmable output polarities. CUPL automatically chooses the output polarity that will result in the fewest number of P terms.

Reduce keystrokes

One of CUPL's (and Abel's) major advantages is macro substitution, the ability to use a single variable name to represent a complex logical equation. For example, if you define "INT_VAR" as "A&B#C," the compiler will insert A&B#C every time it encounters INT_VAR.

Because macro substitution lets you use fewer keystrokes to write equations, it saves time and reduces the probability that you'll make input errors. By using macro substitution, you can write your logic specification in a hierarchical fashion, breaking complex equations into more manageable and readable pieces.

The logic description

The heart of CUPL is the logic-description file (LDF), which contains your logic equations, pin declarations, intermediate variables, and documentation describing the device's function. You must complete the LDF to prepare your logic equations for downloading to

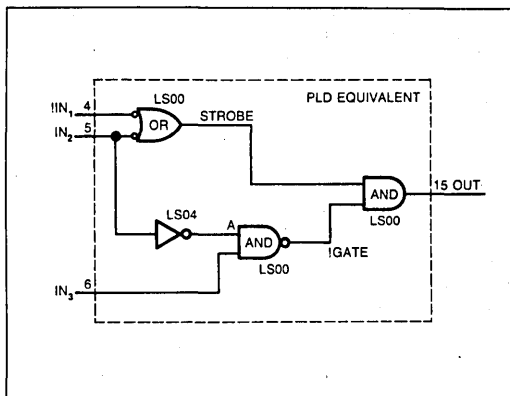


Fig 5—With CUPL, you can often replace a TTL design without understanding its function. You just name the pins and nodes, combine them according to gate relationships in the circuit, and the software does the rest.

2.1.3 CUPL, AmCUPL (Cont'd.)

a programming device. Table 1 shows the format for a CUPL LDF that was written for a memory decoder.

The following example shows you how to complete the logic equation, pin declaration, and intermediate variable portions of an LDF for the design in Fig 5. First, you write the pin declarations using the same names and signal polarities that appear on your schematic. Next, you name the output of each gate in the

schematic. In the example, STROBE, A, and !GATE are the intermediate variables. Using the intermediate variable definitions, you then write an equation for the output:

```
PIN 4=!IN1
PIN 5=IN2
PIN 6=IN3
PIN 15=OUT
A=!IN2
STROBE=!(!IN1)#(!IN2); /*(!IN1) = IN1*1
!GATE=!A&IN3)
OUT=STROBE&!GATE.
```

The following expressions show this strategy applied to the more complicated design in Fig 6:

```
A=!IN1
B=!(!IN1&IN2&IN3)
C=!(!IN3)#(!IN1)
D=!C
E=!C&IN5)
F=!B&!D&!E
G=A#F
!OUT=(G&IN6).
```

The design in Fig 6 illustrates another advantage of using PLDs instead of discrete logic. The propagation delay in the PLD implementation is often less than that in the discrete design. The discrete design for this circuit requires at least three TTL packages and has five levels of delay. The total delay time is 50 nsec (five levels times 10 nsec/level) for LS packages and 26 nsec (4×4 nsec+10 nsec) for a combination of LS and Schottky TTL packages. In an equivalent PLD circuit, the maximum delay is 25 nsec; typical delay is only 15 nsec.

Registered PLDs

Some of the more complicated types of PLDs use flip-flops in their output stages to store information. Most of these PLDs provide integral feedback paths. The simplest registered PLDs contain D-type flip-flops, which transfer the signal at their D input to their Q output after one clock pulse (more specifically, after the application of a positive-going leading edge). The equations for the flip-flop in Fig 7 are

```
OUTPUT.D=G&INPUT /*UPDATE WITH INPUT*/
# !G&OUTPUT; /*MAINTAIN CURRENT OUTPUT*/
/*VIA INTERNAL FEEDBACK DATA*/.
```

For simple registered designs, you can often model

TABLE 1—SOURCE SPECIFICATION FILE FORMAT

FUNCTION	DESCRIPTION
PART NO 900 16487	HEADER INFORMATION:
NAME MEMDEC	IDENTIFIES THE
DATE 07/18/84	PARTICULAR LOGIC
REV 03	SOURCE FILE
DESIGNER OSANN	
COMPANY ATI	
ASSEMBLY PC-RAM	
LOCATION 417	
THIS DEVICE DECODES ADDRESSES FOR THE DYNAMIC RAM AND PROVIDES THE RAS STROBES AS WELL AS A SIGNAL THAT INITIATES CAS.	TITLE BLOCK: DESCRIBES IN PLAIN TERMS WHAT THIS DEVICE DOES.
ALLOWABLE TARGET DEVICE TYPES: PAL 16L8, 625153, EP300.	DEVICE MENU: LISTS ALL TARGET DEVICE TYPES THAT MAY BE USED.
INPUTS:	PIN DECLARATIONS:
PIN [1..6] = [A19..14]	CPU ADDRESS BUS
PIN [7..8] = [MEMW..MEMR]	MEMORY DATA STROBES
PIN 9 = !REF_ADDR_EN	INDICATES REFRESH CYCLE IN PROGRESS
PIN 11 = !REF_RAS	STROBE FOR RAS-ONLY REFRESH
PIN 13 = ALT_LOC	PLACE MEMORY IN ALTERNATE RANGE
OUTPUTS:	
PIN [19..16] = [RAS 3..0]	RAM ROW ADDRESS STROBES
PIN 14 = !CAS_INIT	ENABLE CAS STROBES
DECLARATIONS AND INTER-MEDIATE VARIABLE DEFINITIONS:	WRITE EQUATIONS FOR BIT-FIELD DECLARATIONS AND INTERMEDIATE VARIABLES WHICH WILL BE SUBSTITUTED LATER USING MACRO-SUBSTITUTION:
FIELD MEMADR = [A19..A14]	MEMORY ADDRESS
MEM REQ = MEMW # MEMR	MEMORY REQUEST
LOGIC EQUATIONS:	WRITE EQUATIONS FOR OUTPUTS IN TERMS OF INPUTS AND FEEDBACK AS IN:
	OUTPUT = INPUT 1 & FEEDBACK 1
	# INPUT 2 & FEEDBACK 2
	# INPUTS N & FEEDBACK N
FUNCTION	DESCRIPTION
RAS 3 = MEMREQ & !REF_ADDR_EN & (!ALT_LOC & MEMADR: [0C000...0FFF])	PRIMARY RANGE
# ALT_LOC & MEMADR: [FC000...FFFF]	ALTERNATE RANGE
# REF_ADDR_EN & REF_RAS	REFRESH CYCLE
RAS 2 = MEMREQ & !REF_ADDR_EN & (!ALT_LOC & MEMADR: [08000...0BFFF])	PRIMARY RANGE
# ALT_LOC & MEMADR: [FB000...FBFFF]	ALTERNATE RANGE
# REF_ADDR_EN & REF_RAS	REFRESH CYCLE
RAS 1 = MEMREQ & !REF_ADDR_EN & (!ALT_LOC & MEMADR: [04000...03FFF])	PRIMARY RANGE
# ALT_LOC & MEMADR: [F4000...F3FFF]	ALTERNATE RANGE
# REF_ADDR_EN & REF_RAS	REFRESH CYCLE
RAS 0 = MEMREQ & !REF_ADDR_EN & (!ALT_LOC & MEMADR: [00000...03FFF])	PRIMARY RANGE
# ALT_LOC & MEMADR: [F0000...F3FFF]	ALTERNATE RANGE
# REF_ADDR_EN & REF_RAS	REFRESH CYCLE
CAS_INIT = MEMREQ & !REF_ADDR_EN & (!ALT_LOC & MEMADR: [00000...0FFF])	PRIMARY RANGE
# ALT_LOC & MEMADR: [F0000...FFFF]	ALTERNATE RANGE

2.1.3 CUPL, AmCUPL (Cont'd.)

Compiler-based software for PLD design includes such features as symbolic signal representation and macro substitution.

the circuit with a timing diagram. Using the timing diagram, you can write your logic equations easily. In the Fig 8 timing diagram for a D-type flip-flop, INPUT₂ initiates the input pulse, and INPUT₁ terminates the output pulse. The pin declarations are

```
PIN 3=INPUT1
PIN 6=INPUT2
PIN 1=CLOCK
PIN 14=OUTPUT,
```

and the corresponding logic equations are

```
OUTPUT.D=!OUTPUT&INPUT 2 /*SET FF*/
# OUTPUT&!INPUT 1; /* KEEP FF SET*/
/* UNTIL INPUT 1*/
/*GOES ACTIVE*/.
```

These equations demonstrate one method for using

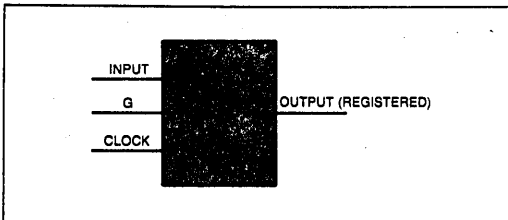


Fig 7—Some PLDs use registered outputs to introduce storage elements into their architecture.

the smallest possible number of product terms to keep a D flip-flop set for several clock cycles. Here, the flip-flop's output is fed back until some condition is met that again enables the flip-flop.

If the registered PLD contained JK flip-flops, the expressions would be

```
OUTPUT.J=INPUT2; /* SET FF*/
OUTPUT.K=INPUT1; /* RESET FF*/.
```

To handle more complicated sequential designs, you can model your circuit as a multiple-flip-flop system that uses a common clock. (Virtually all currently available registered PLDs use common clocks for their flip-flops.) For example, to convert TTL designs that use cascaded flip-flops (in which the outputs of some flip-flops are used to clock other flip-flops), you must find the originating clock in the circuit, which is usually

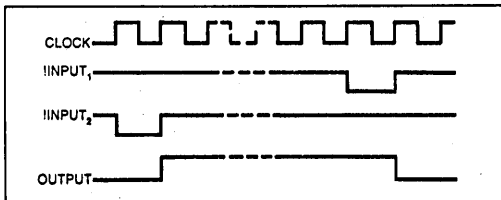


Fig 8—Converting logic designs to PLDs is easy once you've completed a timing diagram for your circuit. This one represents operation of a D-type flip-flop.

TABLE 2—CUPL OPTION FLAGS

A	PRODUCE YOUR_FILE_NAME.ABS FOR LATER USE BY CSIM.
L	PRODUCE YOUR_FILE_NAME.LST WITH LINE NUMBERS AND ERROR MESSAGES.
I	PRODUCE YOUR_FILE_NAME.HL DOWN-LOADABLE HL FORMAT FILE FOR IFL.
H	PRODUCE YOUR_FILE_NAME.HEX MMI PAL ASCII-HEX FORMAT FILE.
F	PRODUCE YOUR_FILE_NAME.DOC WITH FUSE MAP FILE.
X	PRODUCE YOUR_FILE_NAME.DOC WITH FULLY EXPANDED EQUATIONS.
G	PROGRAM SECURITY FUSE.
R	DISABLE GLOBAL PRODUCT-TERM MERGING. (FPLA DEVICES).
M0	PERFORM NO LOGIC MINIMIZATION.
M1	PERFORM LOCAL LOGIC MINIMIZATION.
M2	PERFORM LOGIC MINIMIZATION UNTIL EQUATIONS FIT IN TARGET DEVICE.
M3	PERFORM FULL LOGIC MINIMIZATION.
D	DEACTIVATE UNUSED OR-TERMS. (INCREASES SPEED IN FPLAs).
U	SET ALTERNATE SEARCH PATH FOR PLD DEVICE DATABASE.
J	PRODUCE YOUR_FILE_NAME.JED, THE JEDEC FORMAT DOWNLOADABLE FILE
S	AUTOMATICALLY RUN CSIM AFTER RUNNING CUPL

2.1.3 CUPL, AmCUPL (Cont'd.)

the highest-frequency source in the circuit. In most cases, the timing skew from one flip-flop output to the next is tolerable.

The TTL circuit in Fig 9 contains an LS161 counter whose output is decoded in an LS138. The decoded output sets and resets flip-flops at various points in the timing cycle. The timing diagram in Fig 10 is based on the assumption that the clock rate is sufficiently high that the propagation delays from SYSCLK to OUT₁ and OUT₂ are not significant. If you were to implement this design in a PLD, the pinout would look like the one shown in Fig 11. Outputs Q₀ and Q₁ were added to make all eight time slots in the circuit's cycle a unique combination of the four outputs. Adding Q₀ and Q₁ results in a timing sequence like the one in Fig 12.

You can now write the logic equations by noting, for each output, each place in the timing cycle where the output reads high (the flip-flop is set). For example, OUT₁ is set during time slots 2, 3, and 4. (The equation for the D input should include representations of time slots 1, 2, and 3; these time slots occur immediately before the flip-flop is set.) For time slots 1 through 3, you can now write

```
OUT1.D=!OUT1&!OUT2&Q0&!Q1 /*TIME SLOT 1*/
      #OUT1&!OUT2&!Q0&!Q1 /*TIME SLOT 2*/
      #OUT1&!OUT2&Q0&!Q1 /*TIME SLOT 3*/
```

Writing these equations is easier if you first define each time slot in terms of the register outputs that are fed

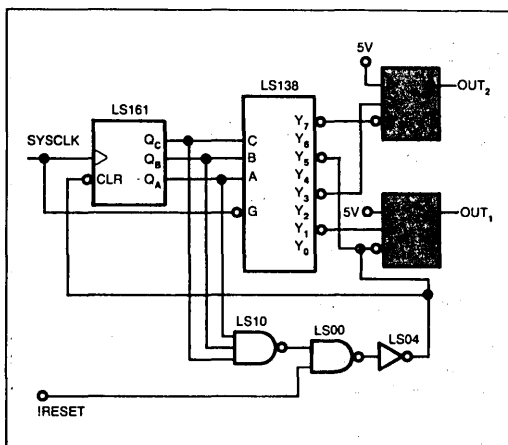


Fig 9—When converting complex sequential designs to PLDs, you can model your circuit as a group of flip-flops driven by a common clock.

Compiler-supported symbolic names can represent pin variable names, internal device nodes, intermediate variables, bit-field representations, and symbolic constants.

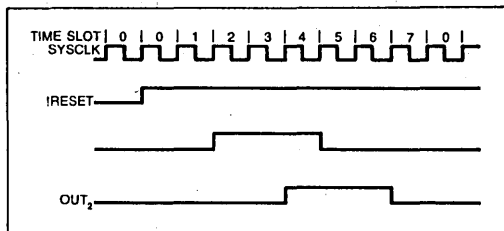


Fig 10—This timing diagram is based on the assumption that the Fig 9 circuit uses a clock rate that is not significantly affected by propagation delays from SYSCLK to OUT₁ and OUT₂.

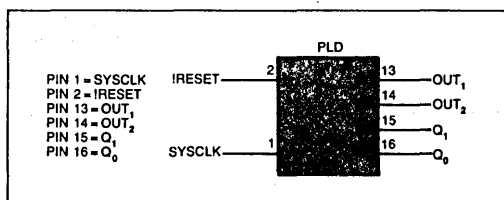


Fig 11—Adding outputs Q₀ and Q₁ of this PLD implementation of the Fig 9 circuit makes each of the eight intervals in the Fig 12 timing cycle a unique combination of the circuit's four outputs.

back into the programmable array:

```
TS0=!OUT1&!OUT2&Q0&!Q1 /*TIME SLOT 0*/
TS1=!OUT1&!OUT2&Q0&Q1 /*TIME SLOT 1*/
TS2=OUT1&!OUT2&!Q0&!Q1 /*TIME SLOT 2*/
TS3=OUT1&!OUT2&Q0&!Q1 /*TIME SLOT 3*/
TS4=OUT1&OUT2&!Q0&!Q1 /*TIME SLOT 4*/
TS5=!OUT1&OUT2&!Q0&Q1 /*TIME SLOT 5*/
TS6=!OUT1&OUT2&Q0&Q1 /*TIME SLOT 6*/
```

You can now write the equations for the four registered outputs in terms of TS₀ through TS₆ (TS₇ is not needed); CUPL performs the following substitutions:

```
OUT1.D=TS1#TS2#TS3
OUT2.D=TS3#TS4#TS5
Q0.D=TS0#TS2#TS5
Q1.D=TS6.
```

Running CUPL

Once you've completed the LDF, you're ready to compile the LDF for downloading to the PLD programmer. To compile the file, you type an expression that follows this format:

```
CUPL [FLAGS] TARGET_DEVICE_CODE
YOUR_FILE_NAME.
```

2.1.3 CUPL, AmCUPL (Cont'd.)

PLDs with an inverting-output architecture complicate selection of signal polarities.

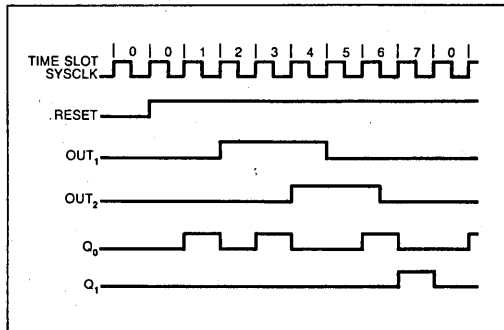


Fig 12—Once you've rewritten the Fig 10 timing diagrams to reflect the PLD configuration in Fig 11, you can write a set of logic equations for implementing the PLD design.

For example, the sequence CUPL -J -A P16L8 RAM-CNTRL compiles the source file for a RAM controller that is targeted for a PAL16L8. The J and A symbols are chosen from a table of CUPL option flags (Table 2). In this case, the compiler produces a JEDEC file and an absolute-format file to be used later by CUPL's simulator, CSIM (Ref 1). The resulting compiled code is downloaded to the programmer, which then blows the appropriate fuses in the PLD.

The designs discussed thus far are simple but useful for describing the PLD design process. The next two articles will extend the discussion to more advanced designs, and finally, to the state-machine approach.

EDN

Reference

Marrin, K, "Programmable logic devices gain software support," *EDN*, February 9, 1984, pg 67.

Author's biography

Bob Osanni is president and chief executive officer of Assisted Technology Inc (Sunnyvale, CA). He received his BSEE from Cornell University and was previously employed at Millennium Systems Inc (Santa Clara, CA). His hobbies include sports cars, airplanes, and music.

Designer's Guide to: Programmable logic—Part 2

Use PLDs to shrink complex, discrete logic designs

As discrete combinatorial and sequential logic circuits become more complex, it becomes more difficult to convert them to PLD equivalents. With the help of compiler-based software, though, you'll be converting complicated logic designs in no time.

Bob Osann, Assisted Technology Inc

Converting complicated discrete designs to their PLD (programmable logic device) equivalents can be an imposing task for the first-time PLD user or for the engineer who's been laboring with outmoded PLD software tools. New compiler-based software, however, makes it easy for you to implement even complex logic designs with PLDs.

This article, the second in a 3-part series on PLD design, introduces a few of the more advanced features of the compiler-based PLD design language CUPL and shows you how to use those features to convert complicated sequential and combinatorial SSI logic designs to PLD equivalent designs. Part 1 of the series demonstrated some elementary features of CUPL and showed you how to apply those features in a few simple designs. Part 3 will introduce CUPL's state-machine syntax and

show you how to move directly from logic ideas to PLD implementations without developing a gate-level description of your system.

CUPL lets you use a systems approach

The CUPL high-level PLD support language enables you to develop your logic designs using a systems approach. This approach not only speeds the design process but facilitates the generation of logic descriptions that are easy to understand.

CUPL supports a systems approach with several advanced features, which give you a self-documenting syntax, allow you to use fewer keystrokes to develop your systems, and let you use symbolic names that correspond to whatever function you're trying to implement. CUPL also gives you a flexible format, which lets you describe several similar systems in less time than it would take to describe the systems using a more rigid format.

One of CUPL's advanced features is its bit-field capability, which allows you to use a single symbolic name to represent a group of bits (such as an address bus or state bit field). This feature saves you keystrokes when you're formulating your design equations and makes the resulting equations easier to read. Once you've defined a symbolic name, you can use that name to represent either a single hexadecimal value or a range of hexadecimal values. For example, in an address-decoding application, you could equate the symbolic name MEMADR with [ADR7, ADR6, ADR5,

2.1.3 CUPL, AmCUPL (Cont'd.)

PLDs are effective replacements for both simple and complex combinatorial and sequential discrete logic designs.

ADR4, ADR3, ADR2, ADR1, ADR0]. You could then substitute [ADR7 . . . 0] for [ADR7, ADR6, ADR5, ADR4, ADR3, ADR2, ADR1, ADR0]. The resulting equation, `FIELD MEMADR=[ADR7 . . . ]`, assigns the name `MEMADR` to the address bus.

CUPL speeds bit-field comparisons

Another CUPL feature is its `..` operator, which can perform bit-field comparisons and operations quickly and efficiently. This feature is particularly useful for describing such features as an address decoder. When the compiler is performing a bit-field comparison, the operator `..` compares a bit field with either a hexadecimal or an octal constant value or a hexadecimal or octal list of constant values (hexadecimal is the default value). When you're describing an address decoder, for example, the statement `MEMADR: [A000 . . . EFFF]` is true if the address `MEMADR` falls in the hexadecimal range `A000` to `EFFF` (inclusively). Note that hexadecimal constant values must contain the proper number of nibbles to include the most significant bit of the bit field. In the above expression, the most significant bit of the `E` in `EFFF` corresponds to `A15` in `MEMADR`.

You can also use the `..` operator for bit-field operations, as in the following equation:

```
IOADR: & REPLACES A7&A6&A5&A4&A3&A2&A1&A0
IOADR: # REPLACES A7#A6#A5#A4#A3#A2#A1#A0.
```

Another timesaving CUPL feature is the preprocess-

or, which lets you write general-purpose logic descriptions that you can tailor to suit more than one application. For example, you might write a general-purpose decoder that you could adapt to 8-, 16-, or 32-bit applications by changing a few symbolic names and ranges.

The CUPL preprocessor is a program that operates on the CUPL source file before it's compiled. The preprocessor's string-substitution function, for example, can replace one symbolic name with another until some condition is met. When it encounters the statement `$DEFINE ARG1 ARG2`, for instance, the preprocessor replaces `ARG1` with `ARG2` until it encounters the statement `$UNDEF ARG1`. You could use the arguments in this example to represent different versions of your decoder. You could make `ARG1` represent, say, the 8-bit decoder, and you could make `ARG2` represent the 16-bit decoder.

The preprocessor also allows you to delay inclusion of a file until compile time. Again, this feature lets you generalize your functions. For example, you could write several files that represent several specific cases of a general application. To implement different functions, you'd just include different file names. In the statement `$INCLUDE FILENAME`, the referenced file becomes part of the LDF (logic description file) only at compile time.

Conditional control structures extend even further the ability to create generalized files. They allow you to

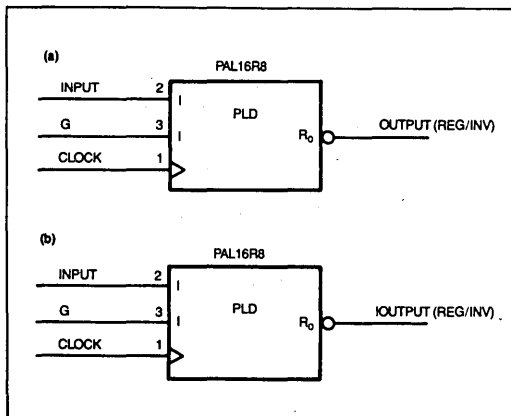


Fig 1—These two PLDs show two possible output configurations for a PLD with a fixed inverting output buffer. PLDs with programmable output polarity eliminate the confusion that fixed output devices cause.

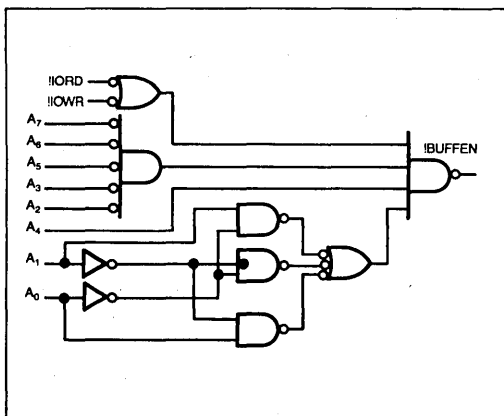


Fig 2—Address decoders are typical targets for first-time PLD users. A simple application like this address decoder shows how you can benefit from software features like macro substitution, range functions, and list notation.

2.1.3 CUPL, AmCUPL (Cont'd.)

TABLE 1 — MEMDEC LOGIC DESCRIPTION FILE

```

PARTNO      2600A00004 ;
NAME        MEMDEC ;
DATE        02/14/84 ;
REV         02 ;
DESIGNER    OSANN ;
COMPANY     ASSISTED TECHNOLOGY ;
ASSEMBLY    PC-RAM ;
LOCATION      U76 ;

/*-----*/
/* THIS DEVICE DECODES ALL MEMORY ACCESSES FOR BOTH PRIMARY AND
/* ALTERNATE LOCATIONS. IT GENERATES THE RAS SIGNALS FOR THE FOUR
/* BANKS OF 16K DYNAMIC RAMS AS WELL AS THE SIGNAL THAT INITIATES
/* THE CAS SIGNALS.
/*-----*/
/* ALLOWABLE TARGET DEVICE TYPES: PAL16L8, 82S153, PAL16P8 */

/* INPUTS */
PIN [1..6]      = [A19..14]      ; /* CPU ADDRESS BUS */
PIN [7,8]       = ! [MEMW,MEMR]   ; /* MEMORY DATA STROBES */
PIN 9           = !REF_ADR_LEN    ; /* INDICATES REFRESH CYCLE IN PROGRESS */
PIN 11          = !REF_RAS        ; /* STROBE FOR RAS-ONLY REFRESH */
PIN 13          = ALT_LOC         ; /* PLACE MEMORY IN ALTERNATE RANGE */

/* OUTPUTS */
PIN [19..16]    = ! [RAS3..0]     ; /* RAM ROW ADDRESS STROBES */
PIN 14          = !CAS_INIT       ; /* ENABLE CAS STROBES */

/* DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS */
FIELD MEMADR    = [A19..14]      ; /* MEMORY ADDRESS */
MEMREQ          = MEMW # MEMR    ; /* MEMORY REQUEST */

/* LOGIC EQUATIONS */
RAS3            = MEMREQ & !REF_ADR_LEN &
                  ( !ALT_LOC & MEMADR:[0C000..0FFFF]
                    # ALT_LOC & MEMADR:[FC000..FFFFF] )
                  ; /* PRIMARY RANGE */
                  ; /* ALTERNATE RANGE */
                  ; /* REFRESH CYCLE */

RAS2            = MEMREQ & !REF_ADR_LEN &
                  ( !ALT_LOC & MEMADR:[08000..0BFFF]
                    # ALT_LOC & MEMADR:[F8000..FBFFF] )
                  ; /* PRIMARY RANGE */
                  ; /* ALTERNATE RANGE */
                  ; /* REFRESH CYCLE */

RAS1            = MEMREQ & !REF_ADR_LEN &
                  ( !ALT_LOC & MEMADR:[04000..07FFF]
                    # ALT_LOC & MEMADR:[F4000..F7FFF] )
                  ; /* PRIMARY RANGE */
                  ; /* ALTERNATE RANGE */
                  ; /* REFRESH CYCLE */

RAS0            = MEMREQ & !REF_ADR_LEN &
                  ( !ALT_LOC & MEMADR:[00000..03FFF]
                    # ALT_LOC & MEMADR:[F0000..F3FFF] )
                  ; /* PRIMARY RANGE */
                  ; /* ALTERNATE RANGE */
                  ; /* REFRESH CYCLE */

CAS_INIT        = MEMREQ & !REF_ADR_LEN &
                  ( !ALT_LOC & MEMADR:[00000..0FFFF]
                    # ALT_LOC & MEMADR:[F0000..FFFFF] )
                  ; /* PRIMARY RANGE */
                  ; /* ALTERNATE RANGE */

```

2.1.3 CUPL, AmCUPL (Cont'd.)

By using symbolic names to represent bit fields such as address buses, you can not only save keystrokes, but you can make your designs virtually self documenting.

compile particular portions of your LDF when you've compiled with certain conditions. When you use the format

```
$IFDEF ARG
... STATEMENTS ...
$ELSE
... STATEMENTS ...
$ENDIF,
```

the statements are compiled only if the argument ARG has been defined. When you use the format

```
$IFNDEF ARG
... STATEMENTS ...
$ELSE
... STATEMENTS ...
$ENDIF,
```

the statements are compiled only if the argument ARG has not been defined.

Output programmability saves space

One CUPL feature that can save you considerable space in your design is the language's ability to support a PLD with programmable output polarity. For PLDs with this feature, the CUPL compiler chooses whichever

output polarity results in logic equations that use the smallest number of product terms. Although output-programmability support is a useful PLD option, many widely used PLDs contain inverting output buffers that are fixed instead of programmable. The examples that follow demonstrate the limitations of PLDs that don't have programmable output polarity.

For instance, Fig 1 illustrates the architecture for a PLD that uses a single D flip-flop and an inverter in its output stage. Fig 1a shows a design that uses an active-high output name, and Fig 1b shows one that uses an active-low output name. The pin declarations for Fig 1a are

```
PIN 1 = CLOCK
PIN 2 = INPUT
PIN 3 = G
PIN 18 = !OUTPUT.
```

To see why support for output programmability is so important, imagine that the flip-flop's output is fed back to keep it set. The polarity used in the output name makes a significant difference in the number of product (P) terms that are fed back.

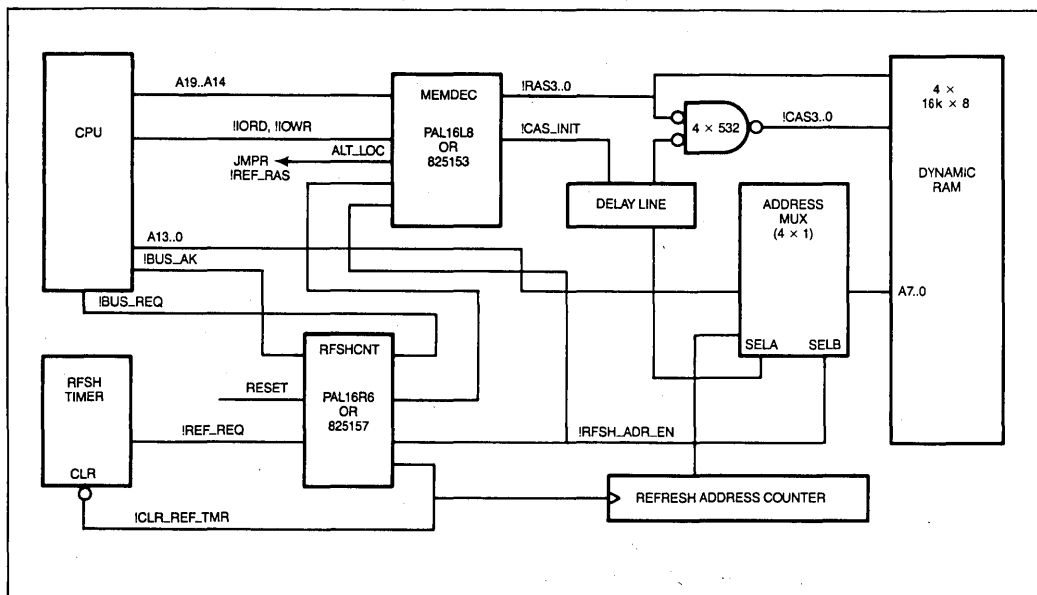


Fig 3—Memory decoders (MEMDEC) pose a challenging application for PLD conversions. This decoder is a portion of a dynamic RAM controller.

2.1.3 CUPL, AmCUPL (Cont'd.)

TABLE 2 — RFSHCNT LOGIC DESCRIPTION FILE

```

PARTNO      2600A00005 ;
NAME        RFSHCNT ;
DATE        02/19/84 ;
REV         02 ;
DESIGNER    OSANN ;
COMPANY     ASSISTED TECHNOLOGY ;

/*-----*/
/* THIS DEVICE RESPONDS TO THE REFRESH REQUEST(REF_REQ) GENERATED */
/* BY THE REFRESH INTERVAL TIMER. IT PRODUCES THE SIGNAL WHICH */
/* GATES THE REFRESH COUNTER ADDRESS INTO THE RAM ADDRESS BUS */
/* AS WELL AS THE REFRESH RAS STROBE AND THE CLEAR PULSE FOR */
/* THE REFRESH INTERVAL TIMER. */
/*-----*/

/* ALLOWABLE TARGET DEVICE TYPES: PAL16R6, 82S157 */

/* INPUTS */
PIN 1      = CLK                ;/* CPU CLOCK */
PIN 2      = REF_REQ            ;/* REFRESH REQUEST FROM INTERVAL TIMER */
PIN 3      = IBUS_AK           ;/* BUS ACKNOWLEDGE FROM CPU */
PIN 4      = RESET             ;/* SYSTEM RESET */
PIN 11     = IOE               ;/* TIED TO GROUND */

/* OUTPUTS */
PIN 18     = IBUS_REQ           ;/* BUS REQUEST TO CPU */
PIN 17     = IREF_ADLEN         ;/* ENABLE REFRESH ADDRESS */
PIN 16     = IREF_RAS           ;/* STROBE FOR RAS-ONLY REFRESH */
PIN 15     = IREF_RAS_DLY1      ;/* REF_RAS DELAYED 1 CLOCK */
PIN 14     = IREF_RAS_DLY2      ;/* REF_RAS DELAYED 2 CLOCKS */
PIN 13     = ICLR_REF_TMR       ;/* PULSE TO CLEAR RFRSH INTERVAL TIMER */

/* DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS */
FIELD ST   = [BUS_REQ,         /* ALL OUTPUTS ARE PART OF */
              REF_ADLEN,       /* THE STATE BIT FIELD. */
              REF_RAS,
              REF_RAS_DLY1,
              REF_RAS_DLY2,
              CLR_REF_TMR] ;

/* LOGIC EQUATIONS */
BUS_REQ.D = IRESET &
           IBUS_REQ & REF_REQ ;/* SET IT */
           # BUS_REQ & ( ST:20 # ST:30 ;/* KEEP IT SET */
                       # ST:38 # ST:3C ;/* KEEP IT SET */
                       # ST:3E ) );/* KEEP IT SET */

REF_ADLEN.D = IRESET &
              ( IREF_ADLEN & BUS_AK & BUS_REQ ;/* SET IT */
                # REF_ADLEN & ( ST:30 # ST:38 ;/* KEEP IT SET */
                              # ST:3C # ST:3E ) );/* KEEP IT SET */

REF_RAS.D = IRESET & ( ST:30 # ST:38 # ST:3C ) ;
REF_RAS_DLY1.D = IRESET & ( ST:38 # ST:3C # ST:3E ) ;
REF_RAS_DLY2.D = IRESET & ( ST:3C # ST:3E # ST:36 ) ;
CLR_REF_TMR.D = IRESET & ST:36 ;

```

2.1.3 CUPL, AmCUPL (Cont'd.)

If you choose an active-high output name, the logic equations are

```
OUTPUT.D = G & INPUT /* UPDATE WITH INPUT */
# 'G & OUTPUT /* MAINTAIN CURRENT OUTPUT */
/* VIA INTERNAL FEEDBACK PATH */
```

Because of the inverting output buffer, the equations that you must program into the array are

```
!OUTPUT.D = !G & INPUT # 'G & OUTPUT
!OUTPUT.D = !G & !OUTPUT # !INPUT & G #
!INPUT & !OUTPUT
```

Notice the extra product terms that are created. If, on the other hand, you choose an active-low output name, the pin declarations are

```
PIN 1 = CLOCK
PIN 2 = INPUT
PIN 3 = G
PIN 18 = !OUTPUT,
```

and the final equations are

```
OUTPUT.D = G & INPUT /* UPDATE WITH INPUT */
# 'G & OUTPUT /* MAINTAIN CURRENT OUTPUT */
/* VIA INTERNAL FEEDBACK PATH */
```

As you can see, when PLDs have fixed inverting buffers, the active-low output condition requires the fewest number of P terms.

Now that you're familiar with CUPL's features, you're ready to apply them to more complicated systems. When a logic designer uses a PLD for the first time in a new design, the designer's target area is often the address-decode function. Fig 2 shows a simple I/O-decoding circuit that creates a buffer-enable signal for I/O reads or writes when the decoded address falls in the hexadecimal range 10 through 12, inclusively. If you were to implement this address-decoding function using assembler-based software, your equations would look like the following ones:

```
BUFFEN=IORD*A7*A6*A5*A4*A3*A2*A1*A0
+IORD*A7*A6*A5*A4*A3*A2*A1*A0
+IORD*A7*A6*A5*A4*A3*A2*A1*A0
+IOWR*A7*A6*A5*A4*A3*A2*A1*A0
+IOWR*A7*A6*A5*A4*A3*A2*A1*A0
+IOWR*A7*A6*A5*A4*A3*A2*A1*A0
```

If you were to implement the address-decoding function using CUPL, your equations would look like this:

```
FIELDADR = [A7 . . 0];
IOREQ = IORD # IOWR;
BUFFEN = IOREQ & ADR:[10 . . 12];
```

To write equations using CUPL, you first define the address bus as a bit field where ADR=[A7 . . 0]. The

Conditional control structures improve compiler flexibility. They allow the compiler to delay decisions until certain predefined conditions are met.

compiler then substitutes [A7 . . 0] whenever it sees ADR. You then combine the strobe signals and give them the arbitrary name IOREQ where IOREQ = IORD#IOWR. Finally, you write an equation for the output BUFFEN in terms of the intermediate variables IOREQ and ADR so that BUFFEN=IOREQ&ADR:[10 . . 12]. The list-notation and range functions, as well as macro substitution, are all used here. The final code takes less time to write and is much easier to read than code written in an assembler-based language, and it's virtually self documenting.

Fig 3 shows the CUPL design technique in a more complicated decoder application, a dynamic RAM controller. The PLD MEMDEC in Fig 3 provides the memory decoder function. It supplies four 16k×8-bit banks of dynamic RAM with RAS (row address strobe) signals and generates a signal that initiates the CAS (column address strobe). The initiating signal first passes through a delay line and then recombines with the RAS signals to produce the CAS.

MEMDEC decodes address bits A19 through A14 of a 20-bit address space and maps the 64k-byte block to either the top or the bottom of the memory map shown in Fig 4. The jumper-selectable input called ALT_LOC

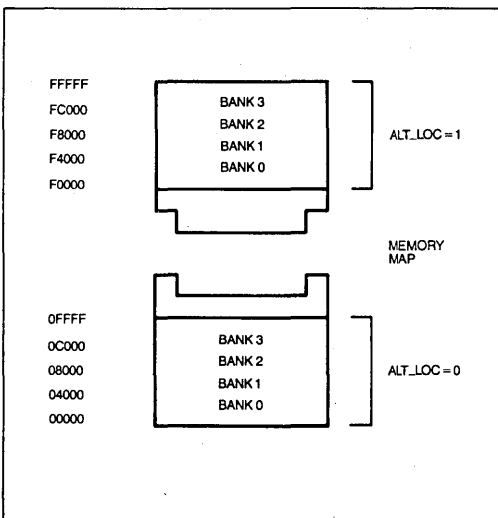


Fig 4—This memory map shows two possible locations for address bits A19 through A14 of a 20-bit address space. MEMDEC decodes the bits and maps the 64k-byte block to either the top or the bottom of the memory map.

2.1.3 CUPL, AmCUPL (Cont'd.)

Programmable-output capability allows the compiler to save PLD space. Thus, you'll need fewer PLDs when you're converting your design.

determines whether the top or the bottom of the memory map is used. Table 1 shows a completed LDF for the memory decoder.

Not only does CUPL simplify combinatorial designs, but it's useful for implementing sequential designs as well. Because PLDs contain both the logic array and registers in the same package, they're particularly powerful for implementing registered logic. The PLD named RFSHCNT in the RAM controller shown in Fig 3 handles the sequential aspects of refresh control for the dynamic RAM in a typical μ P system.

RFSHCNT responds to a refresh signal from the refresh interval timer (usually 14 μ sec) by driving the CPU's bus-request line high. After receiving a bus-acknowledge signal from the CPU, RFSHCNT then generates signals for address MUX control and RAS-only refresh timing.

RFSHCNT also provides a signal that resets the refresh interval timer and clocks the refresh-address counter. Fig 5 shows the timing diagram for RFSHCNT. Note that the registered output signals are shown as logical true even though the actual outputs are active low. Because the equations are based on signals in the timing diagram, in order for the registered outputs to be shown as logical true, the target device must have either an inverting output

buffer or programmable-output-polarity capability.

Table 2 shows the LDF for RFSHCNT. The LDF uses the hexadecimal values that define the time slots shown in the timing diagram. Note the use of CUPL's bit-field capability in the equations that specify the D flip-flop's state.

CUPL's compiler-based techniques simplify the conversion of complicated SSI circuits to their PLD equivalents. Part 3 of this series will show you how to simplify the logic design process even further by using the state-machine approach. **EDN**

Author's biography

Bob Osann is president and chief executive officer at Assisted Technology Inc (San Jose, CA). He received his BSEE from Cornell University and was previously employed at Millennium Systems Inc (Santa Clara, CA). Bob holds three patents in the areas of consumer and industrial electronics. His interests include sports cars, airplanes, and music.

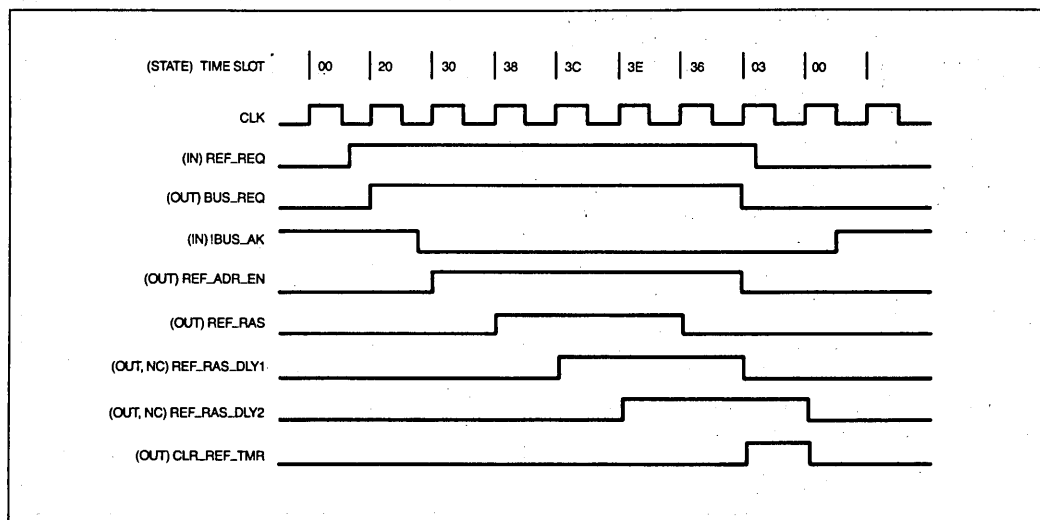


Fig 5—The registered output signals shown in this timing diagram for RFSHCNT are shown as logical true even though the actual outputs are active low. Because the equations are based on the timing diagram, the target device must be inverting, or it must have programmable-output capability.

Designer's Guide to: Programmable logic—Part 3

State-machine approach speeds logic design

2

To exercise a PLD's full potential for shortening design time and improving documentation, use the state-machine approach. This approach lets you formulate a behavioral description of your system and implement it directly in a PLD, without ever developing an equation-level representation.

Bob Osann, Assisted Technology

Using the state-machine approach and a compiler-based PLD design language like CUPL, you can bypass the gate- and equation-level stage in logic design and move directly from a system-level description to a PLD implementation. Unlike assembler-based approaches, the state-machine approach lets you document your design in a manner that's understandable to future users of your design.

Actually, few logic designers currently use the state-machine approach in their logic designs. This isn't surprising: The technique seems difficult to learn at first. But CUPL makes the state-machine approach less formidable by handling many of the decisions you would

normally have to make. Furthermore, CUPL gives you a general and simple state-machine model like the one shown in Fig 1. The software automatically fits the model to your application.

Defining the state model

In general, a state machine is a logic circuit with flip-flops. Because a flip-flop's output can be fed back to its own or some other flip-flop's input, a flip-flop's input value may depend on both its own output and that of other flip-flops. Consequently, the final value for a flip-flop's output depends on its own previous values, as well as those of other flip-flops.

The CUPL state-machine model uses six compo-

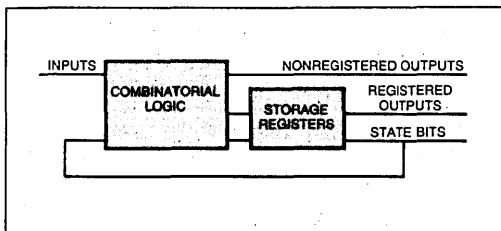


Fig 1—State-machine theory can be complicated, but CUPL allows you to abstract from the theory's complicated details. Using this simple model and an easy-to-learn syntax, you can quickly construct state-machine models of your system.

2.1.3 CUPL, AmCUPL (Cont'd.)

When you use the state-machine approach, you don't have to write a logic-equation-level description of your system before implementing it in a PLD.

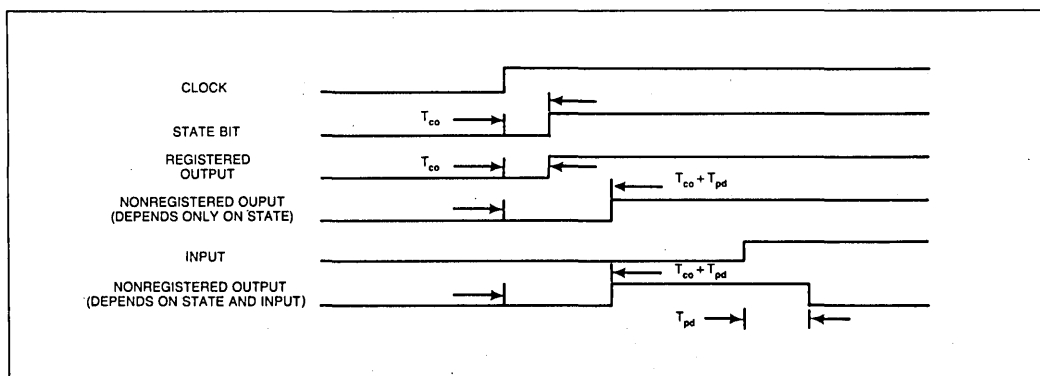


Fig 2—This timing diagram characterizes CUPL's simple state-machine model. The setting or resetting of the registered output depends on the status of the state bit. Conversely, nonregistered outputs can depend either on only the current state bit's status or on both the state bit's status and the input's status.

nents: inputs, combinatorial logic, storage registers, state bits, registered outputs, and nonregistered outputs. Fig 2 shows the timing relationships between these components.

Inputs are signals entering the device that originate in some other device. Combinatorial logic is any combination of logical gates (usually AND-OR) that produces an output signal that's valid T_{pd} (propagation delay time) nsec after any of the signals that drive these gates changes. T_{pd} is the time delay between the initiation of an input or feedback event and the occurrence of a nonregistered output.

State bits are storage-register outputs that are fed back to drive the combinatorial logic. They contain the present-state information. Storage registers are any flip-flop elements that receive their inputs from the state machine's combinatorial logic. Some registers are used for state bits, while others are used for registered outputs. The registered output is valid T_{co} nsec after the clock pulse occurs. T_{co} is the time delay between the initiation of a clock signal and the occurrence of a valid flip-flop output.

For the system to operate properly, you must meet your PLD's requirements for setup and hold times. For most PLDs, the setup time (T_{su}) usually includes both the propagation delay of the combinatorial logic and the actual setup time of the flip-flops. T_{su} is the time it takes for the result of either a feedback or an input event to appear at the input to a flip-flop. A subsequent clock input cannot be applied until this result becomes valid at the flip-flop's input. These flip-flops may be either D,

RS, or JK types (but RS and JK types are used more often in state-machine implementations because they require fewer product (P) terms than D types do).

Nonregistered outputs are outputs that come directly from the combinatorial logic gates. They may be functions of the state bits and the input signals (and have asynchronous timing), or they may be purely dependent on the current state-bit values, in which case they become valid $T_{co} + T_{pd}$ nsec after an active clock edge occurs.

Registered outputs are outputs that come from the storage registers but are not included in the actual state-bit field (ie, a bit field composed of all the state bits). State-machine theory requires that the setting or resetting of these registered outputs depend on the transition from a present state to a next state. This allows a registered output to be either set or reset in a given state, depending on how the machine came to be in that state. Thus, a registered output can assume a "don't care" operation mode. In the "don't care" mode, the registered output will remain at its last value as long as the current state transition does not specify that registered output.

The state-machine syntax

To help you implement this state-machine model quickly, CUPL supplies a general and simple state-machine syntax. This syntax gives you a single, simple format that allows you to describe any function in the state machine. The general format for the state-machine syntax is

2.1.3 CUPL, AmCUPL (Cont'd.)

```
SEQUENCE state_bit_field {
  PRESENT present_state
    IF input_cond NEXT next_state OUT outputs ;
    IF input_cond NEXT next_state OUT outputs ;
    IF ...
  PRESENT present_state
    IF input_cond NEXT next_state OUT outputs ;
    IF input_cond NEXT next_state OUT outputs ;
    IF ...
  PRESENT ...
}
```

Each present-state block within this format describes both asynchronous (present state) and synchronous (transition) activity. Using this format, you can describe any component of the state machine. For example, the formats for registered outputs would be

<u>IF input_cond</u>	<u>NEXT next_state</u>	<u>OUT outputs</u>
CONDITIONAL TRANSITION		OUTPUT ASSOCIATED WITH TRANSITION

or

	<u>NEXT next_state</u>	<u>OUT outputs</u>
UNCONDITIONAL TRANSITION		OUTPUT ASSOCIATED WITH TRANSITION,

depending on whether the transition is conditional or not. To use these equations for describing your system, you need to learn how to use the CUPL keywords. For example, when you use a Next statement, you're telling the compiler that all of the outputs in that block are registered outputs whose values depend on transition

information (ie, information about the transition from the present state to the next state). Using the If statement signifies a conditional event. When you use the If keyword in a nonregistered description, you signify that the input and output events will have an asynchronous dependence. The absence of a Next keyword signifies a nonregistered event.

For nonregistered outputs, you would use the format

<u>IF input_cond</u>		<u>OUT outputs</u>
INPUT CONDITION AFFECTS OUTPUT AFTER T_{pd}	NO STATE TRANSITION	OUTPUTS ASSOCIATED WITH INPUT CONDITION AND PRESENT STATE.

or

		<u>OUT outputs</u>
NO INPUT CONDITION	NO STATE TRANSITION	OUTPUT ASSOCIATED SOLELY WITH PRESENT STATE. VALID $T_{co} + T_{pd}$ AFTER CLOCK.

Much of the reason for choosing either the registered or nonregistered format for an output depends on the system timing. For fully synchronous systems that require tight timing, the registered output provides fast response—it responds within T_{co} nsec after the occurrence of a clock pulse. This quick response gives the circuit time to use that registered output as an input somewhere else in the circuit before the next clock pulse occurs.

Conversely, you would use the nonregistered output in asynchronous applications. You would also use the

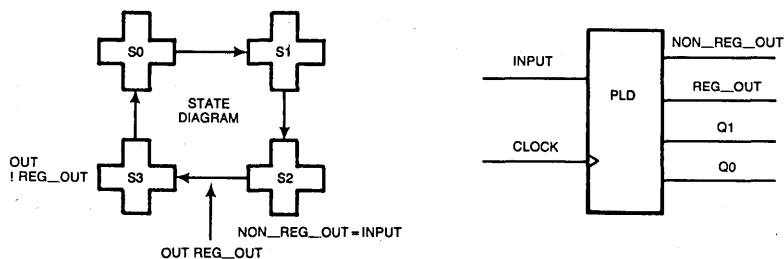


Fig 3—This model for a free-running 2-bit counter demonstrates CUPL's state-machine syntax. The counter has one input, one nonregistered output, and one registered output.

2.1.3 CUPL, AmCUPL (Cont'd.)

nonregistered output in simpler applications, such as present-state decoders.

To better understand the state-machine model and its syntax, consider a simple example: a free-running 2-bit counter with one input, one registered output, and one nonregistered output. Fig 3 shows the state-transition

diagram. The circles represent states (specific combinations of the state bits), and the arrows represent the transitions between states. Because the transitions in this example are unconditional, the counter is free-running. Accordingly, the logic description uses no If keywords in statements that signify a Next state. The

The function-table approach

To design logic systems with PLDs, you could use the function-table approach, which complements the state-machine approach. The function-table approach is useful in applications such as code converters, where input/output relationships are best represented in tabular form.

CUPL's parallel-operation capability makes it easy for you to develop these tabular representations. Using that feature, you can declare bit fields and use them on either the right or left side of the equation.

The parallel operation feature allows you to operate uniformly

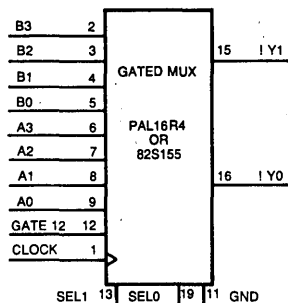


Fig A—In code-conversion applications, like this dual 4-to-1 multiplexer, you can best describe the system using a tabular format.

TABLE A—GATED MUX LOGIC DESCRIPTION FILE

```

PARTNO      PL10007;
NAME        GATED MUX;
DATE        09/17/84;
REV         01;
DESIGNER    ARONSON;
COMPANY     ASSISTED TECHNOLOGY;
ASSEMBLY    PC_IO;
LOCATION     U23;

/* THIS DEVICE FUNCTIONS AS A DUAL 4-TO-1 MUX WITH INVERTING */
/* REGISTERED OUTPUTS. THE MUX OUTPUTS ARE ONLY CLOCKED INTO THE */
/* REGISTERS WHEN THE GATE INPUT IS ACTIVE. */
/* ALLOWABLE TARGET DEVICE TYPES : PAL16R4 , 82S155 */

/* ** INPUTS ** */
PIN 1      = CLOCK      /* SYSTEM CLOCK */
PIN [2..5] = [B3..0]    /* INPUT GROUP B */
PIN [6..9] = [A3..0]    /* INPUT GROUP A */
PIN 13     = SEL1       /* SELECT 1 */
PIN 19     = SELO       /* SELECT 0 */
PIN 12     = GATE       /* GATES MUX OUTPUT INTO REGISTER */
PIN 11     = IOE        /* OUTPUT ENABLE */

/* ** OUTPUTS ** */
PIN 15     = Y1         /* REGISTER OUTPUT FROM GROUP B */
PIN 16     = Y0         /* REGISTER OUTPUT FROM GROUP A */

/* ** DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS ** */
FIELD OUT = [Y1..0] ; /* OUTPUT BITS */
FIELD SEL = [SEL1..0] ; /* SELECT CONTROL BITS */

/* ** LOGIC EQUATIONS ** */
OUT.D = !GATE & OUT
# GATE & ( [B3,A3] & SEL:3
# [B2,A2] & SEL:2
# [B1,A1] & SEL:1
# [B0,A0] & SEL:0 );

/* NOTE:
/* ONE EQUATION DESCRIBES
/* BOTH OUTPUT VARIABLES.

```

2.1.3 CUPL, AmCUPL (Cont'd.)

To make the state-machine approach easier to learn, CUPL uses a model that incorporates both the Mealy and Moore models. You use the same model for all cases.

nonregistered output is active on a count of two (S2) when the input is active. The registered output is set on the transition from S2 to S3 and reset on the transition from S3 to S0. **Table 1** gives the logic description for the counter.

An application that incorporates hysteresis shows the

importance of using transition information in addition to present-state information. Consider, for instance, a circuit that performs threshold detection on an analog signal, but requires a hysteresis band both wider and more accurate than the hysteresis band an analog comparator could achieve. In such an application, you

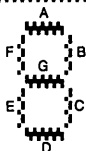
TABLE B—HEXDISP LOGIC DESCRIPTION FILE

PARTNO CT0002;
NAME HEXDISP;
DATE 6/5/84;
REV 01;
DESIGNER T KAHL;
COMPANY ASSISTED TECHNOLOGY INC;
ASSEMBLY DISPLAY_BOARD;
LOCATION U17;

```

/* THIS IS A HEXADECIMAL-TO-SEVEN-SEGMENT
/* DECODER CAPABLE OF DRIVING COMMON-ANODE
/* LEDS. IT INCORPORATES BOTH A RIPPLE-
/* BLANKING INPUT (TO INHIBIT DISPLAYING
/* LEADING ZEROES) AND A RIPPLE-BLANKING
/* OUTPUT TO ALLOW FOR EASY CASCADING OF
/* DIGITS.

```



* ALLOWABLE TARGET DEVICE TYPES: 32 x 8 PROM (82S123 OR EQUIV)

```
/** INPUTS */
```

```
PIN [10..13] = [D0..3] ;      /* DATA INPUT LINES TO DISPLAY */
PIN 14 = !RBI;                /* RIPPLE BLANKING INPUT */
```

```
/** OUTPUTS */
```

```
PIN [7..1] = !{A,B,C,D,E,F,G}; /* SEGMENT OUTPUT LINES */
PIN 9 = !RBO; /* RIPPLE BLANKING OUTPUT */
```

```

/* DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS */

```

```
FIELD DATA = [D3..0];          /* HEXADECIMAL INPUT FIELD          */
FIELD SEGMENT=[A,B,C,D,E,F,G]; /* DISPLAY SEGMENT FIELD          */
```

```
$DEFINE ON 'b'1      /* SEGMENT LIT WHEN LOGICALLY "ON" */
$DEFINE OFF 'b'0     /* SEGMENT DARK WHEN LOGICALLY
                     "OFF" */
```

```
/** LOGIC EQUATIONS **/
```

SEGMENT -	I*	A	B	C	D	E	F	G	I*
'0	/	ON,	ON,	ON,	ON,	ON,	ON,	OFF	& DATA:0 & IRB:
'1	/	OFF,	ON,	ON,	OFF,	OFF,	OFF,	OFF	& DATA:1
'2	/	ON,	ON,	OFF,	ON,	ON,	OFF,	ON	& DATA:2
'3	/	ON,	ON,	ON,	ON,	OFF,	OFF,	ON	& DATA:3
'4	/	OFF,	ON,	ON,	OFF,	OFF,	ON,	ON	& DATA:4
'5	/	ON,	OFF,	ON,	ON,	OFF,	ON,	ON	& DATA:5
'6	/	ON,	OFF,	ON,	ON,	ON,	ON,	OFF	& DATA:6
'7	/	ON,	ON,	ON,	OFF,	OFF,	OFF,	OFF	& DATA:7
'8	/	ON,	ON,	ON,	ON,	ON,	ON,	ON	& DATA:8
'9	/	ON,	ON,	ON,	OFF,	OFF,	ON,	ON	& DATA:9
'A	/	ON,	ON,	ON,	OFF,	ON,	ON,	ON	& DATA:A
'B	/	OFF,	OFF,	ON,	ON,	ON,	ON,	ON	& DATA:B
'C	/	ON,	OFF,	OFF,	ON,	ON,	ON,	OFF	& DATA:C
'D	/	OFF,	ON,	ON,	ON,	ON,	OFF,	ON	& DATA:D
'E	/	ON,	OFF,	OFF,	ON,	ON,	ON,	ON	& DATA:E
'F	/	ON,	OFF,	OFF,	OFF,	ON,	ON,	ON	& DATA:F

RBO = RBI & DATA:0:

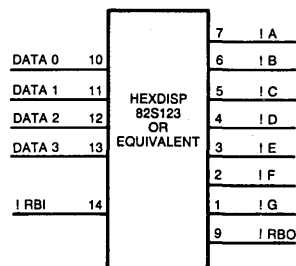


Fig B—Though a PROM was used to implement this simple hexadecimal-to-7 segment decoder, you could use the same function table to implement this state machine in a PLD.

on an entire parallel data path. It's easy to write a description in this manner, as you can see from **Fig A's** 4-to-1 multiplexer, which has an inverting registered output. **Table A** contains the multiplexer's LDF.

The hexadecimal-to-7 segment decoder in **Fig B** also lends itself well to tabular representation. Again, bit-field notation is convenient for describing the logical function. Incidentally, because it's a simple circuit, this implementation uses a PROM as the target device (because 16 pins are sufficient and bipolar PROMs are inexpensive), but you could also implement these function tables in PLDs. **Table B** contains the decoder's LDF.

2.1.3 CUPL, AmCUPL (Cont'd.)

TABLE 1—LOGIC DESCRIPTION FOR 2-BIT COUNTER

```

FIELD COUNT=[Q1, Q0];      /* LET'S CALL THE STATE BIT FIELD "COUNT" */

$DEFINE S0 0 /*.....*/
$DEFINE S1 1 /* DEFINE SYMBOLIC NAMES FOR THE ACTUAL STATE BIT CONSTANT */
$DEFINE S2 2 /* VALUES USING PREPROCESSOR COMMANDS. CONSTANTS DEFAULT */
$DEFINE S3 3 /* TO HEX AND REPRESENT VALUES OF "COUNT" WITHIN THE */
              /* "SEQUENCE" BLOCK BELOW. */
              /*.....*/

SEQUENCE COUNT {           /* NOTE USE OF BRACES FOR ENCLOSING STATE */
                           /* SEQUENCE DESCRIPTION BLOCK. */
    PRESENT S0
    NEXT S1;
    PRESENT S1
    NEXT S2;
    PRESENT S2
    IF INPUT OUT NON_REG_OUT; /* ASYNCHRONOUS WITHIN S2 */
    NEXT S3 OUT REG_OUT;      /* SETS ON TRANSITION */
    PRESENT S3
    NEXT S0 OUT !REG_OUT ; } /* RESETS ON TRANSITION */

```

need transition information in order to achieve hysteresis. One way to solve this problem would be to construct a tracking A/D converter in which the threshold detector output (digital Schmitt-trigger output) is a registered output of the state machine (Fig 4).

The three counter bits that feed the D/A converter compose the state bits. To create the hysteresis, you set the trigger output only on the transition from S5 to S6 and reset the trigger only on the transition from S2 to S1. At all other times, you place the trigger output in a "don't care" state. The trigger output may have different values in states S2 through S5 depending on how the machine arrived at those states.

Fig 5 shows a state diagram for the system. All states in which you can set the trigger output are shown on top and all states in which you can reset the trigger are shown on bottom.

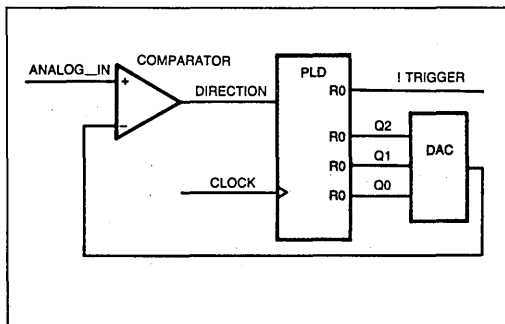


Fig 4—To realize the hysteresis function in this state-machine model for an analog comparator with digital hysteresis, you must set or reset the registered output by using transition information rather than present-state information.

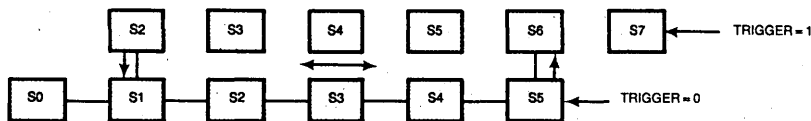


Fig 5—The state diagram for the analog comparator with hysteresis shows that a state's value can be history-dependent. The trigger output can have different values in states S2 through S5 depending on how the machine arrives at those states.

2.1.3 CUPL, AmCUPL (Cont'd.)

The CUPL state-machine syntax allows you to specify any state-machine component with a single format, thus simplifying the state-machine description.

TABLE 2—SCHMITT LOGIC DESCRIPTION FILE

```
PARTNO      CT0001;
NAME        SCHMITT;
DATE        6/30/84;
REVISION    01;
DESIGNER    T KAHL;
COMPANY     ASSISTED TECHNOLOGY INC;
ASSEMBLY    ANALOG_INTERFACE;
LOCATION     U27;

/*.....*/
/* THIS DEVICE RECEIVES A 'COUNT DIRECTION' COMMAND FROM AN ANALOG */
/* COMPARATOR AND RESPONDS BY INCREMENTING OR DECREMENTING AN */
/* INTEGRAL UP/DOWN COUNTER. A REGISTERED OUTPUT IS CREATED AND ACTS */
/* AS A DIGITAL SCHMITT TRIGGER WITH HYSTERESIS. */
/*.....*/

/** INPUTS **/
PIN 1      = CLOCK;      /* CLOCK PIN FOR THE COUNTER */
PIN 2      = DIRECTION;  /* DIRECTION OF COUNT MODE PIN */

/** OUTPUTS **/
PIN [14..16] = !Q0..2;    /* COUNTER STATE BITS */
PIN 17      = !TRIGGER;   /* SCHMITT TRIGGER OUTPUT BIT */

/** DECLARATIONS AND INTERMEDIATE VARIABLE DEFINITIONS **/
UP = DIRECTION;          /* COUNTER MODES */
DOWN = !DIRECTION;;

FIELD COUNT = [Q2..0];   /* FIELD FOR COUNTER STATES */

$DEFINE S0 0
$DEFINE S1 1              /* COUNTER STATES DEFINED AS */
$DEFINE S2 2              /* STATES 0 THRU 7 */
$DEFINE S3 3
$DEFINE S4 4
$DEFINE S5 5
$DEFINE S6 6
$DEFINE S7 7

SEQUENCE COUNT {
PRESENT S0
    IF UP      NEXT S1;
    IF DOWN    NEXT S0;
PRESENT S1
    IF UP      NEXT S2;
    IF DOWN    NEXT S0;
PRESENT S2
    IF UP      NEXT S3;
    IF DOWN    NEXT S1      OUT !TRIGGER;
PRESENT S3
    IF UP      NEXT S4;
    IF DOWN    NEXT S2;

PRESENT S4
    IF UP      NEXT S5;
    IF DOWN    NEXT S3;
PRESENT S5
    IF UP      NEXT S6;
    IF DOWN    NEXT S4      OUT TRIGGER;
PRESENT S6
    IF UP      NEXT S7;
    IF DOWN    NEXT S5;
PRESENT S7
    IF UP      NEXT S7;
    IF DOWN    NEXT S6;
```

2.1.3 CUPL, AmCUPL (Cont'd.)

are shown on the bottom. Note that states S2, S3, S4, and S5 appear twice because they can have two different values. Each state's value depends on the system's previous state.

Note also that the state bits in this application supply information to the outside world; in this case, the information consists of inputs to a D/A converter. When you give the PLD access to the outside world, you deviate from the standard Mealy and Moore state-machine models, but you can squeeze more logic into your PLD.

Table 2 gives the state machine's logic description file (LDF). In the LDF, you declare the state bits as a bit field and give them the symbolic name "Count." Next, you use the input Direction to define names for the Up and Down counter modes. You then complete the numerical state assignment for states S0 through S7 by using the \$Define command from CUPL's pre-processor.

In defining the state machine, you use If and Next keywords for every present-state block. When you use Next, you indicate that the state machine's activity is synchronous; when you use If, you indicate that the transitions are conditional. The transitions' direction depends on the direction the counter counts in, which is in turn determined by the value of the Direction input.

Though applications like counters and comparators with hysteresis may not seem very complicated, they serve to show that designing with PLDs is a straightforward task, whether you're using the devices to replace existing designs or using them in a state-machine design.

EDN

Author's biography

Bob Osann is president and chief executive officer at Assisted Technology Inc (San Jose, CA). He received his BSEE from Cornell University and was previously employed at Millennium Systems Inc (Santa Clara, CA). Bob holds three patents in the areas of consumer and industrial electronics. His hobbies include sports cars, airplanes, and music.



EDN February 7, 1985

2.1.3 CUPL, AmCUPL (Cont'd.)

AmCUPL — The High-Level Software Tool for PAL Designs

To provide a low-cost, high-level design-aid software tool for AMD's programmable logic devices, AMD and Personal CAD Systems have developed AmCUPL. With AmCUPL, you can create custom-design solutions using AMD's PAL devices in a matter of minutes. This shorter design cycle and other benefits such as lower cost, higher performance, and higher reliability result in a significant competitive advantage in your marketplace.

High-Level Design Support

AmCUPL has many features which make it extremely powerful and easy to use:

- Choice of logic description formats:
 - State diagrams
 - High-level Boolean equations
 - Truth tables
- Portfolio of four different logic minimization algorithms
- Automatic enhanced DeMorgan expansion capability
- PALASM-to-AmCUPL translator
- User-defined logic functions
- User-friendly syntax
- Built-in logic simulator

Full Support for AMD PAL Devices at a Very Low Cost

AmCUPL supports all the advanced PAL devices from AMD, including the AmPAL22V10 and AmPAL18P8. All the advanced features of these devices, including programmable output logic macrocells, programmable output polarity, and distributed product terms, can be effectively used with AmCUPL. Future releases of AmCUPL will support upcoming PAL devices from AMD.

Yet, AmCUPL provides all this support at a very low cost. This includes full user support from Personal CAD Systems, Inc., of San Jose, California.

Easy Upgrade Path

When you purchase AmCUPL you have an option to upgrade to CUPL from Personal CAD Systems within one year. The AmCUPL cost can be credited against this upgrade. CUPL offers the same functionality as AmCUPL but also supports PAL devices from other vendors. The PC/MS-DOS version of AmCUPL is distributed by AMD on IBM formatted 5-1/4" floppy disks; a CUPL manual is included.

AmCUPL — The cost-effective, easy-to-use, and complete design tool for your custom design inventions. For additional information on AmCUPL or other AMD PAL devices, contact one of AMD's sales offices, authorized reps, or distributors.

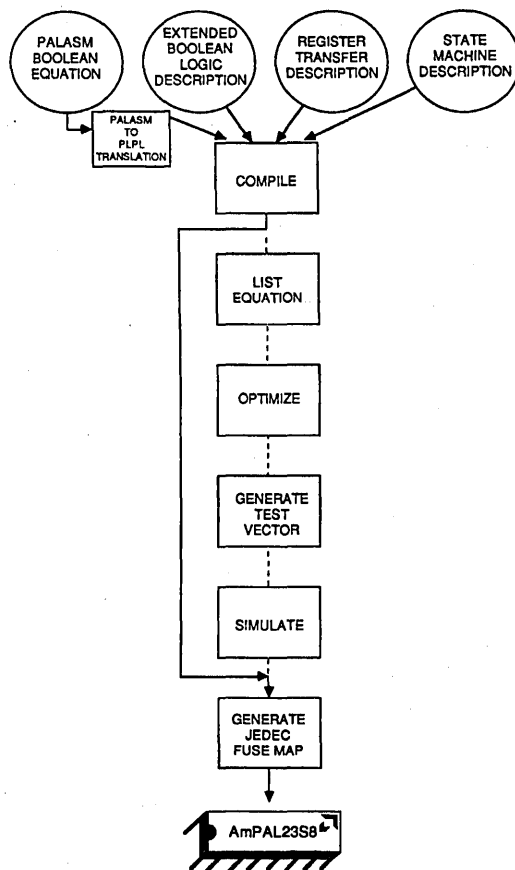
2.1.4 PLPL

Programmable Logic Programming Language (PLPL) is a new design tool that makes it easy to design, verify, and test logic functions that are to be implemented with programmable logic devices. It is an integrated, top-down, hierarchical, and complete design language which provides clear problem definition in a variety of ways. This results in solutions that are self documenting. PLPL allows multiple-input formats which means the function of a device can be expressed in terms of a state transition or truth table, high-level constructs, or Boolean equations.

In a summary, PLPL is the heart of a new programmable logic computer-aided design (CAD) environment that allows you to take full advantage of the benefits associated with using programmable logic devices. It aids in:

- Defining the problems to be solved
- Creating a solution
- Verifying the solution by simulation
- Generating test vectors
- Optimizing/minimizing the intermediate equations
- Providing an interactive- or a batch-mode of operation

Functionally, PLPL accepts a logic-description input file and creates a JEDEC-standard fuse map downloadable to a logic programmer. The flow of this procedure is shown in Figure 2-6.



BD006711

Figure 2-6. PLPL Design Flow

PLPL Features

Major features of PLPL are:

- Employs high-level block-structured hardware-description language optimized for programmable logic devices
- Uses C language for flexibility and portability
- Supports multiple design techniques:
 - PALASM-like sum-of-products Boolean equations
 - Extended Boolean-logic description: Parenthesized equations, DeMorgan's laws, Macro-substitution capability, and pin vectors
 - High-level constructs such as: IF THEN ELSE, FOR, CASE, and FUNCTION CALLS
- Provides a convenient and powerful human interface:
 - Both interactive or batch mode
 - Optional menu-driven capability
 - Extensive error checking
 - Interactive help facility
 - Direct programmer interface

- Supports current AMD PAL devices
- Permits adding new devices quickly because it is data-base driven

Structure of the PLPL Environment

In the PLPL CAD environment, each program is governed by a separate program module, with all the modules tied together in a top-down fashion. The software modules (Figure 2-7) are as follows:

- Operations Processor
- PLPL Compiler
- PLPL Data Base
- PLPL List Equations
- PLPL Optimizer
- PLPL Test-Vector Generator
- PLPL Simulator
- JEDEC Fuse-Map Generator
- PALASM-to-PLPL Translator

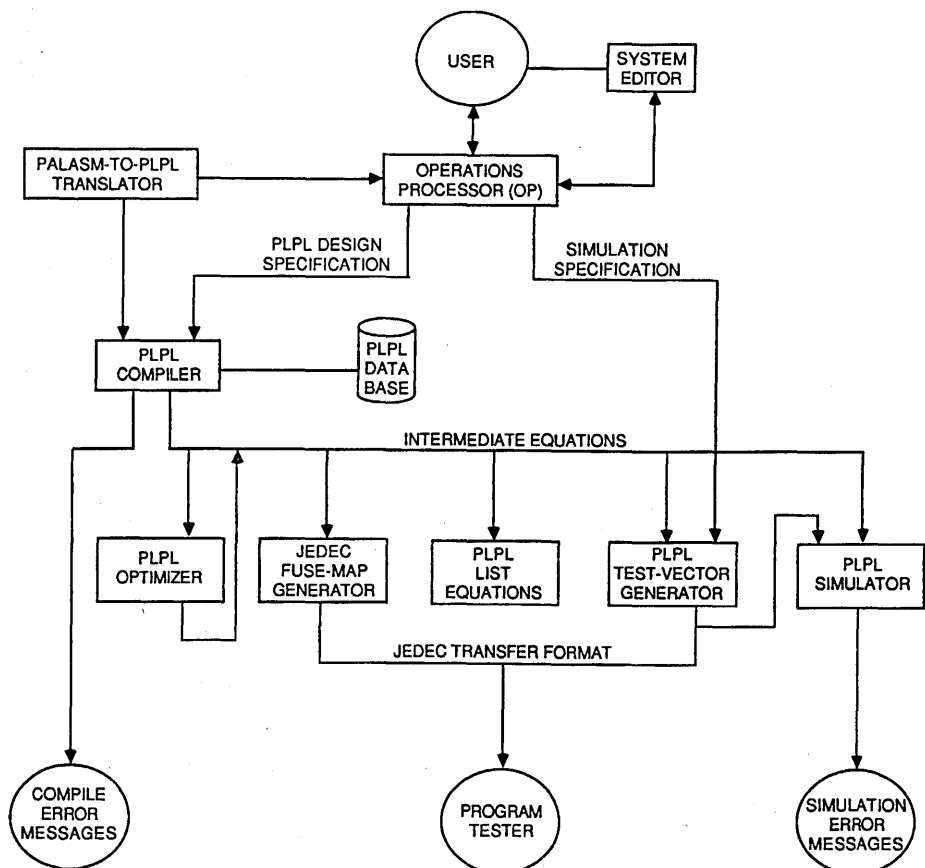


Figure 2-7. Structure of the PLPL Environment

PLPL Operations Processor (OP)

The PLPL OP is the interface module that defines the program structure. It is an interactive, menu-driven shell that provides the user interface to the compiler, the JEDEC fuse-map generator, the simulator, the test-vector generator, the optimizer, and other device-specific utilities. With the operations processor, a comprehensive help facility can be consulted or, if required, a "shell escape" may be executed. Shell-escape capability allows a temporary exit from PLPL for the execution of certain system operations, such as editing or looking at the contents of various other files. It should be noted, however, that the internal state inside the shell is maintained.

PLPL Compiler

The PLPL compiler converts PLPL design specifications into the form needed by the JEDEC fuse-map generator, and other modules (such as the simulator, test-vector generator, and the optimizer). In addition, it flags all syntax and device-limitation errors. For example, a device-limitation error might indicate that a PAL device has an insufficient number of product terms to accommodate a particular logic equation. PLPL compiler is designed to be device-independent, and obtains all of its device data from the data base.

PLPL Data Base

The PLPL data base serves three purposes:

- 1) Stores all the architectural (device-dependent) information associated with supported devices and supplies this data to the compiler and JEDEC fuse-map generator. This arrangement makes the latter modules transparent to variations in the architecture of the devices.
- 2) Provides the flexibility to add new devices — making the PLPL environment more powerful. All it takes to add a new device is adding a description file to the PLPL data base.
- 3) Serves as a standard cell library, thereby automating programmable logic design. The PLPL data base supports all current AMD PAL devices.

PLPL List Equations

The PLPL list equations lists all the PLPL-generated Boolean equations.

PLPL Optimizer

The PLPL-optimizer package optimizes the list of Boolean equations generated by the PLPL compiler. By reducing all

redundant product terms, this package results in an optimum list of equations.

PLPL Test-Vector Generator

The PLPL test-vector generator generates test vectors for a PLPL-design specification. The test vectors can be used by the PLPL simulator to test the device exhaustively.

PLPL Simulator

The PLPL simulator enhances the confidence that the device will function as designed. In PLPL, simulation is performed with user-created test-table files consisting of input and expected output (optional) test vectors. The PLPL simulator uses the test vectors and the outputs of the PLPL compiler (intermediate equation) to model the device's output behavior. The simulator's output is compared to the expected test-vector outputs, if specified, and error messages are issued when results do not match. A test table may be concatenated, or linked, with the design-input specification, or it may be kept in a separate file. Separating the test table from the device specification provides flexibility, allowing more than one test table to be attached to a design.

JEDEC Fuse-Map Generator

The JEDEC fuse-map generator accepts design details from the PLPL compiler. It is employed following successful simulation. It arranges the fuse pattern required for programming the device into a JEDEC-transfer format. This format is supported by all major suppliers of logic programmers. This module generates a fuse map that can be down-loaded directly to a PAL/PROM programmer.

PLPL Input Specification

Figure 2-8 shows a simple PLPL-input specification. It is an example of an 8-bit shift register implemented with the AmPAL22V10. The system editor is used to create the input specification. In general, a minimum PLPL input specification requires a **HEADING**, a **PINLIST**, and an **EQUATION** section. (The example shown contains additional comment strings and an optional macro section.)

Heading (Keyword **DEVICE**)

The heading identifies a title for the design along with the device used. The heading begins with the keyword, **DEVICE**. An optional title may follow the keyword. It is separated by white space (space(s), carriage return(s), or tab(s)). The device used is enclosed within parentheses following the title much like a function call in high-level software. In the example, the title is **SHFT8BIT** and the programmable device used is the AmPAL22V10.

Optional Comment String

In the example, a comment string appears between the heading and pinlist. A comment string is initiated and terminated by the double quote symbol, and all text between double quotes is ignored by the compiler. Comment strings may appear anywhere in the specification, where a white space can legally appear.

Pinlist (Keyword **PIN**)

The pinlist assigns symbolic names to the pins within the device to help describe each pin's actual function. The pinlist begins with the keyword **PIN**. The general format in which the pin name and number is entered is as follows:

PIN NAME = PIN NUMBER

example: CLOCK = 1

The pin name represents the user-specified symbolic name; the pin number is the actual device pin; the equal sign links the

two. Pin names and number may be separated from the equal sign with white space. The pinlist may be entered in any order, with white space as a separator between the keyword and the first entry, and also between succeeding entries. The pinlist is terminated by a semicolon. In addition, multiple pins may be grouped in a single statement designated as a pin vector. Examples from Figure 2-8 are shown below:

example: D[7:0] = 3,4,5,6,7,8,9,10
(same as D[7] = 3, D[6] = 4, D[5] = 5, D[4] = 6, D[3] = 7,
D[2] = 8, D[1] = 9, D[0] = 10)
example: Q[0:7] = 15:22

The symbolic pin-vector name is concatenated with numbers for each symbolic pin enclosed in brackets. There is a one-to-one mapping of the symbolic pin-vector name with the symbolic pin enclosed in brackets (i.e. D[7] = 3, D[6] = 4, etc.). A colon is used to define a sequential series of numbers and a comma is used to concatenate numbers. Colons and commas can be used together for both symbolic and actual pin-number declarations.

Optional Macro Section (Keyword **DEFINE**)

A macro section appears between the pinlist and equation sections of the example. This section begins with the keyword **DEFINE** followed by white space. Each macro definition is terminated with a semicolon, and white space can be used for formatting purposes. A macro allows function(s) with fixed arguments to be defined with a single symbolic name and used repeatedly throughout the equation section. Note that **LOAD**, **SHFTR**, **SHFTL**, and **HOLD** were defined in the macro section and used in the equation section as part of the definition of Q[7] through Q[0].

Equations (Keyword **BEGIN**)

The equation section is used to define the functions assigned to each of the output pins. The equation section begins with the keyword **BEGIN** followed by white space, and is terminated with the keyword **END**. (The period must follow **END**.) The general Boolean format for equations used in the examples is shown below:

PIN NAME := EXPRESSION;

example: Q[7] = LOAD • D[7] + SHFTR • RILO +
SHFTL • Q[6] + HOLD • Q[7];

example: RILO = Q[7];

Where the **EXPRESSION** is a sequence of **PIN NAMES** (or their complements) separated by operators, and the **PIN NAME** is the symbolic input/output (or its complement) taken from the pinlist, the operator "•" defines a sequential expression and "+" defines a combinatorial expression. The operators used in the examples are shown below:

Operator Symbols: • — AND (product)
+ — OR (sum)
/ — NOT (complement, prefix
to an expression)
; — expression terminator
:= — sequential expression
= — combinatorial expression

PLPL Benefits

The major benefit of using PLPL is that it decreases the time and costs associated with creating a design. This is possible because PLPL permits a device to be developed, simulated, and modified before it is programmed. It is also possible because PLPL is structured, self-documenting, and easy to employ.

PLPL has been released into the Public Domain and is available free of charge to any user.

DEVICE

SHFT8BIT (AmPAL22V10)

"This is a simple example of an 8-bit shift register using the AmPAL22V10."

```
PIN      CLOCK    = 1      RESET = 13      SEL[1:0] = 2,11
        RILO      = 23      LIRO  = 14
        D[7:0]    = 3,4,5,6,7,8,9,10
        Q[0:7]    = 15:22
```

```
DEFINE   LOAD      = /SEL[0] * /SEL[1];    "loads data"
        SHFTR      = SEL[0] * /SEL[1];    "shifts right"
        SHFTL      = /SEL[0] * SEL[1];     "shifts left"
        HOLD       = SEL[0] * SEL[1];     "holds data"
```

BEGIN

```
IF (RESET) THEN ARESET ( );
```

```
IF (SHFTL) THEN ENABLE (RILO);
    RILO = Q[7];
```

```
Q[7] := LOAD•D[7] + SHFTR•RILO + SHFTL•Q[6] + HOLD•Q[7];
Q[6] := LOAD•D[6] + SHFTR•Q[7] + SHFTL•Q[5] + HOLD•Q[6];
Q[5] := LOAD•D[5] + SHFTR•Q[6] + SHFTL•Q[4] + HOLD•Q[5];
Q[4] := LOAD•D[4] + SHFTR•Q[5] + SHFTL•Q[3] + HOLD•Q[4];
Q[3] := LOAD•D[3] + SHFTR•Q[4] + SHFTL•Q[2] + HOLD•Q[3];
Q[2] := LOAD•D[2] + SHFTR•Q[3] + SHFTL•1[1] + HOLD•Q[2];
Q[1] := LOAD•D[0] + SHFTR•Q[1] + SHFTL•LIRO + HOLD•Q[0];
```

```
IF (SHFTR) THEN ENABLE (LIRO);
    LIRO = Q[0];
```

END.

Figure 2-8. Sample PLPL Input Specification

2.2 PROGRAMMING HARDWARE

AMD's programmable logic devices are manufactured using IMOX, a high-performance, oxide-isolated process. Platinum-silicide fuses are used for the device programming elements. The platinum-silicide-fuse technology has a superior record of programming yield and reliability. Programming algorithms have been developed by AMD to achieve consistent programming yields in excess of 99%. To maintain this extremely high programming yield, AMD subjects all approved programming equipment to a complete testing and qualification procedure.

The fusing algorithm, which is described in detail in the reliability report, is designed to minimize tight tolerance requirements on the programming equipment. Input pins are used in the programming mode to gate fusing current through the programming path from a programming voltage applied to an output. The delivery of fusing current is therefore controlled by the switching speed of internal circuitry, not programmer circuitry. This minimizes the need for programmer recalibration. However, it is strongly recommended that users maintain a log with each programmer to collect a record of the hours of service use and the programming yield of each lot. The programming equipment should be calibrated after every 50 hours of service or whenever AMD PAL programming yields fall below 98%.

PROGRAMMER APPROVAL CRITERIA

Full details of the required programming parameters, waveforms and addressing schemes are provided on each device data sheet.

The minimum requirements for approval of a programmer by AMD are:

- 1) Must support all current AMD PAL products.
- 2) Must achieve at least 98% programming yield.
- 3) Must accept download of a JEDEC-standard PLDTF file via an RS-232C input port.
- 4) Must generate a JEDEC-standard checksum.
- 5) Must verify at HIGH and LOW V_{CC} extremes after programming.

6) Must program and verify the security fuse.

7) Must be capable of reading a non-AMD PAL device and storing the pattern during any adaptor and/or setup changes necessary for programming the pattern into an AMD device.

Although not required for AMD approval, additional desirable features are:

- 1) Support of JEDEC-standard structured test vectors.
- 2) Support of some form of signature test scheme using the signature of a known good part.
- 3) Continuity test capability.
- 4) Fast programming and verification.
- 5) Handler support.

WHY USE AN AMD-QUALIFIED PROGRAMMER ?

Programmers qualified by AMD have been tested for all the required features. They have been checked for accuracy of the programming voltages, currents, and timing parameters. In addition, a yield sample has been run to confirm that yields meet AMD's high standards. AMD considers the qualification procedure very important to maintaining control of the programming conditions seen by AMD parts and thereby assuring excellent customer programming yields. For this reason, AMD reserves the right to disallow any returns of product programmed on an unqualified programmer.

QUALIFIED PROGRAMMING EQUIPMENT

The list of AMD-qualified PAL programmer models appears in Table 2-2. New programming equipment and vendors are constantly under evaluation. Contact your AMD Field Applications Engineer or the factory to determine the approval status of any equipment not listed here.

AMD is committed to maintaining continued close working relationships with the major programmer manufacturers so that new programmable logic devices will be properly supported in a timely manner.

TABLE 2-2. AMD-QUALIFIED PAL PROGRAMMER MODELS

Vendor	Programmer Models	AMD PAL Personality Module	Socket Adapter
DATA I/O Corporation 10525 Willows Rd. N.E. P.O. Box 97046 Redmond, WA 98073-9746 (206) 881-6444	MODELS 100A, 19, 29A, 29B	LOGICPAK	303A - 004 303A - 011A 303A - 011B
	UniSite 40	Not Required	Not Required
DIGILEC, INC. 1602 Lawrence Ave. Suite 113 Ocean, NJ 07712 (210) 493-2420	803	FAM52	DA53, DA55
KONTRON ELECTRONICS, INC. 1230 Charleston Road Mountain View, CA 94039 (800) 227-8834	MODEL-MPP-80S or EPP80	Not Required	SA37
STAG MICROSYSTEMS 528-5 Weddell Drive Sunnyvale, CA 94086 (408) 745-1991	MODEL-PPZ	ZM2200	Not Required
	ZL30A/ZL32	Not Required	Not Required
STRUCTURED DESIGN, INC. 988 Bryant Way Sunnyvale, CA 94087 (408) 737-7131	SD1040 PAL Burner	Not Required	Not Required
VALLEY DATA SCIENCES 2426 Charleston Road Mountain View, CA 94043 (415) 968-2900	VDS 160	Not Required	Not Required
JMC PROMAC DIVISION 2999 Monterey Highway Monterey, CA 93940 (408) 373-3607	PROMAC - P3	Not Required	Not Required

2

GENERAL GUIDELINES FOR USING PROGRAMMING HARDWARE

There are two common situations when a PAL user wants to program parts:

- 1) The user has a master device and wants to program the master pattern into new unprogrammed parts from the same or from a different manufacturer.
- 2) The user has a file that is in JEDEC-standard Programmable Logic Data Transfer Format (PLDTF) and wants to send the file to a programmer and program parts.

All AMD-approved programmers can accomplish either of these tasks. Here are some general guidelines.

Programming with the Use of a Master Device

Suppose you have a master device and you want to program an AMD device of the same type with exactly the same pattern. The master device can be an AMD device or another manufacturer's functionally equivalent device. Follow these steps:

- 1) Set the programmer to read (or copy) the master device. This may require having a hardware adaptor for the master and entering a product code unique to the manufacturer and device type.
- 2) Install the correct adaptor (if required). Enter the appropriate product-code information. Then place the master device in the correct socket and read its fuse pattern into the programmer memory. Use whatever button-pushing sequence is required by the programmer for this operation.

3) The pattern is now in the programmer memory and will remain there unless the memory is cleared or the programmer power is turned off. Changing an adaptor or product code will not erase the memory. Usually at the end of a copy operation a checksum will be displayed. Make a note of this number. The checksum is an algorithmically calculated code unique to the pattern loaded into memory. It can be very helpful in diagnosing any programming problems. If a part is to be reused frequently as a master device, it is good practice to write the checksum on the top of the part. Never proceed with programming without checksum agreement after reading a master.

Error Detection

As a matter of curiosity, take the part out of the socket once and read an empty socket; also read a known blank part (using the right adaptor). Checksums from these two situations will be helpful in diagnosing two common problems when programming from masters: A) Forgetting to lock down the socket lever to make good contact after loading a part, and B) Loading an unprogrammed part as a master by mistake.

4) Now prepare the programmer for the AMD device to be programmed with the master pattern loaded into memory. Some programmers require different adaptors for different manufacturer's parts. If the programmer being used has this requirement be sure to use an AMD adaptor only for AMD parts. Using a non-AMD adaptor can cause permanent damage to AMD parts. Always check for adaptor compatibility.

5) Everything's okay. You have the AMD adaptor, the right AMD device code, and you wrote down the checksum that you

got after loading the master. Now put the programmer in the mode used for programming from its memory and execute the programming operation.

There is some variation in the sequence of events carried out by different programmers during the programming cycle, but all of them program and verify the appropriate fuses to match the pattern in the programmer memory. Such operations as Blank Checks, Illegal Bit Checks, Test Vector Testing, and Security Fuse Programming can be a part of the programming sequence. Check the Programmer Manufacturer's manual for the availability and appropriate use of these features.

The essential part of the programming cycle is the programming and verification of each fuse followed by a verification of all fuses at both LOW and HIGH V_{CC} . At the very end of the programming sequence you will see the checksum for the part you have just programmed. This checksum should agree with the master-part checksum. You now have an AMD part that is functionally identical to the master.

Programming from a JEDEC File

A JEDEC-standard file is the output of design-software packages used to specify fuse-blowing information to a programmer. All programmers approved for use on AMD parts will accept JEDEC files. A JEDEC file is normally prepared on a computer. The unique aspect of programming from a JEDEC file is the transfer of the file to the programmer. After the file has been transferred into the programmer, the programming task is identical to programming from a master with one exception. The exception is that design software may be used to prepare test vectors to be applied to a device immediately

following the programming cycle. These vectors will be transmitted with the JEDEC fuse file and they have a JEDEC-standard format of their own.

General guidelines for transfer of a JEDEC file and programming are as follows:

- 1) Make sure your file is in the standard JEDEC format. This will not be a problem if you are using software for file preparation that adheres to this standard.
- 2) Connect the JEDEC file source to the programmer with an RS-232C cable. The programmer manual will describe the connection details.
- 3) Prepare the programmer for receiving a JEDEC file over a link. This will generally involve entering the product-code information and putting the programmer into a ready-to-receive mode.
- 4) Transmit the file from the computer source using commercially available communications software or operating-system file-transfer software such as PIP.
- 5) After transmission a checksum should appear on the programmer display. Part of the JEDEC-standard file is a checksum. If the displayed checksum is the same as the JEDEC file-generated checksum, transmission has been successful.
- 6) Program an AMD PAL device now by first installing an AMD adaptor (if needed) and then entering the programming mode. Finally, put a part in the socket and execute the programming operation.

2.3 TESTING INFORMATION

Section 2.3.1 and 2.3.2 describe in detail the general testing requirements for programmable logic devices and how AMD has designed-in special test circuitry to permit complete testing on its programmable logic devices. Section 2.3.3 explains why it is difficult for a programmable-logic device manufacturer to provide specifications for switching delay minimums.

2.3.1 FACTORY TESTING OF PAL DEVICES

Advanced Micro Devices' bipolar PAL devices include special test circuitry to permit thorough AC and DC testing on an unprogrammed unit. The test circuitry is used to ensure good programming yield and to verify that devices will meet all parametric and switching specifications after programming.

Programming circuitry testing includes tests to assure unique addressing of all fuses. The ability of all circuitry in the programming path to handle the large currents and voltages necessary to blow fuses reliably is also thoroughly checked. To accomplish this, special test pads are provided which are accessible only during wafer probing. Using these, AMD confirms that each fuse driver is uniquely decoded and can deliver and sink the necessary current to blow fuses.

Each PAL device has special test fuses. These test fuses are blown during factory testing to prove beyond reasonable doubt that the device is capable of opening all fuses when programmed by the user.

The special probing pads and test fuses are all employed in programmability testing. This testing coupled with AMD's excellent process control gives industry-leading programming yields (> 98%) for all AMD PAL devices.

Other test circuitry, enabled by high voltages on device pins, checks functionality, AC and DC parameters under conditions that simulate post programming operation. All of the circuitry, levels and modes necessary to operation after programming are checked under worst-case conditions. For example, all input buffers are tested for functionality by switching them through a test product term to a single output, and all product term AND gates are switched and sensed for uniqueness and functionality.

Because a large percentage of die area is devoted to fixed-logic circuitry, some percentage of units can fail to function to the desired truth table, even though all fuses are correctly programmed. This problem will vary from manufacturer to manufacturer. Without effective on-chip test circuitry, functional yield after programming is largely dependent on process control. As a result, lot-to-lot variability of AC performance and functionality is to be expected from manufacturers with test-circuitry deficiencies in their products.

AMD's special test circuits and extensive factory-testing procedures have almost entirely eliminated this problem (> 99.9% PPFY). However, if absolute assurance is required, functional testing with test vectors simulating actual operation

can be performed on PAL-device programmers or automatic test equipment.

Test vectors are relatively easy to generate for combinatorial designs using PAL devices. Sequential function testing is more difficult. AMD's PAL devices are designed to provide the capability of loading the output registers to any desired value during testing. This feature, known as PRELOAD, simplifies functional testing of sequential devices. Sequencer products such as the AmPAL23S8 include buried registers. A feature called OBSERVABILITY has been designed into these products along with PRELOAD to allow control and functional test of the buried registers. Other features which AMD verifies with built-in test circuitry are polarity, asynchronous RESET, synchronous RESET and output macrocell functionality.

2.3.2 HOW TESTABILITY IS DESIGNED INTO AMD'S PROGRAMMABLE LOGIC DEVICES

Thorough testing of programmable logic devices by the manufacturer is important to both the performance of programmable logic and its cost of use.

Field programmable logic devices are different from other semiconductor products in that the user must complete the manufacturing process by programming and function testing the parts.

Programming is normally accomplished on commercially available programming equipment. Functional testing may be performed on a programmer, on automatic-test equipment, or at the board or system level. Figure 2-9 illustrates where device failure detection can occur. Clearly, the cost implications of failure become more serious with each advancing step.

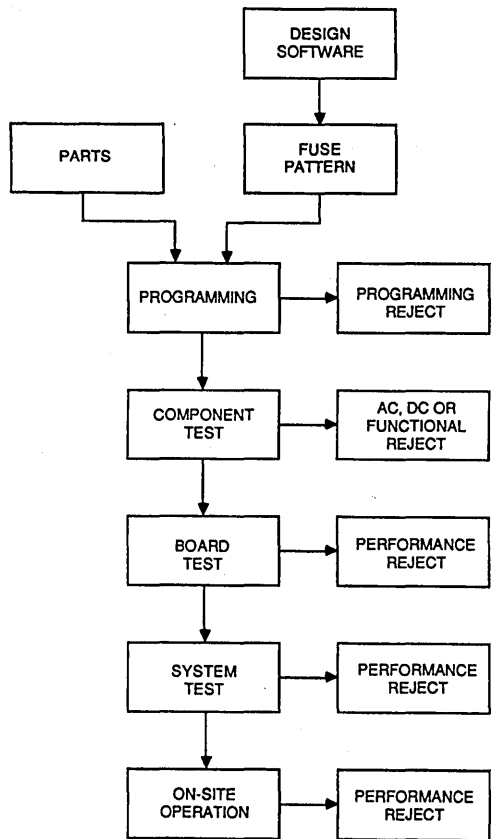
As a result of assuming the responsibility of programming and test, the user gets all the benefits of a custom function with the cost and availability advantages of a standard product. However, the user must also deal with those parts that don't program successfully or don't function to advertised specifications after programming.

The earliest programmable logic devices did not include test circuitry to allow thorough testing of the AC and functional performance prior to programming. AMD was the first manufacturer to add test circuitry to allow thorough device testing.

How well a manufacturer does the job of testing before shipping can make a difference to the user in:

- 1) Programming yield
- 2) Post programming functional yield (PPFY)
- 3) Uniformity of performance

This paper describes the techniques used at Advanced Micro Devices to allow testing of these three important attributes on every device before shipment to the user.



BD006730

Figure 2-9. User-Processing of Programmable Devices

Programming Yield

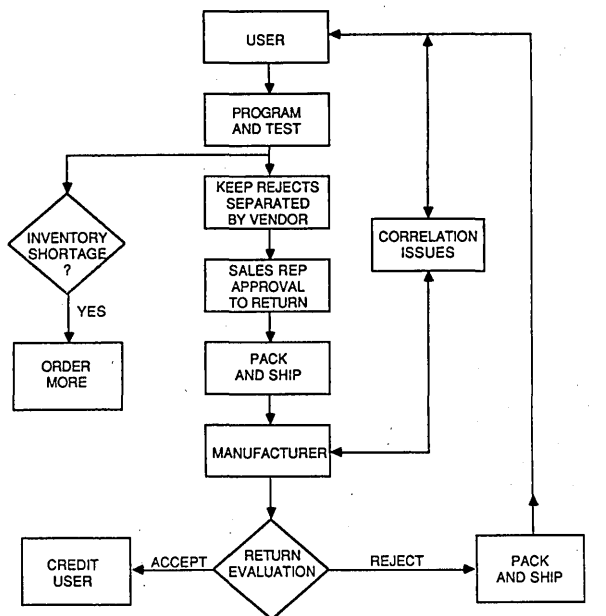
Programming yield is the measure of success of the programming operation. Large-volume users of programmable logic keep records of the programming-yield history of their suppliers' parts. Programming yield is considered by these users to be an important element in judging the overall suitability of different suppliers' parts.

Why do users care? After all, manufacturers offer a "money-back" guarantee on all valid programming rejects. The users

can simply keep the rejects separated from the good parts and send the bad ones back to the manufacturer for credit or a refund.

This sounds simple, but Figure 2-10 shows what could be involved.

Everybody loses in this operation. The manufacturer loses in return handling and evaluation costs; the user loses in return handling costs and also in added purchasing and inventory costs to compensate for programming yield losses.



BD006740

Figure 2-10. Reject Return Processing

Post Programming Functional Yield

Experienced PROM and EPROM users are sometimes puzzled by the fact that not all programmable logic devices function correctly even though they have successfully completed a programming operation and fuse verification check:

With PROMs, a one-for-one relationship exists between address states and programming elements*. That is, the state of each output for each address is dependent on the condition of only one fuse. Sensing a desired fuse state after programming therefore practically guarantees correct functional operation (At least at the voltage and temperature conditions of the programming operation).

With programmable logic devices, the relationship between programming success and post programming functionality is not one-for-one. Except for the simplest of patterns and devices, the relationship is highly complex. Feedback buffers allow the creation of more than one level of logic; latches, counters, shift registers, even oscillators can be created. Special fuse functions such as polarity control, output enables, register/nonregister selection, and buried registers complicate the relationship further.

This is the power of programmable logic—but the test challenge that results from this versatility can be substantial. Logic states for programmable logic devices can be multiple-fuse dependent. The fuse-verification procedure that examines each fuse uniquely is therefore not sufficient, as it is with PROMs, for guaranteeing functionality.

All programmable-logic-device manufacturers must create special on-chip programming circuitry and modes to allow programming and verification of each individual fuse. A review of the data sheets for different manufacturers' products gives a good idea of the special requirements for programming programmable logic devices. The complexity of programming

may vary significantly from manufacturer to manufacturer, but all have one thing in common—successful programming by itself cannot guarantee functionality.

The user's job does not end then with the programming operation. To be assured of a functional part, a comprehensive set of test vectors must be designed by the user and applied to the part. Many programmable-device programmers accept test vectors along with fuse-blowing vectors and will apply the test vectors to the part following the programming operation. AMD's PRELOAD feature greatly simplifies the test generation problem for registered parts.

Test-Vector Generation

The matter of test vector-generation is not trivial. The logic designer can generally write a series of functional states representing the expected operation of the part in the actual application, but what about all of the Don't Care states?

A great deal of work is going into automatic test vector generation for programmable logic devices. Parts manufacturers, programmer manufacturers, design software manufacturers and users all have efforts in progress. Some products are on the market.

Effective test vector generators have been or will be developed, but convenience will be key to their routine application. To be convenient to use generators must run on a wide variety of computing equipment, but best of all, they should run on the same equipment used to process the logic equations into fuse blowing vectors. Efficient algorithms will be needed so that large mainframe computers are not required to generate test vectors for the more complex parts.

Data I/O's Fingerprint

Another alternative for function testing is a signature-test technique such as Data I/O's "Fingerprint". This technique

* Programming elements can be fuses, floating-gate MOS devices, open-base NPN transistors, etc.

applies a pseudorandom series of test vectors to a known good part and generates a Fingerprint value based on its response. Each part tested thereafter must generate the same Fingerprint to be considered a "good" functional part.

To our knowledge, no one has done comprehensive studies of the effectiveness of this technique. Our limited observation indicates that the Fingerprint test is much better than no test at all. However, certain patterns can give unpredictable responses when subjected to random test conditions. Parts with these patterns cannot be Fingerprint tested reliably. Structured vector testing with either automatically or manually generated vectors is needed in these cases. The benefit of the Fingerprint approach is that it requires no effort on the part of the user, other than recognition of non-Fingerprintable patterns.

Post programming functional yield (PPFY) is clearly another distinct measure of the quality of a programmable-logic-device manufacturer's parts. Although the user has the same right of return as with programming rejects, detection of bad parts can be significantly more complex and more costly at this stage.

As shown later in this chapter, the part manufacturer can design-in additional test circuitry that guarantees virtually 100% post programming functional yield.

Uniformity of Performance

The buyer of a programmable logic part has the right to expect that the performance specifications appearing on the manufacturer's data sheet will be met for all legitimate applications of the part. This applies to each and every logic path and function.

A glance at the logic diagram for an unprogrammed part shows that, with the array in its unprogrammed state, no amount of activity of the inputs can make any output switch. Without any fuse programming, the AND gates see both the true and complement of all inputs.

Obviously if post-programming performance is to be guaranteed with confidence, test circuitry must be provided to allow each path to be tested to data-sheet performance before programming. Manufacturers vary in the degree to which they provide this pre-programming testability within their parts. The uniformity of performance of devices will reflect the degree of testability that has been designed-in.

Approach to Designing In Testability In AMD's PAL Devices

AMD's approach to the the design of programmable logic was strongly influenced by the goal to provide users with the industry's best programming yield, post programming functional yield and uniformity of performance.

Designing programmable logic can be viewed as a three-dimensional task involving high-performance logic design, fuse-programming circuit design, and test circuit design.

The first dimension is the design of a high-performance logic circuit with SSI/MSI-competitive switching speeds and very high output drive for bus environments.

The second dimension of programmable logic design is the programming circuit design. The emphasis of this design is to

provide circuitry that will deliver large programming currents to individual fuses. Special decoders, demultiplexers, buffers, and mode-select circuitry are needed. The circuits need not be fast since programming occurs at microsecond speeds. Because the circuitry is not used after programming, it is desirable that it only consume power during programming and not during operation. Since large voltages are required to generate fusing current, survival under high voltage is also a must. All of these requirements are quite different from the logic-circuit requirements but must be achieved within the same part.

Testability is the third dimension of programmable logic design. This overlay of circuitry provides the means to exercise the part through all of the possible paths that might be activated by programming. Another need for test circuitry is to insure that the programming circuitry will function properly. Testability is then important to achieving high programming yields, post-programming functionality and performance to data-sheet specifications through all possible paths.

The unique challenge of programmable logic design is to integrate these three dimensions in the most efficient manner. This is no easy task!

Testability In the Programming Circuitry

Good programming yields are in the high ninety percents. AMD PAL programming yields are typically higher than 98%.

Three things contribute to AMD's high success rate in blowing fuses:

- 1) Uniform fuse cross sections,
- 2) Pre-testing of the current delivery and sink capability of column drivers and row drivers through use of wafer-sort test pads, and
- 3) Sample fusing of test rows.

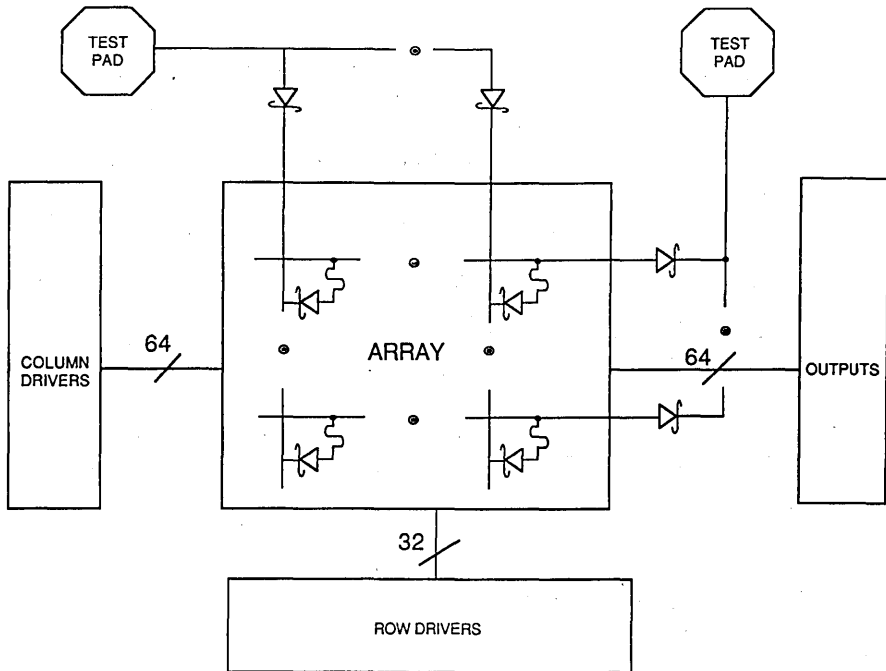
Uniformity of Fuse Cross Sections

The AMD IMOX process gives consistently uniform, platinum-silicide fuse cross sections. Uniformity is monitored by measuring fuse-resistance test patterns on a sample basis in every wafer lot. The data is processed for mean and standard deviation and trend plots are maintained. Material not meeting fuse-width-control limits is scrapped.

Testing for Fusing Current Delivery Capability

On every AMD PAL device there are two extra pads that are probed at wafer sort. These extra pads are used to gain access to the fuse array for special testing at wafer sort. The connection of these pads to the fuse array is shown in Figure 2-11.

The programming process involves selection of individual column and row drivers to deliver and sink programming current through x-y selected fuses. The extra test pads allow easy access for individually testing the source and sink capability of each column and row driver. Also a reverse-leakage check of all of the Schottky diodes in the array is possible by applying bias between the pads. Without the test pads, all of these tests would be impossible or would have to be accomplished in a less direct and less effective manner.



TC003900

Figure 2-11. Extra Test Pads for Wafer-Sort Testing of the Column and Row Drivers and the Fuse Array

Sample Programming

To further assure programmability, the AMD PAL devices include an extra test input buffer with fuses connected to each of the array columns.

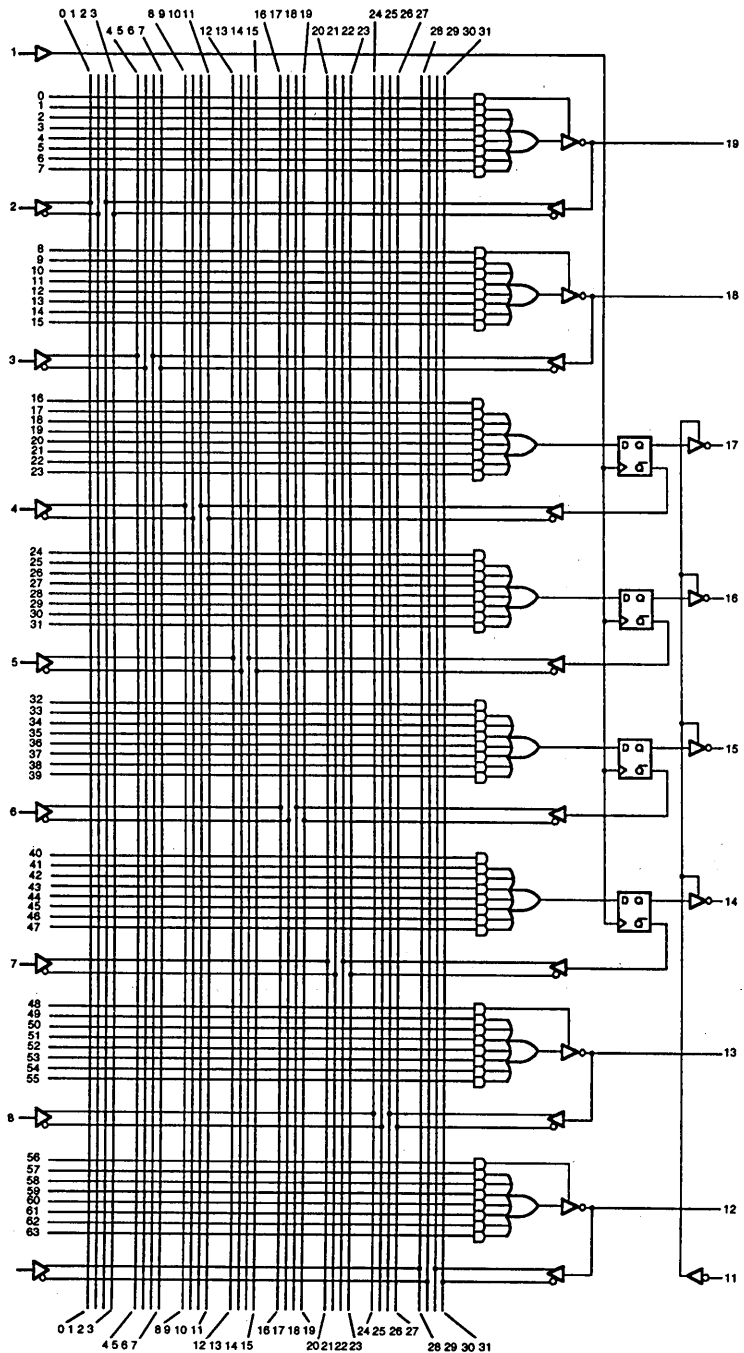
Blowing one test-buffer fuse per column accomplishes two important things. First, a sample fuse has been blown using each of the column drivers. The sample fuse is exactly the same dimension as all of the normal array fuses, and the test-buffer drivers sinking the programming current are of identical design to all of the normal drivers. Before shipment, then, each AMD PAL device has had a sample of fuses blown on

the test buffer. For example, sixty-four fuses are blown on the test word of every AmPAL16L8, one per product term.

The second purpose in blowing the sample fuses is to create a pattern for AC and functional testing.

Testability to Guarantee Functionality After Programming

A typical PAL device, the AmPAL16R4, is shown in Figure 2-12. Not shown in the logic diagram are the components located at each horizontal and vertical line intersection. For AMD bipolar-PAL devices, a fuse and a Schottky diode reside at each cross point as shown in Figure 2-11.



LD000800

Figure 2-12. AmPAL16R4 Logic Diagram

The horizontal or "Product Term" line is then the common anode connection for a 32-wide diode AND gate. The user's job is to figure out which of the thirty-two inputs should be connected to the AND gates. The inputs not needed must be disconnected by blowing the fuse shown in series with the diode.

Thankfully this decision does not have to be made 2,048 (32 x 64) times by a user. Through the wonders of design-aid software (PALASM, ABEL, CUPL, PLPL, etc.), the user simply writes a few Boolean equations describing the desired function of the device. The software then generates fusing

instructions for a programmer and all of the undesired AND-gate connections are blown away.

The obvious problem from a manufacturer's test standpoint is: "How can it be guaranteed through testing that the device will work after fuses are blown?" If the only logic in the device was that shown in Figure 2-12, there would be no chance. With sixteen LOW levels and sixteen HIGH levels presented to each AND gate, the LOWs win. All sixty-four AND outputs are always stuck LOW, and there is no way to get the output to wiggle for AC- or DC-test purposes. This is the raw state of any device before programming.

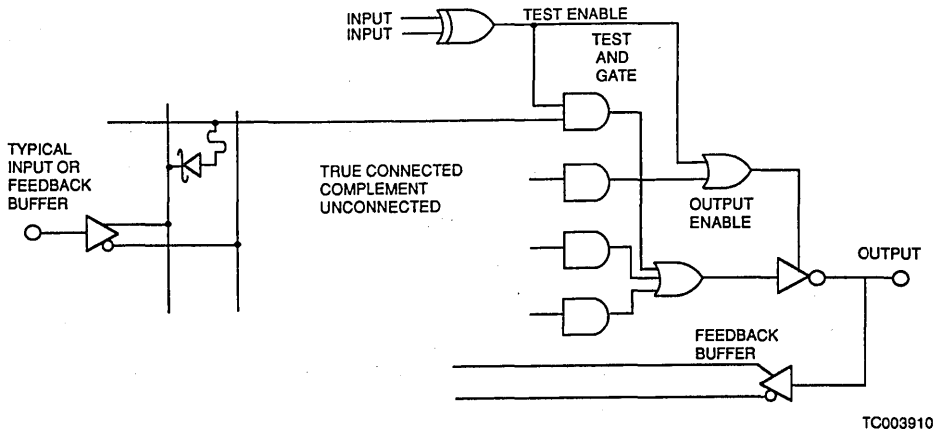


Figure 2-13. Testing All Input and Feedback Buffers through the Special Test AND Gate

Necessary Testability Requirements

Something more is needed in every PAL device to assure 100% functional yield after programming. The AMD PAL designs have an overlay of test circuitry that accomplishes the following:

- 1) Each input and feedback buffer can be checked for functionality.
- 2) Each of the AND gates can be switched HIGH and LOW and uniquely sensed by an output.

Achievement of these two things is necessary to the guarantee of 100% post programming functional yield.

Under normal operating conditions the test circuitry is inactive and consumes very little power. What causes it to come alive? Supervoltages! Supervoltages are levels substantially higher than V_{CC} so that under normal operating conditions accidental activation of a test mode cannot occur.

In this paper a double line on the input side of a logic symbol indicates that the HIGH level must be a supervoltage to activate it.

Checking the Input and Feedback Buffers

Functionality of the input and feedback buffers is checked with the aid of the extra AND gate dedicated to this function. Figure 2-13 illustrates the AND gate and its associated enabling circuitry.

The noninverting or true side of each input and feedback buffer is connected to the special test AND gate. The AND

gate is activated by a supervoltage on one of the input pins. The function actually takes two activating inputs to implement since the use of one for activation prevents that pin from being tested for functionality. Having an alternate pin to activate the function solves this problem.

Only the non-inverting side of each buffer is hooked up to the AND gate because each buffer is constructed from two inverters in series. The first inverter must work for the second one to work, so that checking the second one is sufficient to prove that they both work.

The feedback from the output used for the test cannot be fed to the test AND gate, such a connection would make the test output oscillate. For this reason its feedback input is not connected and is tested by creating another test AND gate on a different output and routing it there.

Since the special AND gate used to test all of the buffers is identical to those used in the normal operating path, switching each input through this path provides the means for testing the switching performance of each buffer.

Testing the AND Gates

The next important test requirement is to make sure that all of the AND gates work and will switch at data-sheet speeds. This test challenge is a little more complex.

What is needed in this case is:

- 1) A means for decoding one AND gate at a time in each output.

2) A way to force all input and feedback buffers to a HIGH level on both true and complement outputs.

3) A special input of identical design to a normal input that can be used to switch the decoded AND gates.

These requirements are met by the circuitry shown in Figure 2-14.

The decoder to select one AND gate at a time in each output serves a dual purpose. It is the same decoder that provides unique selection of product term lines for the programming and fuse-verify operation. It responds to binary combinations of TTL signals at three input pins and provides one of eight active-HIGH level signals to decode the AND gates.

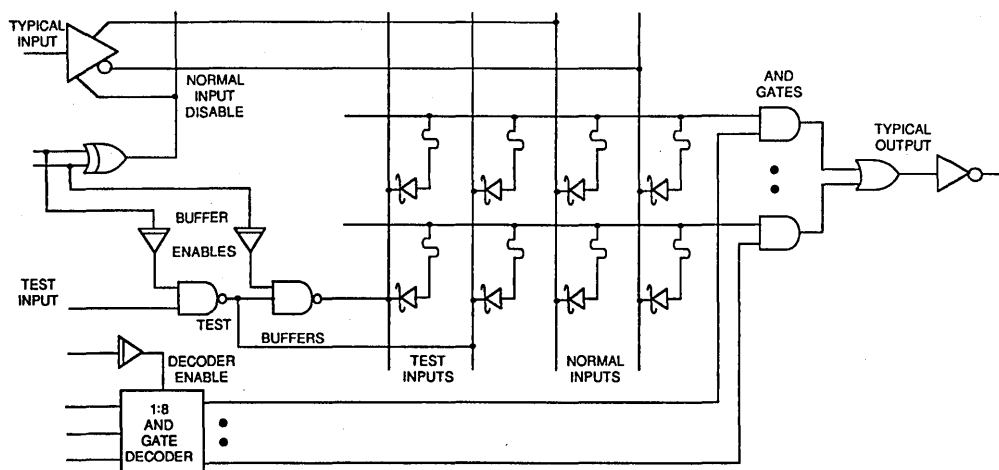
The special test input that is used in this mode also serves a dual purpose. It was mentioned earlier in this paper that a programming sample was performed on each part. This special test input is the input that carries the test fuses. During

the sample programming operation the fuses are blown in a pattern that allows switching of all sixty-four AND gates, one in each output, for each of the eight decode states.

The input to the special buffer for AND-gate testing is one of the normal input pins, but the buffer is inactive for normal operation and must be activated by supervoltage levels applied to two other inputs.

The supervoltage levels also provide the signal to force all of the buffer outputs HIGH, which is one of the three necessary requirements for AND-gate testing.

Since the design of the special buffer is identical to all of the normal input buffers, it serves as a surrogate buffer for speed testing all of the AND gates. In the AND-gate test mode, all eight outputs are switched at once, since one AND gate is selected in each output. For registered outputs, the AND-gate switching path provides the means to test setup and hold times.



TC003920

Figure 2-14. Testing All AND Gates through the Special Test Input Buffers

Uniformity of Performance

Only a complete test to data-sheet parameters by the manufacturer can assure uniformity of performance. There is no other way. Pronouncements of "guaranteed by design" should be read as "too difficult to test or no allowance for testing".

This paper has shown it is possible to design-in the means to exercise all switching paths in AMD PAL devices to data-sheet limits and conditions before programming.

The user of AMD PAL devices, therefore can expect excellent uniformity of performance to the data-sheet parameters after programming.

Summary

The central idea of this paper is that design for testability prior to programming is possible in programmable logic—and it pays off. It pays off for the user in fewer rejects at programming and at functional and AC test. For those that have no means for functional test after programming it pays off in not having to locate a defective part with a board- or system-level test. For the semiconductor manufacturer it pays off in lower returns handling cost.

All Advanced Micro Devices programmable logic devices have designed-in testability and are achieving yields of greater than 98% for programming and better than 99.9% functional and AC test yields after programming. Even higher goals have been set for future products.

2.3.3 SPECIFICATIONS FOR SWITCHING-DELAY MINIMUMS

All system designers would like non-zero minimum-delay specifications, as well as maximum-delay specifications for all AC parameters. With these numbers they could optimally design system timing. Device manufacturers understand this need and would like to meet it. Two major reasons make it impractical to provide minimum specifications.

The first reason is that maximum specifications are based on conditions that create a "worst-case" environment for the device. Maximum loading, longest delay path, multiple output switching, and V_{CC} and temperature at worst-case extremes are examples of these conditions. These conditions can be closely duplicated in an automatic-test-system environment and therefore can be guaranteed by test.

In contrast, minimum-delay specifications must be based on "best-case" conditions for a device. It is true that in a system both completely best-case and worst-case conditions for a group of devices could not practically coexist. However, anything other than the best "best case" cannot be assumed when providing specifications on a data sheet.

The device manufacturer is then faced with trying to create a "best-case" environment for test in order to guarantee

minimums. This requirement is inconsistent with the high-volume test environment of handlers, high-capacitance test heads, etc.

The second major problem with providing minimum specifications is the constant evolution and upgrade of products to achieve better performance. Minimum specifications are an unreasonable constraint to this effort.

Many system lifetimes are longer than the product lifetimes of the ICs from which they are designed. This means that more than one generation of an IC must meet the original system needs. A common reason for IC redesign is to make products faster. Faster products then replace the previous generation slower products. The conflict of this trend with guaranteed minimums is obvious.

A good example of the evolution of product performance is the popular AmPAL16L8. Before AMD entered the PAL-device market this product was originally offered with 35 ns speed. AMD entered the market with 25-ns parts and speed selections of 20 ns. The next evolution was to 15 ns, and now 10- and 7.5-ns parts are under development. Since each new generation can substitute for the previous generation when only maximum AC specifications are guaranteed, the progression to faster parts is not hindered unnecessarily. Minimum AC specifications would have seriously complicated this evolution.

2.4 AMD PROGRAMMABLE ARRAY LOGIC RELIABILITY

INTRODUCTION

Advanced Micro Devices' bipolar Programmable Array Logic (PAL) devices are based on two key technologies with many years of high-volume production experience behind them.

1) IMOX — The basic process technology employed is IMOX, an advanced ion-implanted, oxide-isolated structure. IMOX provides very high-performance devices with predictable manufacturing yields. It has accumulated many millions of hours of life-test history through its application to the Am27S Series of PROMs and the Am2900 Family of bipolar microprocessors.

A comprehensive report on IMOX reliability titled IMOX RELIABILITY REPORT (AMD Publication #03687A) is available for those interested in a detailed presentation on this subject.

2) Platinum-silicide fuses — This fuse structure was originally developed for use on Advanced Micro Devices' families of junction-isolated PROMs. It quickly established a new standard of excellence for high programming yields and long-term reliability. Several years ago it was applied to a new generation of ultra high-performance PROMs based on the IMOX process.

This combination of IMOX and platinum-silicide fuses has an outstanding record of reliability which has been verified repeatedly through in-house life testing and by high-reliability customer-qualification testing and system use.

Advanced Micro Devices' PAL devices are fabricated with this same combined-process technology. Not only is the technology for building PAL devices and PROMs the same, but also the programming algorithm and programming circuitry used to program the platinum-silicide fuses are the same in all characteristics of importance. The result is that the conditions seen by an AMD-PAL fuse are the same as those seen by an AMD-PROM fuse.

Due to the common process technology, fuse-design and fuse-programming circuitry design, reliability and programming-yield results are expected to be the same for PAL devices and PROMs. Data accumulated to date on PAL devices appears to confirm this expectation.

This report describes:

- 1) The characteristics of the platinum-silicide fuse and programming conditions for the fuse.
- 2) Reliability results accumulated to date on IMOX PAL ICs and PROMs.
- 3) The dynamic and static burn-in circuits used for high-temperature reverse-bias (HTRB) reliability testing.
- 4) Thermal resistance values for AMD PAL devices.
- 5) Equivalent gate counts for use in reliability calculations.

PLATINUM-SILICIDE FUSE

Fusing Technique

Advanced Micro Devices' PAL circuits are designed to use a programming algorithm which minimizes the requirements on

the programmer, yet allows the circuit to fuse the platinum-silicide links quickly and reliably.

The sequence of events to program a fuse are:

- 1) V_{CC} power is applied to the chip.
- 2) The address of the fuse to be programmed is selected by TTL/ECL levels on the appropriate address pins.
- 3) The outputs are disabled.
- 4) The programming voltage is then applied to one output.
- 5) A fuse enable is accomplished by raising an input to a level above normal TTL operating voltage. This action gates the current flow through the proper fuse, resulting in an open fuse in a few microseconds.
- 6) The output programming voltage is lowered and then removed.
- 7) The device is enabled and clocked if required. The output state then indicates whether successful programming has occurred. If programming has not occurred a sequence of much longer pulses is applied until programming occurs.
- 8) The sequence of 2 through 7 is repeated for each bit which must be programmed.

There are several advantages to this technique relative to that used by other PAL manufacturers. First, the two high-current power sources, V_{CC} and the voltage applied to the output, do not have critical timing requirements. As the fusing current is gated through the fuse actively, there is no dependence on the rise rate of the programming voltage. A fast application of fusing current is desirable for optimum fusing. Since the output programming voltage does not have to be applied rapidly, breakdown and latchback problems attributed to fast voltage rise times on the output are avoided.

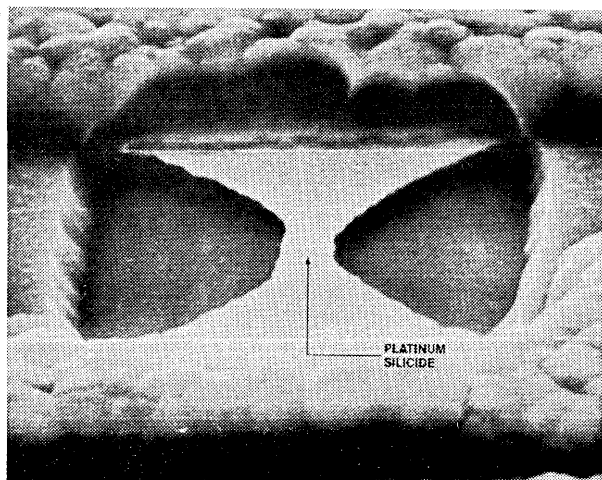
This programming procedure has a second major advantage. If the fuse does not open during the first programming pulse, longer programming pulses are used. With the platinum-silicide fuse, longer programming pulses may be safely applied with no danger of developing a reliability problem. The algorithm can therefore be designed to minimize the time required to program by using a fast first pulse followed by a longer pulse if needed to blow the occasional fuse that does not open with the first short pulse. Most devices do program satisfactorily with all short pulses.

Fuse Characteristics

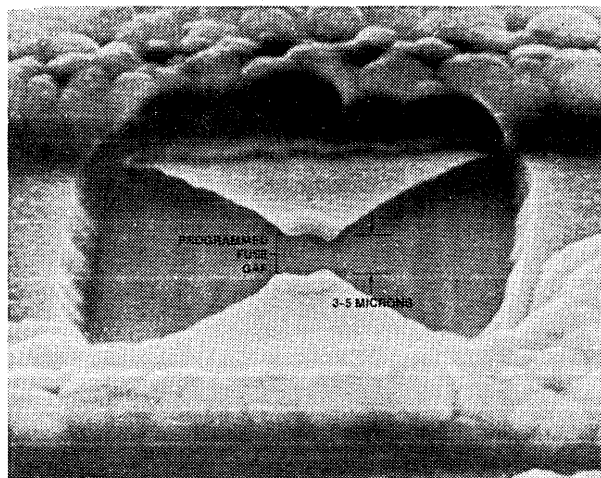
When a fast (less than 500-ns rise time) current pulse is applied to a fuse, the fuse voltage rises abruptly to a value determined by the room temperature resistance. However, it then quickly falls to a value of approximately 2 V. This value is nearly independent of the applied current. During the period of time the fuse is molten, the fuse current drops very abruptly to zero indicating the separation of the platinum-silicide into two distinct sections. Scanning Electron Microscope photographs of the resulting fuses (Figure 2-15) indicate that the typical case is a sharp clean separation in excess of a micron. This separation occurs in the center of the fuse because the "bow-tie" structure (Figure 2-16) concentrates the energy density in

the center away from the aluminum interconnect lines. The energy density in the center of the fuse creates temperatures substantially greater than those required to melt the silicide.

Melted material is then "wicked" from the center of the fuse to either side due to surface tension.

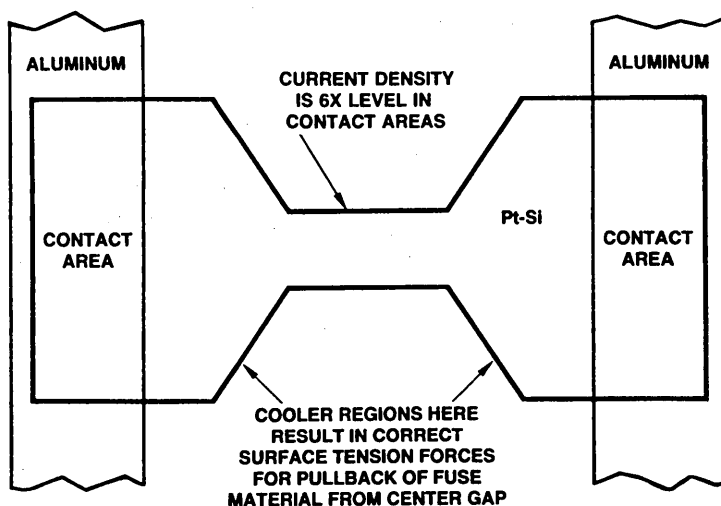


Unprogrammed Fuse



Programmed Fuse

Figure 2-15. Scanning Electron Microscope Photos — Unprogrammed and Programmed Fuses



DF006160

Figure 2-16. Bow-Tie Fuse Design

Reliability of Fuses Programmed Under Non-Optimal Conditions

The marginally opened fuse has been studied at AMD in detail even though it rarely occurs in practice. Under conditions where the fuse is purposely blown at much slower rates, it is possible for the fuse to assume a high-impedance state which is sensed as an open fuse by the circuit. This occurs when the fuse cools before separation is achieved. Electrical and SEM studies of fuses blown under these conditions indicate that a small conductive path of silicon remains of sufficiently high resistance to prevent the power transfer required for complete opening on subsequent programming attempts.

Under these slow-blow conditions, sufficient time exists for the heat flow to carry a significant amount of energy away from the fuse preventing the normal abrupt separation.

To investigate what might happen if a fuse were subjected to these under-blow conditions, a large number of fuses were deliberately programmed this way at AMD. After over two thousand hours of life testing there were no failures. It is clear from the study that partially blown platinum-silicide fuses are

stable even though they will rarely occur in circuits which have been programmed under normal conditions.

It should be noted that most manufacturers carefully specify the conditions under which their devices must be programmed in order to avoid reliability problems. Reliability data available on these devices must be assumed to have been generated using optimally programmed devices.

The study described here, and over forty billion fuse hours of data from life testing many different production lots of PROMs and PAL devices demonstrates the outstanding reliability record of the platinum-silicide fuse under a wide variety of conditions.

RELIABILITY TESTING DATA

Data on the reliability of PAL and PROM devices with platinum-silicide fuses is gathered via AMD's Reliability Monitor Program (RMP). The RMP is an ongoing program conducted on all device types across all product lines, and is designed to ensure that all AMD devices meet acceptable reliability levels. A summary of the RMP tests for hermetic- and plastic-molded packages are shown in Tables 2-3 and 2-4.

TABLE 2-3. RELIABILITY MONITOR PROGRAM FOR DEVICES IN HERMETIC PACKAGES

Test	Conditions	Typical Sample Size
Infant Mortality	160 hours at 125°C ambient. Initial and end-point electrical tests.	300
Operating Life	1000 hrs (1160 total) at 125°C ambient. Initial and end-point electrical tests.	120
Temperature Cycle	1000 cycles, (-65°C to 150°C), 30 min/cycle. end-point-hermeticity and electrical test.	50*
150°C Operating Cycle	1000 hours at 150°C ambient. Initial and end-point electrical tests.	50

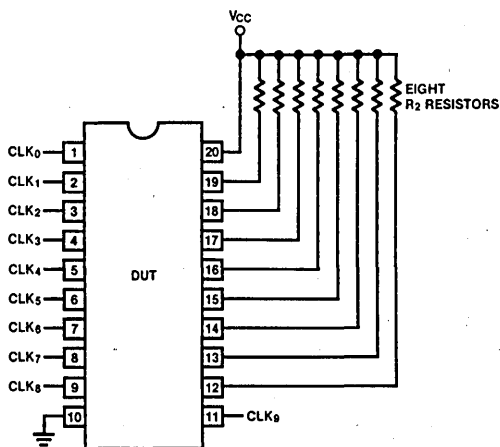
* These units are hermetically tested prior to commencement of test.

TABLE 2-4. RELIABILITY MONITOR PROGRAM FOR DEVICES IN MOLDED PACKAGES

Test	Conditions	Typical Sample Size
Infant Mortality	160 hours at 125°C or 85°C ambient ($T_J < 150^\circ\text{C}$ nominal). Initial and end-point electrical tests.	300
Operating Life	1000 hrs (1160 total) @ 125°C or 85°C ambient ($T_J < 150^\circ\text{C}$, nominal). Initial & end-point electrical tests.	120
Temperature And Humidity	85°C/85% RH/low-power bias, 500 hours and 1000 hrs. Initial, interim, and end-point electrical tests.	50
Temperature Cycle	A. 1000 cycles: -65°C to 150°C, 30 minutes/cycle. High temperature (75°C min) functional end-point electrical test.	50
Pressure Cooker	121°C, 15 psi, 160 hours, unbiased, initial end-point electrical test.	50

Data on AMD PAL and PROM devices has been gathered over millions of device hours and more than 40 billion fuse hours of high-temperature operating life tests (HTOL). The life-test circuits used in this work conform to MIL-STD-883 Method

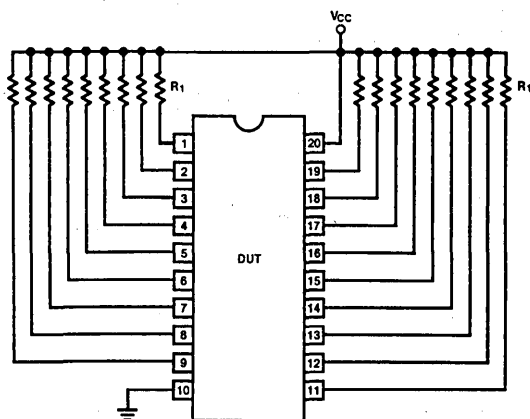
1005, Conditions C and D, and are shown in Figure 2-17. A summary of this data is shown in Table 2-5, which indicates a projected unit-failure rate (at 60% confidence) of 0.0002%/1000 hours at 70°C.



CD010140

$V_{CC} = 5 \text{ V Min.}$
 $R_2 = 270 \Omega (\pm 5\%, \frac{1}{4} \text{ W})$
 $CLK_0 = 100 \text{ kHz @ } 50\% \text{ duty cycle}$
 $CLK_{n+1} = 1/2 \text{ freq of } CLK_n$

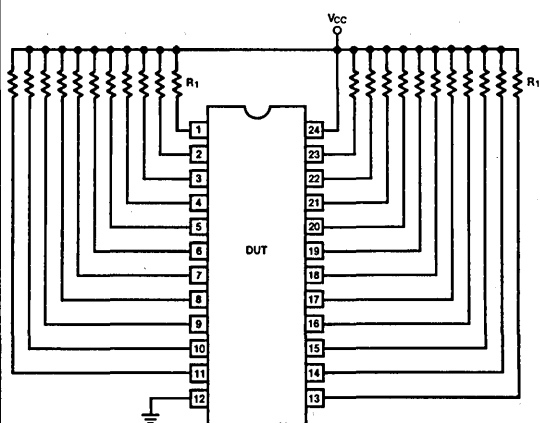
(a) 20-Pin Dynamic Burn-in
MIL-STD-883-C Condition D



CD010130

$V_{CC} = 5 \text{ V Min.}$
 $R_1 = 270 \Omega (\pm 5\%, \frac{1}{4} \text{ W})$

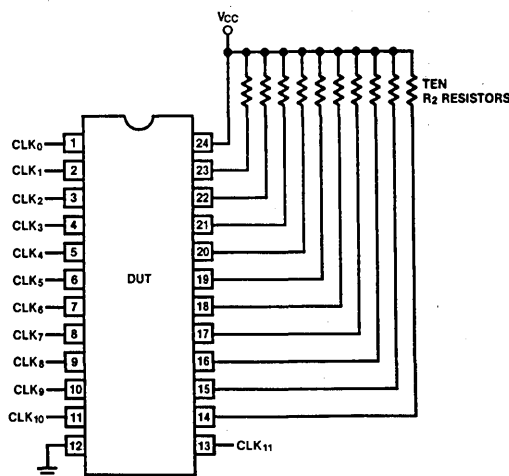
(b) 20-Pin Static Burn-in
MIL-STD-883-C Condition C



CD010150

$V_{CC} = 5 \text{ V Min.}$
 $R_1 = 270 \Omega (\pm 5\%, \frac{1}{4} \text{ W})$

(c) 24-Pin Static Burn-in
MIL-STD-883-C Condition C



CD010160

$V_{CC} = 5 \text{ V Min.}$
 $R_2 = 270 \Omega (\pm 5\%, \frac{1}{4} \text{ W})$
 $CLK_0 = 100 \text{ kHz}$
 $CLK_{n+1} = 1/2 \text{ freq of } CLK_n$

(b) 24-Pin Dynamic Burn-in
MIL-STD-883-C Condition D

Figure 2-17. Life-Test Circuits for AMD PALs

TABLE 2-5. BIPOLAR PAL AND PROM RELIABILITY SUMMARY

Product	Production Lots	Units Tested	Total Unit Hours (Thousands)	Total Fuse Hours (Billions)	Unit Failures	Fuse Related Failures	Unit Failure Rate at 60% Confidence %/1000 HRS at 125°C	Unit Failure Rate at 60% Confidence %/1000 HRS at 70°C
20-pin IMOX PALs	16	2,088	2,088	5.345	0	0	0.0217	0.0002
24-pin IMOX PALs	2	219	219	1.484	0	0	0.1250	0.0009
27S191 IMOX (16K-bit PROM)	7	1,057	1,057	17.318	1	0	0.0922	0.0007
27S180/181 (8K-bit PROM)	12	463	926	7.586	0	0	0.1100	0.0010
27S184/185 IMOX (8K-bit PROM)	15	556	1112	9.109	0	0	0.0900	0.0008
27S25 IMOX (4K-bit PROM)	5	720	720	2.950	0	0	0.0466	0.0003
TOTAL PALs & PROMs	57	5,103	6,122	43.792	1	0	0.0211	0.0002

Results of AMD's RMP are updated periodically and can be obtained through inquiry to any of the AMD Sales Offices listed in the back of this handbook.

THERMAL RESISTANCE

The thermal-resistance values given in Table 2-6 can be used to calculate the junction temperatures (T_J) of a given device at a given ambient or case temperature (T_A or T_C) and power level (P). The formulas below describe the relationship between these variables:

$$T_J = T_C + \theta_{JC}P$$

$$T_J = T_A + \theta_{JA}P$$

TABLE 2-6. THERMAL RESISTANCE VALUES

Package Type	Pins	θ_{JA}	θ_{JC}
Cerdip	20	60	11
Plastic DIP	20	61	30
LCC	20	61	CR
PLCC	20	CR	CR
Cerdip	24	57	15
Plastic DIP	24	60	CR
LCC	28	CR	CR
PLCC	28	58	CR

CR = Consult your local AMD representative.

EQUIVALENT GATE COUNT

Some methods of reliability prediction, such as those outlined in MIL-HDBK-217, incorporate into the reliability formula a variable to account for device complexity. This is based on the assumption that—all other things being equal—as the complexity of a device increases, the probability of failure also increases.

In order for the reliability formula to account for this phenomena, some means of comparing device complexity must be used. The most predominant "measuring stick" used today is

the equivalent gate count, the "equivalent gate" being a two-input NAND gate.

Unfortunately, the only standard adopted to date is the gate itself and not how to translate various logic configurations into equivalent two-input NAND gates.

Take for instance, a 32-input NAND gate. From a reliability standpoint, it is easy to see how a 32-input NAND gate is about as complex as sixteen 2-input NAND gates. Yet from a pure logic-conversion standpoint, it takes at least fifty-three 2-input NAND gates to functionally replace the 32-input NAND gate.

In addition, there are circuit configurations that do not represent any particular traditional logic block, but perform some other functions such as sense amplifiers or input-voltage circuitry.

Programmable Logic Devices complicate matters even more due to the wide range of post-programmed configurations that are possible, ranging from a very simple utilization using few product terms to a more extensive utilization.

Listed in Table 2-7 are equivalent gate counts for AMD's bipolar PAL devices. The equivalent gate count is given as a range for each of the devices to accommodate the range of possible post-programmed configurations and the inherent ambiguities associated with translating the device logic into a 2-input NAND-gate equivalent. The values shown in Table 2-7 are intended to give the user an idea of device complexity based on typical gate utilization of the programmed device.

TABLE 2-7. EQUIVALENT GATE COUNT

Device	Gate Count
16XX	200 – 300
18P8	250 – 350
22V10	700 – 800
20XRP	400 – 600
20EV8	600 – 700
23S8	900 – 1100

2.5 PROGRAMMABLE LOGIC TECHNOLOGY

AMD uses both Bipolar and CMOS technologies to manufacture programmable logic devices. Sections 2.5.1 and 2.5.2 provide a detailed description of these two technologies.

2.5.1 IMOX-III™ — Advanced Bipolar Technology for PAL Devices

In order to meet the next-generation requirements for speed and density in PAL devices, AMD has developed an advanced bipolar technology called IMOX-III. Although IMOX-III represents a major breakthrough which will allow further scaling to the sub-micron region, the technology also shares many features in common with AMD's prior generations of technology, IMOX-II and IMOX-IIS.

The revolutionary breakthrough of IMOX-III is the use of reactive-ion-etched grooves, called slots, to isolate the tran-

sistors. These slots are 1.5 microns wide, over 6 microns deep, and are filled with dielectric material (Figure 2-18). Because the transistors are not isolated by junctions, space for depletion spreading is not necessary. Also, since the slots are etched anisotropically, thicker EPI layers can be isolated without increasing the isolation widths. Essentially, no density penalty is paid to achieve high breakdown voltages. Higher breakdown voltages are needed to support the programming voltages required to blow fuses in bipolar PAL devices. Smaller device sizes translates into faster circuits through smaller die sizes and reduced capacitances of active devices and metal interconnect. Another advantage of the slot isolation is reduced collector to substrate capacitance which offers improved performance in many circuit configurations.

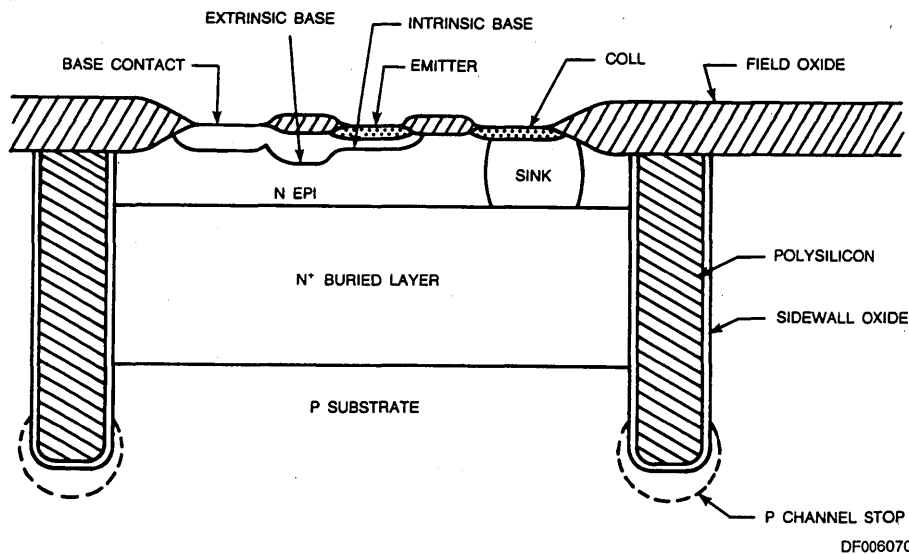


Figure 2-18. Slot Isolation

Overall, the IMOX-III process is a major step forward compared to IMOX-IIS. In addition to the slot isolation, stepper lithography, dry metal, and via etching have been implemented, resulting in a dramatic reduction in design rules. The slot isolation allows the silicon pitch to be reduced by one third. The steppers and plasma metal etching allow the metal pitch to be shrunk by one third also. Furthermore, the IMOX-III process was designed with a 20% shrink in mind. This scaling can be accomplished simply by shrinking the masks.

The IMOX-III process shares many familiar features with its predecessor, IMOX-IIS. Oxide-walled bases and emitters are used to reduce the size and parasitic capacitances of transistors. Ion-implanted emitters and bases are used to achieve the profile control necessary for high-performance transistors. The reliability of the transistor structure used in IMOX-III has been proven over millions of hours of high-temperature tests on AMD products that use IMOX-II and IMOX-IIS processes.

Another key feature familiar to users of older generation AMD-PAL devices is the fuse technology. IMOX-III uses platinum-silicide fuses, identical to the fuse technology used on older-generation IMOX PAL devices. Programming yields are the highest of any fuse technology, and programming times are extremely short (about 300 ns). The reliability of the platinum-silicide fuse is unsurpassed by any other fuse technology.

The IMOX-III technology also features two levels of metallization as does IMOX-II and IMOX-IIS. However, in the IMOX-III technology, both layers are stepper defined and plasma etched.

The IMOX-III technology is being applied to a family of high-performance PAL devices. The first of these is a "D-speed" 20-pin PAL IC, which will run at 10 ns. Also designed in the IMOX-III technology are a family of high-performance ECL PAL devices. The first of these are the 24-pin AmPAL20EV8 and AmPAL20EG8. These parts have 3600 programmable fuses in the AND/OR array. The high performance inherent in the IMOX-III technology gives these ECL PAL devices a propagation delay of 6 ns and a cycle time of 8 ns (3.5-ns clock-to-output plus 4.5-ns setup time). This enables the AmPAL20EV8 or the AmPAL20EG8 to support a 125-MHz system. Table 2-8 lists the processes that are used to fabricate AMD's PAL devices.

IMOX-III technology will enable AMD to develop third and fourth generations of PAL devices that will be significantly faster and more complex than the current devices. It will also reduce the cost of the new devices by significantly reducing die sizes or allowing more features to be added without increasing current die sizes. Faster and more complex PAL devices will permit system designers to build advanced computers, communications systems and instrumentation systems at a much lower cost.

TABLE 2-8. PROCESSES USED TO FABRICATE AMD'S PAL DEVICES

Device	Process	Maximum Propagation Delay/Maximum Frequency
16xx Family	IMOX-II	25 ns
16xxB Family	IMOX-IIS	15 ns
16xxD Family	IMOX-III	10 ns
22V10	IMOX-IIS	25 ns
22V10B	IMOX-III	15 ns
18P8	IMOX-IIS	15 ns
23S8	IMOX-IIS	20 ns/40 MHz
20EV8	IMOX-III	6 ns/125 MHz
20EG8	IMOX-III	6 ns
20XRP Family	IMOX-IIS	15 ns
21VT8B	IMOX-III	15 ns

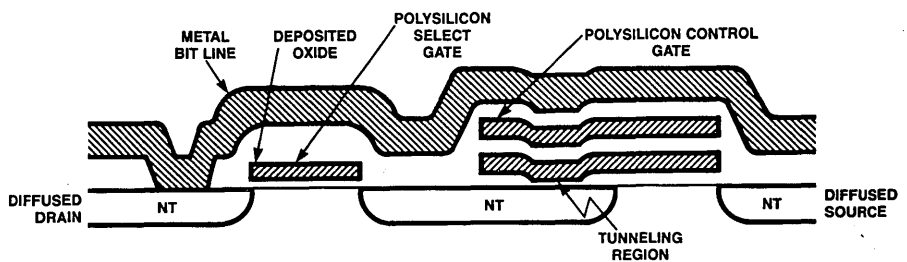
2.5.2 ADVANCED CMOS TECHNOLOGY FOR PAL DEVICES

In addition to the commonly used bipolar TTL and ECL technologies, AMD also manufactures programmable logic devices using an advanced CMOS EEPROM-based technology. This technology offers several significant advantages. Mainly it allows AMD to offer lower power parts of high complexity. PAL devices based on this technology can also be reprogrammed, unlike the bipolar fuse-based parts. In addition, since the EE-cells can be reprogrammed, these devices can be 100% tested at the factory before being shipped to the customer.

This production-tested CMOS process employs state-of-the-art design rules. It uses stepper lithography on all critical levels with a minimum feature size of 1.5 microns. The transistor gate oxide thickness is approximately 300 Å. This advanced process permits volume production of EEPAL devices with state-of-the-art speed-power performance. In addition, contin-

ued technology enhancements are in development that will result in significantly reduced dimensions and increased packing densities, allowing production of even faster circuits at lower cost.

The EE-cell, which can be electrically erased and reprogrammed, contains a floating gate transistor structure (two layers of polysilicon) with a tunnel oxide region of less than 100 Å through which electrons can "tunnel" to either charge or discharge the cell (See Figure below). An additional enhancement transistor has been added in series with the storage cell to prevent leakage in the non-selected discharged cells (as they have a negative threshold) during a charge sensing cycle. This transistor also protects non-selected EE-cells on the same word line during the charge cycle. The tunnel oxide process developed by AMD allows easy manufacturing of ultra-high quality, thermally grown thin oxide capable of withstanding the high fields associated with the tunneling mechanism.



The EEPAL process has a proven history of reliability since it has been used to manufacture N-channel EEPROM devices for several years at AMD. These devices have been in the field for a long period and have gone through extensive testing at the factory (See the Am9864 Reliability Report which can be obtained by contacting an AMD representative).

Users of devices that are based on EE-cell technology have two concerns about the technology: Endurance (number of program or write cycles) and Data Retention (charge storage from the last time the cell was updated). The endurance issue is of little significance to EEPAL users, as opposed to EEPROM users, since the PAL devices are usually reprogrammed only several times, whereas EEPROMS are written up to 10,000 times. AMD's process is capable of supporting higher impedance levels than are specified for PAL devices.

The second concern arises over the leakage of charge from the EE-cell over a period of time, thus potentially degrading the device's performance. At AMD this issue is resolved by designing the PAL circuits so that performance is guaranteed to the factory specifications under worst-case conditions for a minimum of 10 years. If the device is reprogrammed with the same or a different pattern during this period, functionality is assured for another 10 years from that time.

The CMOS PAL Family could have been implemented with either EPROM or EEPROM technologies, both of which are in production at AMD. AMD chose the EEPROM technology to

manufacture PAL devices because of the superior flexibility it offered without compromising performance.

One of the major benefits of EEPROM technology is 100% testability. The EEPAL devices can be fully and more easily tested since they are electrically erasable. Compared to EEPAL devices, EPAL (based on EPROM technology) devices cannot be easily reprogrammed since they require a lengthy amount of exposure to ultraviolet light to be erased. Also, if EPAL devices are placed in a conventional non-windowed package (which is easier and cheaper to produce), they cannot be reprogrammed. EEPAL devices do not face these problems. EPAL devices also require additional circuitry to improve testability due to the lack of flexibility in programming. This adds significantly to the design cycle time since additional design and debugging time are required. Overall, EE-cell based technology is more attractive in the long run to manufacture PAL devices.

With both bipolar (TTL and ECL) and CMOS technology capabilities, AMD is in an ideal position to meet customers' needs for different programmable logic devices. AMD can choose the proper technology to produce the PAL device that the customer requires. Where blazing speed is needed bipolar is the technology of choice, whereas for low-power high-complexity devices CMOS technology is appropriate. Over the next several years enhancements will be made to both these technologies allowing production of even faster, lower power, and more complex PAL devices. The flexibility to choose technology permits AMD to better serve its customers.

SECTION 3 — APPLICATIONS



3.1 OVERVIEW

3.2 COMBINATORIAL LOGIC

3.2.1 Multiplexers

3.2.2 Demultiplexers

3.2.3 Encoders/Decoders

3.2.4 Comparators

3.2.5 Address Decoding and Chip Select Generation Simplified
with Combinatorial PAL Devices

3

3.3 SEQUENTIAL LOGIC

3.3.1 Counters

3.3.2 Shifters

3.4 MICROPROCESSOR INTERFACE LOGIC

3.4.1 The Interface Problem

3.4.2 Interfacing to the 8086/80186/80286

3.4.3 Interfacing to the 68000/68020

3.4.4 Interfacing to the 8088/80188

3.4.4 Interfacing to the Z80/Z8000

3.5 BUS INTERFACE LOGIC

3.5.1 MULTIBUS to Am9516 Interface

3.5.2 Z-Bus and 8088/8086 Interface

3.5.3 An AMD PAL MULTIBUS Arbiter

(Continued)



3.6 MISCELLANEOUS LOGIC FUNCTIONS

3.6.1 8088 to Am2968 Interface

3.6.2 A General-Purpose Interface for the Am2968

3.6.3 MC68000 to Am2968 Interface

3.6.4 General-Purpose Dual-Port Arbiter

3.6.5 Customized a Flexible DRAM Controller Using Second-Generation PAL Devices

3.6.6 Dynamic Memory Control State Sequencer

3.6.7 82284 and 82288 Emulation in an IBM PC/AT Computer Using Two AmPAL16R8B Devices

3.6.8 Interfacing the 80186 Microprocessor to the 8087 with the AmPAL22V10A Device

3.6.9 A MULTIBUS Arbiter Design for 10 MHz Processors

3.7 PROGRAMMABLE LOGIC DEVICES — ARTICLE REPRINTS

3.7.1 "Logical Alternatives in Supermini Design," reprinted by permission of Computer Design, November 1983. All rights reserved.

3.7.2 "PLDs as Semicustom Substitutes," reprinted by permission of VLSI Design, June 1985. All rights reserved.

3.7.3 "Programmable logic chip rivals gate arrays in flexibility," reprinted by permission of Electronic Design, December 1983. All rights reserved.

3.7.4 "PAL device buries state registers, brings state machines to life," reprinted by permission of Electronic Design, July 1986. All rights reserved.

3.7.5 "Mixing Data Paths Expands Options in System Design," reprinted by permission of Computer Design, January 1985. All rights reserved.

(Continued)



- 3.7.6 "High Performance DMA for the VMEbus," reprinted by permission of Digital Design, November 1985. All rights reserved.
- 3.7.7 "A Demultiplexed Analog Subsystem," reprinted by permission of Digital Design, November 1985. All rights reserved.
- 3.7.8 "Advanced Programming Language for Programmable Logic Devices," from Southcon 1985.
- 3.7.9 "Test Methods for Programmable Logic, " from Southcon 1984.
- 3.7.10 "Fuse-programmable chip takes command of distributed systems," reprinted by permission of Electronic Design, October 17, 1985. All rights reserved.
- 3.7.11 "Programmable event generator conquers timing restraints," reprinted by permission of Electronic Products, July 1, 1986. All rights reserved.
- 3.7.12 "PLDs Implement encoder/decoder for disk drives," reprinted by permission of Electronic Design News, September 18, 1986. All rights reserved.

3.8 BIBLIOGRAPHY

3.1 OVERVIEW

Programmable Logic Devices (PLDs) are off-the-shelf custom devices. Combining the flexibility of custom logic and off-the-shelf availability of standard products, programmable logic devices offer attractive logic design solutions. Because their higher levels of integration and other features such as programmable I/O pins, polarity controls, flexible output structures, etc. (see Section 1 for detailed discussion), designs based on PAL devices result in lower parts count compared with those based on discrete SSI/MSI logic. Existing PAL devices have been widely used to consolidate random logic into a structured form in many digital systems. PLDs are used in MOS microprocessors, single-chip microcomputer systems, and in bipolar bit-slice microprocessor-based systems. Typically, domain of PLD applications includes all SSI/MSI logic. Wherever SSI/MSI logic is used, PLDs are likely candidates for those applications.

Applications for PAL devices can be in one of the following categories:

SSI/MSI Logic Replacement: This includes combinatorial logic functions (such as special-purpose decoders/priority encoders, multiplexers/demultiplexers, comparators, adders/subtractors), synchronous or asynchronous sequential functions (such as special-purpose counters—binary counters, decade counters, gray-code counters, Johnson counters, modulo counters using registered flip-flops) or logic using latches and glue logic (for replacing many discrete flip-flops).

State-Machine Applications: These can be very simple state machines such as counters or more complex state machines such as bus controllers, bus arbiters, DRAM controllers, DMA controllers, FIFO controllers, or dual-port memory controllers.

Microprocessor/Single-Chip Microcomputer Interface: Interface to various LSI/VLSI peripheral controller chips.

Other Miscellaneous Functions: Such as memory-refresh generation, wait-state generation, timer/counter functions, error detection/control and memory scrubbing functions. These functions may also include various I/O interface and support—such as intelligent I/O ports, data-communications interface (for protocol conversion), display or front-panel interface, keyboard scanning, disk and tape drive controls, standalone microprocessor-based controllers (stepper-motor control, sensor monitor, etc.), or support for image processing and signal processing.

APPLICATION OF PAL DEVICES IN DIGITAL SYSTEMS

A typical digital system can be partitioned into three general areas: 1) data path, 2) control path, and 3) interface path (glue area).

Data-path functions typically include data manipulation (ALUs), data storage (register file, pipeline registers, etc.) and data-steering/selection (multiplexing/demultiplexing). Typically, the data-path portion of a system is the most structured portion. This is likely to be defined relatively early in the design cycle, and is less likely to be changed. Usually performance and density are important for the data-path area.

Control-path functions include the timing, sequencing and decision making segments of the digital system. These sections are normally implemented with state machines—either with random logic, PROMs, or RAMs. The control section is the most complex portion of a digital system. It is likely to contain subtle errors and requires many changes during prototyping. Performance and design turnaround time tend to be critical for the control path, while density considerations are less important.

Interface or glue applications fulfill miscellaneous functions. Typically, interface circuitry connects various LSI modules, such as microprocessors, peripherals, memories, and gate arrays. For interface applications, logic flexibility and design turnaround time are critical.

PAL devices, because of their general-purpose nature, are used in all of these areas.

In the data path they can be used for data steering and data manipulation. Using PAL devices for data steering simplifies the implementation of a multiple-bus architecture. Data manipulation may include functions such as 16/32/64-bit customizable bidirectional shifters, barrel shifters, constant-generation logic, and sign-extension logic.

In control path, PAL devices are used extensively for optimizing control functions, such as instruction predecoding, double pipelining control, register-file control, and special instruction control.

Interface and glue-logic applications include interfacing various peripherals to different microprocessors, and interfacing different peripherals to different buses. PAL devices are ideal for replacing SSI/MSI devices, for saving board space and providing user customizability. There are several problems associated with interfacing a general-purpose peripheral device to a microprocessor. The system designer has to watch for various control signals that each chip uses and its compatibilities.

One of the most common use of PAL devices for interfacing applications is to perform chip-select decoding and address decoding from address and status signals. PAL devices offer the flexibility of timing generation and doing timing changes. Timing changes can be simply implemented by adding or changing a term in the logic equations and reprogramming the device.

For interfacing different peripherals to microprocessors, PAL devices can be used for generating address strobes, data strobes, Read/Write and interrupt-acknowledge signals based on the status information available from the microprocessor. They can be used for generating coordinated timing signals—such as simultaneous assertion/disassertion of various signals, automatic insertion of a variable number of wait states to match different-speed microprocessors and peripherals.

For various bus-interface applications, the most common use of the PAL device is for implementing bus-arbitration and bus-control functions. Bus arbitration and bus grant control function is done by monitoring various transfer-request signals and deciding which request gets the bus and when it gets the bus. In multi-master systems, some sort of arbitration scheme and prioritization scheme needs to be incorporated. This can be done by assigning different priorities to different bus masters. Bus arbitration functions include request synchronization, bus grant/access control functions and generation of various control signals.

This application section is organized into the following subsections:

Section 3.2 describes simple combinatorial functions (such as multiplexers/demultiplexers, encoders/decoders and comparators) implemented with PAL devices.

Section 3.3 describes some sequential applications such as counters (Modulo counters, Johnson counter, dual 4-bit BCD counter, and a Grey-binary counter) and shifters.

Section 3.4 describes various microprocessor interfaces to different peripheral chips using PAL devices.

Section 3.5 describes various bus interface and bus functions (such as bus arbitration, bus control) using PAL devices.

Section 3.6 describes various miscellaneous functions implemented with PAL devices.

Many of the examples shown are paper designs and modifications may be necessary when using them in actual applications. AMD assumes no responsibility for the use of any application described.

3.2 COMBINATORIAL LOGIC

3.2.1 MULTIPLEXERS

The multiplexer (also called data selector) is used to selectively route data from several inputs to one output.

Three 4-to-1 Multiplexers (AmPAL18P8)

A simple 4-to-1 multiplexer has four data input lines and two control lines that select which one of the four data inputs is to be passed to the output. The function table for this device is shown in Table 3-1.

Each AND gate has a data input and a two-input combination of select inputs. Given one of four possible combinations of select bits, only the AND gate corresponding to this combination is enabled, allowing the desired input data to pass to the output. This can easily be expanded to accommodate more data by adding more select lines and data inputs. For every n select lines there can be 2^n data inputs, each of which require one product term.

TABLE 3-1. FUNCTION TABLE FOR 4-TO-1 MULTIPLEXER

Inputs						Output
S1	S0	D3	D2	D1	D0	Y
0	0	X	X	X	0	0
0	0	X	X	X	1	1
0	1	X	X	0	X	0
0	1	X	X	1	X	1
1	0	X	0	X	X	0
1	0	X	1	X	X	1
1	1	0	X	X	X	0
1	1	1	X	X	X	1

Three 4-to-1 multiplexers require twelve inputs, three outputs, and two control select pins, for a total of seventeen input/output pins. Each output needs four product terms. These requirements are satisfied by a single AmPAL18P8 device. The corresponding PLPL specification and the sum-of-products equations generated by the compiler are shown in Figure 3-1.

Four 3-to-1 Multiplexers (AmPAL18P8)

Four 3-to-1 multiplexers require twelve inputs, four outputs, and two control select pins, for a total of eighteen input/output pins. Each output needs three product terms. These requirements are satisfied by a single AmPAL18P8 device. The corresponding PLPL specification and the sum-of-products equations generated by the compiler are shown in Figure 3-2.

16-to-1 Multiplexer (AmPAL22V10)

The 16-to-1 multiplexer requires sixteen inputs, one output, and four control select pins, for a total of twenty-one input/output pins. These requirements are satisfied by a single AmPAL22V10 device. The corresponding PLPL specifications and the sum-of-products equations generated by the compiler are shown in Figure 3-3.

3.2.2 DEMULTIPLEXER

A decoder with an enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines.

Dual 2-to-4 Demultiplexers (AmPAL16H8)

This design implements dual 2-to-4 demultiplexers on an AmPAL16H8. Each demultiplexer is individually enabled but shares the same select input lines with the other (see Table 3-2 and 3-3). The corresponding PLPL specification and the sum-of-products equations generated by the compiler are shown in Figure 3-4.

TABLE 3-2. FUNCTION TABLE FOR FIRST DEMULTIPLEXER

Inputs		Outputs					
Enable	Select						
G1	C1	B	A	Y1[0]	Y1[1]	Y1[2]	Y1[3]
H	X	X	X	H	H	H	H
X	L	X	X	H	H	H	H
L	H	L	L	L	H	H	H
L	H	L	H	H	L	H	H
L	H	H	L	H	H	L	H
L	H	H	H	H	H	H	L

TABLE 3-3. FUNCTION TABLE FOR SECOND DEMULTIPLEXER

Inputs		Outputs					
Enable	Select						
G2	C2	B	A	Y2[0]	Y2[1]	Y2[2]	Y2[3]
H	X	X	X	H	H	H	H
X	H	X	X	H	H	H	H
L	L	L	L	L	H	H	H
L	L	L	H	H	L	H	H
L	L	H	L	H	H	L	H
L	L	H	H	H	H	H	L

```

DEVICE THREE_4_TO_1_MULTIPLEXER (PAL18P8)
PIN  DAO = 1  DA1 = 2  DA2 = 3  DA3 = 4
      DB0 = 5  DB1 = 6  DB2 = 7  DB3 = 8
      DCO = 9  DC1 = 11 DC2 = 12 DC3 = 13
      OUTA = 14
      OUTB = 15
      OUTC = 16
      SELO = 17
      SEL1 = 18 ;

```

```

DEFINE
  S0 = /SELO * /SEL1 ;
  S1 = /SELO * SEL1 ;
  S2 = SELO * /SEL1 ;
  S3 = SELO * SEL1 ;

```

```

BEGIN
  IF (S0) THEN BEGIN
    OUTA = DAO ;
    OUTB = DB0 ;
    OUTC = DCO ;
  END ;
  IF (S1) THEN BEGIN
    OUTA = DA1 ;
    OUTB = DB1 ;
    OUTC = DC1 ;
  END ;
  IF (S2) THEN BEGIN
    OUTA = DA2 ;
    OUTB = DB2 ;
    OUTC = DC2 ;
  END ;
  IF (S3) THEN BEGIN
    OUTA = DA3 ;
    OUTB = DB3 ;
    OUTC = DC3 ;
  END ;
END.

```

TEST_VECTORS

```

IN      DAO DA1 DA2 DA3 DB0 DB1 DB2 DB3
        DCO DC1 DC2 DC3 SELO SEL1 ;
OUT     OUTA OUTB OUTC ;

```

```

BEGIN
"DA[0:3] DB[0:3] DC[0:3] SELO SEL1 OUTA OUTB OUTC"
  1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 H L H ;
  0 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 H L H ;
  0 0 1 0 1 1 0 1 0 0 1 0 1 0 0 0 H L H ;
  0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 H L H ;
  0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 L H L ;
  1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 0 L H L ;
  1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 L H L ;
  1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 0 L H L ;
END.

```

Figure 3-1. AmPAL18P8 PLPL Specification—Three 4-to-1 Multiplexers

DEVICE FOUR_3_TO_1_MULTIPLEXER (PAL18P8)

```
"
  FOUR 3:1 MULTIPLEXER
"
PIN  DA0 = 1  DA1 = 2  DA2 = 3
      DB0 = 4  DB1 = 5  DB2 = 6
      DC0 = 7  DC1 = 8  DC2 = 9
      DDO = 11 DD1 = 12 DD2 = 13
      OUTPUTA = 14  OUTPUTB = 15
      OUTPUTC = 16  OUTPUTD = 17
      SELECT0 = 18  SELECT1 = 19 ;
```

```
BEGIN
  IF (/SELECT0) THEN BEGIN
    OUTPUTA = DA0 ;
    OUTPUTB = DB0 ;
    OUTPUTC = DC0 ;
    OUTPUTD = DDO ;
  END ;
  IF (SELECT0*/SELECT1) THEN BEGIN
    OUTPUTA = DA1 ;
    OUTPUTB = DB1 ;
    OUTPUTC = DC1 ;
    OUTPUTD = DD1 ;
  END ;
  IF (SELECT0*SELECT1) THEN BEGIN
    OUTPUTA = DA2 ;
    OUTPUTB = DB2 ;
    OUTPUTC = DC2 ;
    OUTPUTD = DD2 ;
  END ;
END.
```

TEST_VECTORS

```
IN      DA0 DA1 DA2 DB0 DB1 DB2 DC0 DC1 DC2 DD0 DD1 DD2
        SELECT0 SELECT1 ;
OUT     OUTPUTA OUTPUTB OUTPUTC OUTPUTD ;

BEGIN
"D D D  D D D  D D D  D D D S S                "
"A A A  B B B  C C C  D D D E E                "
"0 1 2  0 1 2  0 1 2  0 1 2 L L  OUTPUT OUTPUT OUTPUT OUTPUT"
"              0 1  A    B    C    D  "
"
1 0 0  0 1 1  1 0 0  0 1 1 0 X  H    L    H    L ;
0 1 0  1 0 1  0 1 0  1 0 1 1 0  H    L    H    L ;
0 0 1  1 1 0  0 0 1  1 1 0 1 1  H    L    H    L ;
0 1 1  1 0 0  0 1 1  1 0 0 0 X  L    H    L    H ;
1 0 1  0 1 0  1 0 1  0 1 0 1 0  L    H    L    H ;
1 1 0  0 0 1  1 1 0  0 0 1 1 1  L    H    L    H ;
END,
```

Figure 3-2. AmPAL18P8 PLPL Specification—Four 3-to-1 Multiplexers

Figure 3-3. AmPAL22V10 PLPL Specification—16-to-1 Multiplexer

DEVICE DUAL_2_TO_4_DEMULTIPLEXER (PAL18P8)

```

PIN    A = 2  B = 1
      G1 = 3  G2 = 4
      C1 = 5  C2 = 6
      Y1[0:3] = 19:16
      Y2[0:3] = 15:12 ;

BEGIN
  IF (G1 + /C1) THEN Y1[0:3] = #B1111 ;
  IF (/G1 * C1) THEN BEGIN
    CASE (B,A)
    BEGIN
      #B00) BEGIN
        Y1[0] = 0 ;
        Y1[1:3] = #B111 ;
      END ;
      #B01) BEGIN
        Y1[1] = 0 ;
        Y1[0,2,3] = #B111 ;
      END ;
      #B10) BEGIN
        Y1[2] = 0 ;
        Y1[0,1,3] = #B111 ;
      END ;
      #B11) BEGIN
        Y1[3] = 0 ;
        Y1[0:2] = #B111 ;
      END ;
    END ;
  END ;
  IF (G2 + C2) THEN Y2[0:3] = #B1111 ;
  IF (/G2 * /C2) THEN BEGIN
    CASE (B,A)
    BEGIN
      #B00) BEGIN
        Y2[0] = 0 ;
        Y2[1:3] = #B111 ;
      END ;
      #B01) BEGIN
        Y2[1] = 0 ;
        Y2[0,2,3] = #B111 ;
      END ;
    END ;
  END ;

```

```

#B10) BEGIN
  Y2[2] = 0 ;
  Y2[0,1,3] = #B111 ;
END ;
#B11) BEGIN
  Y2[3] = 0 ;
  Y2[0:2] = #B111 ;
END ;

```

END ;

END.

TEST_VECTORS

```

IN      B A G1 C1 G2 C2 ;
OUT     Y1[0:3] Y2[0:3] ;

```

BEGIN

"B	A	G1	C1	G2	C2	Y1[0:3]	Y2[0:3]
X	X	1	X	1	X	H H H H	H H H H
0	0	0	1	0	0	L H H H	L H H H
0	1	0	1	0	0	H L H H	H L H H
1	0	0	1	0	0	H H L H	H H L H
1	1	0	1	0	0	H H H L	H H H L
X	X	X	0	X	1	H H H H	H H H H

END.

Figure 3-4. AmPAL16H8 PLPL Specification—Dual 2-to-4 Multiplexers

Gray Code to Decimal Demultiplexer (AmPAL22V10)

This design implements an excess-3-gray-to-decimal decoder (see Table 3-4). It is functionally equivalent to '44A and 'L44 MSI circuits and can be implemented in a single AmPAL22V10 device. The corresponding PLPL specification and the sum-of-products equations are shown in Figure 3-5.

TABLE 3-4. FUNCTION TABLE FOR GRAY-CODE-TO-DECIMAL DEMULTIPLEXER

No.	Excess-3-Gray-Input				Decimal Outputs									
	D	C	B	A	y0	y1	y2	y3	y4	y5	y6	y7	y8	y9
0	L	L	H	L	L	H	H	H	H	H	H	H	H	H
1	L	H	H	L	H	L	H	H	H	H	H	H	H	H
2	L	H	H	H	H	H	L	H	H	H	H	H	H	H
3	L	H	L	H	H	H	H	L	H	H	H	H	H	H
4	L	H	L	L	H	H	H	H	L	H	H	H	H	H
5	H	H	L	L	H	H	H	H	H	L	H	H	H	H
6	H	H	L	H	H	H	H	H	H	H	L	H	H	H
7	H	H	H	H	H	H	H	H	H	H	H	L	H	H
8	H	H	H	L	H	H	H	H	H	H	H	H	L	H
9	H	L	H	L	H	H	H	H	H	H	H	H	H	L
I	H	H	H	H	H	H	H	H	H	H	H	H	H	H
N	H	L	H	H	H	H	H	H	H	H	H	H	H	H
V	H	L	L	H	H	H	H	H	H	H	H	H	H	H
A	H	L	L	L	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	L	H	H	H	H	H	H	H	H	H
I	L	L	L	H	H	H	H	H	H	H	H	H	H	H
D	L	L	H	H	H	H	H	H	H	H	H	H	H	H

DEVICE GRAY_CODE_TO_10_DEMULTIPLXER (PAL22V10)

PIN A = 4 B = 3 C = 2 D = 1
Y[0:9] = 14:23 ;

BEGIN

CASE (D,C,B,A)

BEGIN

```
#B0010) Y[0:9] = #B011111111 ;
#B0110) Y[0:9] = #B101111111 ;
#B0111) Y[0:9] = #B110111111 ;
#B0101) Y[0:9] = #B111011111 ;
#B0100) Y[0:9] = #B111101111 ;
#B1100) Y[0:9] = #B111110111 ;
#B1101) Y[0:9] = #B111111011 ;
#B1111) Y[0:9] = #B111111101 ;
#B1110) Y[0:9] = #B111111110 ;
#B1010) Y[0:9] = #B111111110 ;
#B0000) Y[0:9] = #B111111111 ;
#B0001) Y[0:9] = #B111111111 ;
#B0011) Y[0:9] = #B111111111 ;
#B1000) Y[0:9] = #B111111111 ;
#B1001) Y[0:9] = #B111111111 ;
#B1011) Y[0:9] = #B111111111 ;
```

END ;

END.

TEST_VECTORS

IN D C B A ;
OUT Y[0:9] ;

BEGIN

```
"D C B A Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 Y9"
0 0 1 0 L H H H H H H H H H ;
0 1 1 0 H L H H H H H H H H ;
0 1 1 1 H H L H H H H H H H ;
0 1 0 1 H H H L H H H H H H ;
0 1 0 0 H H H H L H H H H H ;
1 1 0 0 H H H H H L H H H H ;
1 1 0 1 H H H H H H L H H H ;
1 1 1 1 H H H H H H H L H H ;
1 1 1 0 H H H H H H H H L H ;
1 0 1 0 H H H H H H H H H L ;
0 0 0 X H H H H H H H H H H ;
1 0 0 X H H H H H H H H H H ;
X 0 1 1 H H H H H H H H H H ;
```

END.

Figure 3-5. AmPAL22V10 PLPL Specification—Gray Code to Decimal Demultiplexer

3.2.3 ENCODERS/DECODERS

A binary code of n bits is capable of representing up to 2^n distinct elements of coded information. A decoder is a combinatorial circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit decoded information has unused or don't-care combinations, the decoder output will have less than 2^n outputs.

An encoder is a digital function that produces a reverse operation from that of a decoder. An encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code for the 2^n variables.

A priority encoder establishes an input priority to ensure that only the highest-priority input line is encoded.

16-to-4 Priority Encoder (AmPAL22V10)

The 16-to-4 priority encoder requires sixteen inputs and four outputs, for a total of twenty input/output pins. Each output needs eight product terms. These requirements are satisfied by a single AmPAL22V10 device. The function table (Table 3-5), PLPL specification and sum-of-products equations generated by the compiler are shown in Figure 3-6.

TABLE 3-5. FUNCTION TABLE FOR 16-to-4 PRIORITY ENCODER

Inputs																Outputs			
I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	103	102	101	100
X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	H	H	H	H
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	L	L	L	L
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	L	L	L
X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	L	L	H
X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	L	L	H
X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	L	H	L
X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	L	H	L
X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	L	H	L
X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	1	L	H	L
X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	1	1	L	H	L
X	X	X	X	X	X	0	1	1	1	1	1	1	1	1	1	1	L	H	L
X	X	X	X	X	0	1	1	1	1	1	1	1	1	1	1	1	L	H	L
X	X	X	X	0	1	1	1	1	1	1	1	1	1	1	1	1	L	H	L
X	X	X	0	1	1	1	1	1	1	1	1	1	1	1	1	1	L	H	L
X	X	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	L	H	L
X	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	L	H	L

DEVICE PRIORITY_ENCODER_16_TO_4 (PAL22V10)

"ENCODE 16 DATA LINES DECIMAL INTO 4 LINE BINARY"

```

PIN    I1 = 1   I2 = 2   I3 = 3   I4 = 4   I5 = 5
        I6 = 6   I7 = 7   I8 = 8   I9 = 9   I10 = 10
        I11 = 11  I12 = 13  I13 = 14  I14 = 15
        I15 = 16   I0 = 23
        /O[3:0] = 17:20 ;

BEGIN
  IF (I1*I2*I3*I4*I5*I6*I7*I8*I9*I10*I11*I12*I13*I14*I15) THEN
    O[3:0] = #80000 ;
  IF (/I15) THEN O[3:0] = 15 ;
  IF (I15*/I14) THEN O[3:0] = 14 ;
  IF (I15*I14*/I13) THEN O[3:0] = 13 ;
  IF (I15*I14*I13*/I12) THEN O[3:0] = 12 ;
  IF (I15*I14*I13*I12*/I11) THEN O[3:0] = 11 ;
  IF (I15*I14*I13*I12*I11*/I10) THEN O[3:0] = 10 ;
  IF (I15*I14*I13*I12*I11*I10*/I9) THEN O[3:0] = 9 ;
  IF (I15*I14*I13*I12*I11*I10*I9*/I8) THEN O[3:0] = 8 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*/I7) THEN O[3:0] = 7 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*/I6) THEN O[3:0] = 6 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*I6*/I5) THEN
    O[3:0] = 5 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*I6*I5*/I4) THEN
    O[3:0] = 4 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*I6*I5*I4*/I3) THEN
    O[3:0] = 3 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*I6*I5*I4*I3*/I2) THEN
    O[3:0] = 2 ;
  IF (I15*I14*I13*I12*I11*I10*I9*I8*I7*I6*I5*I4*I3*I2*/I1) THEN
    O[3:0] = 1 ;
END.
```

TEST_VECTORS

```

IN      I0 I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15 ;
OUT     /O[3:0] ;
```

```

BEGIN
  "I 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0"
  "O 1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 3 2 1 0"
  "                                0 1 2 3 4 5"
  "                                "
  "X 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H H;
  "X X X X X X X X X X X X X X X 0 L L L L;
  "X X X X X X X X X X X X X X 0 1 L L L H;
  "X X X X X X X X X X X X X 0 1 1 L L H L;
  "X X X X X X X X X X X X 0 1 1 1 L L H H;
  "X X X X X X X X X X X 0 1 1 1 1 L H L L;
  "X X X X X X X X X 0 1 1 1 1 1 L H L H;
  "X X X X X X X X 0 1 1 1 1 1 1 L H H L;
  "X X X X X X X 0 1 1 1 1 1 1 1 L H L L;
  "X X X X X X 0 1 1 1 1 1 1 1 1 H L L L H;
  "X X X X X 0 1 1 1 1 1 1 1 1 1 H L H L;
  "X X X X 0 1 1 1 1 1 1 1 1 1 1 H L H H;
  "X X X 0 1 1 1 1 1 1 1 1 1 1 1 H L L L;
  "X X 0 1 1 1 1 1 1 1 1 1 1 1 1 H H L H;
  "X 0 1 1 1 1 1 1 1 1 1 1 1 1 1 H H H L;
  "
END.
```

Figure 3-6. AmPAL22V10 PLPL Specification—16-to-4 Encoder

GCR (4B-5B) Encoder/Decoder

One of the more common logic functions performed on serial data is the data encode/decode function. Usually it is desirable to map (encode) the logical bit stream to a physical bit stream, adjusting for the peculiarities of the particular transmission or storage media.

Noise, bandwidth, and reliability considerations may mean that a different data format would be desirable when data is sent along to or stored on a given media. For example, group-coded recording (GCR) formats take a given number of data bits and encode them with a larger number of bits. A 4B-5B

GCR code would take 4 data bits and encode them into 16 states with 5 new bits. A particular 4B-5B code is shown in Table 3-6.

This mapping allows at most two zeros to occur in succession. Also note that data combinations with more than one zero at the beginning and end of the word are excluded. This is necessary to insure that when data words are serialized, no more than two zeros occur in succession at any point in the bit stream. Finally, the data combination 11111 is reserved as a synchronization mark. In tape systems, this results in increased bit density and eases clock synchronization.

TABLE 3-6. 4B-5B CODE

4B-5B Code	
4-Bit Data	5-Bit Data
0 0 0 0	1 1 0 0 1
0 0 0 1	1 1 0 1 1
0 0 1 0	1 0 0 1 0
0 0 1 1	1 0 0 1 1
0 1 0 0	1 1 1 0 1
0 1 0 1	1 0 1 0 1
0 1 1 0	1 0 1 1 0
0 1 1 1	1 0 1 1 1
1 0 0 0	1 1 0 1 0
1 0 0 1	0 1 0 0 1
1 0 1 0	0 1 0 1 0
1 0 1 1	0 1 0 1 1
1 1 0 0	1 1 1 1 0
1 1 0 1	0 1 1 0 1
1 1 1 0	0 1 1 1 0
1 1 1 1	0 1 1 1 1

03862A-93

The system diagram in Figure 3-7 shows how the GCR Encoder/Decoder (GCR E/D) interfaces to a tape drive and tape controller. Parallel input data is given to the GCR E/D, converted to the 5-bit format, serialized, and written to the tape. On a read, the serial data from the tape is parallelized, converted back to the 4-bit format and output to the output data bus. Additionally, during a read, two status signals are developed. The first signal, $\overline{INV}$, indicates the presence of an invalid input, i.e., too many zeros in succession. The second status signal, $\overline{H}$, indicates the detection of the synchronization mark (11111).

The operation modes for the GCR E/D are shown in the Data-Flow Diagrams of Figure 3-8. The control signal definition and operation functions are indicated for each operation mode. In particular, the data flow between each bit of the output register is indicated schematically.

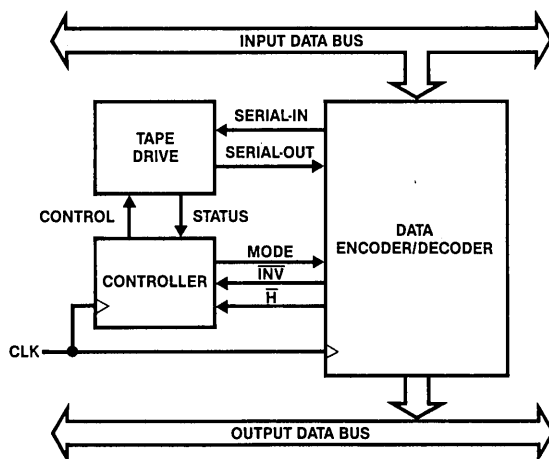
The first mode of operation of the GCR E/D is the HOLD mode. When $\overline{ENABLE}$ is HIGH, all data operations on the output register are disabled, independent of the two mode controls, M_1 and M_0 . The output data is simply fed-back to the register inputs. Thus the register content is retained after the clock transition.

When the $\overline{ENABLE}$ input is LOW, the operations indicated by the M_1 and M_0 mode bits are executed on the clock transition. When M_1 and M_0 are both LOW, the SERIAL SHIFT IN mode is selected. In this mode the output register is configured as a serial shift register. The serial input is consecutively shifted into the register until all 5 bits from the tape have been stored, MSB at Y_3 and LSB at SERIAL OUT.

The CONVERT SERIAL INPUT AND LOAD operation is selected when $\overline{ENABLE}$ is LOW, M_1 is HIGH and M_0 is LOW. After the 5 bits of data have been serialized by the SERIAL SHIFT IN instruction, the 5B code must be converted to a 4B code. This is accomplished by taking the outputs of the 5 register bits and converting them to 4 bits with combinatorial logic. On the clock transition, the result is loaded into the Y register. On the same clock transition that loads the converted data into the Y register, the serial input is loaded into the serial output register. Because the serial data is being read continuously, one data bit per clock transition, the conversion must be done without missing a serial data bit.

The CONVERT PARALLEL INPUT AND LOAD operation is selected when $\overline{ENABLE}$ is LOW, M_1 is HIGH and M_0 is HIGH. This mode takes the 4 input data bits and converts them to the 5 bit representation. The result is loaded into the output register on the clock transition. The LSB of the 5B representation is loaded into the Y_3 bit of the output register and the MSB is loaded into the serial output bit. This configuration, in conjunction with the next instruction, allows the serial data to be written to the tape drive one bit per clock transition.

The final operation, SERIAL SHIFT OUT, is selected when $\overline{ENABLE}$ is LOW, M_1 is LOW and M_0 is HIGH. After the CONVERT PARALLEL INPUT AND LOAD operation is executed, the SERIAL SHIFT OUT operation outputs the converted data to the tape drive. A series of one convert operation followed by 4 shift operations will transfer a sequence of 5-bits to the tape drive, one bit per clock cycle.



03862A-94

Figure 3-7. Typical Tape Storage System

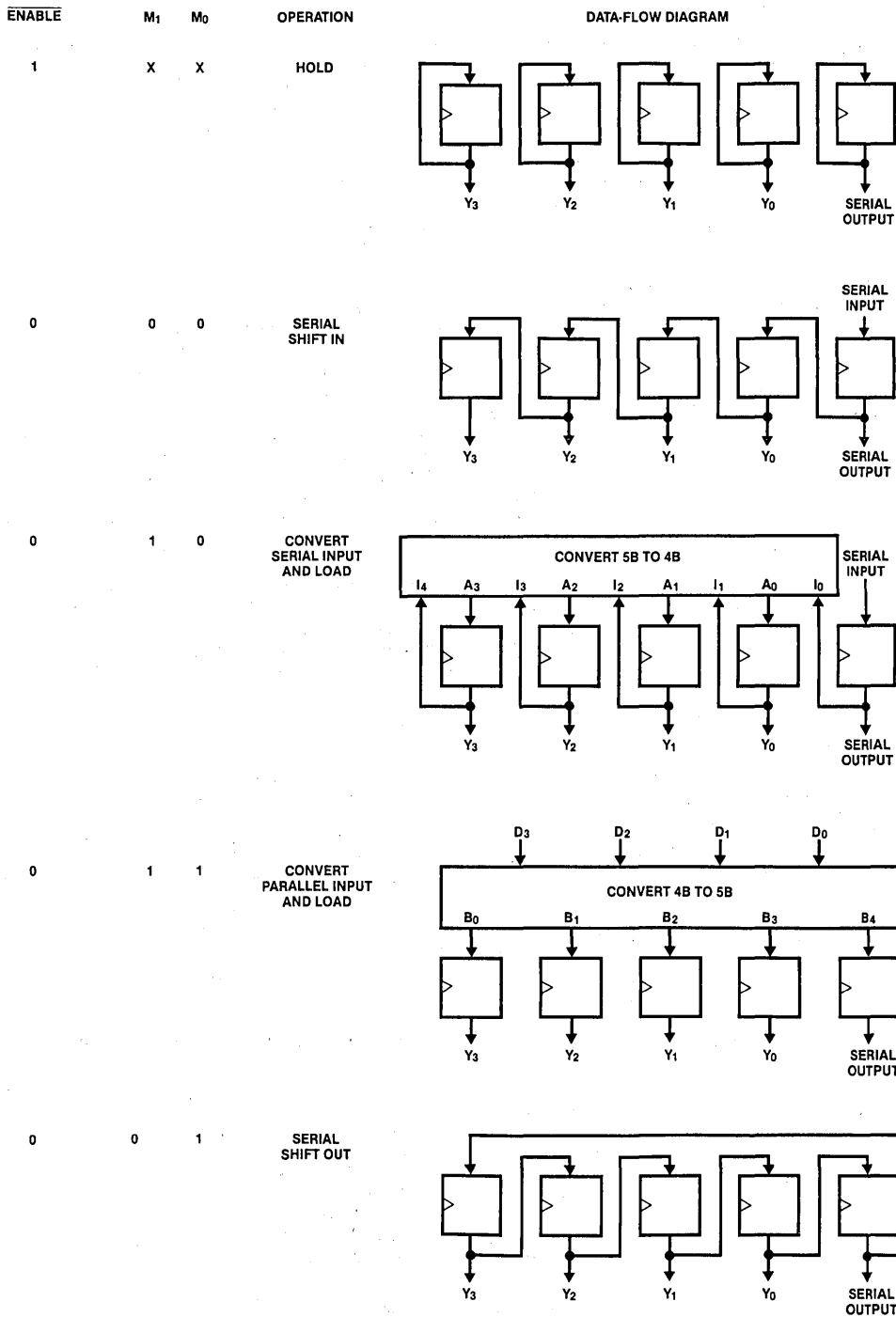


Figure 3-8. GCR E/D Mode Definitions

03862A-95

Design Approach (AmPAL16R6)

The PAL implementation of the GCR Encoder/Decoder takes advantage of the multiplexer-like structure of the AND-OR array. Each valid combination of $\overline{\text{ENABLE}}$, M_1 and M_0 selects a different set of AND terms. In some cases, only one term is selected (in data steering operations for example). In other cases, multiple AND terms are selected to implement a combinatorial logic function (the 5B-to-4B conversion for example). This concept, using the control inputs to enable one or more AND terms, allows the direct implementation of the PAL design from the mode Data-Flow Diagrams (with a little Karnaugh map help). The K-Maps for the 5B-to-4B conversion logic and the 4B-to-5B conversion logic for the Y_3 output are shown in Figures 3-9 and 3-10. Given these maps and the flow diagrams in Figure 3-8, the Boolean equations can be constructed for the Y_3 output. The resulting equation, in PALASM format, is shown in Figure 3-11.

It is important to note that the equation in Figure 3-11 is written for the inverse of the Y_3 output ($\overline{Y_3}$). This is necessary if true data is desired on the output pin because of the inverting nature of the output buffer on the PAL. The inverted form of the equation is easily implemented by selecting the negative version of the data ($\overline{Y_3}$ in the hold operation for example) or by grouping zeros in a combinatorial logic function (see Figures 3-9 and 3-10). Notice that the multiplexer strategy works equally well for active-LOW or active-HIGH logic functions.

Once the transformation of the Data-Flow Diagrams and K-Maps to Boolean equations is understood, the interested reader should be able to construct K-Maps for the other Y outputs and, in conjunction with the Data-Flow Diagrams of

Figure 3-8, write the PALASM equations for the resulting logic functions. This exercise will help the reader to fully appreciate the advantages of the Data-Flow Diagram/Multiplexer method of PAL design. Consult the full PALASM listing (Figures 3-15 and 3-16) for the complete solutions.

It is important to note that the diagrams and equations in Figures 3-12, 3-13, and 3-14 specify the true output for invalid signal ($\overline{\text{INV}}$), rather than $\overline{\text{INV}}$ which appears in the system diagram of Figure 3-7. This is necessary if the correct data is desired on the output pin and is due to the inverting nature of the output buffer on the PAL device.

Once the data portion of the Encoder/Decoder is completed, only the two status outputs, $\overline{H}$ and $\overline{\text{INV}}$, need to be implemented. $\overline{H}$ indicates the synchronization mark (1111) has been detected and is simply an AND of Y_3 through S_{OUT} . $\overline{\text{INV}}$ indicates an invalid serial input was received.

The $\overline{\text{INV}}$ signal is registered and held until the clear $\overline{\text{INV}}$ flag input (CIF) is brought LOW, deactivating the flag. Only during a 5B-4B conversion operation ($M_1 = \text{HIGH}$, $M_0 = \text{LOW}$) is the $\overline{\text{INV}}$ flag activated. Figures 3-12 and 3-13 show the $\overline{\text{INV}}$ flag mode definitions and the intermediate INVALID logic equation respectively.

In this case, an active-LOW output is desired so the active-HIGH form of the $\overline{\text{INV}}$ signal is developed internally. Ones are grouped in the intermediate combinatorial logic function (INVALID) and the true version of the data is selected. The complete PALASM equation for $\overline{\text{INV}}$ is given in Figure 3-14.

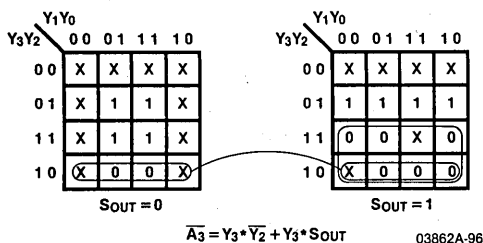


Figure 3-9. 5B-to-4B Conversion K-Map for Y_3 Output

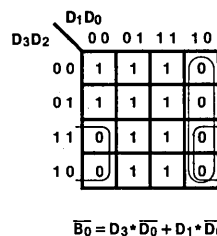


Figure 3-10. 4B-to-5B Conversion K-Map for Y_3 Output

```

 $\overline{Y_3} :=$  EN *  $\overline{Y_3}$ 
      EN *  $\overline{M_1} \cdot \overline{M_0} \cdot \overline{Y_2}$ 
      EN *  $\overline{M_1} \cdot M_0 \cdot S_{\text{OUT}}$ 
      EN *  $M_1 \cdot \overline{M_0} \cdot Y_3 \cdot S_{\text{OUT}}$ 
      EN *  $M_1 \cdot \overline{M_0} \cdot Y_3 \cdot Y_2$ 
      EN *  $M_1 \cdot M_0 \cdot D_3 \cdot \overline{D_0}$ 
      EN *  $M_1 \cdot M_0 \cdot D_1 \cdot \overline{D_0}$ 
+ ;HOLD
+ ;SERIAL SHIFT IN
+ ;SERIAL SHIFT OUT
+ ;CONVERT SERIAL
+ ;INPUT AND LOAD
+ ;CONVERT PARALLEL
+ ;INPUT AND LOAD

```

Figure 3-11. PALASM Equation for Y_3

03862A-98

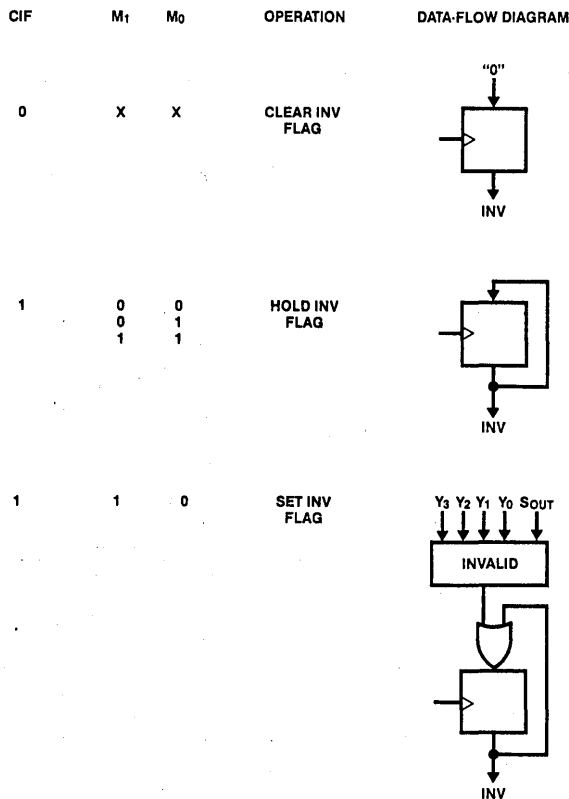


Figure 3-12. INV Flag Mode Definitions 03862A-99

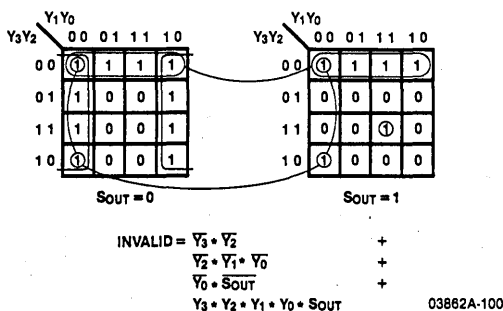


Figure 3-13. PALASM Equation for INVALID

$$\begin{aligned}
 \text{INV} &:= \overline{\text{CIF}} \cdot \text{INV} && \text{;HOLD INV FLAG} \\
 &\quad \overline{\text{CIF}} \cdot M_1 \cdot \overline{M_0} \cdot \overline{Y_3} \cdot \overline{Y_2} && + \text{;SET INV FLAG IF INVALID IS TRUE} \\
 &\quad \overline{\text{CIF}} \cdot M_1 \cdot \overline{M_0} \cdot \overline{Y_0} \cdot \text{SOUT} && + \\
 &\quad \overline{\text{CIF}} \cdot M_1 \cdot \overline{M_0} \cdot \overline{Y_2} \cdot \overline{Y_1} \cdot \overline{Y_0} && + \\
 &\quad \overline{\text{CIF}} \cdot M_1 \cdot \overline{M_0} \cdot Y_3 \cdot Y_2 \cdot Y_1 \cdot Y_0 \cdot \text{SOUT}
 \end{aligned}$$

Figure 3-14. PALASM Equation for INV

03862A-101

TABLE 3-7. FUNCTION TABLE FOR GCR (4B-5B) ENCODER/DECODER

FUNCTION TABLE

CK	/E	/EN	M1	M0	D3	D2	D1	D0	SIN	/CIF	Y3	Y2	Y1	Y0	SOUT	/INV	/H
; HIGH Z TEST																	
X	H	X	X	X	X	X	X	X	X	X	Z	Z	Z	Z	Z	Z	X
; INITIALIZE /INV																	
C	L	X	X	X	X	X	X	X	X	L	X	X	X	X	X	H	X
; SERIAL SHIFT IN TEST																	
C	L	H	H	H	H	H	H	H	X	H	H	H	H	H	L	H	H
C	L	H	L	L	X	X	X	X	H	H	H	H	H	L	H	H	H
C	L	H	L	L	X	X	X	X	H	H	H	H	L	H	H	H	H
C	L	H	L	L	X	X	X	X	H	H	H	L	H	H	H	H	H
C	L	H	L	L	X	X	X	X	H	H	L	H	H	H	H	H	H
C	L	H	L	L	X	X	X	X	H	H	H	H	H	H	H	H	L
; SYNCHRONIZATION MARK DETECT TEST																	
C	L	H	H	H	L	L	L	L	X	H	H	L	L	H	H	H	H
; 4B TO 5B CONVERSION FOLLOWED BY 5B TO 4B RECONVERSION																	
; NOTE THAT DURING RECONVERSION THE BIT ORDER IS REVERSED																	
C	L	H	H	L	X	X	X	X	L	H	L	L	H	H	L	H	H
C	L	H	H	H	L	L	L	H	X	H	H	H	L	H	H	H	H
C	L	H	H	L	X	X	X	X	L	H	L	L	H	H	L	H	H
C	L	H	H	H	L	L	H	L	X	H	L	H	L	L	H	H	H
C	L	H	H	L	X	X	X	X	L	H	H	L	L	H	L	H	H
C	L	H	H	H	L	L	H	H	X	H	H	H	L	L	H	H	H
C	L	H	H	L	X	X	X	X	L	H	L	L	L	L	H	H	H
C	L	H	H	H	L	H	L	L	X	H	H	L	H	H	H	H	H
C	L	H	H	H	L	L	H	H	X	H	H	L	L	H	L	H	H
C	L	H	H	L	X	X	X	X	L	H	L	L	L	L	H	H	H
C	L	H	H	H	H	L	L	L	X	H	L	H	L	H	H	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	H	H	H
C	L	H	H	H	H	H	H	L	X	H	L	H	H	H	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	L	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H
C	L	H	H	L	X	X	X	X	L	H	H	H	H	L	L	H	H
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	L	H	H

TABLE 3-7. FUNCTION TABLE FOR GCR (4B-5B) ENCODER/DECODER (Continued)

; SERIAL SHIFT IN TEST																
C	L	H	H	H	H	H	H	H	X	H	H	H	H	L	H	H
C	L	H	L	H	X	X	X	X	X	H	L	H	H	H	H	H
C	L	H	L	H	X	X	X	X	X	H	H	L	H	H	H	H
C	L	H	L	H	X	X	X	X	X	H	H	H	L	H	H	H
C	L	H	L	H	X	X	X	X	X	H	H	H	H	L	H	H
C	L	H	L	H	X	X	X	X	X	H	L	H	H	H	H	H

DESCRIPTION

THIS PART IMPLEMENTS A 4B 5B ENCODER/DECODER FOR TAPE DRIVES. ON A WRITE IT ENCODES THE 4B INPUT DATA TO THE 5B FORMAT AND SERIALIZES THE DATA. ON A READ THE 5B DATA IS SHIFTED IN, RECONVERTED TO THE 4B FORMAT, AND OUTPUT TO THE DATA BUS.

PAL16R6

PAT003

4B-5B ENCODER/DECODER

ADVANCED MICRO DEVICES

CK M1 MO D3 D2 D1 DO /EN /CIF GND

/E SIN /INV Y0 Y1 Y2 Y3 SOUT /H VCC

PAL DESIGN SPECIFICATION

WARREN K. MILLER 2/15/82

```

/SOUT := EN*/SOUT          +      ;HOLD
        /EN*/M1*/MO*/SIN    +      ;SERIAL SHIFT IN
        /EN*/M1* MO*/Y0     +      ;SERIAL SHIFT OUT
        /EN* M1*/MO*/SIN    +      ;CONVERT SERIAL INPUT AND LOAD
        /EN* M1* MO* D3* D1  +      ;CONVERT PARALLEL INPUT AND LOAD
        /EN* M1* MO* D3* DO  +      ;
;

/YO := EN*/Y0              +
      /EN*/M1*/MO*/SOUT    +
      /EN*/M1* MO*/Y1      +
      /EN* M1*/MO*/SOUT    +
      /EN* M1*/MO* Y3* Y2*/YO +
      /EN* M1* MO*/D3* D1  +
      /EN* M1* MO*/D3* D2* DO

/Y1 := EN*/Y1              +
      /EN*/M1*/MO*/Y0      +
      /EN*/M1* MO*/Y2      +
      /EN* M1*/MO*/Y0      +
      /EN* M1*/MO* Y3* Y2  +
      /EN* M1* MO*/D2

/Y2 := EN*/Y2              +
      /EN*/M1*/MO*/Y1      +
      /EN*/M1* MO*/Y3      +
      /EN* M1*/MO*/Y1      +
      /EN* M1* MO*/D3*/D1*/DO +
      /EN* M1* MO*/D3* D2*/D1 +
      /EN* M1* MO* D3*/D1* DO

/Y3 := EN*/Y3              +
      /EN*/M1*/MO*/Y2      +
      /EN*/M1* MO*/SOUT    +
      /EN* M1*/MO* Y3* SOUT +
      /EN* M1*/MO* Y3*/Y2  +
      /EN* M1* MO* D3*/DO   +
      /EN* M1* MO* D1*/DO

INV := /CIF* INV           +      ;HOLD INV FLAG
      /CIF* M1*/MO*/Y3*/Y2 +      ;SET INV FLAG IF INVALID TRUE
      /CIF* M1*/MO*/Y2*/Y1*/YO +
      /CIF* M1*/MO*/Y0*/SOUT +
      /CIF* M1*/MO* Y3* Y2* Y1* YO* SOUT

H    = Y3* Y2* Y1* YO* SOUT

```

Figure 3-15. PALASM Listing (pg. 1 of 3)

PAL16R6

PAT003

4B-5B ENCODER/DECODER

ADVANCED MICRO DEVICES

*D9724

FO

PAL DESIGN SPECIFICATION

WARREN K. MILLER 2/15/82

```
L0000 1111 1111 1111 1111 1111 1111 1111 1111 *
L0032 1111 1101 1101 1101 1101 1101 1111 1111 *
L0256 1111 1110 1111 1111 1111 1111 1011 1111 *
L0288 1011 1011 1111 1111 1111 1111 0111 1110 *
L0320 1011 0111 1111 1111 1111 1110 0111 1111 *
L0352 0111 1011 1111 1111 1111 1111 0111 1110 *
L0384 0111 0111 0111 1111 0111 1111 0111 1111 *
L0416 0111 0111 0111 1111 1111 0111 0111 1111 *
L0512 1111 1111 1110 1111 1111 1111 1011 1111 *
L0544 1011 1011 1111 1110 1111 1111 0111 1111 *
L0576 1011 0110 1111 1111 1111 1111 0111 1111 *
L0608 0111 1001 1101 1111 1111 1111 0111 1111 *
L0640 0111 1011 1101 1110 1111 1111 0111 1111 *
L0672 0111 0111 0111 1111 1111 1011 0111 1111 *
L0704 0111 0111 1111 1111 0111 1011 0111 1111 *
L0768 1111 1111 1111 1110 1111 1111 1011 1111 *
L0800 1011 1011 1111 1111 1110 1111 0111 1111 *
L0832 1011 0111 1110 1111 1111 1111 0111 1111 *
L0864 0111 1011 1111 1111 1110 1111 0111 1111 *
L0896 0111 0111 1011 1111 1011 1011 0111 1111 *
L0928 0111 0111 1011 0111 1011 1111 0111 1111 *
L0960 0111 0111 0111 1111 1011 0111 0111 1111 *
L1024 1111 1111 1111 1111 1110 1111 1011 1111 *
L1056 1011 1011 1111 1111 1111 1110 0111 1111 *
L1088 1011 0111 1111 1110 1111 1111 0111 1111 *
L1120 0111 1011 1111 1111 1111 1110 0111 1111 *
L1152 0111 1011 1101 1101 1111 1111 0111 1111 *
L1184 0111 0111 1111 1011 1111 1111 0111 1111 *
L1280 1111 1111 1111 1111 1111 1110 1011 1111 *
L1312 1011 1010 1111 1111 1111 1111 0111 1111 *
L1344 1011 0111 1111 1111 1110 1111 0111 1111 *
L1376 0111 1010 1111 1111 1111 1111 0111 1111 *
L1408 0111 1011 1101 1101 1111 1110 0111 1111 *
L1440 0111 0111 1011 1111 0111 1111 0111 1111 *
L1472 0111 0111 1011 0111 1111 0111 0111 1111 *
L1536 1111 1111 1111 1111 1111 1111 1110 0111 *
L1568 0111 1011 1110 1110 1111 1111 1111 0111 *
L1600 0111 1011 1111 1110 1110 1110 1111 0111 *
L1632 0111 1010 1111 1111 1111 1110 1111 0111 *
L1664 0111 1001 1101 1101 1101 1101 1111 0111 *
C8E23*
V0001 XXXXXXXX01XZZZZZX1 *
V0002 CXXXXXX000XHXXXXXX1 *
V0003 C1111111100XHHHHHLH1 *
V0004 COOXXXX11001HLHHHHH1 *
V0005 COOXXXX11001HHLHHHH1 *
V0006 COOXXXX11001HHHLHHH1 *
V0007 COOXXXX11001HHHHLHH1 *
V0008 COOXXXX11001HHHHHHL1 *
```

3

Figure 3-15. PALASM Listing (pg. 2 of 3)

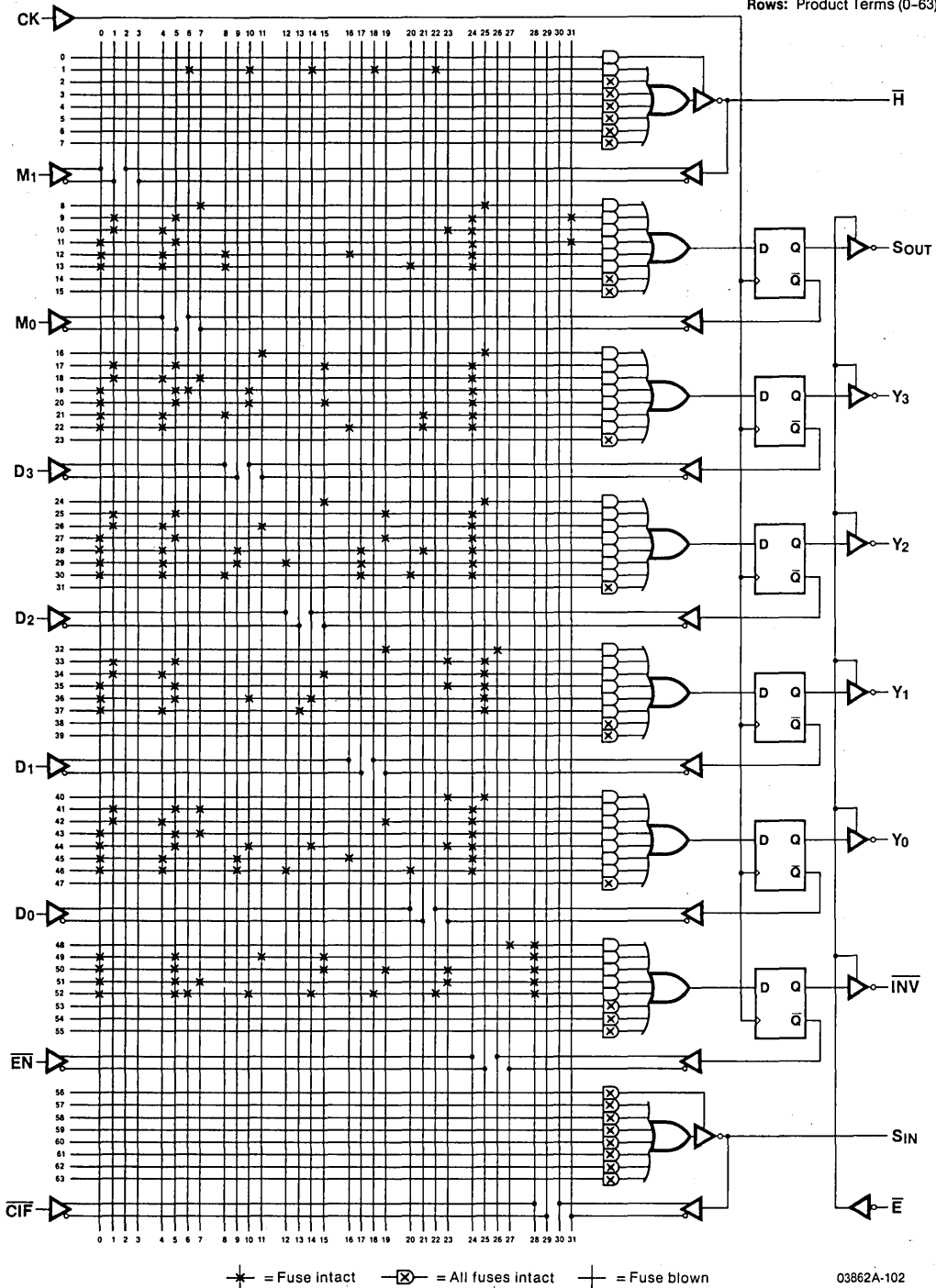
```

V0009 C1100001100XHHLLHHH1 *
V0010 C10XXXX11000HHHLLHL1 *
V0011 C1100011100XHHLLHHH1 *
V0012 C10XXXX11000HLLLLHL1 *
V0013 C1100101100XHLHLHLH1 *
V0014 C10XXXX11000HHLLHLH1 *
V0015 C1100111100XHLHLHHH1 *
V0016 C10XXXX11000HLLLLHL1 *
V0017 C1101001100XHHHLHHH1 *
V0018 C10XXXX11000HHHLLHL1 *
V0019 C1101011100XHLHLHHH1 *
V0020 C10XXXX11000HHLHLHL1 *
V0021 C1101101100XHLHLHLH1 *
V0022 C10XXXX11000HHLHLHL1 *
V0023 C1101111100XHLHHHHH1 *
V0024 C10XXXX11000HLHLHLH1 *
V0025 C1110001100XHHHLHLH1 *
V0026 C10XXXX11000HHLHLHL1 *
V0027 C1110011100XHHLLHLH1 *
V0028 C10XXXX11000HLHLLHL1 *
V0029 C1110101100XHHHLHLH1 *
V0030 C10XXXX11000HLHLHLH1 *
V0031 C1110111100XHHHLHLH1 *
V0032 C10XXXX11000HLLHLHL1 *
V0033 C1111001100XHHHHLH1 *
V0034 C10XXXX11000HHHHHLH1 *
V0035 C1111011100XHHHLHLH1 *
V0036 C10XXXX11000HLHHLHL1 *
V0037 C1111101100XHHHHLH1 *
V0038 C10XXXX11000HLHHHLH1 *
V0039 C1111111100XHHHHHLH1 *
V0040 C10XXXX11000HLHLHLH1 *
V0041 C1111111100XHHHHHLH1 *
V0042 C01XXXX1100XHHHHLH1 *
V0043 C01XXXX1100XHHHLHHH1 *
V0044 C01XXXX1100XHHHLHHH1 *
V0045 C01XXXX1100XHLHHHHH1 *
V0046 C01XXXX1100XHHHHHLH1 *
V0047 C01XXXX1100XHHHHLH1 *
CA76

```

Figure 3-15. PALASM Listing (pg. 3 of 3)

Columns: Inputs (0-31)
Rows: Product Terms (0-63)



03862A-102

Figure 3-16. Logic Diagram for GCR Encodes/Decodes Using AmPAL16R6

3.2.4 COMPARATORS

The comparison of two numbers is an operation that determines if one number is equal to the other number.

6-Bit Comparator (AmPAL18P8)

The 6-bit comparator establishes when two 6-bit data strings (A0–A5 and B0–B5) are equivalent ($NE = L$). Because the maximum number of product terms for each output pin is eight, we use one additional output pin (UPNE) for the comparison of the three most significant bits ($UPNE = L$ if the upper three bits are equivalent). It requires twelve inputs and two outputs, for a total of sixteen input/output pins. These requirements are

satisfied by a single AmPAL18P8 device. The corresponding PLPL specification and the sum-of-products equations generated by the compiler are shown in Figure 3-17.

Octal Comparator (AmPAL22V10)

The octal comparator establishes when two 8-bit data strings (A0–A7 and B0–B7) are equivalent ($NE = L$). It requires sixteen inputs, and one output pin. The single output needs sixteen product terms. These requirements are satisfied by one single AmPAL22V10. The corresponding PLPL specification and sum-of-products equations generated by the compiler are shown in Figure 3-18.

DEVICE SIX_BIT_MAGNITUDE_COMPARATORS (PAL18P8)

PIN A5 = 1 A4 = 2 A3 = 3 A2 = 4 A1 = 5 A0 = 6
 B5 = 7 B4 = 8 B3 = 9 B2 = 11 B1 = 12 B0 = 13
 UPNE = 14 "THE UPPER THREE BITS NOT EQUAL"
 NE = 15 ;

BEGIN

UPNE = A5*/B5 + /A5*B5
 + A4*/B4 + /A4*B4
 + A3*/B3 + /A3*B3;

NE = UPNE
 + A2*/B2 + /A2*B2
 + A1*/B1 + /A1*B1
 + A0*/B0 + /A0*B0 ;

END.

TEST_VECTORS

IN A5 A4 A3 A2 A1 A0 B5 B4 B3 B2 B1 B0 ;
 OUT UPNE NE ;

BEGIN

"A A A A A A B B B B B B "
 "5 4 3 2 1 0 5 4 3 2 1 0 UPNE NE"

1 0 0 0 0 0 0 0 0 0 0 0 H H ;
 0 1 0 0 0 0 0 0 0 0 0 0 H H ;
 0 0 1 0 0 0 0 0 0 0 0 0 H H ;
 0 0 0 1 0 0 0 0 0 0 0 0 L H ;
 0 0 0 0 1 0 0 0 0 0 0 0 L H ;
 0 0 0 0 0 1 0 0 0 0 0 0 L H ;
 0 0 0 0 0 0 1 0 0 0 0 0 H H ;
 0 0 0 0 0 0 0 1 0 0 0 0 H H ;
 0 0 0 0 0 0 0 0 1 0 0 0 H H ;
 0 0 0 0 0 0 0 0 0 1 0 0 L H ;
 0 0 0 0 0 0 0 0 0 0 1 0 L H ;
 0 0 0 0 0 0 0 0 0 0 0 1 L H ;
 0 0 0 0 0 0 0 0 0 0 0 0 L L ;
 1 1 1 1 1 1 1 1 1 1 1 1 L L ;
 1 0 1 0 1 0 1 0 1 0 1 0 L L ;
 0 1 0 1 0 1 0 1 0 1 0 1 L L ;
 0 0 1 0 0 1 0 0 1 0 0 1 L L ;

END.

Figure 3-17. AmPAL18P8 PLPL Specification—8-Bit Comparator

DEVICE OCTAL_COMPARATOR (PAL22V10)

PIN A7 = 1 A6 = 2 A5 = 3 A4 = 4
 A3 = 5 A2 = 6 A1 = 7 A0 = 8
 B7 = 9 B6 = 10 B5 = 11 B4 = 13
 B3 = 14 B2 = 15 B1 = 16 B0 = 17
 NE = 18;

BEGIN

NE = A7*/B7 + /A7*B7
 + A6*/B6 + /A6*B6
 + A5*/B5 + /A5*B5
 + A4*/B4 + /A4*B4
 + A3*/B3 + /A3*B3
 + A2*/B2 + /A2*B2
 + A1*/B1 + /A1*B1
 + A0*/B0 + /A0*B0 ;

END.

TEST_VECTORS

IN A7 A6 A5 A4 A3 A2 A1 A0 B7 B6 B5 B4 B3 B2 B1 B0;

OUT NE;

BEGIN

"A A A A A A A A B B B B B B B B N"

"7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 E"

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 H ;

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 H ;

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 H ;

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 H ;

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 H ;

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 H ;

0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 H ;

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 H ;

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 H ;

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 H ;

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 H ;

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 H ;

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 H ;

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 H ;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 H ;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 H ;

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 L ;

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 L ;

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 L ;

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 L ;

END.

Figure 3-18. AmPAL22V10 PLPL Specification—Octal Comparator

3.2.5 ADDRESS DECODING AND CHIP SELECT GENERATION SIMPLIFIED WITH COMBINATORIAL PAL DEVICES

Combinatorial PAL devices are used extensively in digital computer systems for address decoding and chip-select generation functions. For these functions, a combinatorial PAL device with a large number of inputs/outputs and programmable polarity for its outputs is preferable. The AmpPAL18P8, a second-generation combinatorial PAL device is ideal for these applications. Compared to the standard 20-pin combinatorial PAL devices such as 16L8/16H8, it offers three significant advantages: two extra bidirectional I/Os, increased number of product terms for every output, and programmable polarity for each of its eight outputs. Its extra inputs allows it to decode a larger number of address bits in a single device. Its programmable polarity allows generation of both active-LOW and active-HIGH chip selects from the same device for different peripherals. This obviates the need for separate PAL devices with dedicated polarity (such as 16L8/16H8) and also results in faster speed.

For chip-select functions, a system designer often has to generate stable latched chip selects, and chip selects for memories without OE pins; whereas address decoders, implemented with PAL devices, can also be used to implement certain system-specific tricks such as boot-address generation, and switching between Real and Virtual modes of operation.

Stable Latched Chip-Select Generation

The address decoding function often requires latching the "decoded chip select (CS)" line. This is needed for most memory and peripheral devices. To generate this stable CS, designers often have to take into account possible hazard conditions. These hazard conditions can be handled easily by a PAL device such as the AmpPAL18P8.

Hazard Definitions and Brief Explanations

Hazards are timing problems and unwanted switching transients that arise due to gate and wiring delays. These unwanted transients may occur at the outputs of combinatorial or asynchronous sequential networks because different paths of the network may have different propagation delays.

A network is said to have a "STATIC 1 Hazard" if its output momentarily goes to 0 when it should remain at constant 1, due to some input change; and it is said to have a "STATIC 0 Hazard" if its output momentarily goes to 1 when it should remain at constant 0. A network is said to have "DYNAMIC Hazards" when its output oscillates a few times (from 1 $\rightarrow$ 0 or 0 $\rightarrow$ 1) before it returns to a stable state. In all of these cases, the steady-state output of the network is correct, but switching transients cause "glitches" to appear at the outputs. Figures 3-19.a through 3-19.c show various types of hazards.

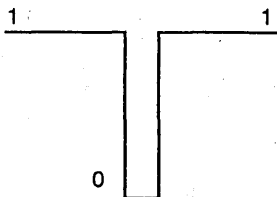


Figure 3-19.a STATIC 1 Hazard

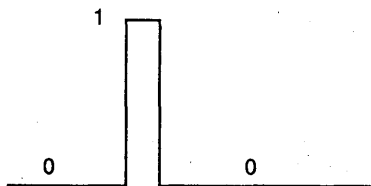


Figure 3-19.b STATIC 0 Hazard

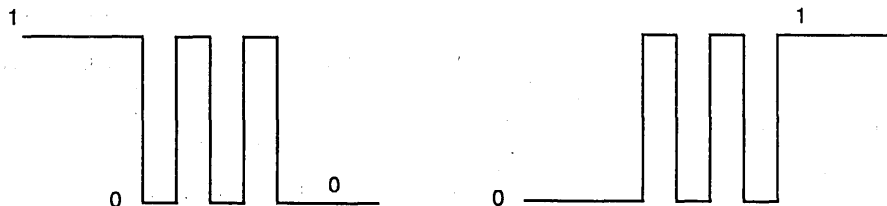


Figure 3-19.c DYNAMIC Hazards

Figure 3-20 shows the example of a network with a "STATIC 1 Hazard." Here, output Y is equal to $X_1 * X_2 + /X_2 * Y$. This is a sum-of-products equation. It can be implemented with either

NAND gates (shown in Figure 3-20), or with simple AND-OR gates in a PAL device.

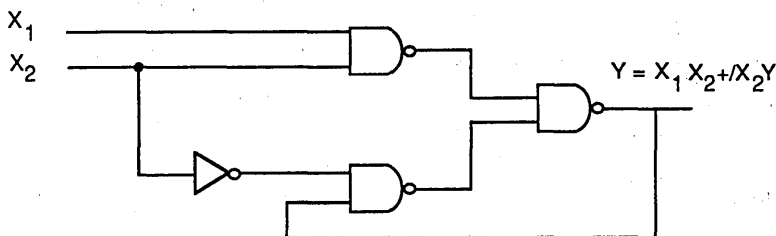


Figure 3-20. NAND Implementation of $Y = X_1 * X_2 + /X_2 * Y$

The logic in Figure 3-20 shows the sum-of-product form of design. Signal X2 can be an LE signal (usually ALE or AS in most microprocessors), Y is the feedback from the output for generating the stable CS signal. X1 can be an AND term of a number of address signals (e.g., $X_1 = A_3 * A_4 * A_5$). For simplicity sake, we would use X1 as a single signal. Figure 3-21 shows the Karnaugh Map for this function.

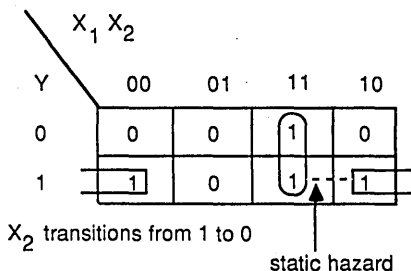


Figure 3-21. Karnaugh Map

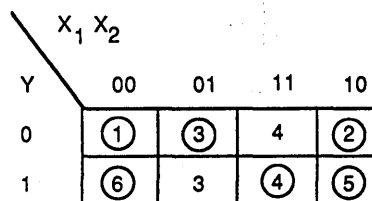
As we can see in the Karnaugh Map, there is a possibility of a hazard when signal X2 transitions.

Consider the case when initially X_1 & $Y = 1$, and X_2 is also 1. Now let's say X_2 transitions to 0. The output Y should remain a constant 1; however, the output may momentarily go to 0 if the gates switch in the following sequence: gate 1 switches to 0, then Y goes to 0, then the inverter output goes to 1, gate 2 switches to 0, and finally, Y goes to 1. Note that there is possibility of this hazard only because of different gate delays.

This kind of hazard results in momentary false outputs, and may cause serious problems if the output is used as an input

for other asynchronous networks. For example, if the signal Y is used as the CS signal for some memory or peripheral device, it is likely to result in erroneous behavior of the system because of this momentary false output. These kinds of hazards cause the network to go into wrong stable states.

Figure 3-22 shows the Karnaugh Map and its associated stable states. If the network is in stable state 4 (represented by $X_1 = 1$, $X_2 = 1$, and $Y = 1$), and if the X_2 input is changed to 0, the next stable state should be 5 and not 2. However, if Y momentarily goes to 0 and is fed back before $/X_2$ goes to 1, the output of gate 2 will remain 1 and the output might be 0.



X_2 transitions from 1 to 0

Figure 3-22. Stable States for the Function

$$Y = X_1 * X_2 + /X_2 * Y$$

Avoiding Hazard Conditions

These hazard conditions can be avoided easily with PAL devices. The hazard condition of Figure 3-22 can be eliminated by an extra AND gate as shown in Figure 3-23. This extra AND gate prevents the false output for Y. Providing this extra AND gate with a PAL device is very easy. It is simply an unused product term.

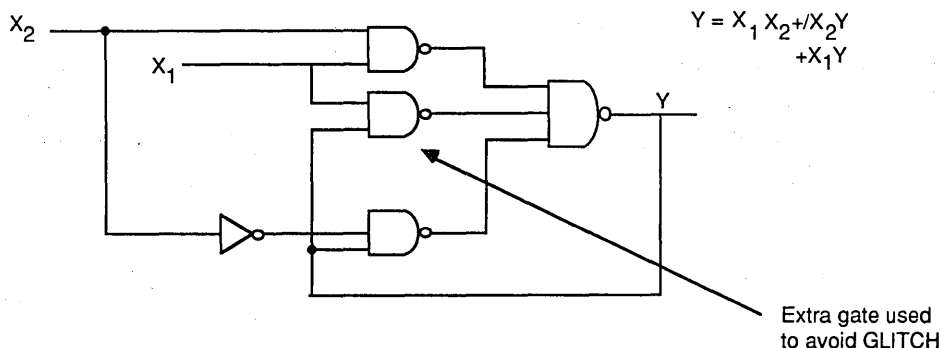


Figure 3-23. Hazard Elimination with an Extra Gate

Generating Chip Selects (CS) for Memories—With or Without Output Enable (OE)

Most of the second-generation memories have both output enable (OE) and chip select (CS) pins. This separate and independent OE line eliminates the bus-contention problem completely. The OE line is used by the microprocessor for getting data “on” or “off” the system data bus, and the microprocessor is always in control. Without this the system is always asynchronous with respect to the microprocessor/memory interface.

However, some memories do not have a separate OE pin. Generating CS for these memories could cause serious bus-contention problems in such situations. Often this problem is solved by gating the CS with READ or WRITE signals as follows:

$$CS = X \cdot /R + X \cdot /W \text{ (X is combination of certain address lines)}$$

$$WE = /W$$

This approach allows usually sufficient time for other buffers on the data bus to tristate. This works well for the read cycles in which the memory/peripheral is expected to be the only bus driver. However, for write cycles where usually the processor is the data bus driver, this doesn't quite solve the problem completely.

In the above equations, CS is generated either from the READ or WRITE line. There is still no way to ensure that CS is generated after the WE line to the peripheral/memory is asserted LOW since CS is not controlled by WE at all. This problem however can be solved by controlling the CS line from the WE signal, by gating the $/W$ signal with the WE signal (for CS) as follows:

$$CS = X \cdot /R + X \cdot /W \cdot /WE$$

This ensures that the WE of peripheral/memory is LOW before the CS is asserted. This, however, solves part of the problem.

On the other edge, WE should be disasserted only after CS is disasserted. This can be accomplished by controlling $/WE$ from the $/CS$ —by gating $/CS$ along with the $/WE$ line (as follows):

$$WE = /W \cdot /CS$$

This ensures the assertion of $/WE$ as long as CS is asserted and W is LOW. As soon as the $/CS$ is disasserted, the $/WE$ of the peripheral/memory will be disasserted.

These signals $/CS$ and $/WE$ used for controlling each other are nothing but the feedback from the appropriate I/O pins. The AmPAL18P8 provides feedback from all of its I/O pins to implement these functions. These feedbacks ensure logic safety, and provide smooth functioning of the design without using any additional components.

The PAL devices can be used also for providing flexible address-decoding functions.

Mapping Boot Address Difficulty Solved with a PAL Device

Usually single-board systems use one or two ROMs for implementing booting software. For example, a single 1-Mbit ROM provides 64K words (64K x 16) of memory.

In iAPX86 microprocessor-based systems, the bootstrap address is at the end of the address space. This can present problems in systems with only one or two ROMs.

In such cases, a slight trick in the address decoder can usually save one extra ROM chip to be placed at the boot location. During initialization, the boot-strap address can be decoded to a location in the ROM and the boot code is placed in that ROM in addition to usual software. After booting is complete, an external flip-flop (START) can be asserted to allow normal addressing of the ROM address space (see Figure 3-24).

$$CS2 = A19 * A18 * A17 * A16 * /START \quad \text{Boot Address after booting is complete} \\ + /A19 * /A18 * /A17 * /A16 * START$$

One has to be careful and not execute the boot code again. An extra product term of the PAL device allows use of the same ROM for both functions.

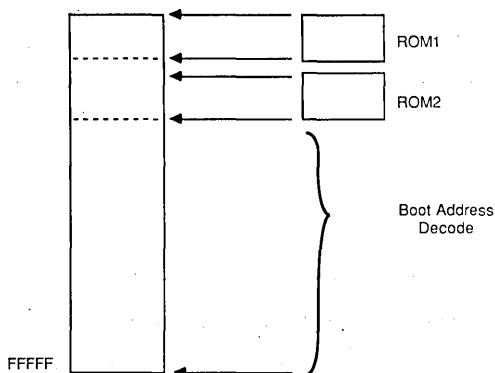


Figure 3-24. Boot Address Decoding Scheme

Switching Between Real and Virtual Modes of Operation

In the 80286 microprocessor, for maintaining upward-compatibility with the 8086/8088 Microprocessor Family, two modes of operation are provided: Real and Virtual/Protected mode.

The Real mode is compatible to iAPX 88/86 and has 1 Mbyte of physical address space: The 80286 is initialized in this Real mode with the bootstrap address of FFFF0 (A0-A19) at the bottom of this address space. Switching to the Virtual/Protected mode of operation allows for the A0-A23 address bits with 16 Mbytes of physical address space. In the Real mode, the system bootstrap ROMs must respond to the address available (with 1 Mbyte), ignoring the upper four A20-A23 address lines. In Protected mode, these same address lines address only the top 1 Mbyte of the available 16 Mbytes of address space. Thus, based on control signal PROT, the address decoder decides which address bits to use to generate CS for the same location in memory. The circuit should allow the capability of resetting to restart the chip from Real mode. The bootstrap-ROM CS can be generated as follows:

$$CS = A19 * A18 * A17 * A16 * PROT \\ + /A23 * /A22 * /A21 * /A20 * A19 * A18 * A17 * A16 * PROT *$$

$$PROT = /MODE + RESET * PROT$$

Note that MODE indicates Real mode, and /MODE indicates Virtual mode.

During START, the signal PROT initializes as LOW, which allows Real-mode address decoding. When switching to Virtual mode, the processor explicitly changes the state of a register mode to LOW.

$$MODE = H = \text{Real mode} \\ MODE = L = \text{Virtual/Protected mode}$$

This allows the PAL-based internal latch to store the state of the Virtual mode in output PROT whose feedback is used to address the Virtual mode address space (Note that this flexibility is again provided by a simple product term of the PAL device).

A similar approach can be used even for the iAPX 386. The switching between Real and Virtual/Protected mode is identical except that in Virtual mode, the addressing needs to be based on extra bits A20-A32, whereas in iAPX 286, only A20-A23 need to be implemented.

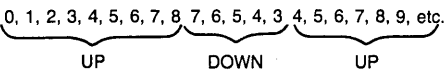
3.3 SEQUENTIAL LOGIC

3.3.1 COUNTERS

Nine-Bit Up-Down Counter

An up-down counter is also known as a bidirectional counter, and is capable of counting in either direction through a certain sequence. For example, a 4-bit up-down binary counter can count up through its sequence (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) and then can be reversed to count down (15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0).

In general, most up-down counters can be reversed at any point in their counting sequence. For example, the 4-bit binary up-down counter can be made to go through the following sequence:



Design Requirements

Figure 3-25 shows the block diagram of a 9-bit up-down counter. In addition to the nine registered outputs for the counter, there are five other inputs: one clock pin, one UP/DOWN control, one CLEAR, one Output Enable and one HOLD input.

The counter counts up, when a HIGH is asserted on the UP/DOWN control signal and clock pulses are applied. It counts down when a LOW is asserted on the UP/DOWN control signal and the clock pulses are applied. HOLD, when asserted, holds the state of the last count. CLEAR resets the entire counter to 0. Output Enable enables the value of counter on the output pins.

Design Approach

For the ease of implementation, the counter is partitioned into two separate counters: one 4-bit and the other 5-bit. The 5-bit counter is enabled every time the counter reaches the value 15 (when the counter is counting up) and value 0 (when counting down).

To minimize product-term requirements, an external pin is used to generate a "ONE" signal based on the least significant 4 bits of the counter value ($Q[3] * Q[2] * Q[1] * Q[0]$).

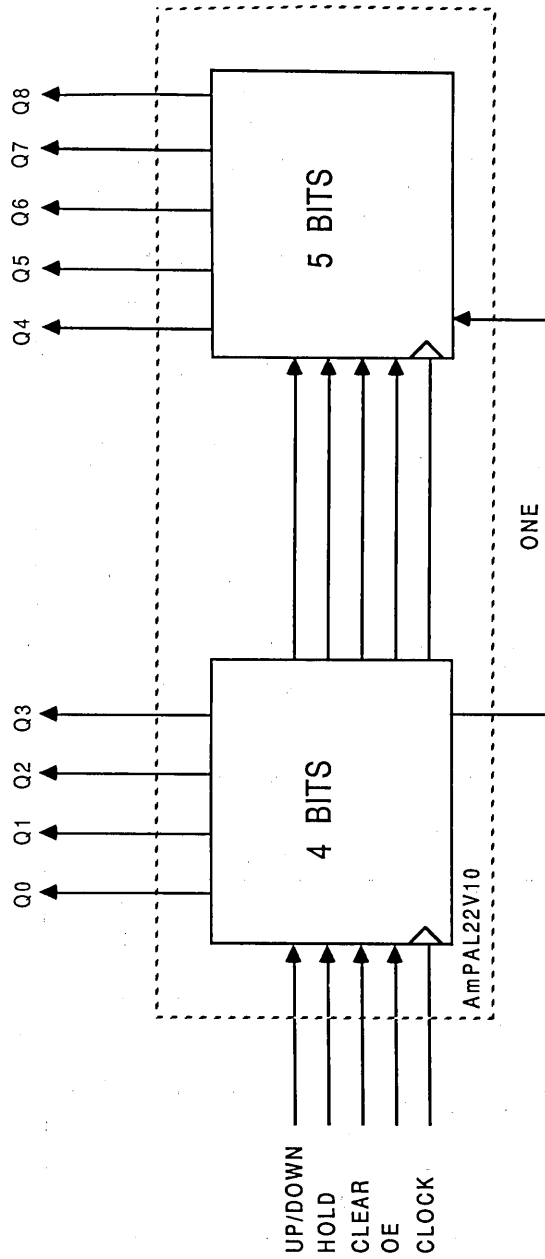
Table 3-8 shows the function table for the 9-bit up-down counter. Figure 3-26 shows the PLPL specification and Figure 3-27 shows the fuse map of a 9-bit up-down counter implemented with single AmPAL22V10 device.

As shown in the fuse map, a total of 106 product terms are required to implement this function. The most significant bit of the counter (Q8) requires 16 product terms.

TABLE 3-8. FUNCTION TABLE FOR THE 9-BIT UP-DOWN COUNTER

CLK	OE	CLR	HLD	UP/DWN	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
X	0	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z	Z
X	1	1	X	X	0	0	0	0	0	0	0	0	0
1	1	0	1	X	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
1	1	0	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	1
COUNT UP													
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1	1	1	1	0
1	1	0	0	0	1	1	1	1	1	1	1	0	1
COUNT DOWN													
1	1	0	0	0	0	0	0	0	0	0	0	0	0

OUTPUTS TRISTATED
CLEAR
HOLD
COUNT UP
COUNT DOWN



08479A-29

Figure 3-25. Block Diagram

```

DEVICE NINE_BIT_UP_DOWN_COUNTER (PAL22V10)
PIN   CLOCK = 1  UP = 2  CLEAR = 3  EN = 4
      HOLD = 5   ONE = 14
      Q[8:0] = 19,18,17,16,15,20,21,22,23 ;
DEFINE ZERO = /Q[3] * /Q[2] * /Q[1] * /Q[0] ;
      DOWN = /UP ;

" NOTE: DUE TO PRODUCT TERM LIMITATIONS, MUST GENERATE 'ONE'
SIGNAL ON A PIN TO FIT THE DESIGN IN THE PAL. 'ZERO' CAN
BE DETECTED WITH INTERNAL PRODUCT TERMS. "

```

BEGIN

```

IF ( EN ) THEN ENABLE (Q[8:0]) ;
ONE = Q[3] * Q[2] * Q[1] * Q[0] ;
IF ( CLEAR ) THEN ARESET(Q[8:0]) ;
IF ( HOLD ) THEN Q[8:0] := Q[8:0] ;
IF ( UP ) THEN BEGIN
    CASE (Q[3:0])
    BEGIN
        #B0000) Q[3:0] := #B0001 ;
        #B0001) Q[3:0] := #B0010 ;
        #B0010) Q[3:0] := #B0011 ;
        #B0011) Q[3:0] := #B0100 ;
        #B0100) Q[3:0] := #B0101 ;
        #B0101) Q[3:0] := #B0110 ;
        #B0110) Q[3:0] := #B0111 ;
        #B0111) Q[3:0] := #B1000 ;
        #B1000) Q[3:0] := #B1001 ;
        #B1001) Q[3:0] := #B1010 ;
        #B1010) Q[3:0] := #B1011 ;
        #B1011) Q[3:0] := #B1100 ;
        #B1100) Q[3:0] := #B1101 ;
        #B1101) Q[3:0] := #B1110 ;
        #B1110) Q[3:0] := #B1111 ;
        #B1111) Q[3:0] := #B0000 ;
    END ;
    IF ( ONE ) THEN BEGIN
        CASE (Q[8:4])
        BEGIN
            #B00000) Q[8:4] := #B00001 ;
            #B00001) Q[8:4] := #B00010 ;
            #B00010) Q[8:4] := #B00011 ;
            #B00011) Q[8:4] := #B00100 ;
            #B00100) Q[8:4] := #B00101 ;
            #B00101) Q[8:4] := #B00110 ;

```

```

#B00110) Q[8:4] := #B00111 ;
#B00111) Q[8:4] := #B01000 ;
#B01000) Q[8:4] := #B01001 ;
#B01001) Q[8:4] := #B01010 ;
#B01010) Q[8:4] := #B01011 ;
#B01011) Q[8:4] := #B01100 ;
#B01100) Q[8:4] := #B01101 ;
#B01101) Q[8:4] := #B01110 ;
#B01110) Q[8:4] := #B01111 ;
#B01111) Q[8:4] := #B10000 ;
#B10000) Q[8:4] := #B10001 ;
#B10001) Q[8:4] := #B10010 ;
#B10010) Q[8:4] := #B10011 ;
#B10011) Q[8:4] := #B10100 ;
#B10100) Q[8:4] := #B10101 ;
#B10101) Q[8:4] := #B10110 ;
#B10110) Q[8:4] := #B10111 ;
#B10111) Q[8:4] := #B11000 ;
#B11000) Q[8:4] := #B11001 ;
#B11001) Q[8:4] := #B11010 ;
#B11010) Q[8:4] := #B11011 ;
#B11011) Q[8:4] := #B11100 ;
#B11100) Q[8:4] := #B11101 ;
#B11101) Q[8:4] := #B11110 ;
#B11110) Q[8:4] := #B11111 ;
#B11111) Q[8:4] := #B00000 ;

```

```

END ;
END ;
ELSE Q[8:4] := Q[8:4] ;

```

END ;

IF (DOWN) THEN BEGIN

```

    CASE (Q[3:0]) BEGIN
        #B0000) Q[3:0] := #B1111 ;
        #B0001) Q[3:0] := #B0000 ;
        #B0010) Q[3:0] := #B0001 ;
        #B0011) Q[3:0] := #B0010 ;
        #B0100) Q[3:0] := #B0011 ;
        #B0101) Q[3:0] := #B0100 ;
    END ;

```

Figure 3-26. PLPL Specification for a 9-Bit Up-Down Counter (1 of 2)

```

#B0110) Q[3:0] := #B0101 ;
#B0111) Q[3:0] := #B0110 ;
#B1000) Q[3:0] := #B0111 ;
#B1001) Q[3:0] := #B1000 ;
#B1010) Q[3:0] := #B1001 ;
#B1011) Q[3:0] := #B1010 ;
#B1100) Q[3:0] := #B1011 ;
#B1101) Q[3:0] := #B1100 ;
#B1110) Q[3:0] := #B1101 ;
#B1111) Q[3:0] := #B1110 ;

END ;
IF ( ZERO ) THEN BEGIN
CASE ( Q[8:4] ) BEGIN
#B00000) Q[8:4] := #B11111 ;
#B00001) Q[8:4] := #B00000 ;
#B00010) Q[8:4] := #B00001 ;
#B00011) Q[8:4] := #B00010 ;
#B00100) Q[8:4] := #B00011 ;
#B00101) Q[8:4] := #B00100 ;
#B00110) Q[8:4] := #B00101 ;
#B00111) Q[8:4] := #B00110 ;
#B01000) Q[8:4] := #B00111 ;
#B01001) Q[8:4] := #B01000 ;
#B01010) Q[8:4] := #B01001 ;
#B01011) Q[8:4] := #B01010 ;
#B01100) Q[8:4] := #B01011 ;
#B01101) Q[8:4] := #B01100 ;
#B01110) Q[8:4] := #B01101 ;
#B01111) Q[8:4] := #B01110 ;
#B10000) Q[8:4] := #B01111 ;
#B10001) Q[8:4] := #B10000 ;
#B10010) Q[8:4] := #B10001 ;
#B10011) Q[8:4] := #B10010 ;
#B10100) Q[8:4] := #B10011 ;
#B10101) Q[8:4] := #B10100 ;
#B10110) Q[8:4] := #B10101 ;
#B10111) Q[8:4] := #B10110 ;
#B11000) Q[8:4] := #B10111 ;
#B11001) Q[8:4] := #B11000 ;
#B11010) Q[8:4] := #B11001 ;
#B11011) Q[8:4] := #B11010 ;
#B11100) Q[8:4] := #B11011 ;
#B11101) Q[8:4] := #B11100 ;
#B11110) Q[8:4] := #B11101 ;
#B11111) Q[8:4] := #B11110 ;

END ;
END ;
ELSE Q[8:4] := Q[8:4] ;
END ;
END.
TEST_VECTORS
IN  CLOCK UP CLEAR EN HOLD ;
OUT ONE Q[8:0] ;
BEGIN
"C U C   E H O   Q Q Q Q Q Q Q Q Q "
"L P L   N O N   8 7 6 5 4 3 2 1 0 "
"D   E       L E           "
"C   A       D           "
"K   R           "
P X X   X X X   L L L L L H L L L L ;
C 1 0   1 0 L   L L L L L H L L L H ; "COUNT UP"
C 1 0   1 0 L   L L L L L H L L L H ;
C 1 0   1 0 L   L L L L L H L L L H ;
P X X   X X L   L L L L L L L L L H ; "PRELOAD REGS"
C 1 0   1 0 L   L L L L L L L L L H ; "COUNT UP "
C 1 0   1 0 L   L L L L L L L L L H ;
C 1 0   1 0 L   L L L L L L L L L H ;
C 1 0   1 0 L   L L L L L L L L L H ;
C X 0   1 1 L   L L L L L L L L L H ; "HOLD "
C 0 0   1 0 L   L L L L L L L L L H ; "COUNT DOWN "
C 0 0   1 0 L   L L L L L L L L L H ;
C 0 0   1 0 L   L L L L L L L L L H ;
C 0 0   1 0 L   L L L L L L L L L H ;
C 0 0   1 0 L   L L L L L L L L L H ;
C 0 0   1 0 L   L L L L L L L L L H ;
C 0 0   1 0 H   H H H H H H H H H H ; "COUNT DN/WRAP AROUND"
C X X   0 X X   Z Z Z Z Z Z Z Z Z Z ; "DISABLE OUTPUTS"
C X 1   1 0 L   L L L L L L L L L L ; "CLEAR "
P X X   X X H   H H H H H H H H H H ; "PRELOAD REGS"
C 1 0   1 0 L   L L L L L L L L L L ; "COUNT UP/WRAP AROUND"
C 1 0   1 0 L   L L L L L L L L L H ;
C 1 0   1 0 L   L L L L L L L L L H ;
END.

```

Figure 3-26. PLPL Specification—9-Bit Up-Down Counter (2 of 2)

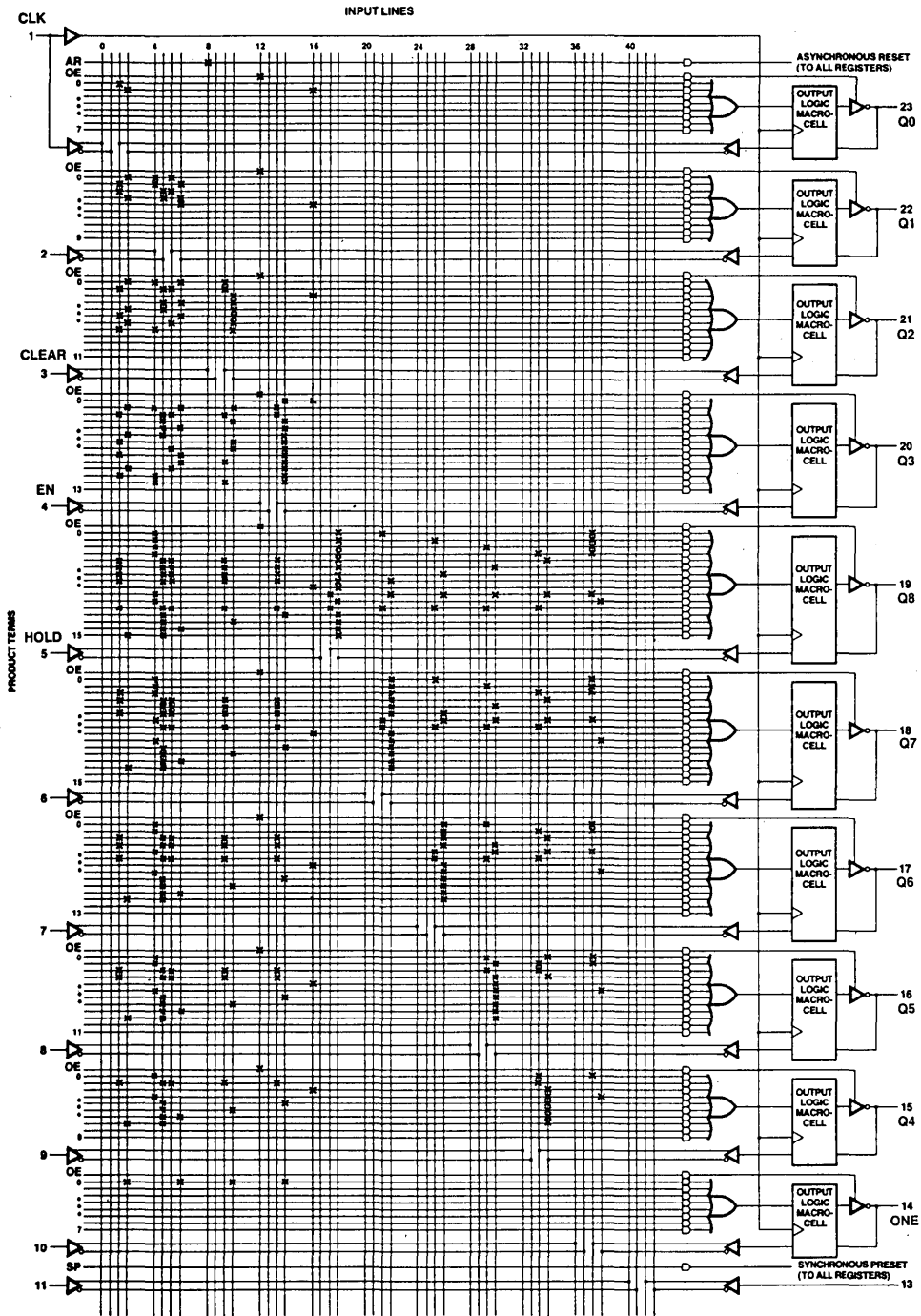


Figure 3-27. Fuse Map of a 9-Bit Up-Down Counter

Modulo-360 Counter

A typical n -bit binary counter has a maximum modulus equal to 2^n , where n is the number of stages. However, it can be modified to have a lesser modulus, without changing the number of stages. This may be necessary for applications where a non-power-of-two modulus is required.

Often it is necessary to introduce signal delays into logic design to meet tight timing requirements such as to allow for bus skew, access time or different propagation delays through devices along two different signal paths. A typical example of this is the introduction of wait states to allow for access times of different memory elements. Counters or delay lines are commonly used to introduce the delay. Counters have the advantage of programmability to generate the required delay and are possibly easier to synchronize with the master device. A modulo counter, implemented with a PAL device can be used for these kind of applications.

Design Requirements

For example, if a counter with modulus of 17 is required, five flip-flops will be required because a four-stage counter has maximum modulus of 16. Similarly, to implement modulo-360 counter, nine flip-flops will be required.

However, to achieve a lesser modulus in an n -stage counter, some of the "natural" states have to be skipped. For example, a five-stage counter has a natural counting sequence from 0 to 31. If the counter is to be redefined as modulus 17 counter,

then the states 18 to 31 have to be skipped. That means the counter circuitry has to be modified. Whenever the counter reaches state 17 it must be reset to state 0, rather than counting up to state 18.

Design Approach

Design approach for modulo-counters of any desired length is quite similar to that of typical binary or BCD counters. In a simple case, Boolean equations can be derived from the function table covering all possible states of the counter. However, the size of the function table increases with the number of states. For large counters, generating Boolean equations can become tedious and time consuming.

Another approach for implementing the modulo-counter is to partition the design into smaller state machines. We know that a modulo-360 counter requires nine flip-flops. However, instead of implementing this as a straight 9-bit counter from the function table we can implement this as two counters: one 4-bit counter (counting from 0 to 14) and another 5-bit counter (counting from 0 to 23). Together the two counters count up to 360. The signal M_OUT is asserted when the counter counts up to 360.

Figure 3-28 shows the block diagram of a 9-bit loadable, modulo-360 counter. It needs nine inputs, nine outputs, one clock pin, one LOAD pin, one RESET, and one M_OUT (module output) pin. Figure 3-29 shows the PLPL specification of this modulo-360 counter. Figure 3-30 shows the implementation of this function in a single AmPAL22V10 device.

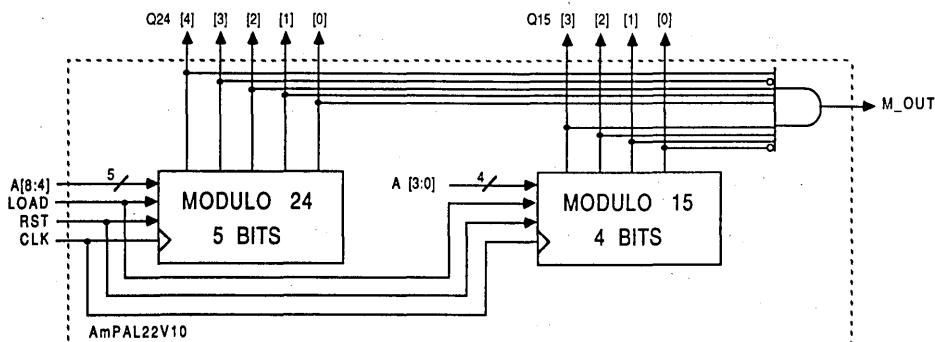


Figure 3-28. Block Diagram

DEVICE MODULO_360_COUNTER (PAL22V10)

```

PIN    CLK = 1   RST = 2
      LOAD = 3
      A[8:0] = 4:11,13
      Q24[4:0] = 18:14
      Q15[3:0] = 22:19
      M_OUT = 23; "ASSERTED WHEN COUNT BECOMES 360"

BEGIN
  IF (RST) THEN ARESET();
  IF (LOAD) THEN BEGIN
    Q24[4:0] := A[8:4];
    Q15[3:0] := A[3:0];
  END;
  ELSE BEGIN
    IF (Q24[4]*Q24[3]*Q24[2]*Q24[1]*Q24[0]*
      Q15[3]*Q15[2]*Q15[1]*Q15[0]) THEN M_OUT = 1;
    CASE (Q24[4:0]) BEGIN " MODULO 24 "
      #B00000) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00001;
      END;
      #B00001) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00010;
      END;
      #B00010) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00011;
      END;
      #B00011) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00100;
      END;
      #B00100) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00101;
      END;
      #B00101) BEGIN
        Q15[3:0] := Q15[3:0];
        Q24[4:0] := #B00110;
      END;
    END;
  END;
END;

```

```

#B00110) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B00111;
END;
#B00111) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01000;
END;
#B01000) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01001;
END;
#B01001) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01010;
END;
#B01010) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01011;
END;
#B01011) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01100;
END;
#B01100) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01101;
END;
#B01101) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01110;
END;
#B01110) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B01111;
END;
#B01111) BEGIN
  Q15[3:0] := Q15[3:0];
  Q24[4:0] := #B10000;
END;

```

Figure 3-29. PLPL Specification for a Modulo 360 Counter (1 of 2)

```
#B10000) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10001;
END;
#B10001) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10010;
END;
#B10010) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10011;
END;
#B10011) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10100;
END;
#B10100) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10101;
END;
#B10101) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10110;
END;
#B10110) BEGIN
    Q15[3:0] := #Q15[3:0];
    Q24[4:0] := #B10111;
END;
#B10111) BEGIN
    Q24[4:0] := #B00000;
    CASE (Q15[3:0]) BEGIN
        #B0000) Q15[3:0] := #B0001;
        #B0001) Q15[3:0] := #B0010;
        #B0010) Q15[3:0] := #B0011;
        #B0011) Q15[3:0] := #B0100;
        #B0100) Q15[3:0] := #B0101;
        #B0101) Q15[3:0] := #B0110;
        #B0110) Q15[3:0] := #B0111;
        #B0111) Q15[3:0] := #B1000;
        #B1000) Q15[3:0] := #B1001;
        #B1001) Q15[3:0] := #B1010;
        #B1010) Q15[3:0] := #B1011;
        #B1011) Q15[3:0] := #B1100;
        #B1100) Q15[3:0] := #B1101;
        #B1101) Q15[3:0] := #B1110;
        #B1110) Q15[3:0] := #B0000;
    END;
END.
```

Figure 3-29. PLPL Specification for a Modulo 360 Counter (2 of 2)

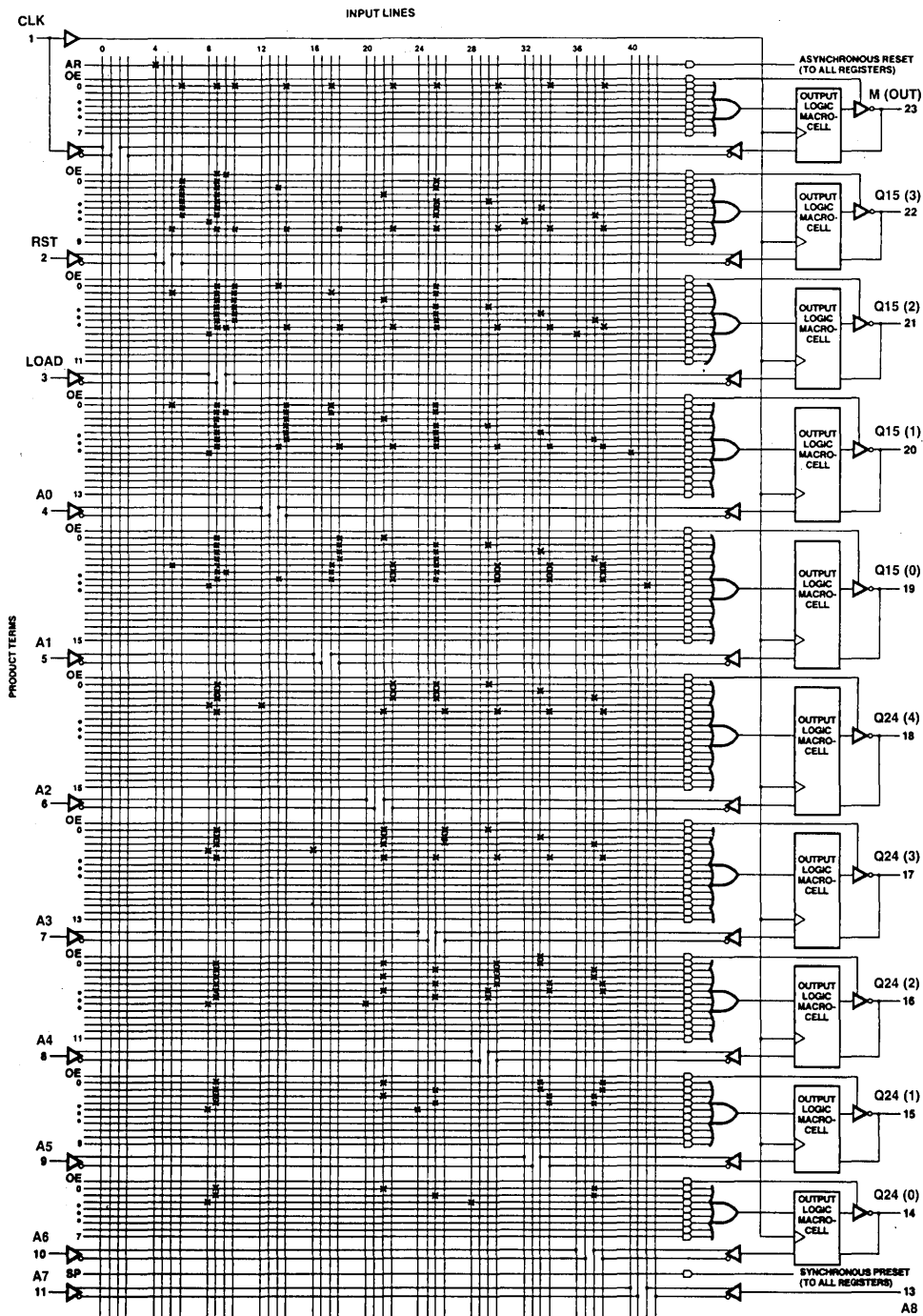


Figure 3-30. Fuse Map of a Modulo-360 Counter

Nine-Bit Johnson Counter

The Johnson counter is also known as a "circular-shift counter" and produces a special counting sequence. The sequence for a five-stage Johnson counter is shown in Table 3-9. Note that, the five-stage sequence has a table of 10 valid states and 22 invalid states (Table 3-9). In general an n-stage (bit) Johnson counter will produce a modulus of $2n$. Figure 3-31 shows the state diagram of a five-stage counter.

As seen from the sequence table of the five-stage counter, the counter first fills up with 1's from left to right and then it fills up with 0's again. Note from the counting sequence that unlike a normal binary counter, Johnson counter is a unit distance counter. Like a Gray code counter, its output changes only one bit at a time. One major advantage of Johnson counter's counting sequence is that it can be readily decoded with two input AND gates, and hence, is suitable for high-speed applications where output can be decoded safely.

TABLE 3-9. FIVE-STAGE JOHNSON COUNTER SEQUENCE

Q0	Q1	Q2	Q3	Q4
0	0	0	0	0
1	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
1	1	1	1	1
0	1	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

VALID STATES

Q0	Q1	Q2	Q3	Q4
0	1	0	0	0
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
0	0	0	1	0
1	0	0	1	0
0	1	0	1	0
1	1	0	1	0
0	0	1	1	0
1	0	1	1	0
0	1	1	1	0
1	0	0	0	1
0	1	0	0	1
1	1	0	0	1
0	0	1	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	0	1
1	0	0	1	1
0	1	0	1	1
1	1	0	1	1
1	0	1	1	1

INVALID STATES

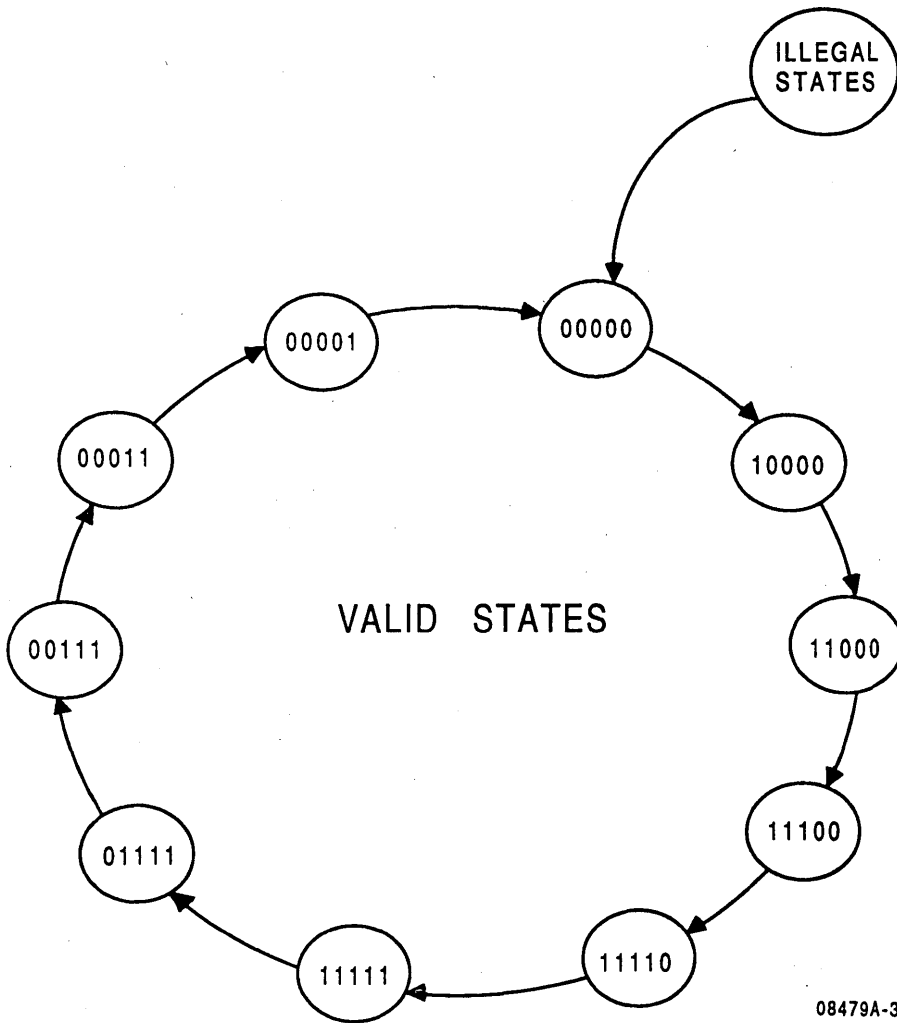


Figure 3-31. 5-Stage Johnson Counter State Diagram

Design Considerations

The implementation of a Johnson counter is relatively straight forward and is the same regardless of the number of stages. Implementing this with D-type flip-flops, the Q output of each flip-flop is connected to the D input of the following stage. The single exception is that the Q output of the last stage is complemented and connected to the D input of the first stage.

One disadvantage of this counter is the number of invalid states. For an n-stage counter, the number of invalid states is equal to $2^n - 2n$. For example, a three-stage Johnson counter has two invalid states, a four-stage counter has eight invalid states, and a five-stage counter has 22 invalid states. The number of invalid states increases almost exponentially with the length of the counter (for example, a nine-stage counter has 494 invalid states). The bigger the counter becomes, the more are the chances of its entering an illegal state. Once the counter enters an invalid state, it may be difficult to recover.

Care should be taken when designing Johnson counter to prevent it from entering invalid states. Often even self-correcting logic should be incorporated to allow the counter to get out of illegal states.

Design Requirement

Figure 3-32 shows the block diagram of a 9-bit Johnson counter. The counting sequence shown in Table 3-9 for 5-bit counter can easily be expanded to 9-bits. This counter (with load capability) requires nine input pins, nine output pins, a LOAD pin, one RESET, and invalid status pin and one clock pin. During power-on, the counter resets to zero state.

Figure 3-33 shows the PLPL specification for this counter. Figure 3-34 shows the fuse map for this counter implemented with a single AmPAL22V10 instead of four SSI/MSI devices. Additionally, an error flag for invalid states is incorporated in the design to indicate error condition.

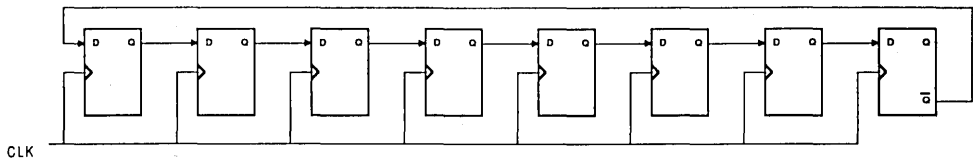


Figure 3-32. Block Diagram of a 9-Bit Johnson Counter

DEVICE NINE_BIT_JOHNSON_COUNTER (AmPAL22V10)

```

PIN    CLK = 1
      RST = 2
      A[8:0] = 3:11
      Q[8:0] = 14:21,23
      VF = 22 "VALID STATE FLAG"
      LOAD = 13 ;

BEGIN
  IF (RST) THEN ARESET(Q[8:0]);
  IF (LOAD) THEN Q[8:0] := A[8:0] ;
  ELSE BEGIN
    CASE (Q[8:0])
      BEGIN
        #8000000000) Q[8:0] := #8100000000 ;
        #8100000000) Q[8:0] := #8110000000 ;
        #8110000000) Q[8:0] := #8111000000 ;
        #8111000000) Q[8:0] := #8111100000 ;
        #8111100000) Q[8:0] := #8111110000 ;
        #8111110000) Q[8:0] := #8111111000 ;
        #8111111000) Q[8:0] := #8111111100 ;
        #8111111100) Q[8:0] := #8111111110 ;
        #8111111110) Q[8:0] := #8111111111 ;
        #8111111111) Q[8:0] := #8011111111 ;
        #8011111111) Q[8:0] := #8001111111 ;
        #8001111111) Q[8:0] := #8000111111 ;
        #8000111111) Q[8:0] := #8000011111 ;
        #8000011111) Q[8:0] := #8000001111 ;
        #8000001111) Q[8:0] := #8000000111 ;
        #8000000111) Q[8:0] := #8000000011 ;
        #8000000011) Q[8:0] := #8000000001 ;
        #8000000001) Q[8:0] := #8000000000 ;

      END;
    END;
  CASE (Q[8:0])
    BEGIN
      #8000000000) VF = 1 ;
      #8100000000) VF = 1 ;

```

```

#8110000000) VF = 1 ;
#8111000000) VF = 1 ;
#8111100000) VF = 1 ;
#8111110000) VF = 1 ;
#8111111000) VF = 1 ;
#8111111100) VF = 1 ;
#8111111110) VF = 1 ;
#8111111111) VF = 1 ;
#8011111111) VF = 1 ;
#8001111111) VF = 1 ;
#8000111111) VF = 1 ;
#8000011111) VF = 1 ;
#8000001111) VF = 1 ;
#8000000111) VF = 1 ;
#8000000011) VF = 1 ;
#8000000001) VF = 1 ;

```

END;

TEST_VECTORS

IN CLK RST LOAD A[8:0];

OUT Q[8:0] VF;

BEGIN

```

" L "
"C R O "
"L S A A A A A A A A A A Q Q Q Q Q Q Q Q V"
"K T D 8 7 6 5 4 3 2 1 0 8 7 6 5 4 3 2 1 0 F"

```

```

X 1 X X X X X X X X L L L L L L L L H; "RESET"
C 0 1 1 1 1 1 1 1 1 H H H H H H H H H; "LOAD"
C 0 0 X X X X X X X X L H H H H H H H H; "COUNT"
C 0 0 X X X X X X X X L L H H H H H H H;
C 0 0 X X X X X X X X L L L H H H H H H; "COUNT"
C 0 1 1 0 1 0 1 0 1 0 1 H L H L H L H L H; "LD INVALID ST"
C 0 0 X X X X X X X X L L L L L L L L L; "INVALID ST"
C 0 0 X X X X X X X X H L L L L L L L L; " RECOVERY"
C 0 1 1 1 1 1 1 1 1 0 H H H H H H H H L; "LOAD"
C 0 0 X X X X X X X X H H H H H H H H H; "COUNT"
C 0 0 X X X X X X X X L H H H H H H H H;
C 0 1 0 0 0 0 0 0 0 1 L L L L L L L L L;
C 0 0 X X X X X X X X L L L L L L L L L;

```

END.

Figure 3-33. PLPL Specification for 9-Bit Johnson Counter

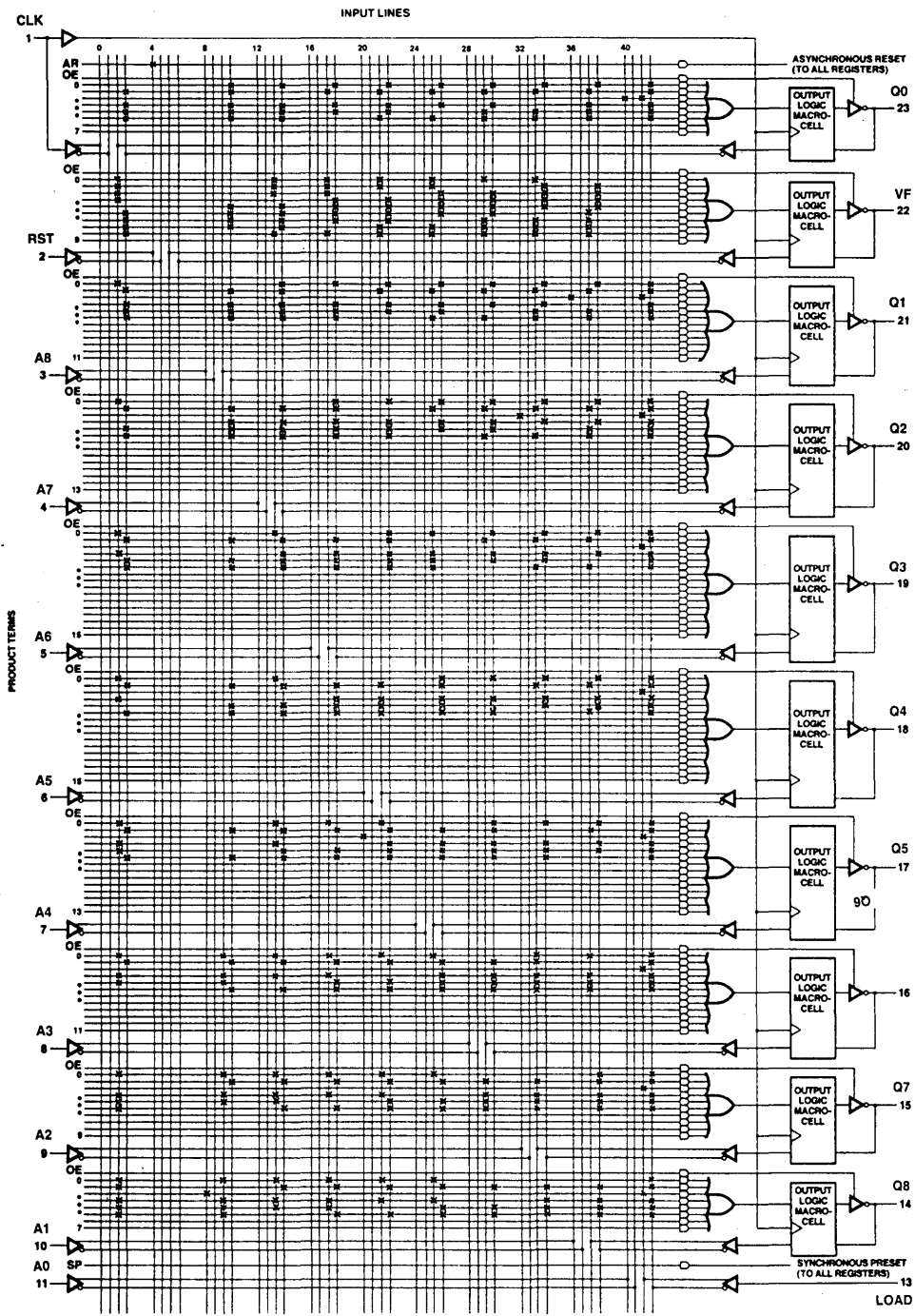


Figure 3-34. Fuse Map for a 9-Bit Johnson Counter

Gray Code Counter and Gray-to-Binary Code Conversion

Gray code is an important unweighted code which exhibits only a single bit change from one code number to the next. This feature is especially important in high-speed data communication applications, where data is transmitted from one part of a system to another and where the error susceptibility increases with the number of bit changes between adjacent numbers in a sequence. It is also commonly used in applications such as real-time process control and shaft encoders where each segment is assigned a separate bit.

Table 3-10 shows a listing of five-bit Gray code for decimal numbers 0 through 31. Note that there is only single bit change between any two Gray code numbers. The corresponding binary numbers are shown for reference. For instance, going from decimal value 11 (Binary 01011) to 12 (Binary 01100), the Gray code changes from 01110 to 01010. Only the third bit changes; the others remain the same.

The Gray code can be extended to any number of bits. Conversion from Gray to binary (or visa versa) is relatively simple.

Conversion from Gray-to-Binary Code

To convert from Gray code to binary, the following rules apply:

- The most significant bit (the left-most) in the binary code is the same as the corresponding bit in the Gray code.
- Going from left to right, add the next Gray code bit to the previous binary bit to generate the next binary bit. Disregard the carries (Figure 3-35).

This can be conveniently implemented with exclusive-OR gates. Figure 3-36 shows the logic diagram for converting 5-bit Gray code to binary code.

TABLE 3-10. 5-BIT GRAY CODE

DECIMAL	GRAY	BINARY
0	0 0 0 0 0	0 0 0 0 0
1	0 0 0 0 1	0 0 0 0 1
2	0 0 0 1 1	0 0 0 1 0
3	0 0 0 1 0	0 0 0 1 1
4	0 0 1 1 0	0 0 1 0 0
5	0 0 1 1 1	0 0 1 0 1
6	0 0 1 0 1	0 0 1 1 0
7	0 0 1 0 0	0 0 1 1 1
8	0 1 1 0 0	0 1 0 0 0
9	0 1 1 0 1	0 1 0 0 1
10	0 1 1 1 1	0 1 0 1 0
11	0 1 1 1 0	0 1 0 1 1
12	0 1 0 1 0	0 1 1 0 0
13	0 1 0 1 1	0 1 1 0 1
14	0 1 0 0 1	0 1 1 1 0
15	0 1 0 0 0	0 1 1 1 1
16	1 1 0 0 0	1 0 0 0 0
17	1 1 0 0 1	1 0 0 0 1
18	1 1 0 1 1	1 0 0 1 0
19	1 1 0 1 0	1 0 0 1 1
20	1 1 1 1 0	1 0 1 0 0
21	1 1 1 1 1	1 0 1 0 1
22	1 1 1 0 1	1 0 1 1 0
23	1 1 1 0 0	1 0 1 1 1
24	1 0 1 0 0	1 1 0 0 0
25	1 0 1 0 1	1 1 0 0 1
26	1 0 1 1 1	1 1 0 1 0
27	1 0 1 1 0	1 1 0 1 1
28	1 0 0 1 0	1 1 1 0 0
29	1 0 0 1 1	1 1 1 0 1
30	1 0 0 0 1	1 1 1 1 0
31	1 0 0 0 0	1 1 1 1 1

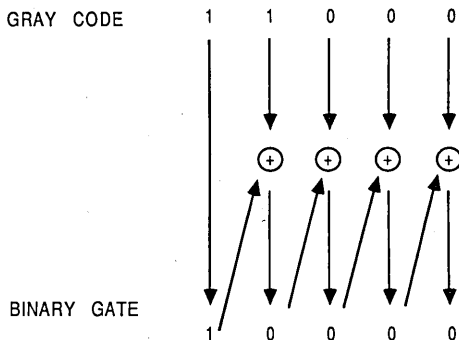
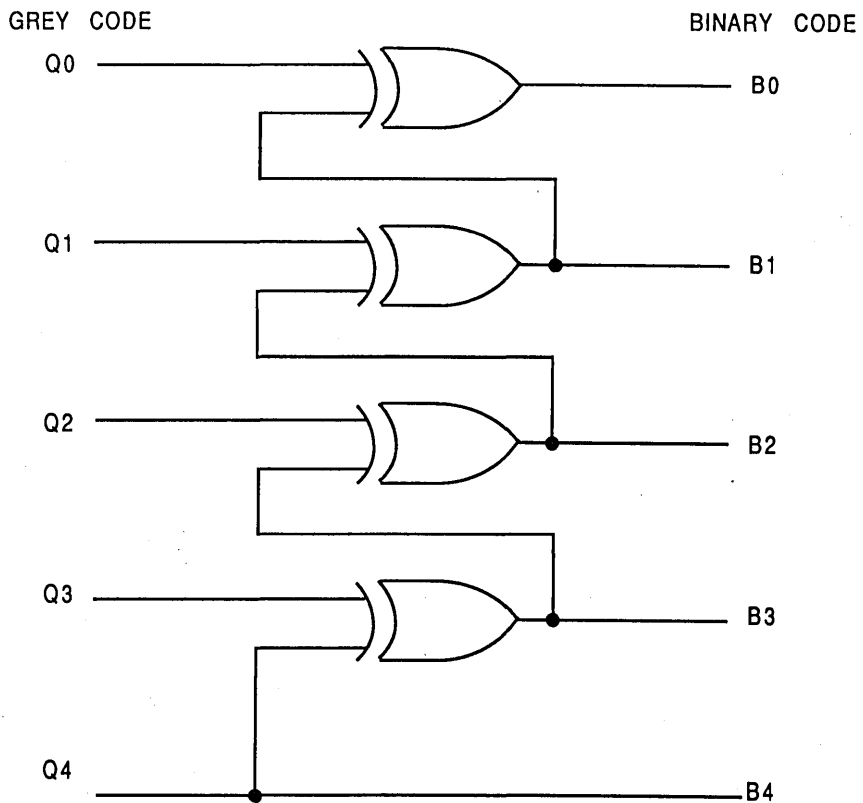


Figure 3-35. Gray Code to Binary Code Conversion



08470A-34

Figure 3-36. Logic Diagram for Gray Code-to-Binary Conversion

Design Requirements and Approach

Implementation of a 5-bit Gray code counter requires at least five flip-flops and some extra logic. Implementation of 5-bit Gray-to-binary code conversion can be accomplished with XOR gates or in a PAL device that has enough product terms.

Figure 3-37 shows the PLPL specification of 5-bit Gray code counter and 5-bit Gray-to-binary converter. Figure 3-38 shows the corresponding fuse map of this function implemented in a single AmpAL22V10 device. As shown in the fuse map the least significant bit requires 16 product terms for Gray-to-binary conversion.

```

DEVICE FIVE_BIT_GRAY_CODE_COUNTER (PAL22V10)
PIN

```

```

CLK = 1
RST = 2
Q[4:0] = 18:14
B[4:0] = 23:19;

```

```

BEGIN

```

```

IF (RST) THEN ARESET(Q[4:0]) ;
CASE (Q[4:0])
BEGIN

```

```

    #B00000) Q[4:0] := #B00001 ;
    #B00001) Q[4:0] := #B00011 ;
    #B00011) Q[4:0] := #B00010 ;
    #B00010) Q[4:0] := #B00110 ;
    #B00110) Q[4:0] := #B00111 ;
    #B00111) Q[4:0] := #B00101 ;
    #B00101) Q[4:0] := #B00100 ;
    #B00100) Q[4:0] := #B01100 ;
    #B01100) Q[4:0] := #B01101 ;
    #B01101) Q[4:0] := #B01111 ;
    #B01111) Q[4:0] := #B01110 ;
    #B01110) Q[4:0] := #B01010 ;
    #B01010) Q[4:0] := #B01011 ;
    #B01011) Q[4:0] := #B01001 ;
    #B01001) Q[4:0] := #B01000 ;
    #B01000) Q[4:0] := #B11000 ;
    #B11000) Q[4:0] := #B11001 ;
    #B11001) Q[4:0] := #B11011 ;
    #B11011) Q[4:0] := #B11010 ;
    #B11010) Q[4:0] := #B11110 ;
    #B11110) Q[4:0] := #B11111 ;
    #B11111) Q[4:0] := #B11101 ;
    #B11101) Q[4:0] := #B11100 ;
    #B11100) Q[4:0] := #B10100 ;
    #B10100) Q[4:0] := #B10101 ;
    #B10101) Q[4:0] := #B10111 ;
    #B10111) Q[4:0] := #B10110 ;
    #B10110) Q[4:0] := #B10010 ;
    #B10010) Q[4:0] := #B10011 ;
    #B10011) Q[4:0] := #B10001 ;
    #B10001) Q[4:0] := #B10000 ;
    #B10000) Q[4:0] := #B00000 ;

```

```

END;

```

```

" GRAY CODE TO BINARY CODE CONVERSION "

```

```

B[4] = Q[4] ;
B[3] = Q[3] XOR Q[4] ;
B[2] = Q[2] XOR Q[3] XOR Q[4] ;
B[1] = Q[1] XOR Q[2] XOR Q[3] XOR Q[4] ;
B[0] = Q[0] XOR Q[1] XOR Q[2] XOR Q[3] XOR Q[4] ;

```

```

END.

```

```

TEST_VECTORS

```

```

IN      CLK RST;
I_O     Q[4:0];
OUT      B[4:0];
BEGIN
"C R   Q Q Q Q Q   B B B B B "
"L S   4 3 2 1 0   4 3 2 1 0 "
"K T           "
C 1   L L L L L   L L L L L ;
C 0   L L L L H   L L L L H ;
C 0   L L L H H   L L L H L ;
C 0   L L L H L   L L L H H ;
C 0   L L H H L   L L H L L ;
C 0   L L H H H   L L H L H ;
C 0   L L H L H   L L H H L ;
C 0   L L H L L   L L H H H ;
C 0   L H H L L   L H L L L ;
P 0   L H L L L   L H H H H ;
C 0   H H L L L   H L L L L ;
C 0   H H L L H   H L L L H ;
P 0   H L L L L   H H H H H ;
C 0   L L L L L   L L L L L ;
C 0   L L L L H   L L L L H ;

```

```

END.

```

Figure 3-37. PLPL Specification for a 5-Bit Gray Code Counter and 5-Bit Gray-to-Binary Converter

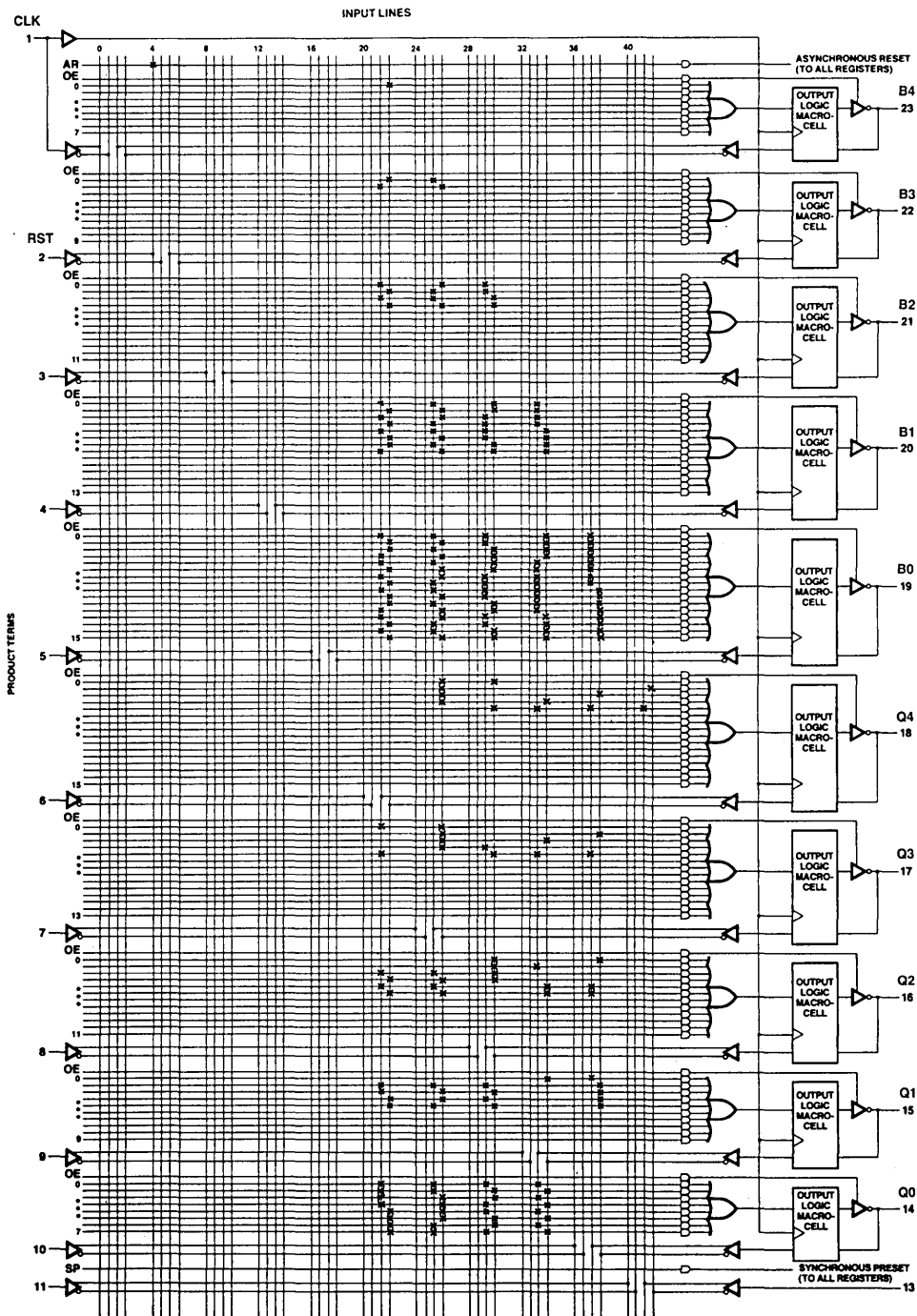


Figure 3-38. Fuse Map for a Gray Code Counter

3.3.2 SHIFTERS

Eight-Bit Arithmetic/Logic Shifter

Most commonly used logical operators are: AND, OR, XOR and COMPLEMENT. An AND operator is used for masking or clearing chosen bits of certain operands. An OR operator can be used for "setting" some required bits to value 1. An XOR operator can be used for generating check-sums or comparing two operands. A COMPLEMENT operator can be used for inverting logic levels.

All these four operators, together with "shift" operation can be used for functions such as: serial-to-parallel conversion, scaling, normalization, or for matching bit patterns. Shift operator can also be used for doing multiplication or division. There are four major shift operators: Logical SHIFT, Arithmetic SHIFT, ROTATE, and ROTATE with CARRY.

Logical-SHIFT operation shifts the data either right or left. Typically it fills the shifted bit with zero, and loads the CARRY bit from the bit shifted out. In Logical-SHIFT operation, the sign-bit is usually not preserved.

Arithmetic SHIFT is very similar to the Logical SHIFT except that the sign is preserved. For example, for shifting right, the sign-bit is copied into the next position. Often this is referred to as sign extension and can be used to either normalize a number or scale a two's-complement number. For shifting left, the sign-bit is left unchanged.

Simple ROTATE just rotates the data. The MSB and the LSB are linked together. In ROTATE with CARRY, the CARRY bit also acts as part of the shift register. Figure 3-39.a shows the diagrammatic representation of Arithmetic and logic shifter.

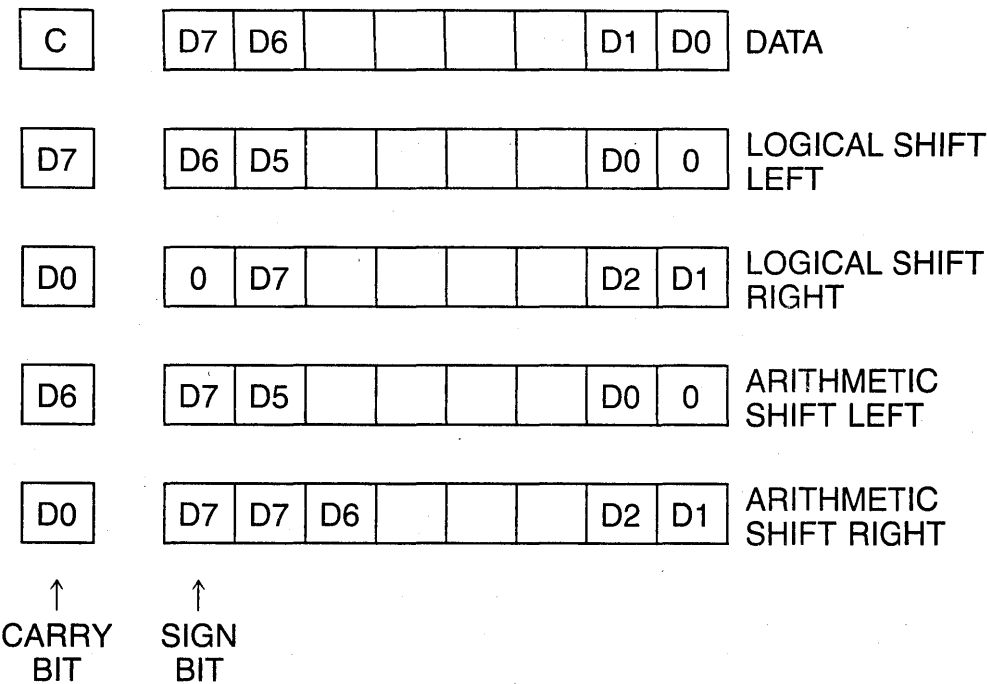


Figure 3-39.a Arithmetic and Logic Shifter

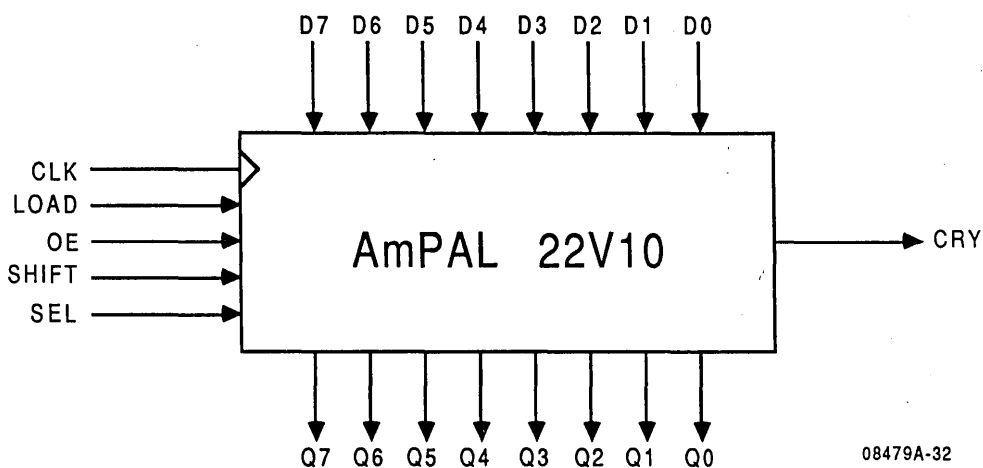
Design Requirements and Approach

An 8-bit combined arithmetic/logic registered shifter shifts data left or right both logically or arithmetically, maintaining the sign-bit appropriately.

Such a shifter requires eight data inputs, eight data outputs, a clock pin, a LOAD pin, an Output-Enable pin, a CARRY (CRY) output pin, and two control pins: one control pin to signify Arithmetic or Logical SHIFT, and one control pin to specify direction

(left or right). For ROTATE with CARRY function, it uses the CARRY pin. Figure 3-39.b shows the block diagram of an 8-bit registered arithmetic/logical shifter while Table 3-11 shows the function table.

This function can be easily implemented in a single AmPAL22V10 device. The AmPAL22V10 can provide up to 22 inputs and 10 outputs, enough for 9 registered outputs and for all other inputs.



08479A-32

Figure 3-39.b Block Diagram of 8-Bit Registers—Arithmetic Logic Shifter

TABLE 3-11. FUNCTION TABLE FOR 8-BIT ARITHMETIC LOGIC SHIFTER

EN	SEL	SHIFT	LD	D7	D6	D5	D4	D3	D2	D1	D0	CRY	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	X	X	X	X	X	X	X	X	X	X	X	Z	Z	Z	Z	Z	Z	Z	Z	Z
1	X	X	1	D7	D6	D5	D4	D3	D2	D1	D0	X	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	X	X	X	X	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0	0
1	0	1	1	X	X	X	X	X	X	X	X	D0	0	D7	D6	D5	D4	D3	D2	D1
1	1	0	1	X	X	X	X	X	X	X	X	D6	D7	D5	D4	D3	D2	D1	D0	0
1	1	1	1	X	X	X	X	X	X	X	X	D0	D7	D7	D6	D5	D4	D3	D2	D1

Figure 3-40 shows the PLPL specification for this function. Figure 3-41 shows the fuse map of 8-bit registered arithmetic/logical shifter function implemented in a single AmPAL22V10. As shown in the figure, each output requires only

five logical product terms: one for LOAD, one for arithmetic shift right, one for arithmetic shift left, one for logical shift right, and one for logical shift left.

DEVICE EIGHT_BIT_ARITHMETIC_LOGIC_SHIFTER (PAL22V10)

```
PIN    CK    = 1
      SEL    = 2    " HIGH -> ARITHMETIC SHIFT, LOW -> LOGICAL SHIFT"
      SHIFT  = 3    " HIGH -> RIGHT SHIFT,      LOW -> LEFT SHIFT"
      EN     = 4    " OUTPUT ENABLE "
      LOAD   = 5
      D[7:0] = 6:11,13,14
      CRY    = 15
      Q[7:0] = 23:16;
```

```
DEFINE LOGIC = /SEL ;
      ARITH = SEL ;
      LEFT  = /SHIFT ;
      RIGHT = SHIFT ;
```

```
BEGIN
  IF (EN) THEN ENABLE() ;
  IF (LOAD) THEN Q[7:0] := D[7:0] ;
  ELSE BEGIN
    IF (ARITH*RIGHT) THEN BEGIN
      Q[7] := Q[7] ; " Q[7] IS A SIGN BIT "
      Q[6] := Q[7] ;
      Q[5:0] := Q[6:1] ;
      CRY := Q[0] ;
    END ;
    IF (ARITH*LEFT) THEN BEGIN
      Q[7] := Q[7] ;
      Q[6:1] := Q[5:0] ;
      Q[0] := 0 ;
      CRY := Q[6] ;
    END ;
    IF (LOGIC*RIGHT) THEN BEGIN
      Q[7] := 0 ;
      Q[6:0] := Q[7:1] ;
      CRY := Q[0] ;
    END ;
    IF (LOGIC*LEFT) THEN BEGIN
      Q[7:1] := Q[6:0] ;
      Q[0] := 0 ;
      CRY := Q[7] ;
    END ;
  END ;
END ;
```

END.

TEST_VECTORS

```
IN    CK SEL SHIFT EN LOAD D[7:0] ;
OUT   CRY Q[7:0] ;
```

BEGIN

```
"CK SEL SHIFT EN LOAD D7 D6 D5 D4 D3 D2 D1 D0 CRY Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0"
C X X 0 0 X X X X X X Z Z Z Z Z Z Z Z ;
C X X 1 1 1 1 0 1 1 0 1 1 L H H L H H L H H ;
C 1 1 1 0 X X X X X X X X H H H H L H H L H ;
C 1 0 1 0 X X X X X X X X H H H L H H L H L ;
C 1 0 1 0 X X X X X X X X H H L H H L H L L ;
C 0 1 1 0 X X X X X X X X L L H L H H L H L ;
C 0 1 1 0 X X X X X X X X L L L H L H H L H ;
C 0 0 1 0 X X X X X X X X L L H L H H L H L ;
C 0 0 1 0 X X X X X X X X L H L H H L H L L ;
```

END.

Figure 3-40. PLPL Specification for an 8-Bit Arithmetic Logic Shifter

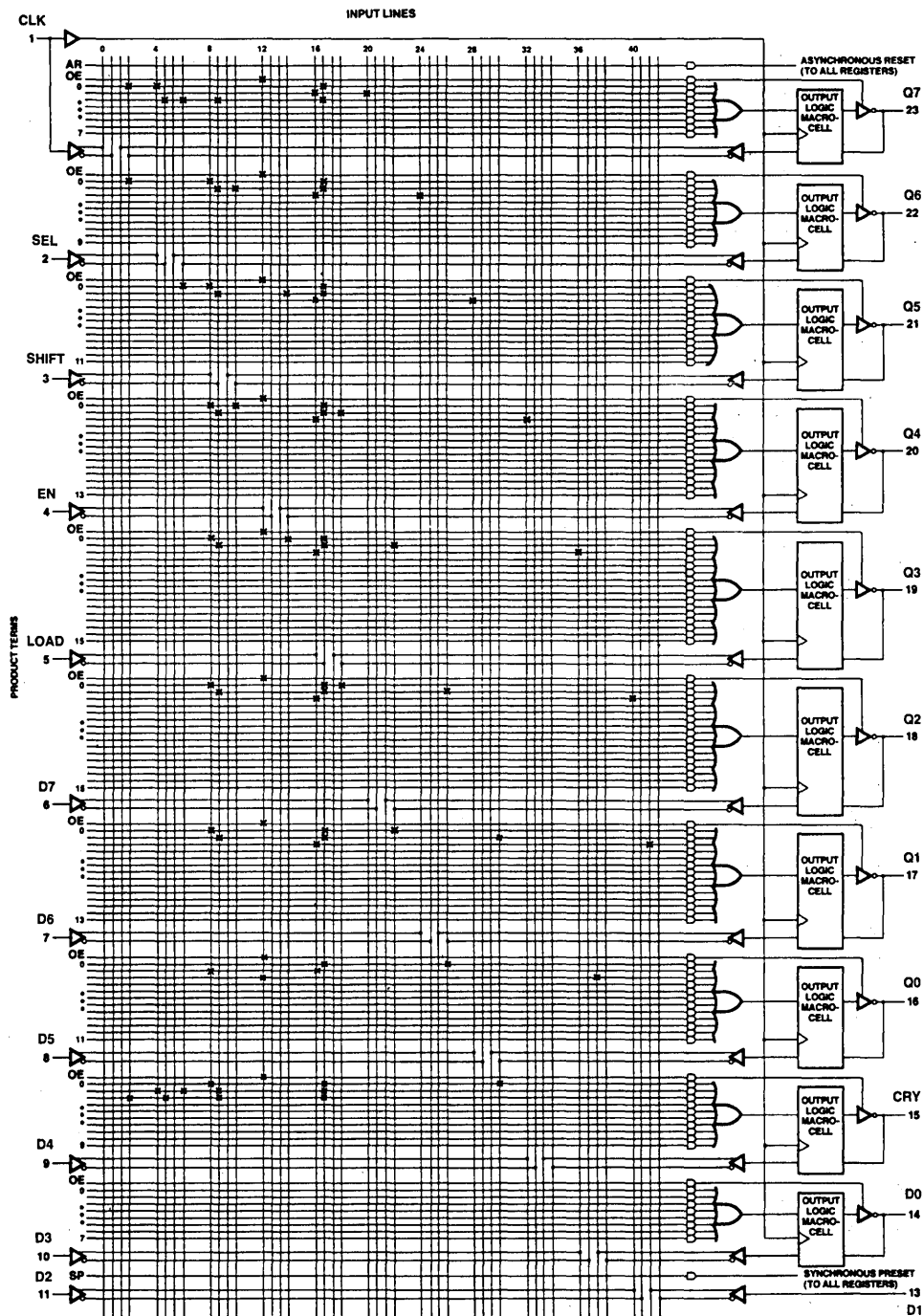


Figure 3-41. Fuse Map for an 8-Bit Arithmetic Logic Shifter

Eight-Bit Registered Barrel Shifter

In most data processing systems, some form of data shifting or rotation is necessary. This elementary function is used in such diverse applications as floating-point arithmetic and string

manipulation. In a typical computer, the shifter is located on the output of the ALU. This architecture allows single-cycle add and shift, and mask and shift operations (See Figure 3-42.a for a typical computer ALU architecture).

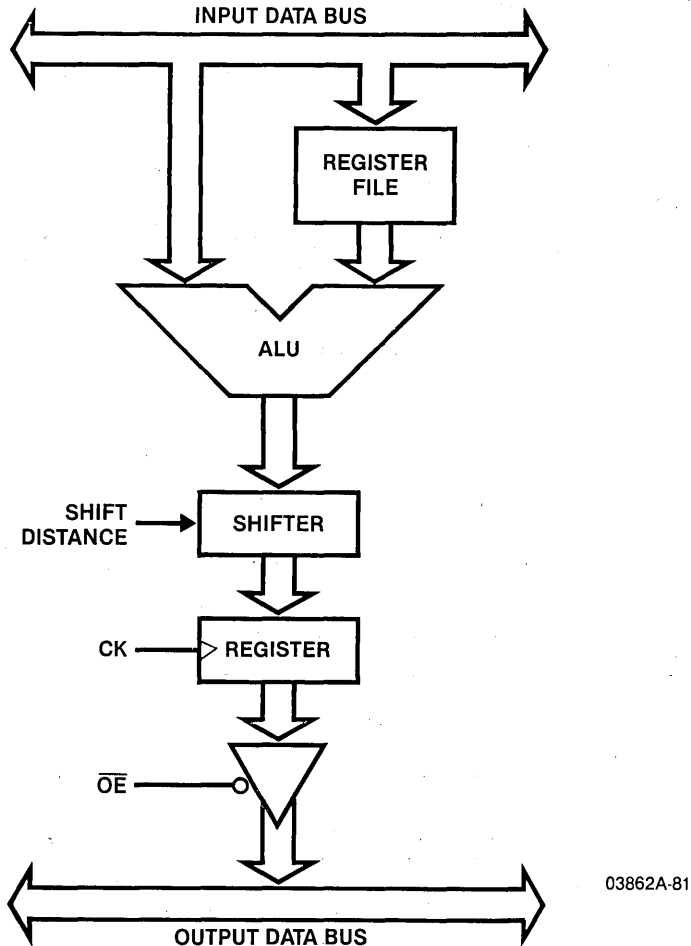


Figure 3-42.a Typical ALU Architecture

Design Requirements

A barrel shifter takes input data and cyclically rotates it by an arbitrary number of bit positions. This cyclic rotation implies that the data rotated from the most significant end of the shifter is returned to the least significant end. The name barrel shifter is used because of the circular nature of the shift operation.

The storage register on the output of the shifter is used in this architecture to pipeline the data operation, increasing throughput. The three-state buffer on the register output is necessary to interface the ALU to the output data bus.

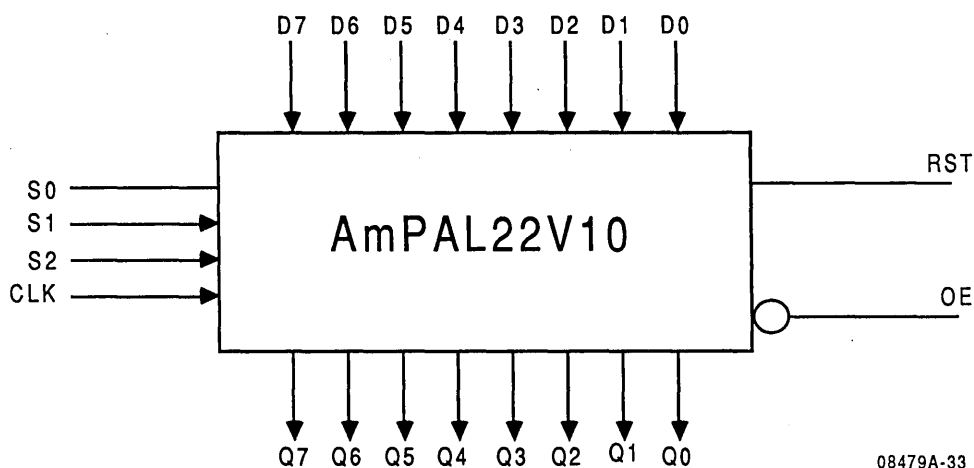
Design Approach

An 8-bit registered barrel shifter requires at least eight data inputs, eight registered data outputs, three control lines to specify shift distance, a clock input, a Reset input and an Output Enable that controls the tri-state buffer on the register output.

Figure 3-42.b shows the block diagram for an 8-bit registered barrel shifter while Table 3-12 shows the function table.

As seen from Figure 3-42.b, the registered barrel shifter function requires a total of 14 inputs and 8 outputs. This function can be easily implemented in a single AmPAL22V10 device. The AmPAL22V10 can provide up to 22 inputs and 10 outputs. The dynamic I/O programmability of the AmPAL22V10 permits it to meet the I/O requirement.

Figure 3-43 shows the PLPL specification for this function. Figure 3-44 shows the fuse map of 8-bit registered barrel shifter function implemented in AmPAL22V10.



08479A-33

Figure 3-42.b Block Diagram for an 8-Bit Registered Barrel Shifter

TABLE 3-12. FUNCTION TABLE FOR AN 8-BIT REGISTERED BARREL SHIFTER

Control In			Input To Output Mapping							
S2	S1	S0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

DEVICE EIGHT_BIT_BARREL_SHIFTER (AmPAL122V10)

```
PIN    CLK = 1
      D[7:0] = 2:9      /Q[7:0] = 23:16
      SEL2 = 13      SEL1 = 11      SEL0 = 10
      OE = 14      RST = 15;
```

BEGIN

```
IF (RST) THEN ARESET();
IF (OE) THEN ENABLE(Q[7:0]);
IF (/SEL2 * /SEL1 * /SEL0) THEN Q[7:0] := D[7:0];
IF (/SEL2 * /SEL1 * SEL0) THEN Q[7:0] := D[6:0], D[7];
IF (/SEL2 * SEL1 * /SEL0) THEN Q[7:0] := D[5:0], D[7:5];
IF (/SEL2 * SEL1 * SEL0) THEN Q[7:0] := D[4:0], D[7:4];
IF (SEL2 * /SEL1 * /SEL0) THEN Q[7:0] := D[3:0], D[7:4];
IF (SEL2 * /SEL1 * SEL0) THEN Q[7:0] := D[2:0], D[7:3];
IF (SEL2 * SEL1 * /SEL0) THEN Q[7:0] := D[1:0], D[7:2];
IF (SEL2 * SEL1 * SEL0) THEN Q[7:0] := D[0], D[7:1];
```

END.

" FUNCTION TABLE SPECIFICATION "

TEST_VECTORS

IN CLK RST D[7:0] SEL2 SEL1 SEL0;

I_O OE;

OUT Q[7:0];

BEGIN

```
"CLK RST D[7:0]      SEL2 SEL1 SEL0 OE Q[7:0] "
```

"

```
C 1 00000000 1 0 1 0 ZZZZZZZZ;
C 0 11111111 0 1 0 1 HHHHHHHH;
C 1 XXXXXXXX X X X 1 LLLLLLLL;
C 0 XXXXXXXX X X X 0 ZZZZZZZZ;
C 0 00000000 0 0 0 1 LLLLLLLL;
C 0 11111111 0 0 0 1 HHHHHHHH;
C 0 01111111 1 0 0 1 HHHHLHHH;
C 0 01111111 0 1 0 1 HHHHHHLH;
C 0 01111111 1 1 0 1 HHLHHHHH;
```

END.

3

Figure 3-43. PLPL Specification for an 8-Bit Registered Barrel Shifter

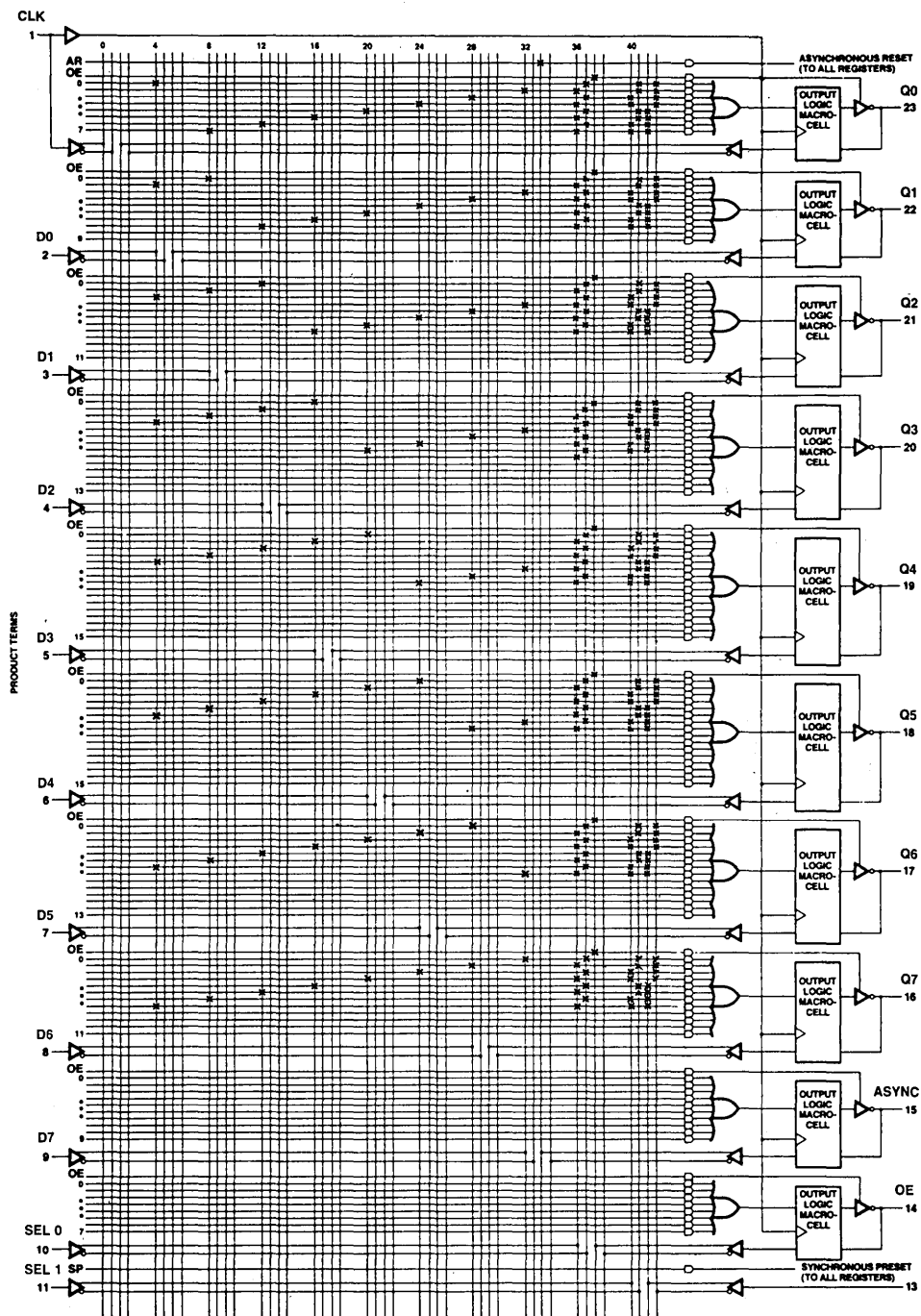


Figure 3-44. Fuse Map for an 8-Bit Registered Barrel Shifter

3.4 MICROPROCESSOR INTERFACE LOGIC

3.4.1 THE INTERFACE PROBLEM

Several problems are associated with interfacing a peripheral device to a CPU when the two families are not designed to be pin-to-pin compatible. One major problem involves the various control signals that each chip uses. What's more, the popular MOS microprocessors differ not only in their instruction set and internal architecture, but also in their I/O interface signals and timing, and in the way they handle DMA and interrupts. Part of the pin incompatibility involves genuine signal differences or timing differences. Some pin-to-pin differences require nothing more than name changes.

Problems to Look for When Interfacing

The following is a list of areas to watch out for when interfacing, and are discussed in detail:

I. Timing

A. Setup and Hold Times

- i. $\overline{CS}$ setup and hold to strobe.
 - ii. Address setup and hold to strobe.
 - iii. Data setup and hold to strobe (rising or falling)
- Note: Strobe may be $\overline{RD}$, $\overline{WR}$, or $\overline{DS}$.

B. Pulse Widths

- i. $\overline{AS}$ or $\overline{ALE}$ pulse widths.
 - ii. $\overline{RD}$, $\overline{WR}$, $\overline{DS}$ pulse widths.
- Pulse widths can usually be stretched by inserting Wait States if necessary.

C. Interrupts

Interrupt structures vary. Mixing interrupt structures is the toughest because two or more different sets of timings may need to be generated.

II. Metastable Operation

When to expect a cause of problem and the solutions.

III. Bus Structures

- i. MULTIBUS
- ii. ILBX
- iii. ISBX Multichannel

I. Timing

Setup and Hold Times

If one remembers nothing else from this book, one should remember the following: *The most common interfacing error is the violation of setup and hold times!* More specifically, three areas of setup and hold time relationships need close attention, and we will discuss them after a brief overview.

Setup and hold time are the minimum time the proper signal must be on the bus before and after the strobe. Minimum setup and hold time criterion must be met for proper interface operation. When interfacing involves direct hookup, individual specification can be looked up and the worse case calculated to meet timing requirements. If the interface involves additional components, such as buffers or transceivers, the data to strobe timing relationship may change and correction by delay devices such as flip-flops or latches may be necessary to bring the system circuit within timing requirements. The following are areas that need special attention:

- i) Chip select ($\overline{CS}$) setup and hold time to strobe. (Figure 3-45).

Here we use $\overline{RD}$ as the strobe. t_s is the minimum $\overline{CS}$ to strobe setup time. t_H is the minimum hold time. Comparisons are made in conditions where these requirements are not met and could lead to the wrong device being selected (Figures 3-46 and 3-47).

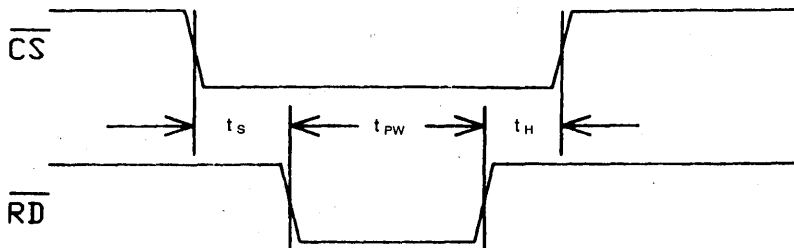


Figure 3-45.

02188A-1

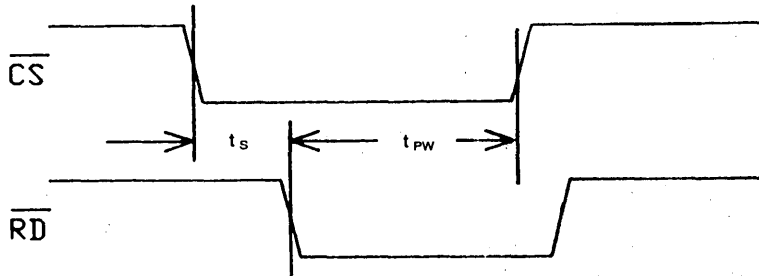


Figure 3-46.

02188A-2

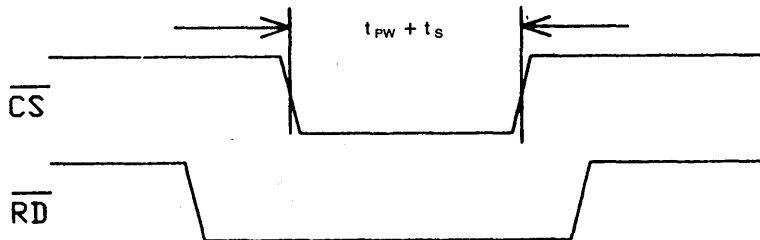


Figure 3-47.

02188A-3

Note that it is often assumed that $\overline{CS}$ and $\overline{RD}$ or $\overline{CS}$ and $\overline{WR}$ are simply gated together. This is true for most peripherals but not in all cases. Check the manufacturer's specification. It is often possible to implement and simplify the design but manufacturers do not guarantee operation in these unusual modes.

- ii) Address setup and hold times to strobe.
(Figures 3-48 and 3-49)

The figures illustrate the relationships between strobe to address setup and hold, and $\overline{CS}$. Address setup time is measured with respect to the leading edge of the strobe. If the strobe is not sufficiently delayed, after the address is valid, the address setup time requirement will not be met.

Address hold time will not be met if $\overline{CS}$ is, due to internal delay, late in going HIGH. This can be compensated by delaying the $\overline{WR}$ strobe. Failure in meeting either requirement will result in incorrect addressing.

- iii) Data setup and hold times to strobe ($\overline{WR}$).
(Figures 3-48 and 3-49)

The data setup and hold times are measured with respect to the rising edge of the strobe. The data is represented here as a window. If the strobe goes HIGH too late, data hold time will not be met, and incorrect data may be read. This may happen if the strobe is purposely delayed to meet Address setup and hold times. The data can be latched to allow a larger window so that data hold time can be met.

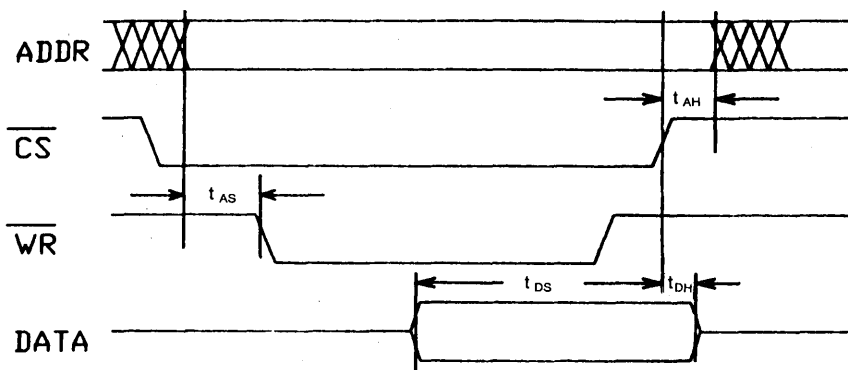


Figure 3-48.

02188A-4

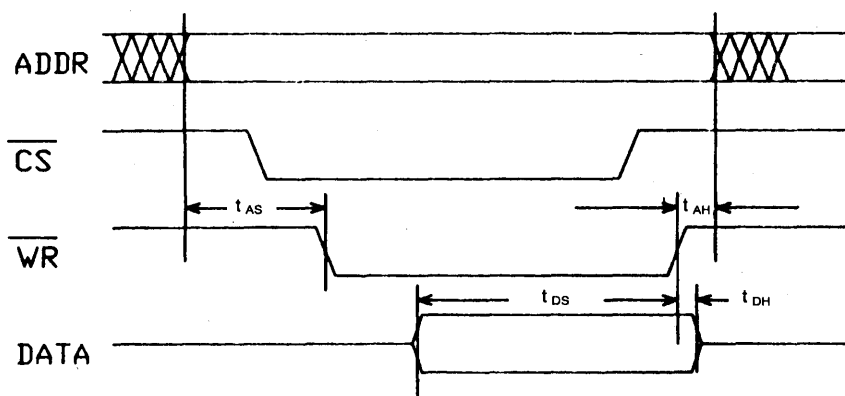


Figure 3-49.

02188A-5

II. Metastable Operation

Metastable conditions occur in all flip-flops when the active clock edge samples the input at exactly the same time the input changes state. When this happens, the cross-coupled latch at the output can reach a balanced, symmetrical condition in which it will remain for some arbitrary time before returning to its proper state. This problem is faced by designers when interfacing asynchronous digital signals. Although most difficulties can be overcome somewhat easily, there is a more fundamental problem that defies a perfect solution. The following is a general overview of the metastable problem.

Latches and flip-flops are normally considered bi-stable devices, since they have two unconditionally stable operating points, either HIGH or LOW. There is, however, a third operating point when the cross-coupled arrangement is exactly balanced. This operating point is stable only if there is no noise in the system and the system is perfectly balanced. The condition is called metastable (meta=Greek for "between"). A metastable condition will last only long enough for the circuit to fall into one of the two stable operating points. This period can be many micro-seconds, or milliseconds, for devices as fast as a 74S74 flip-flop. If a flip-flop has reached the balanced, metastable condition, it may remain in this state for an undetermined time, perhaps 1000 times longer than its normal response speed. Previously, the designer of asynchronous system had only one remedy for the metastable problem; two or more synchronizer flip-flops could be cascaded. This can reduce the probability of a metastable output but will also increase the throughput delay.

When to Expect this to Cause a Problem?

In most digital systems, certain asynchronous events (keystrokes, incoming data, interrupts) must be synchronized to the computer clock. The textbook solution is a fast, clocked flip-flop, like the 74S74, in which the asynchronous signal is applied to the D input and clocked with the system clock. This usually results in a perfectly synchronized output.

If the data sheet specified a setup time requirement (3 ns), this means that any signal that arrives at least 3 ns before the clock edge will achieve the intended result, i.e., a HIGH will set, a LOW will reset the flip-flop. Great for synchronous systems! But what happens when the asynchronous input violates this setup time requirement and changes less than 3 ns before the clock edge? Most of the time, nothing. The actual moment where the flip-flop samples the D input is somewhere in the guaranteed range, i.e., somewhere less than 3 ns before the clock. So the flip-flop makes the decision. It either senses the change on the asynchronous input and therefore changes the Q output, or it ignores the change and doesn't change the Q output. So the only thing lost is one clock cycle. Unfortunately that's not always true. If the D input changes exactly at the same moment that the flip-flop makes its decision, it might transfer exactly the amount of energy to kick the output latch into the metastable balanced condition, from which it will recover after an unpredictable delay (measured in nanoseconds, microseconds or even milliseconds).

Any latch, flip-flop or register has a "moment of truth" somewhere inside the guaranteed range of setup time where it actually makes up its mind. If the input changes at that very moment, the output is no longer synchronous. This "moment of truth" is a very short window. For TTL flip-flops, it is of the order of 10 ps; for MOS devices, it is more like 50 ps to 100 ps. For purposes of this discussion, this timing window will be called "t".

Here are two extreme examples. In each case there is a need to synchronize asynchronous inputs that have no phase or frequency relationship with the computer clock.

- * Data signal derived from a disk, roughly 6 MHz with enough frequency modulation and jitter to make it totally asynchronous to the 10-MHz computer clock.

Every time the Data Signal falls into the "window", the probability of hitting the window is t divided by the clock period, or even simpler: clock frequency times t .

$$\begin{aligned} M &= \text{Metastable Rate} = f(D) \cdot f(C) \cdot t \\ f(D) &= \text{Device Frequency} = 6 \text{ MHz} \cdot 10 \text{ MHz} \cdot 10 \text{ ps} \\ f(C) &= \text{Clock Frequency} = 600 \text{ Hz} \end{aligned}$$

The synchronizer goes metastable 600 times per second.

- * Keyboard entry: one keystroke per second synchronized with a 100-kHz clock.

$$M = \text{Metastable Rate} = 1 \text{ Hz} \cdot 10^5 \text{ Hz} \cdot 10 \text{ ps} = 10^{-6} \text{ Hz}$$

The synchronizer goes metastable with a statistical probability of once per 10^6 sec., i.e., once every six weeks (assuming 5 eight-hour days/week).

"Going metastable" here means that the synchronizer output is within a mid-level or oscillation range for an unpredictable time. Most occurrences will last less than 50 ns, but may occasionally last much longer - perhaps many microseconds. This certainly can upset the timing chain.

A metastable latch or flip-flop has an unpredictable delay and will therefore change its output at a time that differs from the value obtained from the worst case timing analysis. In a slow system this usually doesn't matter, but in a fast system it can lead to a "crash".

Solutions

We cannot eliminate the basic problems but we can reduce the probability in two ways:

- 1) Cascade two or more synchronizer flip-flops, which is the method employed in all "metastable free" systems.
- 2) Use flip-flops that are less prone to metastable operation than the popular 74S74. For example, the AMD Am29821-26 registers are "metastable-hardened". They show no oscillations and only a minimal increase in output delay when hit right in the window.

Metastable operation is an inherent, so far incurable disease of all asynchronous interfaces. Once understood, the problem can be handled by reducing its probability to an acceptable level. AMD's Am29821-26 registers vastly minimize this problem. The Am29800 registers, while not totally immune to this problem, are "metastable hardened" by means of a unique circuit design that reduces both the probability and the delay of any metastable condition. An artificially induced metastable condition that failed to produce any output oscillation merely increased the clock-to-output delay by 6 ns. This is an improvement of many orders of magnitude over previously available designs.

III. Bus Structures

In the course of interfacing, special attention must be applied to the different standard bus systems. They are standard because they have the acceptance and support of the industry; different because each evolved to satisfy a particular function.

Bus systems are developed to increase the flexibility and expandability of the mother board. They are also developed to cut cost and hardware overheads. Since each bus structure has its unique area of operation, each must be carefully matched for proper interfacing.

The following are brief explanations on different major bus structures and areas of their involvement:

Big MULTIBUS vs. Little MULTIBUS

The big MULTIBUS architecture connects the single-board computer with its CPU, memory and I/O to the outside world. The so-called little MULTIBUS is the bus between the CPU and the other components on the single-board computer.

MULTIBUS

The MULTIBUS evolved, in structure, from 8-bit to 16-bit capability; and in architecture, expansion via the iSBX bus and Multi-channel bus, and iLBX bus. Expansions were necessary because a single system bus structure was no longer capable of supporting the demands of today's high-speed, high-performance VLSI microprocessor technology, and its increasingly complex configurations.

iLBX

The iLBX bus provides users the architectural solution that extends the high-performance benefits of a processor's on-board local bus to off-board memory resources. Powerful iLBX system modules can be created using the bus to connect a single board computer and multiple memory cards. The iLBX bus preserves the advantages in performance and architecture of on-board memory, while allowing a wide range of memory capabilities to match application requirements.

iSBX

The iSBX bus provides users with a low cost, on-board expansion solution for Multibus single board computers. The iSBX boards allow addition in the areas of parallel I/O, serial I/O, peripheral controllers, and high-speed math, without going to the expense of additional full Multibus board. iSBX bus compatible boards enable users to buy exactly the capabilities they require for their Multibus systems, which keeps both system size and system cost at a minimum.

Multichannel

Multichannel bus provides users with a separate path for DMA I/O block transfers. The new VLSI microprocessors that process data at very high rates require the connection of numerous high-speed I/O devices to the system bus. The Multichannel bus provides for high-speed block transfers of data over an 8/16-bit wide data path between peripherals and single board computer resources.

Motorola has its own equivalents to these busses. The signals and interfaces are slightly different but the functions are similar, and the objective of higher throughput is the same.

3.4.2 INTERFACING TO THE 8086/80186/80286

Overview

The 8086 is a general-purpose 16-bit microprocessor CPU. The CPU has a 16-bit data bus multiplexed with 16 address outputs. There are 4 additional address lines (segment addresses which are multiplexed with STATUS) that increase the memory range to 1 Mbyte. 8086 addresses are specified as bytes. In a 16 bit word, the least significant byte has the lower address and the most significant byte has the higher address. This is compatible with 8080, 8085, Z80 and PDP11 addressing schemes but differs from the Z8000* and 68000 addressing.

The data bus is "asynchronous", i.e., the CPU machine cycle can be stretched without clock manipulation by inserting Wait states between t2 and t3 of a read or write cycle to accommodate slower memory or peripherals. Unlike the 68000, the 8086 has separate address space for I/O (64 kBytes).

The 8086 can operate in MIN. or MAX. mode. Maximum mode offloads certain bus control functions to a peripheral device and allows the CPU to operate efficiently in a co-processor environment. A brief discussion on both the MIN. and MAX. modes are as follows:

MIN. mode: I/O addressing is defined by a HIGH or the IO/M output, and activated by the RD output for reading from memory, or I/O or activated by the WR output for writing to memory or I/O.

DMA: The Bus is requested by activating the HOLD input to the 8086. Bus Grant is confirmed by the HLDA output from the 8086.

MAX. mode: I/O operation is controlled by two outputs (8086 plus 8288)
IORC: active during Read from I/O
IOWC: active during Write to I/O
MRDC: active during Read from memory
MWTC: active during Write to memory

DMA: The Bus is requested and Bus Grant is acknowledged on the same pin (RQ/GT0 OR RQ/GT1) through a pulsed handshake.

Interrupts In Min. and Max. Modes:

Interrupt is requested by activating the INTR or NMI inputs to the 8086.

Interrupt is acknowledged by the $\overline{\text{INTA}}$ pin on a MIN. mode 8086 or by the $\overline{\text{INTA}}$ pin on the 8288 in MAX. mode.

Note: There is no $\overline{\text{RD}}$ or $\overline{\text{IORC}}$ during the interrupt-acknowledge sequence.

8086 and Z8000 Peripherals

Z8000 peripherals may be interfaced to the 8086 in many ways. One thing distinguishes the Z80xx peripherals from others is that they require data to be valid prior to the falling edge of $\overline{\text{DS}}$ during a write. The way to go about meeting this requirement is to use a 74LS74 to delay the falling edge of WR and OR RD with WR. By using the asynchronous ready mode of the 8284A and the other half of the 74LS74, one Wait State is inserted in both the read and write cycles, and thus stretches $\overline{\text{DS}}$ to meet the minimum pulse width requirements. In the examples that follow, one assumes that there are other devices in the system that require the use of the Am9519A. In another example, the Am9519A is not needed to interface the peripherals interrupt structure. $\overline{\text{INTA}}$ is tied HIGH since the vector will be supplied by the Am9519A. The interrupt service routine can read the peripheral status to determine the cause of the interrupt. The service routine should also clear IP and IUS before returning to mainline code. Note, since $\overline{\text{INTA}}$ is tied HIGH, IUS needs not be cleared. If $\overline{\text{INTA}}$ is activated, IUS must also be cleared.

Many of the design examples in this book use PAL devices. PAL devices, in many cases, will be the most cost-effective solution. The SSI examples show what is being accomplished much clearer than by using a PAL device, which is a black box to many. Gates are often left over from one circuit which can be used to implement other circuits. Whether to use a PAL device or do a discreet design is an individualistic choice and is not always clear-cut. As a general rule, if you can replace four or more packages, use a PAL device.

8086 and Am7990 LANCE

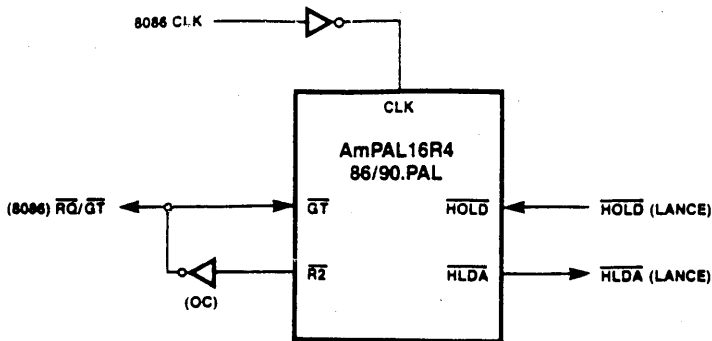
The LANCE, Am7990, has been designed to be interfaced easily with the popular 16-bit microprocessors (8086/80186, 68000, Z8000, LSI-11). Most of the interface logic is embedded inside the chip and is program selectable.

Although the LANCE itself has a multiplexed bus, it can easily be interfaced to demultiplexed buses with a minimal amount of effort. The following designs assume that the processor and the LANCE reside on the same board. Address buffers and data transceivers are set up to be shared between the processor and the LANCE. All of these designs use PALs to reduce the parts count.

The 8086 to LANCE interface requires a different Bus Request handshake, depending on whether the 8086 is configured in

MAX. or MIN. mode. The 8086 has a bidirectional signal for both Bus Request and Bus Grant ($\overline{RQ}/\overline{GT}$). Both Bus Request ($\overline{RQ}$), and Bus Grant ($\overline{GT}$) to/from 8086 are one CPU clock wide, and are synchronous to the CPU clock. Figure 3-50 shows a PAL design for the conversions in MAX. mode. This PAL device is utilized to include other external logic requirements for interfacing the LANCE to 8086. The interface diagram is similar to the one for the 80186 to LANCE interface except for the changes made in programming the PAL device. Interface timing diagram is shown in Figure 3-51.

Figure 3-52 shows a block diagram on the 8086 to Am7990 interface, in MIN. mode. The interface also employs a PAL device to minimize parts count. The PAL equations are given in Figure 3-53.



AMPAL16R4
PAT001

RASOUL OSKOUY
MARCH 20, 1984

8086 RQ/GT CONVERSION TO LANCE HOLD/HLDA
ADVANCED MICRO DEVICES

CLK	NC	HOLD	$\overline{GT}$	NC	HC	NC	NC	NC	GND
NC	NC	NC	HLDA	D2	R2	R1	NC	NC	VCC

R1 := HOLD
D2 := R1
R2 := $R1 \cdot \overline{D2} + \overline{R1} \cdot D2$
HLDA := $\overline{GT} \cdot R2 + HLDA \cdot \overline{D2}$

This PAL converts the request and grant ($\overline{RQ}/\overline{GT}$) of 8086, when configured in maximum mode, to the LANCE Am7990 HOLD/HLDA. Both request (input to 8086) and grant (output from 8086) are one clock wide and are synchronous to the CPU clock.

Figure 3-50. 8086 $\overline{RQ}/\overline{GT}$, Am7990 HOLD/HLDA Conversion PAL Device

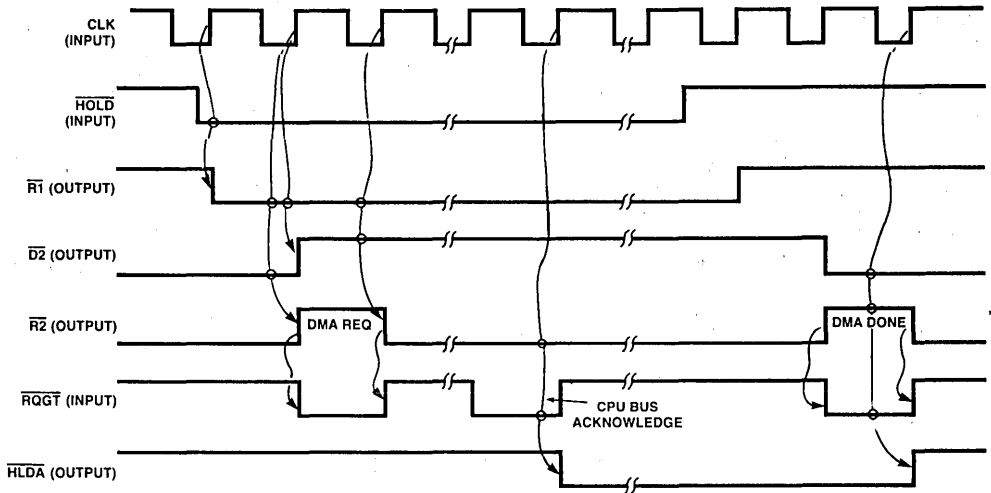


Figure 3-51. AmPAL16R4, 8086 (Max. Mode) LANCE Interface Timing Diagram

02188A-20

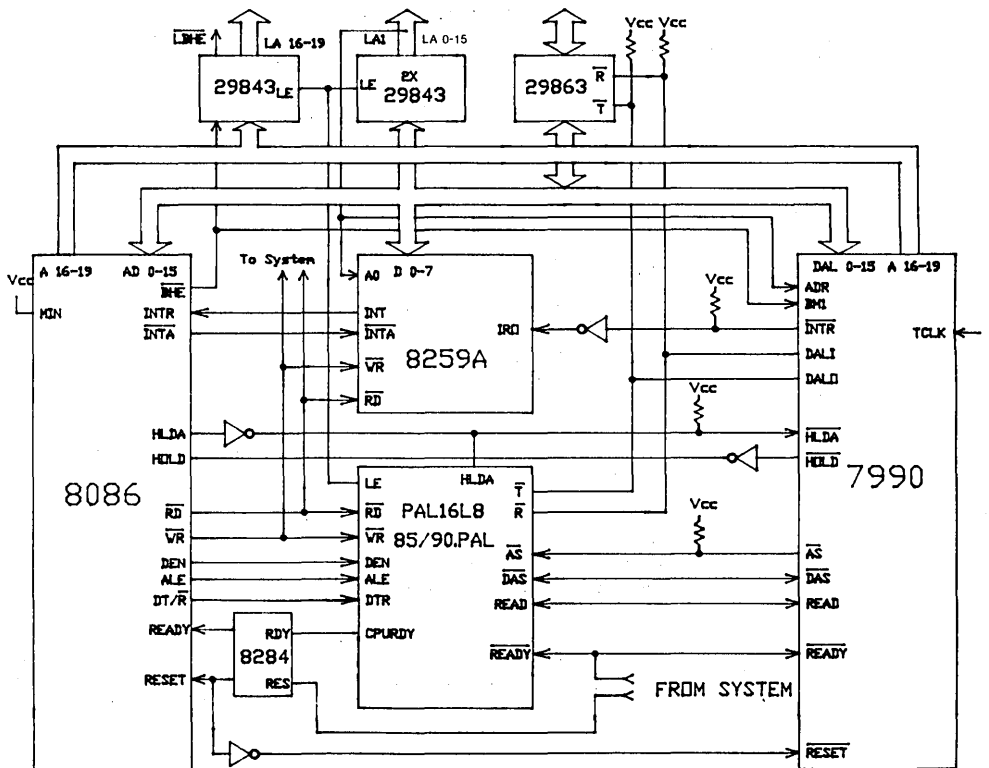


Figure 3-52. Am7990 to 8086 Interface

02188A-21

PAL16L8

PAT

FILE: 86-90-A.PAL

KHUYNH NGUYEN

AUG '85

8086 MINIMUM MODE TO LANCE INTERFACE
AMD

ALE /AS DTR NC NC DEN NC /READY HLDA GND
NC /DPURDY READ /R /T /DAS /WR /RD LE VCC

/LE = /ALE + /AS

/CPURDY = /READY

; CPU (86) IS BUS MASTER

IF(/HLDA) DAS = RD + WR

IF(/HLDA) /READ = DTR

IF(/HLDA) T = DTR

IF(/HLDA) R = /DTR * DEN

; LANCE IS BUS MASTER

IF(HLDA) RD = READ * DAS

IF(HLDA) WR = /READ * DAS

3

Figure 3-53. Source Listing for Example of Figure 3-52

80186 to Am7990 LANCE Interface

Similar to the 8086 to Am7990, this interface uses a PAL device design to reduce the parts count. Figure 3-54 shows the interface block diagram. 80186/LANCE address and data buses can be connected directly together since they both have multiplexed buses. It seems natural to program the LANCE for ALE output. However, the PAL device equations or indeed a discrete design is easier if $\overline{AS}$ (CSR3, ACON=1) is used. This

is because the LANCE tristates ALE, the 186 does not. The $\overline{INTR}$, $\overline{READY}$, and $\overline{HOLD}$ signals from the LANCE are open drain and should be pulled up. The $\overline{BM1}$ signal from the LANCE or $\overline{BHE}$ from the 186 along with A0 can be used to decode the data transfer type (Word/Byte). The external address buffers and data transceivers are enabled by the LANCE and the 186. The buffers and transceivers are enabled by whichever device is the master. The user should program the BCON, BSWP to 0, and ACON to 1 in CSR3.

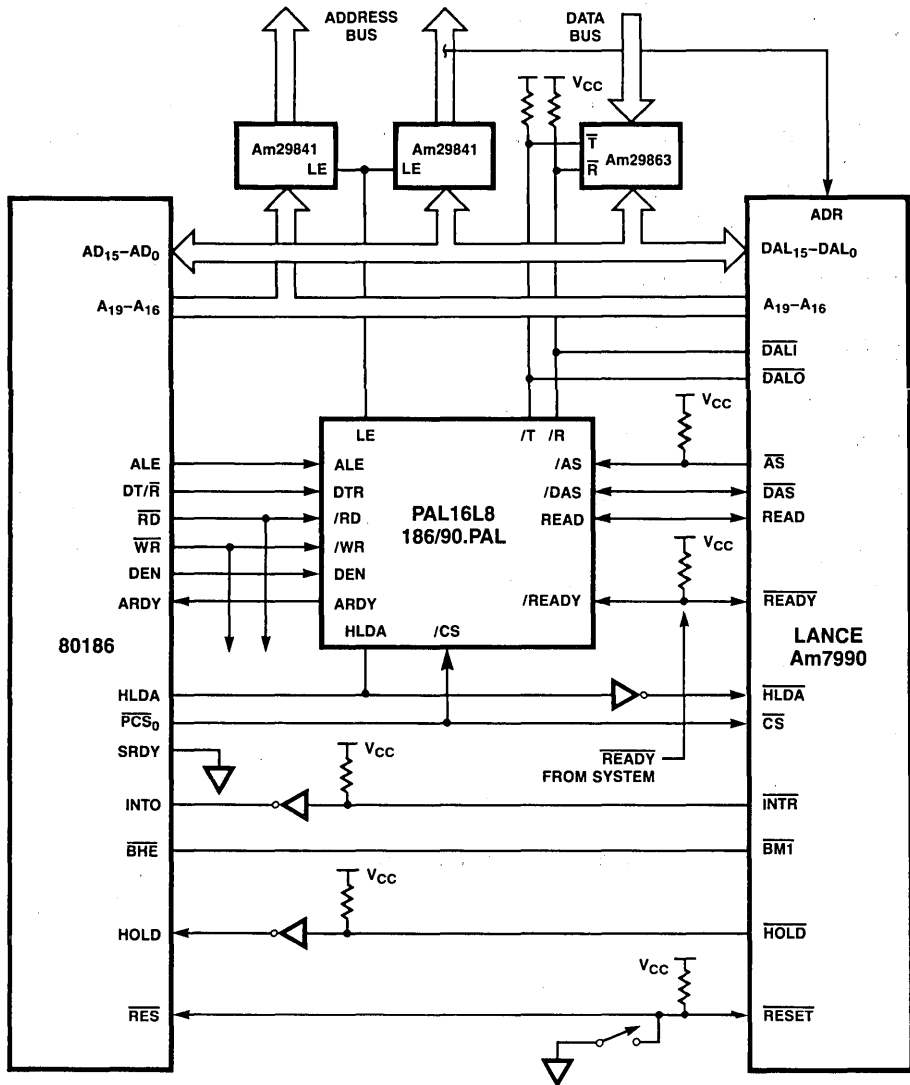


Figure 3-54. 80186 to Am7990 Interface

02188A-22

8086 and Am9516 Universal DMA Controller Interface

Am9516 in MIN. Mode

Figure 3-55 illustrates the interface of the Am9516 to the 8086 in the MIN. mode configuration. Figure 3-57 illustrates the interface in MAX. mode. The interfaces could be accomplished by using rather complex implementation of standard SSI/MSI logics. Examples here replace the logic portion with a PAL device. The MIN. Mode uses the PAL device 16L8. This is a good example of "GARBAGE COLLECTION". It reduces the amount of real estate, interconnections, parts, and part types.

Both interface examples accomplish two major functions. First, when the Am9516 is bus master, it converts $\overline{RD}$ and $\overline{WR}$ into $R/\overline{W}$ and $\overline{DS}$, and vice versa when the Am9516 is not the bus master. Secondly the buffer controls, $\overline{TBEN}$ and $\overline{REBN}$, are generated from $\overline{DEN}$ and $\overline{DT/R}$.

The two examples show different types of latches and transceivers and there are many more to choose from. The designer selects those that best meet system requirements while trying to minimize the number of different parts that must be stocked. Figure 3-56 shows the PAL equations for this example.

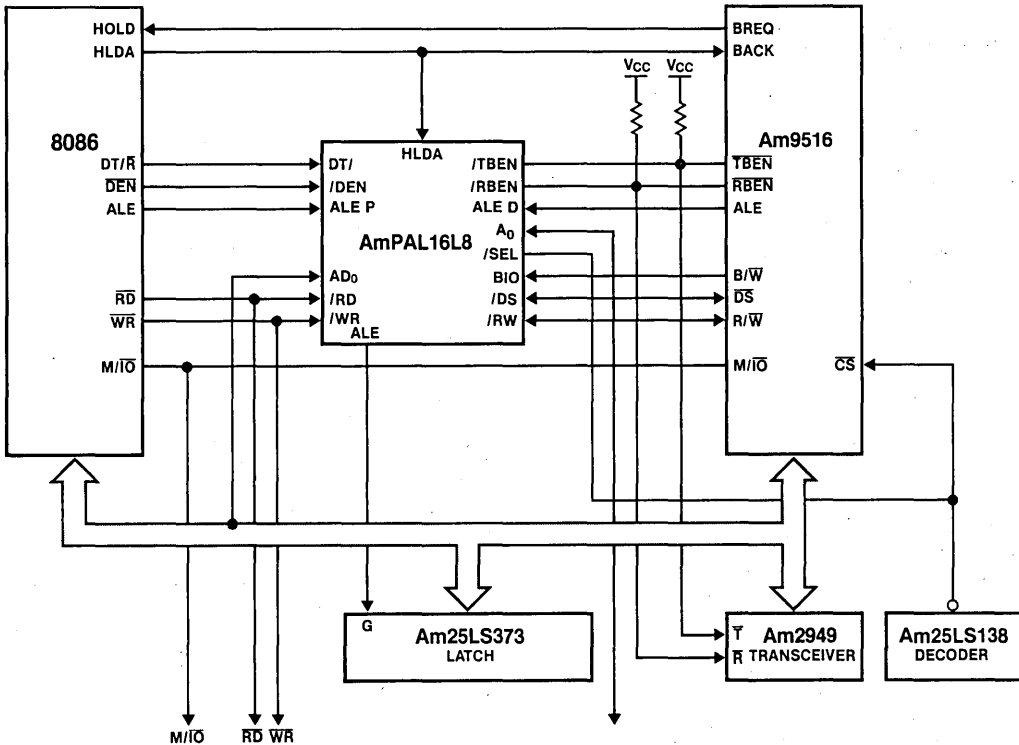


Figure 3-55. The Am9516 UDC to 8086 CPU Interface (Minimum Mode)

AmpAL16L8 PALASM FILE

PAL16L8

PAT001

Am9516 to 8086 min mode interface chip
Advanced Micro Devices

NC ALED ALEP HLDA BW ADO DT /DEN /SEL GND
NC /RBEN /RD ALE AO /RW /DS /WR /TBEN VCC

```
IF (/HLDA) DS = RD + WR
IF (/HLDA) RW = DT
IF (/HLDA) TBEN = /DT * /SEL * DEN
IF (/HLDA) RBEN = DT * /SEL * DEN
IF (HLDA) RD = /RW * DS
IF (HLDA) WR = RW * DS
ALE = /ALEP * /ALED
AO = /ADO * /BW * HLDA * ALED      +
    ADO * BW * HLDA * ALED        +
    /ADO * /HLDA * ALEP            +
    AO * /ALEP + AO * /ALED
```

DESCRIPTION

THIS PAL CONVERTS THE CONTROL SIGNALS TO INTERFACE THE 8086 IN MIN MODE TO THE Am9516 DMA CONTROLLER. ANOTHER EXAMPLE SHOWS HOW THIS IS DONE IN MAX MODE.

Figure 3-56. Source Listing for Example of Figure 3-55.

Am9516 in MAX. Mode

This MAX. mode interface between the 8086 and Am9516 (Figure 3-57) also uses PAL devices to reduce component count.

The example makes several assumptions which result in a slightly more complex design than absolutely necessary. The 8086 is assumed to be in MAX. Mode, and the design has to be compatible with a MULTIBUS or similar interface; the 16R4 and 8288 could be eliminated if this is not the case.

The 16R6 (Figure 3-57) is a PAL device that performs a function similar to the 8288, that is, it converts the processor's status signals and clock into control signals. In this case, the signals are, R/W, and Data Strobe (DS), Interrupt Acknowledge (INTA) and Peripheral Acknowledge (PACK). This PAL device, Figure 3-59, basically generates ZBUS signals for Zilog peripherals. In the example, it connects to the Am9516 (UDC). The DMA Controller is very similar to the AmZ8016 except its bus interface has been modified to interface to non-multiplexed buses. This was changed from the previous design using a 16R8 to eliminate problems due to skew between OSC and CLK.

The 16R4, Figure 3-58, has two functions. The first is connecting the MIN. Mode protocol of the Am9516 or similar

device to the MAX. Mode protocol for bus exchange. The 74LS03s are used to aid in this function. The timing waveform illustrates what happens in detail. The basic philosophy is that a rising edge on the HOLD input generates a pulse that is one clock wide. The CPU samples this pulse and, in response, issues a one-clock-wide pulse. The 16R4 uses this response pulse to generate HLDA. When the Am9516 has completed the necessary transfer, HOLD transitions HIGH to LOW. This generates another pulse to the CPU signalling that the Am9516 is done and that the CPU may continue.

The second function of the 16R4 is the conversion of R/W, DS and M/I $\bar{O}$ into the MULTIBUS-compatible signals /MRDC, /MWTC, /IORC, and /IOWC, when HLDA is HIGH. It is possible to collapse the 74LS03s into the PAL device, thus reducing the external logic required to only one open collector inverter. The disadvantage, however, is that it adds two additional clocks for the bus exchange overhead, one during acquisition of the bus and one on bus release. For block transfers, this is not significant but it may be undesirable when performing single transfers, or short burst, or when in Demand Mode. To eliminate the 74LS03, change the equation for R2 to

$$/R1 \cdot /D2 + R1 \cdot D2,$$

and then drive the [/RQ/GT] line with a 74LS05 from the R2 output.

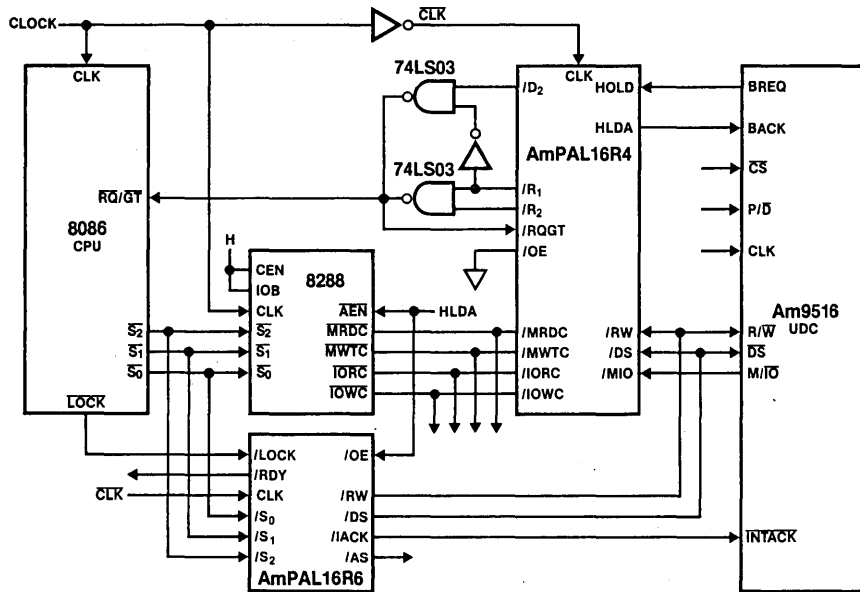


Figure 3-57.

02188A-30

AmPAL16R4 PALASM File

B> Type Am9516 PAL
PAL16R4

8086 to Am9516 interface
Advanced Micro Devices

CLK /RQGT HOLD NC NC NC /RW /DS MIO GND
/OE /MWTC /MRDC HLDA /D2 /R2 /R1 /IOWC /IORC VCC

IF (HLDA) IORC = /MIO * DS * /RW
IF (HLDA) IOWC = /MIO * DS * RW
IF (HLDA) MRDC = MIO * DS * /RW
IF (HLDA) MWTC = MIO * DS * RW

R1 := HOLD

R2 := /R1

D2 := R1

/HLDA := /R1 + /D2 * /HLDA +
/RQGT * /HLDA

DESCRIPTION

THIS DEVICE CONVERTS THE MIN MODE SIGNALS HOLD AND HLDA TO THE MAX MODE /RQGT PROTOCOL. ADDITIONALLY IT GENERATES THE 8288 EQUIVALENT CONTROL OUTPUTS /MRDC, /MWTC, /IORC, AND /IOWC. THIS PAL WAS USED TO CONNECT THE Am9516 TO THE 8086 IN MAX MODE.

Figure 3-58. Source Listing for Example of Figure 3-57

AmPAL16R6

PAT 005

8086 TO 85XX PERIPHERAL INTERFACE
ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION

BY JOE BRICCH 5/10/83

& JAMES WILLIAMSON 7/21/83

CLOCK	RESET	CLK	/S0	/S1	/S2	/LOCK	NC	NC	GND
/OE	/AS	/P1	/RW	/DS	/P0	/IACK	/RDY	CLKD	VCC
P0 :=	/RESET * S0 *	/P0 *	/P1						+
	/RESET * S1 *	/P0 *	/P1						+
	/RESET * S2 *	/P0 *	/P1						+
	/RESET * S0 *	P1							+
	/RESET * S1 *	P1							+
	/RESET * S2 *	P1							+
P1 :=	/RESET * P0 *	/P1							+
	/RESET * P1 *	S0							+
	/RESET * P1 *	S1							+
	/RESET * P1 *	S2							+
DS :=	/IACK * /P0 *	P1 *	S0 *	/S1 *	S2				+
	/IACK * /P0 *	P1 *	/S0 *	S1 *	S2				+
	IACK *	S0 *	S1 *	S2 *	P0 *	/P1 *	LOCK		+
	DS *	S0 *	S1 *	S2					
	DS *	S0 *	/S1 *	S2					
	DS *	/S0 *	S1 *	S2					
RW :=	S0 *	/S1 *	S2						
IACK :=	/RESET * S0 *	S1 *	S2 *	/P0 *	/P1 *	/LOCK			
	/RESET * IACK *	S0 *	S1 *	S2 *	P0 *	/P1 *	/LOCK		
	/RESET * IACK *	LOCK *	/DS						
	/RESET * IACK *	/LOCK *	DS *	/P0 *	P1				
RDY :=	/RESET * S0 *	/S1 *	S2 *	P0 *	P1				+
	/RESET * /S0 *	S1 *	S2 *	P0 *	P1				+
	/RESET * RDY *	S0 *	/S1 *	S2					+
	/RESET * RDY *	/S0 *	S1 *	S2					+
	/RESET * IACK *	S0 *	S1 *	S2 *	DS				+
	/RESET * RDY *	S0 *	S1 *	S2					
/CLKD = CLK									
AS = /CLKD * P0 * /P1 * /IACK * CLK									

DESCRIPTION

THIS PAL TRANSLATES 8086 BUS SIGNALS INTO COMPATIBLE SIGNALS FOR THE 9516. IT IS ALSO APPLICABLE TO 85XX PERIPHERALS BY ALTERING /RW AND /DS TO /RD AND /WR. ONE FLIP FLOP IS AVAILABLE TO GIVE THE NECESSARY DELAY TO THE FALLING EDGE OF /WR. THE DATA STROBE TIMING FOR A WRITE CYCLE IS DELAYED UNTIL THE FALLING EDGE OF T2 TO MEET THE REQUIREMENTS OF THE 85XX PARTS. THIS DESIGN ASSERTS RDY TO DEMAND ONE WAIT STATE FROM THE 8086. THIS WAIT STATE IS NOT LONG ENOUGH FOR DESIGNS WHICH USE AN 8 MHZ 8086. THEREFORE, WITH AN 8 MHZ CPU, 85XXA PERIPHERALS SHOULD BE USED. AS AN ALTERNATIVE, THREE WAIT STATES CAN BE USED BY ALTERING THE RDY EQUATION. THIS PAL ALSO TRANSFORMS THE 8086 TWO CYCLE INTERRUPT ACKNOWLEDGE INTO A SINGLE CYCLE OF THE TYPE NECESSARY FOR 85XX PARTS. THIS IS MADE POSSIBLE BY SAMPLING THE LOCK STATUS, P0, P1, AND IACK SIGNALS.

NOTE; THE CLK SIGNAL MUST BE EXTERNALLY INVERTED.

Figure 3-59. Source Listing for Example of Figure 3-57.

80186 to Am9516 Universal DMA Controller

The addition of the Am9516 to an 80186 design is a natural choice in systems requiring four channels of DMA. Figure 3-60 shows the interface between the 80186 and the Am9516 with a PAL device. PCS₅ is programmed to provide a latched A₁ signal.

This interface accomplishes two major control transformations. First, it converts $\overline{RD}$ and $\overline{WR}$ into R/W and $\overline{DS}$ when the 80186 is Bus Master, and vice versa when the Am9516 is Bus Master. Secondly, the transceiver control signals, TBEN and RBEN, are generated from DEN and DT/R. This example shows only one possible configuration. Other configurations can be made as dictated by system requirements. A PAL device is used here to reduce board space.

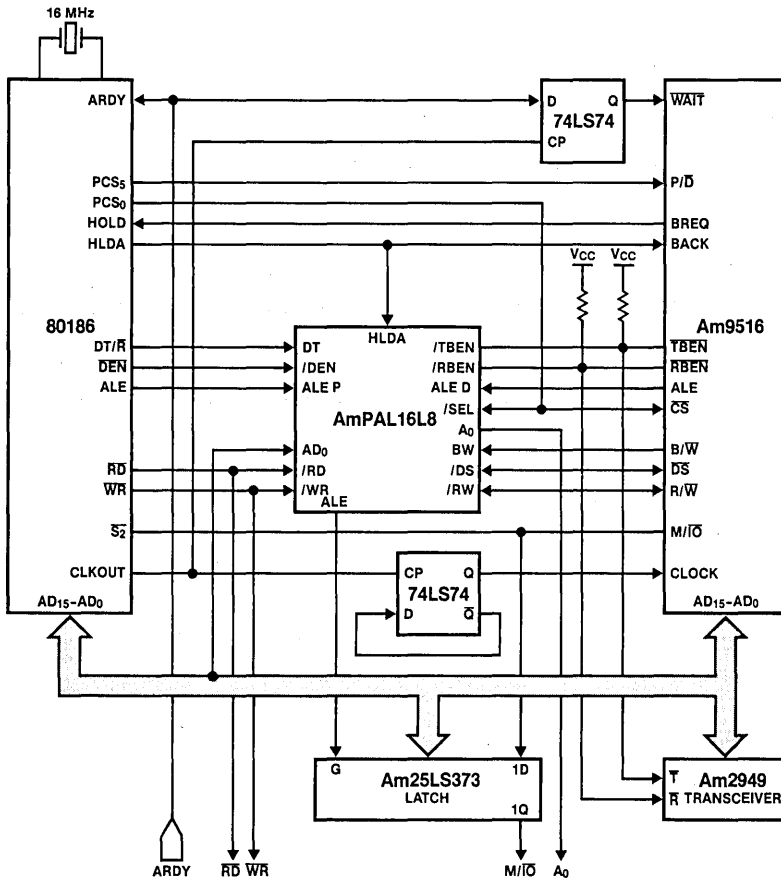


Figure 3-60. Am9516 to 80186 Interface

AmPAL16L8 PALASM File

PAL16L8

PAT001

Am9516 to 80186 interface chip

Advanced Micro Devices

NC	ALED	ALEP	HLDA	BW	ADO	DT	/DEN	/SEL	GND
NC	/RBEN	/RD	ALE	AO	/RW	/DS	/WR	/TBEN	VCC

```
IF (/HLDA) DS = RD + WR
IF (/HLDA) RW = DT
IF (/HLDA) TBEN = /DT * /SEL * DEN
IF (/HLDA) RBEN = DT * /SEL * DEN
IF (HLDA) RD = /RW * DS
IF (HLDA) WR = RW * DS
ALE = /ALEP * /ALED
AO = /ADO * /BW * HLDA * ALED +
    ADO * BW * HLDA * ALED +
    /ADO * /HLDA * ALEP +
    AO * /ALEP +
    AO * /ALED
```

DESCRIPTION

THIS PAL CONVERTS THE CONTROL SIGNALS TO INTERFACE THE 80186 TO THE Am9516 DMA CONTROLLER. OTHER EXAMPLES SHOW HOW THIS IS DONE FOR THE 8086 IN MIN AND MAX MODES.

Figure 3-61. Source Listing for Example of Figure 3-60

iAPX286 to Am9568 Data Ciphering Processor Interface

This interface is designed for an 8-MHz CPU where the DCP is synchronously operating at the maximum clock rate of 4 MHz. Block diagram for the interface is shown in Figure 3-62. The Am9568 requires a narrower width of address strobe than the Am9518. This works comfortably with the 60-ns address strobe width of an 8-MHz CPU.

The MULTIBUS Mode Select input of the Bus Controller 82288 is tied LOW to optimize the command and control signals for short bus cycles. The Command Delay (CMDLY) becomes active-HIGH for one 16-MHz clock cycle whenever the DCP is selected to delay the Read and Write strobes by 125 ns. This satisfies the timing requirement of the minimum delay between ALE inactive and Read or Write strobe active of the DCP. An open-collector gate must be added to allow other peripherals to drive this input.

The ALE, $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ outputs of the 82288 are wired directly to the DCP. ALE strobes a D-Flip-Flop to store the state of Chip Select for the entire cycle.

$\overline{\text{Q}}_3$ and the latched Chip Select CSL are ANDed externally to generate the Synchronous Ready for the 82284. The 82284 samples the line at the falling edge of the clock. The registered

output $\overline{\text{Q}}_3$ is clocked with the rising edge of the same clock, thus satisfying the setup and hold time requirements of the 82284. Two Wait States are inserted.

Half of the PAL device operates as a bidirectional Address/Data Multiplexer. During the Address Latch Enable active phase, the state of A_1 and A_2 is transferred to the AD_1 and AD_2 pin of the PAL device. The DCP latches this two bit-address with the falling edge of ALE.

When $\overline{\text{IORC}}$ and CSL are active, the states of AD_1 and AD_2 are passed to D_1 and D_2 respectively. The DCP Register can be read. If $\overline{\text{IOWC}}$ and CSL are active, the data path is turned around: D_1 and D_2 are inputs, AD_1 and AD_2 are outputs.

The Address Hold Time of the PAL device is sufficient because the address information is passed to AD_1 and AD_2 whenever $\text{IORC} \cdot \text{CSL}$ or $\text{IOWC} \cdot \text{CSL}$ are not true, i.e. whenever data is not transferred between the CPU and the DCP.

The Read Data Hold Time requirement of 5 ns of the Am9568 is satisfied by the propagation delay of the PAL device.

The Read Data Hold Time requirement of 5 ns of the iAPX286 is also satisfied by the PAL device.

The Master Port Chip Select ($\overline{\text{MCS}}$) input of the DCP is connected to the unlatched address decoder output.

Figure 3-64 shows the source listing and Figure 3-65 provides the pin descriptions.

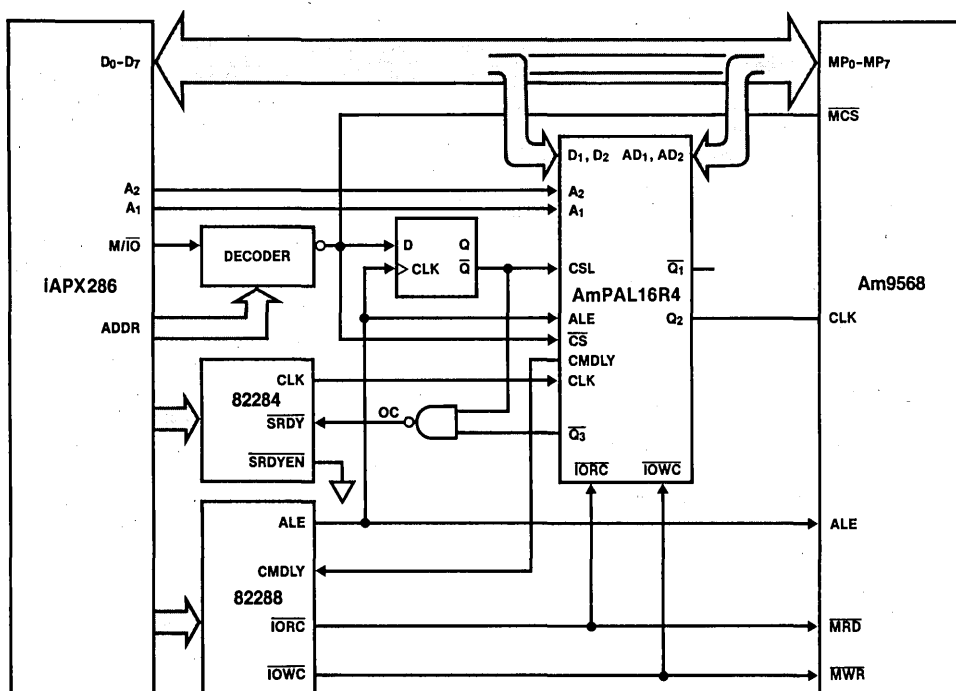


Figure 3-62. IAPX286-Am9568 Interface

02188A-43

The DCP Clock

The PAL device synchronizes the DCP clock to the Data Strobe $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ (Figure 3-63). It also divides the 16-MHz system clock (8-MHz CPU clock) down to the maximum DCP clock rate of 4 MHz. At this clock rate, the Data Strobe Delay to the DCP clock must be 0-30 ns. The Bus Controller is specified to generate a Data Strobe timing of 3-15 ns to the falling edge of CLK (16 MHz). Because of the higher propagation delay of a standard PAL device, the registered outputs are toggled at the rising edge of CLK before the Data Strobe becomes inactive. This gives additional 32.5 ns for the DCP clock signal path.

Q_1 to Q_3 are three outputs of the PAL device state machine. The registered outputs are clocked with the rising edge of the 16-MHz 82284 clock. Whenever ALE and $\overline{\text{CS}}$ are active, Q_1 to Q_3 are set to the initial state. Q_1 to Q_3 are outputs of a 3-bit down counter, with Q_3 as the most significant bit.

Q_3 is used to generate the $\overline{\text{SRDY}}$ signal for the 82284 as mentioned above.

Q_2 is the DCP clock. This design must guarantee that the minimum DCP clock HIGH or LOW time is at least 115 ns or two 16-MHz clock cycles. This is done by toggling Q_2 only

during phase 2 cycles of the CPU. The CPU design guarantees that there is always a phase 1 cycle between two phase 2 cycles.

Assuming a typical PAL device propagation delay of 25 ns, timing parameter tCDS (Time Clock Data Strobe) is 10.5 to 22.5 ns (3 + 32.5 - 25 ns to 15 + 32.5 - 25 ns). This satisfies the 0 to 30 ns requirement.

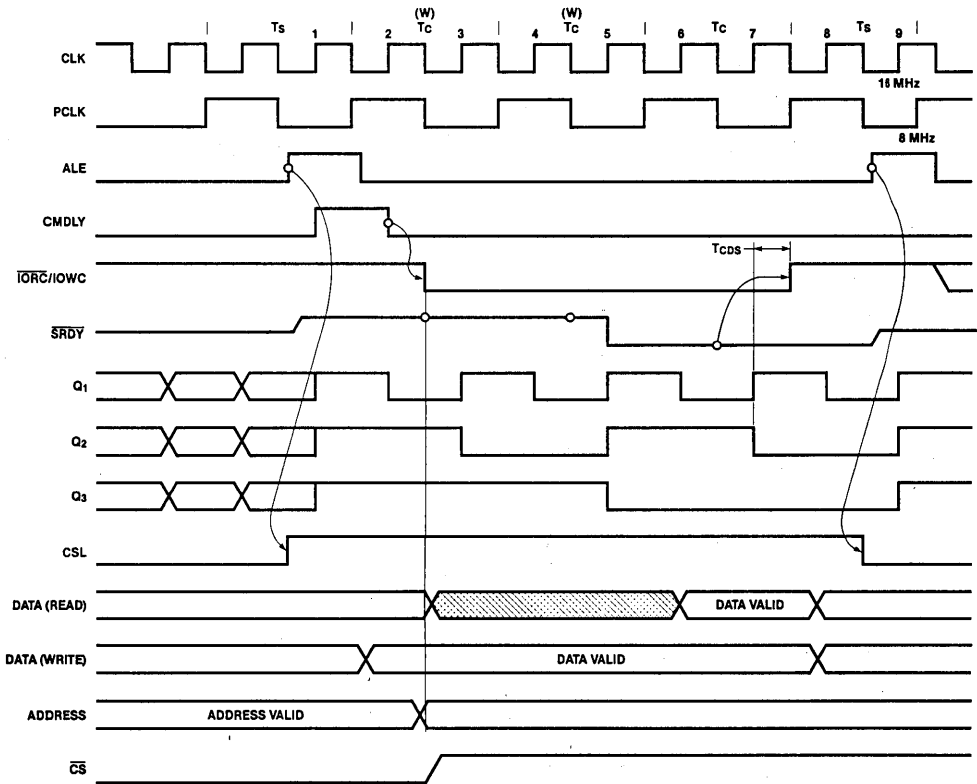
The AmPAL16R4 has active-LOW outputs. But one output, Q_2 , should be active-HIGH. The equation for Q_2 was derived to be

$$Q_2 = \text{ALE} * \text{CS} * Q_1 * \overline{Q_3} * \overline{Q_2}$$

To compensate for the inversion in the PAL device, either de Morgan Theorem or Karnaugh-Veitch diagrams can be used to convert it to the form shown in PAL device Design Specification.

Improvements

The DCP needs two Wait States only when the Control Registers are read. Data Register read or writes and Control Register reads can be executed with only one Wait State, which improves the Data Ciphering speed of this interface. The more sophisticated Wait control logic and the two external TTL gates can be integrated into one AmPAL22V10 device.



02188A-44

Figure 3-63. Timing Diagram

PAL16R4
DCP043
IAPX286 - Am9568 (DCP) INTERFACE DEVICE
ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION
JUERGEN STELBRINK 8-23-83

```
CLK /CS CSL ALE /IORC /IOWC A1 A2 NC GND
/OE D1 D2 /Q1 Q2 /Q3 CMDLY AD1 AD2 VCC

Q1      := ALE*CS + /Q1
/Q2      := Q1*/Q2*/ALE + Q1*/Q2*/CS + /Q1*Q2*/ALE + /Q1*Q2*/CS
Q3      := ALE*CS + Q1*Q2*Q3 + /Q1*Q2*Q3 + Q1*/Q2*Q3 + /Q1*/Q2*/Q3
/CMDLY   := /ALE+/CS

IF(CSL*IORC) /D1 = /AD1
IF(CSL*IORC) /D2 = /AD2
IF(CSL*/IORC) /AD1 = /A1*ALE + /D1*/ALE
IF(CSL*/IORC) /AD2 = /A2*ALE + /D2*/ALE
```

FUNCTION TABLE

CLK	/CS	CSL	ALE	/IORC	A1	A2	D1	D2	AD1	AD2	/Q1	/Q2	/Q3	CMDLY	
;				/											C
;				I											M
;	C	/	C	A	O			A	A	/	/	/			D
;	L	C	S	L	R	A	A	D	D	D	Q	Q	Q	L	
;	K	S	L	E	C	1	2	1	2	1	2	1	2	3	Y

;	C	L	H	H	H	L	L	Z	Z	L	L	L	L	L	H ; 1 (/CS ACTIVE)
	X	L	H	H	H	L	H	Z	Z	L	H	L	L	L	H
	X	L	H	H	H	H	H	Z	Z	H	H	L	L	L	H
	C	L	H	L	H	H	L	L	H	L	H	H	L	L	L ; 2 (WRITE CYCLE)
	X	H	H	L	L	L	L	L	H	L	H	H	L	L	L ; (READ CYCLE)
	C	H	H	L	L	H	L	L	H	L	H	L	H	L	L ; 3
	C	H	H	L	L	H	L	L	L	L	L	H	H	L	L ; 4
	C	H	H	L	L	H	L	H	H	H	H	L	L	H	L ; 5
	C	H	H	L	H	H	L	H	H	H	H	H	L	H	L ; 6
	C	H	H	L	H	H	L	L	L	L	L	L	H	H	L ; 7
	C	H	H	L	H	H	L	H	L	H	L	L	H	H	L ; 8
;	C	H	L	H	H	X	X	Z	Z	Z	Z	L	L	L	L ; 1 (NO /CS)
;	-----														

Figure 3-64. Source Listing for the Example of Figure 3-62

DESCRIPTION:

INPUT SIGNALS:

CLK 16 MHZ SYSTEM CLOCK OF THE 82284 SYSTEM TIMING CONTROLLER.
 THIS CLOCKS TRIGGERS THE D-FLIP-FLOPS OF FOUR PAL OUTPUTS

/CS ACTIVE LOW UNLATCHED CHIP SELECT OF THE ADDRESS DECODER

CSL ACTIVE HIGH LATCHED CHIP SELECT. IT HAS TO BE ACTIVE TO THE
 RISING EDGE OF ALE OF THE NEXT CYCLE

ALE ADDRESS LATCH ENABLE OF THE 82288 BUS CONTROLLER

A1,A2 DEMULTIPLEXED ADDRESS INPUTS. THEY CARRY THE 2-BIT REGISTER
 ADDRESS FOR THE DCP

/IORC INPUT/OUTPUT READ CONTROL OF THE 82288

/IOWC INPUT/OUTPUT WRITE CONTROL OF THE 82288

OUTPUT SIGNALS:

/Q1 INTERNAL STATE SIGNAL. IT IS DIVIDED BY TWO FROM CLK AND
 SYNCHRONIZED TO ALE

/Q2 INTERNAL STATE SIGNAL. IT IS DIVIDED BY TWO FROM /Q1 AND
 SYNCHRONIZED TO ALE. IT IS THE INVERTED DCP CLOCK (4MHZ).
 THE RIGHT EDGE OF Q2 IS SYNCHRONOUS TO THE DATA STROBES
 /IORC AND /IOWC, IF TWO WAIT STATES ARE INSERTED.

/Q3 INTERNAL STATE SIGNAL. IT IS DIVIDED BY TWO FROM /Q2 AND
 SYNCHRONIZED TO ALE. IT IS USED TO GENERATE THE SYNCHRONOUS
 READY (/SRDY) FOR THE 82284. EXTERNALLY IT HAS TO BE
 LOGICALLY AND'ED WITH THE THE LATCHED CHIP SELECT (CSL).

CMDLY COMMAND DELAY GOES ACTIVE FOR ONE CLOCK WIDTH TO DELAY THE
 DATA STROBES. THE AM9568 REQUIRES A DELAY BETWEEN ALE
 INACTIVE AND DATA STROBE ACTIVE.

BIDIRECTIONAL SIGNALS:

D1,D2 DEMULTIPLEXED DATA BUS LINES TO 8086 CPU

AD1,AD2 MULTIPLEXED ADDRESS/DATA BUS LINES FOR THE DCP

Figure 3-65. Pin Description for the Example of Figure 3-62

Am8530 to 80286 Interface

The Am8530 is a high-speed, dual-channel serial communications controller that supports a variety of advance communication protocols. It supports data rates up to 1.5M bps.

This design note shows an Am8530 to an 80286 CPU interface when no interrupts are used (Figures 3-66 and 3-67). The design is for a CPU running at 5 MHz with a 100-ns system clock cycle time. This clock is used to generate the clock for the 6-MHz Am8530 SCC.

Chip select generated by the external decoder in an 80286 system that is enabled by the M/IO- line will go inactive with the falling edge of Tc-phase 1. However, the chip enable for the

SCC should stay valid for the entire read or write cycle. The PAL device generates the chip enable signal for the SCC using a structure equivalent to two cross-coupled NOR gates.

The Am8530 requires a minimum pulse width of 250 ns for $\overline{RD}$ and $\overline{WR}$ control signals. For a 100-ns cycle time 80286 with no wait state, these controls will be active for 200 ns only. SRDY control is generated for the 82284 bus controller to guarantee minimum $\overline{RD}$ and $\overline{WR}$ pulse widths for the Am8530.

CMDLY is not used in this design because the address to read or write control setup time required is only 80 ns for the Am8530; with a 100-ns system cycle time, the time between ALE valid and $\overline{IORC}$ or $\overline{IOWC}$ valid is 100 ns. Even after accounting for the address latch delay, 80-ns timing will still be met.

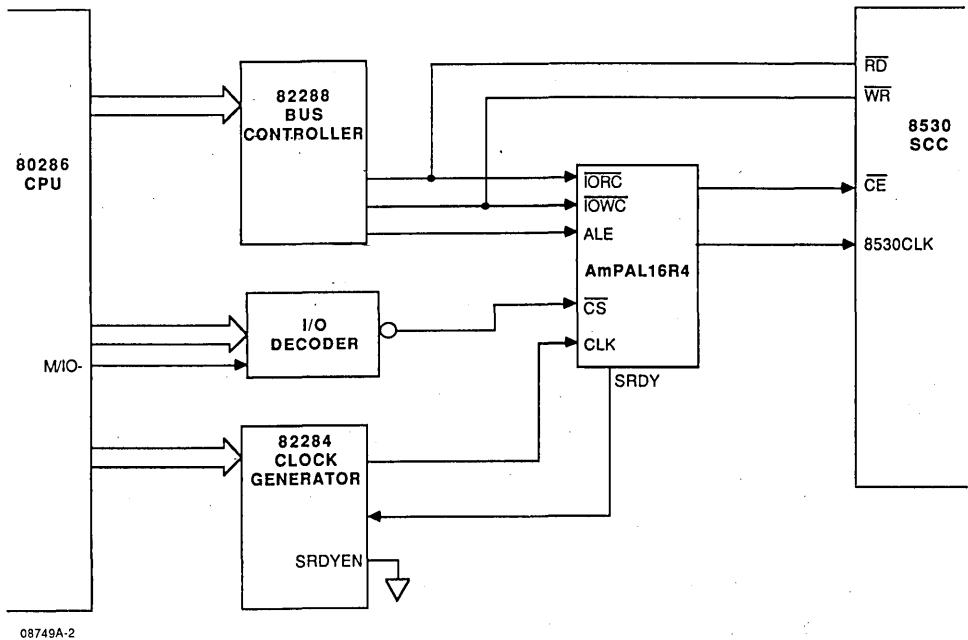


Figure 3-66. Am8530—IAPX286 Interface

" THIS PLPL FILE IS FOR A 16R4 THAT IMPLEMENTS THE LOGIC
NECESSARY TO INTERFACE AN Am8530 (SCC) TO AN 80286 SYSTEM. "

DEVICE Am8530_to_80286 (PAL16R4)

PIN

CLK	= 1	VCC	= 20
/CS	= 2	/CE	= 19
ALE	= 3	/INTSO	= 18
/IORC	= 4	/Q[1]	= 17
/IOWC	= 5	/Q[2]	= 16
NC1	= 6	/Q[3]	= 15
NC2	= 7	NC5	= 14
NC3	= 8	/SRDY	= 13
NC4	= 9	NC6	= 12
GND	= 10	NC7	= 11 ;

BEGIN

" CHIP ENABLE FOR THE Am8530 IS DERIVED FROM ALE AND
THE EXTERNAL DECODER CHIP SELECT OUTPUT. "

CE = CS * ALE + INTSO ;
INTSO = /(IORC * IOWC) + CE ;

" SYNCHRONIZE THE COUNTER WITH ALE. "

IF (CS * ALE) THEN Q[3:1] = 7 ;

" COUNT DOWN SPECIFICATION FOR THE COUNTER. "
" Q2 OF THE COUNTER IS THE CLOCK INPUT OF THE Am8530. "

CASE (Q[3:1])

BEGIN

0) Q[3:1] := 7 ;
1) Q[3:1] := 0 ;
2) Q[3:1] := 1 ;
3) Q[3:1] := 2 ;
4) Q[3:1] := 3 ;
5) Q[3:1] := 4 ;
6) Q[3:1] := 5 ;
7) Q[3:1] := 6 ;
END ;

" SRDY IS GENERATED FOR WAIT STATE GENERATION, IT IS SENT
TO THE 82284 CLOCK GENERATOR. "

SRDY = /(Q[3] * CE) ;

END.

Figure 3-67. Source Listing for AmPAL16R4 for the Example of Figure 3-66

Am9518 to 80286 Interface

The Am9518 Data Ciphering Processor encrypts and decrypts data using the National Bureau of Standards encryption algorithm. The DCP can be operated in either Multiplexed-control or the Direct-control mode. This design shows the logic necessary to interface the Am9518 Data Ciphering Processor to an 80286 microprocessor system with a system clock cycle time of 100 ns. The DCP is operated in the Multiplexed-control mode with the C/K- pin tied LOW. (A HIGH level on C/K- will place the device in Direct-control mode. This mode is suitable for ciphering data using a high-speed controller for high-speed applications.)

Two PAL devices are used to interface the Am9518 DCP to the 80286. The DCP has a multiplexed address/data port. Note that only two bits (MP2 and MP1) are used for DCP register

addressing. PAL device 1 performs the necessary address/data multiplexing since the 80286 has non-multiplexed address and data busses. Memory read/write and data strobe control signals for the DCP are also generated by this device.

PAL device 2 generates the clock for the Data Ciphering Processor by dividing the system clock that is synchronized with ALE and $\overline{CS}$ active. It also generates CMDLY for the 82288 bus Controller to insert a delay between $\overline{MAS}$ inactive and $\overline{MRD}$ or $\overline{MWR}$ active for the Am9518. Note that ALE cannot be inverted and tied to the Am9518 Master Address Strobe. Master Address Strobe requires a minimum width of 115 ns—device 2 generates $\overline{MAS}$ to meet this width requirement. This device also generates the SRDY signal for the 82284 Clock Generator to insert wait states during Am9518 read and write cycles.

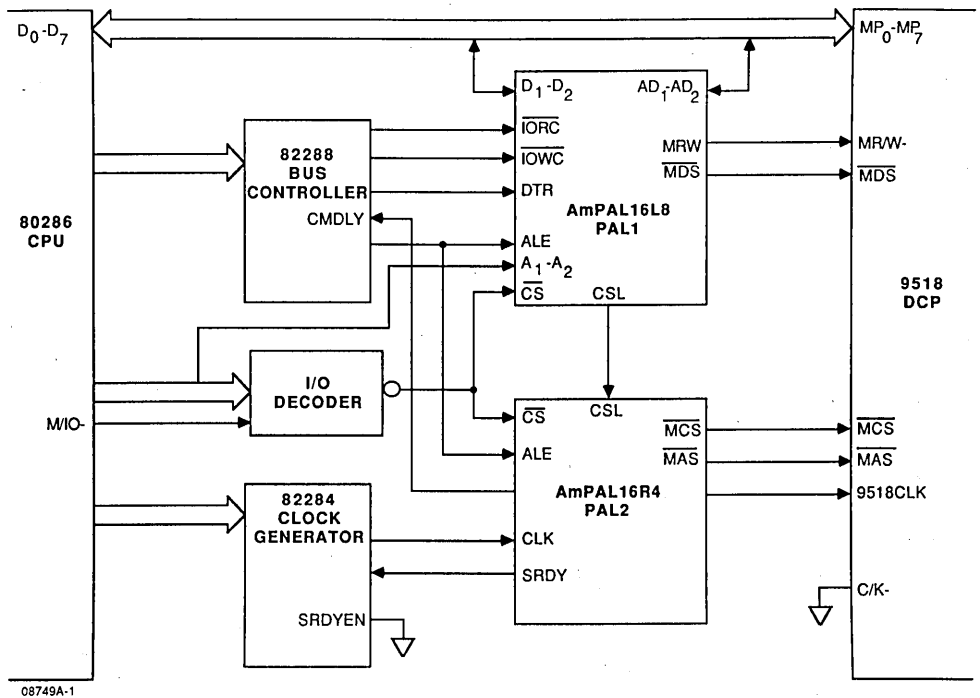


Figure 3-68. Am9518—IAPX286 Interface

" THIS PLPL FILE IS FOR A 16R4 PAL DEVICE THAT IMPLEMENTS
PART OF THE LOGIC NECESSARY TO INTERFACE AN Am9518 (DCP)
TO AN 80286 SYSTEM. ANOTHER PAL DEVICE (16L8) IS NEEDED
FOR THE COMPLETE DESIGN. "

DEVICE Am9518_TO_80286_2 (PAL16R4)

PIN

CLK	= 1	VCC	= 20
/CS	= 2	/MCS	= 19
ALE	= 3	/MAS	= 18
CSL	= 4	/Q[1]	= 17
NC1	= 5	/Q[2]	= 16
NC2	= 6	/Q[3]	= 15
NC3	= 7	/CMDLY	= 14
NC4	= 8	/SRDY	= 13
NC5	= 9	NC6	= 12
GND	= 10	NC7	= 11 ;

BEGIN

" MASTER CHIP SELECT FOR THE Am9518 IS DERIVED FROM
UNLATCHED CHIP SELECT AND INTERNAL STATE. "

MCS = Q[2] * CS ;

" MASTER ADDRESS STROBE MUST MEET THE MINIMUM PULSE
WIDTH REQUIREMENT, IT IS DERIVED FROM AN INTERNAL
STATE BECAUSE ALE IS NOT WIDE ENOUGH. "

MAS = /CMDLY ;

" COMMAND DELAY TO DELAT /IORC AND /IOWC GENERATED BY
THE 82288 BUS CONTROLLER. "

CMDLY := /(Q[2] * CS) ;

" SYNCHRONIZE THE COUNTER WITH ALE. "

IF (CS * ALE) THEN Q[3:1] = 7 ;

" COUNT DOWN SPECIFICATION FOR THE COUNTER. "

" Q2 OF THE COUNTER IS THE CLOCK INPUT OF THE Am9518. "

CASE (Q[3:1])

BEGIN

0) Q[3:1] := 7 ;

1) Q[3:1] := 0 ;

2) Q[3:1] := 1 ;

3) Q[3:1] := 2 ;

4) Q[3:1] := 3 ;

5) Q[3:1] := 4 ;

6) Q[3:1] := 5 ;

7) Q[3:1] := 6 ;

END ;

" SRDY IS GENERATED FOR WAIT STATE GENERATION, IT IS SENT
TO THE 82284 CLOCK GENERATOR. "

SRDY = /(Q[3] * CS) ;

END.

Figure 3-69. Source Listing for AmPAL16R4 for the Example of Figure 3-68

" THIS PLPL FILE IS FOR A 16L8 PAL DEVICE THAT IMPLEMENTS
PART OF THE LOGIC NECESSARY TO INTERFACE AN Am9518 (DCP)
TO AN 80286 SYSTEM. ANOTHER PAL DEVICE (16R4) IS NEEDED
FOR THE COMPLETE DESIGN. "

DEVICE Am9518_TO_80286_1 (PAL16L8)

PIN

DTR	= 1	VCC	= 20
A1	= 2	/MRW	= 19
A2	= 3	/AD1	= 18
/IORC	= 4	/AD2	= 17
/IOWC	= 5	/INTSO	= 16
ALE	= 6	/CSL	= 15
/CS	= 7	/D2	= 14
NC1	= 8	/D1	= 13
NC2	= 9	/MDS	= 12
GND	= 10	NC3	= 11 ;

BEGIN

" MEMORY R-/W FOR THE Am9518 IS DERIVED FROM DT/R-. "

MRW = /DTR ;

" MEMORY DATA STROBE FOR THE Am9518 IS DERIVED FROM
IORC- AND IOWC-. "

MDS = /(IORC + IOWC) ;

" LATCHED CHIP SELECT IS DERIVED FROM ALE AND THE
EXTERNAL DECODER CHIP SELECT OUTPUT. "

CSL = CS * ALE + INTSO ;

INTSO = /(IORC * IOWC) + CSL ;

" THE FOLLOWING SPECIFICATION IS FOR MULTIPLEXING THE
ADDRESS AND DATA SIGNALS OF THE 80286 FOR THE MASTER
PORT OF THE Am9518. "

IF (CSL * IORC) THEN D1 = AD1 ;

IF (CSL * /IORC) THEN AD1 = A1 * ALE + D1 * /ALE ;

IF (CSL * IORC) THEN D2 = AD2 ;

IF (CSL * /IORC) THEN AD2 = A2 * ALE + D2 * /ALE ;

END.

Figure 3-70. Source Listing for AmPAL16L8 for the Example of Figure 3-68

3.4.3 INTERFACING TO THE 68000/68020

The 68000 has an asynchronous, 16-bit, bidirectional, data bus. Data types supported by the 68000 are: bit data, integer data of 8, 16, or 32 bit, 32-bit addresses and binary-coded decimal data. It can transfer and accept data in either words or bytes. The $\overline{DTACK}$ input indicates the completion of a data transfer. When the processor recognizes $\overline{DTACK}$ during a read cycle, the data is latched and the bus cycle terminates. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle also terminates. An active transition of $\overline{DTACK}$ indicates the termination of a data transfer on the bus. All control and data lines are sampled during the 68000's clock HIGH time. The clock is internally buffered, which results in some slight differences in the sampling and recognition of various signals. The 68000 mask sets prior to CC1 and allows $\overline{DTACK}$ to be recognized as early as S2, and all devices allow $\overline{BERR}$ or $\overline{DTACK}$ to be recognized in S4, S6, etc., which terminates the cycle. If the required setup time is met during S4, $\overline{DTACK}$ will be recognized during S5 and S6, and data will be captured during S6. $\overline{DTACK}$ signal is internally synchronized to allow for valid operation in an asynchronous system. If an asynchronous control signal does not meet the required setup time, it is possible that it may not be recognized during that cycle. Because of this, synchronous systems must not allow $\overline{DTACK}$ to precede data by more than 40 to 240 nanoseconds, depending on the speed of the particular processor. I/O is memory-mapped, i.e., there are no special I/O control signals, any peripheral is treated as a memory location.

DMA: This Bus is requested by activating the $\overline{BR}$ input of the 68000. Bus Arbitration is started by the $\overline{BG}$ output going active. The Bus is available when AS becomes inactive. The requesting device must acknowledge bus mastership by activating the $\overline{BGACK}$ input to the CPU.

The 23-bit address ($A_1 \dots A_{23}$) is on a unidirectional, three-state bus, and can address 8 M words (16 M bytes) of memory or I/O. It provides the address for bus operation during all cycles, except the interrupt cycles. During interrupt cycles, address lines A1, A2 and A3 provide information about the level of interrupt being serviced. Instead of A_0 and BYTE/WORD, there are two separate data strobe lines for the two bytes in a word. A note of caution here, the 68000 treats the MSB of the lower byte as an even byte, or word address. The same goes with processors such as the Z8000. Processors such as the 8086 treats the lower byte as the odd byte.

Interrupt is requested by activating any combination of the interrupt inputs to the 68000 (IPL0...2), indicating the encoded priority level of the interrupt requester (inputs at or below the current processor priority are ignored). The 68000 automatically saves the status register, switches to supervisor mode, fetches a vector number from the interrupting device, and displays the interrupt level on the address bus. For interfacing with old 68000 peripherals, the 68000 issues an Enable signal at one-tenth of the processor clock frequency. There are a number of AMD proprietary third generation peripherals that can be interfaced to the 68000 CPU, to improve system performance. This section deals mainly with the interfacing of the 68000 and some of the AMD proprietary peripherals.

The 68000 and AmZ8530 Interface with Interrupts

This example shows how a Programmable Array Logic (PAL) device simplifies the task of interrupt generation compared to the MSI implementation. The block diagram for the interface via a PAL device is shown in Figure 3-71. The timing diagram (Figure 3-72) illustrates the Interrupt Acknowledge cycle. As in the other designs, $\overline{RD}$ is generated during Interrupt Acknowledge to place the vector on the bus.

The timing during register programming is not shown. The PAL device allows selection of one or two Wait States by making W_0 HIGH or LOW, respectively. The table below shows the appropriate number of Wait States as a function of CPU speed.

Part	CPU Speed	Wait States
85XX	4 MHz	1
85XX	6 MHz	2
85XXA	8 MHz	2
85XXA	10 MHz	2

PAL device equations are shown in Figure 3-73.

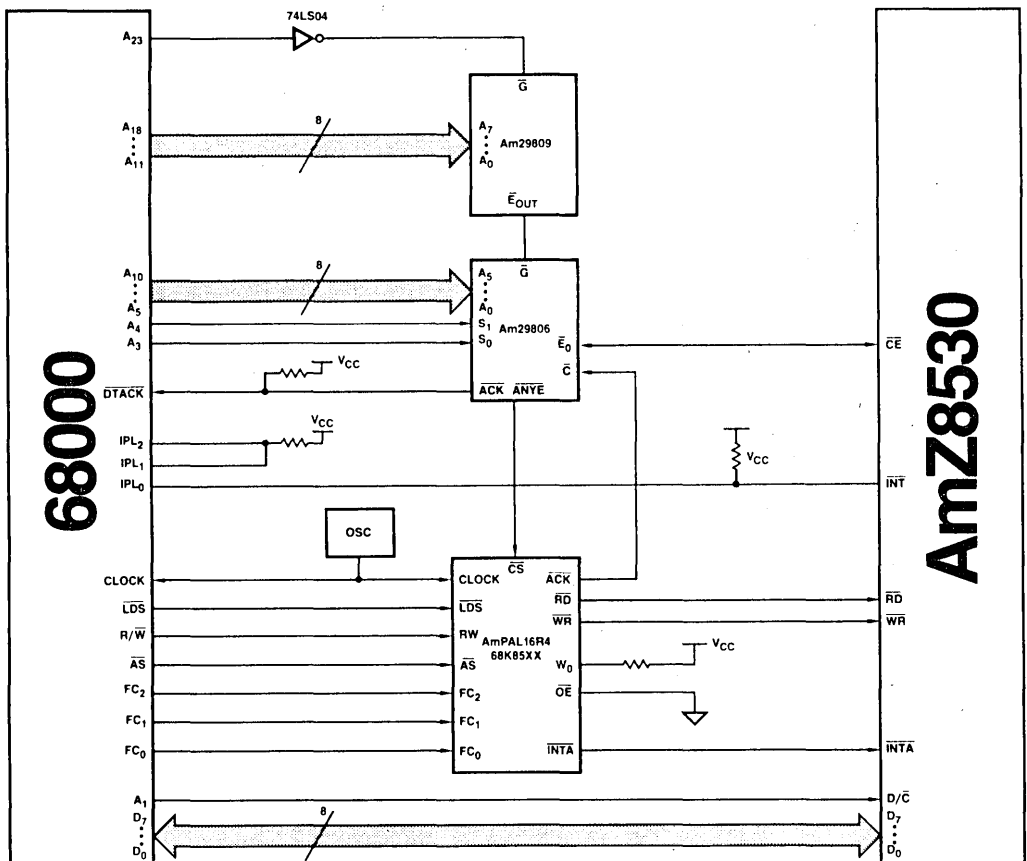
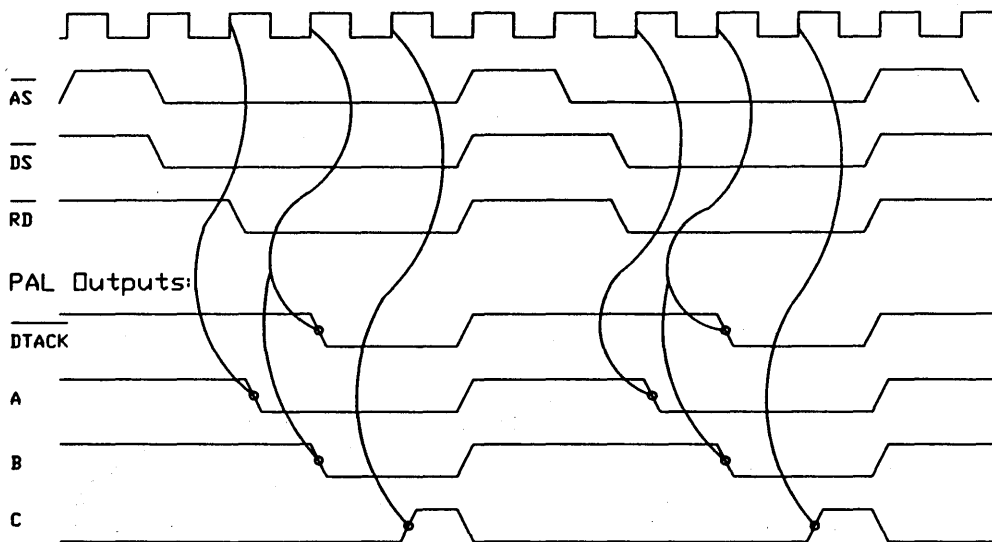


Figure 3-71. 68000 to AmZ8530 Connection Using a PAL

02188A-68



02188A-69

Figure 3-72.

PAL16R4
 PAT 002
 68000 TO 8500 OR 9500 PERIPHERALS
 ADVANCED MICRO DEVICES

PAL DESIGN SPEC
 JOE BRCICH 9 SEPT 83

CLOCK /CS RW /LDS /W0 /AS FC0 FC1 FC2 GND
 /OE /INTA /ACK /C /B /A /DLDS /RD /WR VCC

$A := A*/B + B * C + /AS$

$B := A*/C + /A*C + /AS$

$C := /A*/B*AS + B*C*AS$

$DLDS := LDS$

$RD = LDS*DLDS*RW*/INTA + A*C*INTA*AS + A*/B*INTA*AS$

$WR = LDS * /RW$

$INTA = FC0*FC1*FC2*AS$

$ACK = /INTA*/A*/B*/C*/W0 + /INTA*/A*/B*C*W0 + INTA*/B*A + ACK * LDS$

DESCRIPTION

THIS PAL DEVICE INTERFACES 85XX TYPE PERIPHERALS TO THE 68000 MICRO PROCESSOR. IT INSERTS 1 OR 2 WAIT STATES AS SELECTED BY /W0=0 IS ONE AND /W0=1 IS TWO WAIT STATES. FOUR WAIT STATES ARE INSERTED DURING INTERRUPT ACKNOWLEDGE CYCLES. ALSO THE RD OUTPUT GENERATED DURING INTA IS A FUNCTION OF THE INTERNAL STATE MACHINE AND NOT A FUNCTION OF LDS. OE CAN BE LEFT OPEN SINCE THE FLIP FLOP OUTPUTS ARE NOT USED DIRECTLY. THE FALLING EDGE OF RD IS DELAYED IN ORDER TO GUARENTEE THE CS TO RD SETUP TIME REQUIREMENTS.

Figure 3-73. Source Listing for the Example of Figure 3-71

68000 and AMD Proprietary Peripherals Interface

AMD manufactures a large number of peripherals which can be interfaced with a number of microprocessor devices. The user is advised to verify that the appropriate microprocessor interface specification is met. Two of the important parameters are setup and hold times to ensure that peripherals will work with both fast and slow CPUs. In some cases the insertion of a wait state is all that is required. In the following sections, the interface between a number of the AMD proprietary peripheral products with the 68000 are discussed.

The Am29809 Comparator and the Am29806 Comparator/Decoder provide high-speed address selection as well as an open collector acknowledge driver. This allows memories and peripherals to be conveniently wire ORed to the processor's DTACK pin.

68000 and Am7990 LANCE Interface

The design of the LANCE has made it easier for the user to interface the device with demultiplexed buses. The exaxmple shown here is an interface to be compatible with an 8 MHz or faster 68000 (Figure 3-74). The two flip-flops are needed to adapt the LANCE bus request handshake to the 68000.

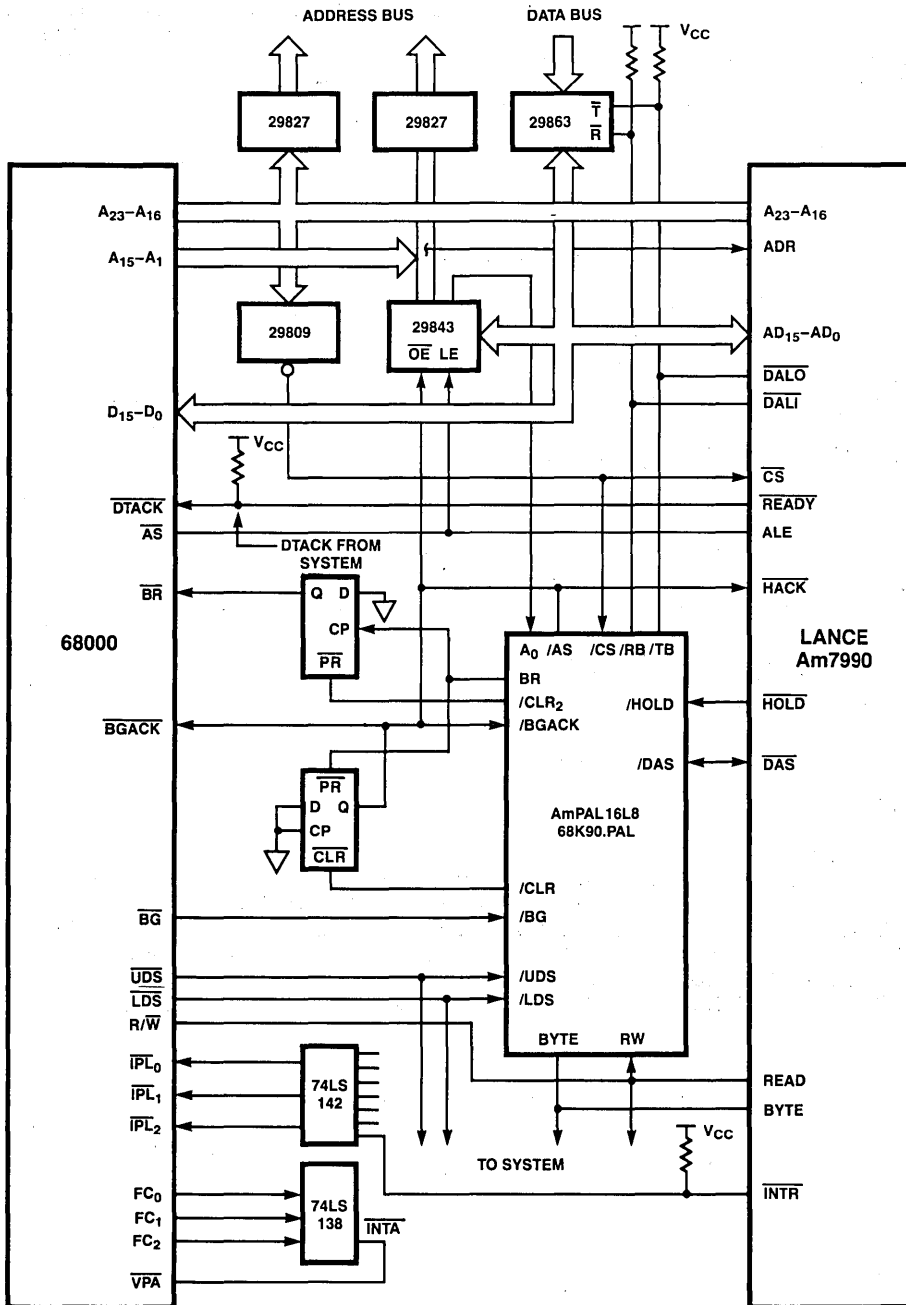


Figure 3-74. Am7990 to 68000 Interface

02188A-70

Autovectoring is used since the Am7990 does not return a vector during interrupt acknowledge cycles. The BYTE and DAS signals of LANCE are used to generate the UDS and LDS when LANCE is in Bus Master mode; the UDS and LDS is used to generate the DAS when LANCE is in Bus Slave mode. It

takes two latches to demultiplex the LANCE address/data lines to adapt to the 68000 address bus. The flip-flops can be replaced by an AMPAL22V10 to minimize parts count. Equations for the AmPAL16L8 are shown in Figure 3-75.

```

PAL16L8                                JOE BRCICH
PAT002                                2 FEB 84
68000 TO LANCE INTERFACE
ADVANCED MICRO DEVICES

/AS RW BYTE /HOLD NC /BG A0 NC /BGACK GND
/CS /TB /UDS /DAS /CLR1 BR /CLR2 /LDS /RB VCC

IF (/BGACK) RB = CS*RW*UDS + CS*RW*LDS

IF (/BGACK) TB = CS*/RW

IF (BGACK) UDS = DAS*/A0*BYTE + /BYTE*DAS

IF (BGACK) LDS = DAS*A0*BYTE + /BYTE*DAS

IF (/BGACK) DAS = UDS*LDS

CLR1 = /AS*BG

CLR2 = BGACK ;DELAY

/BR = /HOLD

```

DESCRIPTION

THE GOAL OF THIS INTERFACE WAS TO BE COMPATIBLE WITH 8 MHZ AND FASTER 68000'S WHILE MINIMIZING PARTS COUNT. THE AM22V10 COULD BE USED TO ELIMINATE THE TWO FLIP-FLOPS SHOWN. AUTOVECTORIZING IS USED SINCE THE 7990 DOES NOT RETURN A VECTOR DURING INTERRUPT ACKNOWLEDGE CYCLES. NOTE PROGRAM BSWP, BCON TO 1 ,AND ACON TO 0 IN CSR3 REG.

Figure 3-75. Source Listing for the Example of Figure 3-74

68000 to AmZ8068 Data Ciphering Processor Interface

Figures 3-76 and 3-77 show the 68000-DCP interface and the interface timing. This interface provides a two-chip solution to add high-speed data ciphering to a 68000-based system. About 500 kbyte/sec are possible in a CPU-controlled transfer. The ciphering rate can be increased with a sophisticated DMA controller, or with several DCPs operating in parallel. The CPU operates at 8 MHz and the DCP operates synchronously at 4 MHz. The Interface Controller, a PAL device, generates the Address and Data Strokes for the DCP and the Data Acknowledge for the CPU. It also divides the CPU clock by two and synchronizes it to the Data Strokes.

The main features of this interface are:

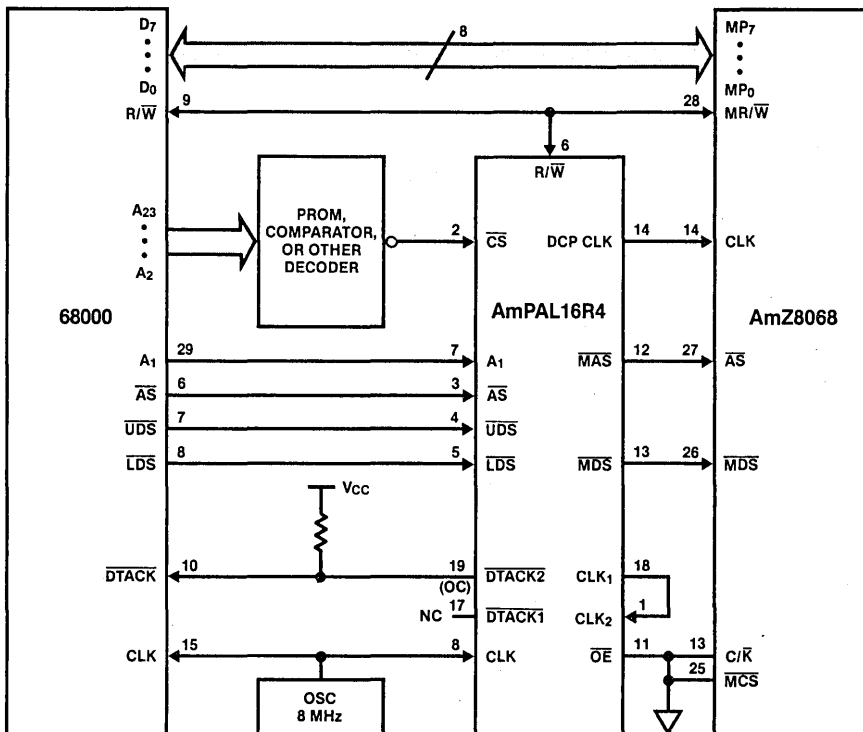
- Multiplexed Control Mode
- Demultiplexed address and data bus
- Two-Cycle Operation

— Clock Synchronization with two Low Cycles after the Data Strokes

— About 500 kbyte/sec ciphering speed

Data transfers between the CPU and the DCP are accomplished by a two-cycle operation. First the address of an internal register is latched in, then the data is transferred. This causes a small overhead in the initialization phase, but improves the ciphering rate in a high-speed data ciphering session. The rate of 500 kbyte/sec can be reached only if a high-speed peripheral device is connected to the Slave Port and the DCP is programmed for dual port configuration.

The PAL device is programmed to allow only DCP transfers to the DCP. The PAL device equations are shown in Figure 3-80. A_0 must be odd to make the CPU transfer the data on the Low byte of the data bus. A "0" on A_1 indicates an Address Latch Cycle, whereas a "1" on A_1 indicates a Data Transfer Cycle. A_0 must be "1" in both cycles.



02188A-73

Figure 3-76. AmZ8068 to 68000 Interface

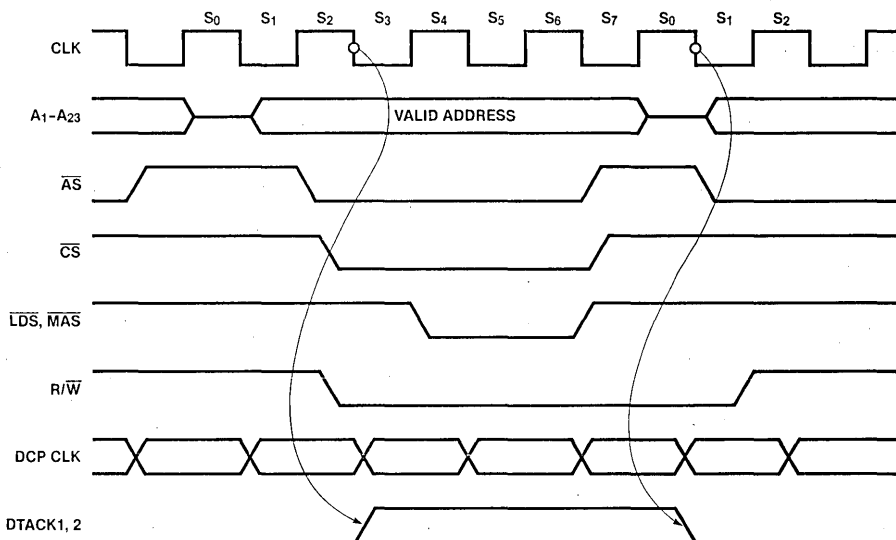


Figure 3-77. 68000-AmZ8068 Address Latch Cycle ($A_1 = \text{Low}$)

02188A-74

An address decoder generates the Chip Select for the DCP. The Address Strobe indicates a valid address. The PAL device is only activated if the Lower Data Strobe becomes active while the Upper Data Strobe stays inactive. This means that data is transferred in MOVE.B instructions with an odd peripheral address.

The PAL device provides two Data Acknowledge outputs. $\overline{\text{DTACK}}_1$ is an active Low TTL output. $\overline{\text{DTACK}}_2$ has the same timing as $\overline{\text{DTACK}}_1$, but is an Open Collector output. (The Open Collector output is realized by a three-state output which has only two states, Low or Floating.)

Address Latch Cycle

In this cycle only a Master Port Address Strobe ($\overline{\text{MAS}}$) is generated. Master Port Chip Select ($\overline{\text{MCS}}$) is tied to Low. $\overline{\text{LDS}}$ is sent to the $\overline{\text{MAS}}$ output. The minimum pulse width of $\overline{\text{LDS}}$ is 115 ns; 80 ns are required for the AmZ8068.

$\overline{\text{DTACK}}$ is activated with the falling edge of the CPU clock after cycle S_2 . The CPU inserts no Wait states. $\overline{\text{DTACK}}$ is deactivated with the first edge of CLK after $\overline{\text{AS}}$ becomes inactive.

Data Read Cycle: (Figure 3-78)

The generation of $\overline{\text{MDS}}$ in a Data Read Cycle is similar to the Data Write Cycle. Because the CPU activates $\overline{\text{LDS}}$ one cycle earlier, there is no need for a Wait State. The minimum pulse width of $\overline{\text{LDS}}$ is 240 ns; the DCP requires 200 ns for a Status Register read. $\overline{\text{DTACK}}$ is activated using the same logical condition as in the Data Write cycle. Because of the earlier activation of $\overline{\text{LDS}}$, $\overline{\text{DTACK}}$ becomes active earlier and the CPU inserts no Wait states.

Data Write Cycle: (Figure 3-79)

A Data Write Cycle is performed with A_0 is HIGH, $\overline{\text{AS}}$, $\overline{\text{CS}}$ and $\overline{\text{LDS}}$ are LOW. The minimum pulse width of $\overline{\text{LDS}}$ is not sufficient for the DCP which requires at least 125 ns. One Wait state or a slower system clock will satisfy this parameter. In this interface, one Wait state is inserted by activating $\overline{\text{DTACK}}$ at the end of S_4 .

The DCP clock is synchronized in Data Read or Write Cycles by forcing it Low when $\overline{\text{DTACK}}$ becomes active. This guarantees that the DCP clock has a falling edge just before $\overline{\text{LDS}}$ (MDS) rises. The delay of the DCP clock to CLK is typically 8 ns for a normal speed PAL device. The delay of $\overline{\text{LDS}}$ to $\overline{\text{MDS}}$ is typically 12 ns. The delay of $\overline{\text{LDS}}$ to the system clock is 0-70 ns for the 8 MHz version. This results in a delay of 4-74 ns of $\overline{\text{MDS}}$ to the DCP clock. The DCP requires 0-50 ns when operating at the maximum clock rate.

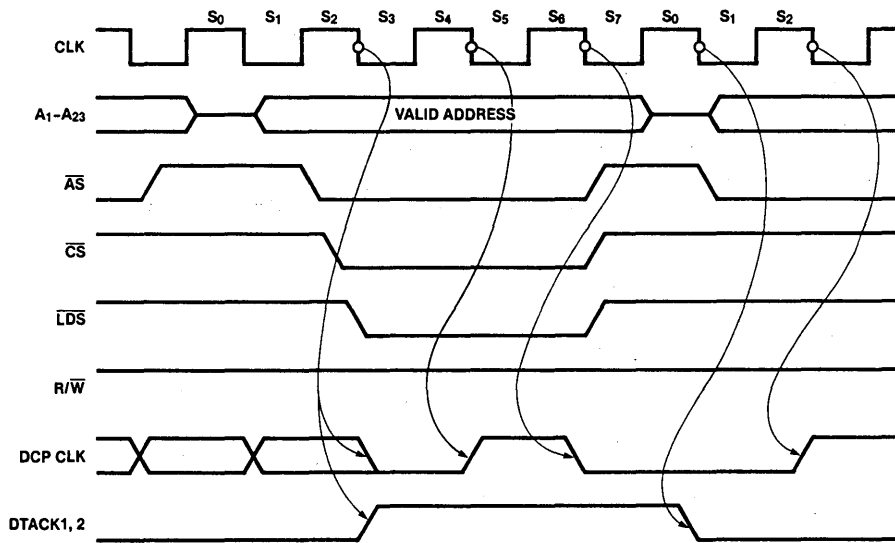
This problem is solved by stretching the clock for one cycle. The DCP clock stays LOW for two cycle in the end of a transfer cycle. This is done automatically by the PAL device (see Timing Diagram).

A Sample Data Ciphering Session

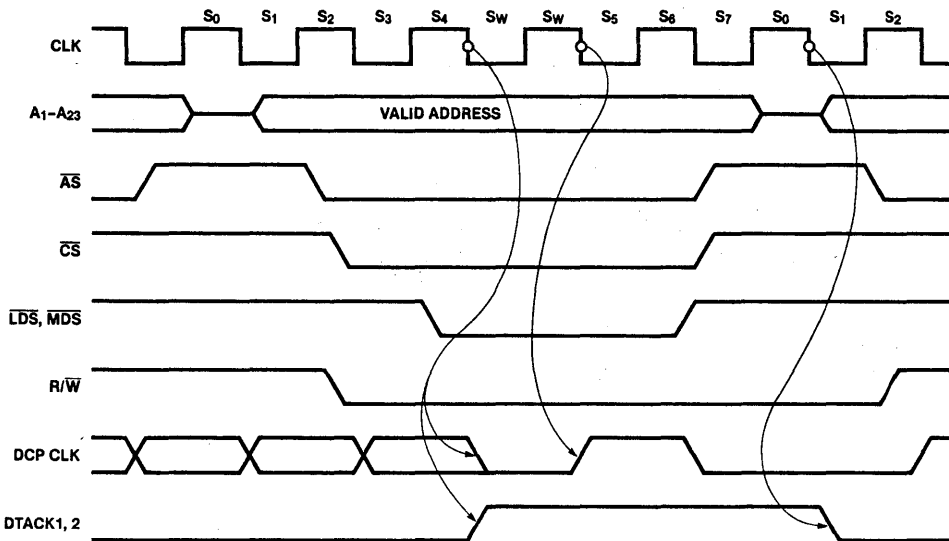
The interface was built on a Motorola 68000 evaluation board. The instructions are shown below. These instructions were executed with the "MACSBUG 1.32" evaluation board monitor.

Address Register:

A_0 Address Latch Address ($A_1=0, A_0=1$)
 A_1 Data Transfer Address ($A_1=1, A_0=1$)



02188A-75

Figure 3-78. 68000-AmZ8068 Data Read Cycle ($A_1 = \text{HIGH}$)

02188A-76

Figure 3-79. 68000-AmZ8068 Data Write Cycle ($A_1 = \text{HIGH}$)

PAL16R4
 DCP044
 68000 - Am28068 (DCP) INTERFACE DEVICE
 ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION
 JUERGEN STELBRINK 8-24-83

CLK2 /CS /AS /UDS /LDS RW A1 CLK NC GND
 /OE /MAS /MDS DCPCLK NC NC /DTACK1 CLK1 /DTACK2 VCC

/CLK1 = CLK ; INVERT CLOCK TO TRIGGER THE REGISTERED
 ; OUTPUTS WITH THE FALLING EDGE OF CLK

MAS = AS*LDS*/UDS*/RW*/A1*CS

MDS = AS*LDS*/UDS*A1*CS

/DCPCLK := DCPCLK + ; DIVIDE BY TWO
 /DTACK1*CS*AS*LDS*/UDS +
 DTACK1*/AS*/LDS*/UDS ; TWO CLOCKS LOW IN
 ; THE END OF A DATA CYCLE

DTACK1 := AS*LDS*/UDS*A1*CS + ; DATA TRANSFER CYCLE
 AS*/RW*/A1*CS ; ADDRESS LATCH CYCLE

IF (DTACK1*AS*CS) DTACK2 = DTACK1

FUNCTION TABLE

CLK2 CLK CLK1 /CS /AS /LDS /UDS RW A1
 DCPCLK /MAS /MDS /DTACK1 /DTACK2

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figure 3-80. Source Listing for the Example of Figure 3-76

```

C X X L L L H H H H H L L L ; S4
C X X L L L H H H H L H L L L ; S6
X X X L H H H H H L H H L Z ; S7
C X X L H H H H H L H H H Z ; S0
C X X X H H H H H H H H Z ; S2
; ADDRESS LATCH CYCLE
C X X L L H H L L X H H L L ; S2
C X X L L L H L L X L H L L ; S4
C X X L L L H L L X L H L L ; S6
X X X L H H H L L X H H L Z ; S7
C X X X H H H L L X H H H Z ; S0
;

```

DESCRIPTION:

INPUT SIGNALS:

CLK2 CLOCK FOR THE REGISTERED OUTPUTS OF THE PAL. IT IS CONNECTED TO CLK1

CLK 8 MHZ 68000 SYSTEM CLOCK

/CS CHIP SELECT FOR DCP (A2-A23 ARE RELEVANT)

/AS ADDRESS STROBE

/LDS LOWER DATA STROBE USED TO TIME THE MASTER PORT DATA STROBE

/UDS UPPER DATA STROBE HAS TO BE INACTIVE DURING ALL TRANSFERS

A1 ADDRESS BIT 1 DISTINGUISHES BETWEEN ADDRESS LATCH AND DATA TRANSFER CYCLES

A1=LOW ADDRESS LATCH
A1=HIGH DATA TRANSFER

RW READ/ WRITE CONTROL

OUTPUT SIGNALS:

/MAS MASTER PORT ADDRESS STROBE

/MDS MASTER PORT DATA STROBE

CLK1 INVERTED CLOCK CLK

/DTACK1 LOW ACTIVE DATA ACKNOWLEDGE FOR 68000
ONE WAIT STATE IS INSERTED IN A DATA WRITE CYCLE

/DTACK2 LOW ACTIVE DATA ACKNOWLEDGE FOR 68000 (OPEN COLLECTOR)

DCPCLK 4 MHZ DCP CLOCK, IT IS SYNCHRONIZED TO THE MASTER PORT DATA STROBE. IN A DATA TRANSFER CYCLE DCPCLK STAYS TWO CLK CYCLES LOW TO DELAY THE FIRST RISING EDGE OF THE

Figure 3-80. Source Listing for the Example of Figure 3-76 (Continued)

68000 and a Single Am9516 DMA Controller Interface

The Am9516 Universal DMA Controller (UDC) is a high-performance peripheral interface circuit for 8086 and 68000 CPUs. The UDC was designed to interface with non-multiplexed address and data bus systems. However, because it is basically a modified 8016, several incompatibilities remain. ALE is more like the 8086 than the 68000. Although the timing of ALE closely matches the $\overline{AS}$ of the 68000, it does not tristate when the 9516 is not the bus master. The major obstacle in this design was generating the proper Data Hold time for a slave write; the 8-MHz and 10-MHz CPU provides 30 ns and 20 ns, respectively, while the 9516 requires 40 ns. The newer, 8 and 10-MHz versions only require 10-ns Hold time, thus simplifying the design.

The interface design shown here (Figure 3-81) uses an AmPAL16R4 to insert a Wait State and truncate $\overline{DS}$ to the 9516 early. PAL device equations are shown in Figure 3-82. In this case the shortened $\overline{DS}$ strobes data into the 9516 while the CPU holds data valid for almost 1 clock cycle afterwards. This Wait State is inserted only during slave write. This Wait State is

actually a small penalty to pay, because most of the registers are being loaded by the UDC itself by means of a linked-list control structure. This results in very few writes being done directly to the UDC.

Another interface problem to be addressed is the bus exchange protocol. The BREQ and BACK handshake of the UDC is converted to the three-wire handshake of the 68000 by the PAL device and a 7403 package (open-collector NAND Gate). This is done by a state machine internal to the PAL device. The state assignment was done to minimize external logic. This is very different from the discrete design in the Am9516 data sheet because the PAL doesn't have preset and clear on its flip-flops.

The UDC can support the bus error function by means of its $\overline{EOP}$ pin. If a bus error occurs during a DMA transfer, an $\overline{EOP}$ will cause the DMA operation to stop and interrupt the CPU. The CPU can read status, address, word count, or anything else it may need to take corrective action. The UDC may then be restarted or reprogrammed appropriately.

The control signals $\overline{LDS}$ and $\overline{UDS}$ are generated by the PAL device when the UDC is the bus master. This is a straightforward combination of $\overline{DS}$, A0, and B/W.



Figure 3-81. The Am9516 UDC to 68000 CPU Interface

the only extra component required to make the 9516 a low-cost alternative. Additionally, the 6-MHz 9516 has the same data throughput as the 8-MHz 68450.

PAL16R4
 PAT006
 68000 TO Am9516 INTERFACE WITH DATA HOLD CORRECTION
 ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION
 JOE BRCICH 9/01/83

CLK RW AO BREQ /BG /DSI /AS /BW /CS GND
 /OE /LDS /UDS DSO /C BACK /BR /TB /RB VCC

IF(/BACK) RB = /CS * RW * UDS +
 /CS * RW * LDS

IF(/BACK) TB = /CS * /RW

IF(BACK) UDS = DSI * /AO * /BW +
 BW * DSI

IF(BACK) LDS = DSI * AO * /BW +
 BW * DSI

BR := BREQ * BG * BR * AS +
 BREQ * /BG * /BACK

/BACK := /BREQ +
 /BREQ * /BG +
 /BREQ * AS +
 /BREQ * /BACK +
 /BG * /BACK +
 AS * /BACK

C := UDS * /BACK +
 LDS * /BACK

/DSO := BACK +
 /BACK * /RW * C

DESCRIPTION

IF BREQ*BACK IS TRUE THE Am9516 HAS THE BUS, OTHERWISE THE 68000 HAS THE BUS. THIS PAL CONNECTS THE Am9516 TO THE 68000 WITH ONE WAIT STATE DURING WRITES WHILE SHORTENING /DS TO ACHIEVE PROPER DATA HOLD TIME. IT ALSO CONVERTS THE BUS EXCHANGE PROTOCOL INTO 68000 FORMAT. THIS DESIGN ASSUMES NO OTHER BUS MASTERS IN THE SYSTEM. /RB AND /TB CONTROL THE TRANSCEIVERS WHEN CPU IS BUS MASTER. /CS MUST BE A FUNCTION OF ALL DEVICES CONNECTED TO THE CPU BUS NOT JUST THE Am9516 /CS AS SHOWN HERE.

The /CS to /DS set-up time of 30 ns is met in the following ways:

1) During a read cycle the only effect from not meeting this set-up time is that the data valid access time from the Am9516 will be delayed by a proportional amount. Since the minimum /DS Low width from the 10-MHz 68000 (during a read) is 193 ns and the minimum /DS Low width to the Am9516 is 150 ns, we have 43 ns margin not counting gate delays which will further increase this margin.

2) During a write cycle this is not an issue since the /DS comes later and is stretched longer due to the Wait state.

Figure 3-82. Source Listing for the Example of Figure 3-81

68000 and Dual Am9516 DMA Controllers Interface

There has been interest shown in connecting two Am9516 DMA Controllers to obtain four channels. The example here shows that such a system can be built by incorporating one PAL device. AMD's new 22V10 (Figure 3-83). Address and data buses are not shown as they are straightforward and require no explanation. The PAL device, designated 68K16D2, converts the two DREQs into the 68000 three-wire handshake,

prioritizes the request, and converts the control signals appropriately. Equations for the PAL device are shown in Figure 3-84.

The key parameters are: 1) data hold with respect to the rising edge of $\overline{DS}$ during a write, and 2) $\overline{DTACK}$ setup time. Control for a data bus transceiver is shown because it will be required in most systems. The PAL device provides these signals when the CPU is bus master; the Am9516 generates these control signals directly when it is bus master.

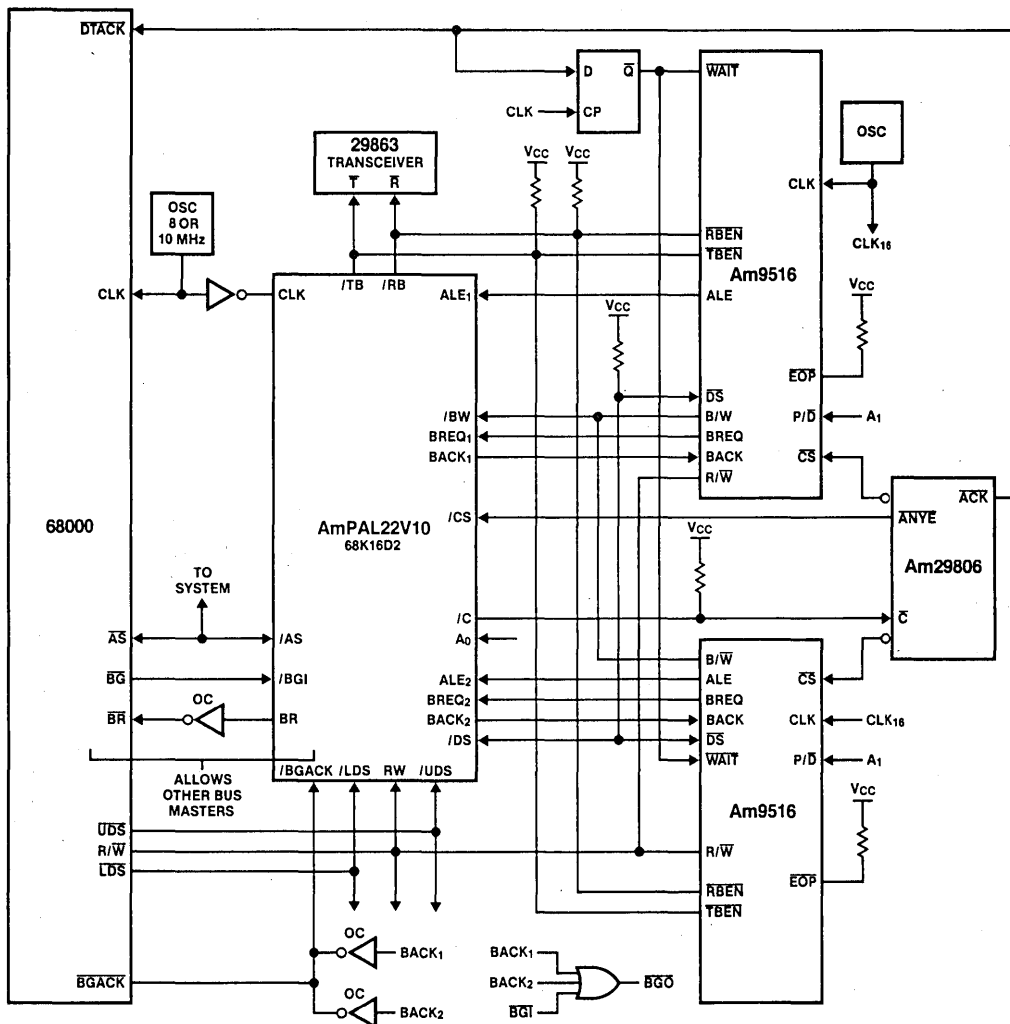


Figure 3-83. Dual Am9516 UDCs to 68000 CPU Interface

PAL 22V10
 PAT 001
 68000 TO DUAL 9516 INTERFACE
 ADVANCED MICRO DEVICES

JOE BRCICH
 5 APRIL 83

CLK RW A0 BREQ1 BREQ2 /BG NC ALE1 /BW ALE2 /BGACK GND
 /CS /LDS /UDS /DS /C /AS /BR BACK2 BACK1 /TB /RB VCC

BR = BREQ1*/BGACK + BREQ1*BG +
 BREQ2*/BGACK + BREQ2*BG

BACK1 = BREQ1*BG*/AS + /BG*BGACK

BACK2 = BREQ2*/BREQ1*BG*/AS + /BG*BGACK

IF (/BGACK) RB = CS*RW*UDS + CS*RW*LDS

IF (/BGACK) TB = CS*/RW

IF (/BGACK*AS) DS := AS*/C*/RW + AS*RW

IF (BGACK) AS = ALE1 + ALE2

IF (BGACK) UDS = DS*/A0*/BW + BW*DS

IF (BGACK) LDS = DS*A0*/BW + BW*DS

IF (AS) C := UDS*BGACK + LDS*BGACK

DESCRIPTION

THE GOAL IN THIS DESIGN WAS TO BE COMPATIBLE WITH 8 MHZ AND FASTER 68000'S. BECAUSE THE EQUATIONS FOR DS AND C WOULD EXTEND INTO THE NEXT CYCLE, THE OUTPUTS WERE TRI-STATED TO TERMINATE THESE SIGNALS EARLIER. THIS REQUIRES THAT PULL UP RESISTORS BE PLACED ON THESE OUTPUTS. THE INVERTED CLOCK WAS USED TO PROVIDE MORE SETUP TIME FOR DTACK IN THESE HIGH SPEED SYSTEMS. AT 8MHZ THE EQUATIONS CAN BE MODIFIED AND THE PULL UP RESISTORS ELIMINATED. THIS EXAMPLE IMPLEMENTS FIXED PRIORITY WHERE BREQ1 IS HIGHEST. NOTE THAT /AS SHOULD NOT BE INCLUDED IN THE ADDRESS DECODER.

Figure 3-84. Source Listing for the Example of Figure 3-83

The Am9516s are shown with independent clocks. The clocks may be divided from the CPU clock or may be generated independently of the CPU. Because WAIT must meet the set-up and hold times, DTACK may need synchronization by the use of a flip-flop, as shown. This flip-flop may be eliminated in synchronous systems.

The clock is inverted to the PAL device to meet DTACK set-up time in systems of 10 MHz or faster. This inverter may be deleted in slower systems, if appropriate changes to the PAL device equations are made. This PAL device implements fixed priority between the Am9516s. BREQ1 is the highest priority. Rotating priority can be implemented by adding another PAL device.

EOPs are pulled up separately, but could be tied together since they affect a channel only if it is active. The bus-error function can be supported by connecting BERR and EOP.

If a bus error occurs, EOP will stop the current transfer and interrupt the CPU. The Interrupt Service routine can read the status to determine if EOP caused the interrupt or if termination was normal. If EOP caused the interrupt, the Address Register can be read to determine where the bus error occurred. After the problem is corrected, the CPU can program the Am9516 to complete the transfer or do an alternate transfer, as appropriate.

When operating the DMA in interleave mode, an external EOP should be gated with DACK to prevent affecting the wrong channel. This is unnecessary if interleave is not used, since the UDC releases the bus.

This arbiter design supports both serial and parallel-expansion techniques and is therefore compatible with VME bus protocol. Bus grant out was implemented with an external gate due to a shortage of pins. The VME/BCLR function was not implemented because the Am9516 does not support preemption.

Am8500 to 68000 Interface

Modern 16-bit microprocessors, such as the 8086, Z8000, and 68000, are being used to form the nucleus of powerful personal/business computers and engineering workstations. However, support peripheral chips are virtually non-existent for the most recently introduced 16-bit CPUs such as the Motorola 68000. Since the modern microprocessor system depends as much on the peripheral controllers as it does on the CPU, it is important for a system designer to have a large variety of peripheral chips available. The 8500 family of peripheral chips from AMD provides users of non-multiplexed bus microprocessors, such as the 68000, a variety of powerful peripheral chips that can be interfaced easily with a single programmable array logic (AMD PAL) device.

The Am8500 Family

The Am8500 Family is a group of programmable peripheral chips which offload a variety of system functions from the main CPU. They support a variety of operating modes which are specified by writing to their control registers. The current members of the family include the Am8536 Counter/Timer and Parallel I/O Unit (CIO), the Am8038 FIFO Input/Output Interface Unit (FIO), and the Am8530 Serial Communications Controller (SCC). While the object of this article is not to discuss the capabilities of the Am8500 Family, a brief overview is necessary to fully understand its interface requirements.

The Am8536 is a counter/timer chip which has available three 16-bit counters. These timer/counters have features such as duty-cycle control (pulsed, one-shot, or square waved), retrigging options, and external access control lines. The CIO also provides up to 20 lines of programmable I/O ports. The Am8038 FIO is an asynchronous 128-byte buffer specially designed to be used by two CPUs or a CPU and a peripheral device as a communication or data buffer. It supports a variety of handshake interfaces on both I/O ports. Finally, the Am8530 SCC is a dual-channel, multi-protocol data communications peripheral. The SCC functions as a serial-to-parallel, parallel-to-serial converter/controller. It supports a wide variety of serial communications protocols and includes extensive on-board hardware such as baud rate generators, digital phase-locked-loops, and crystal oscillators to reduce the need for external logic.

The Control and frequently used Data registers are accessed in a different manner. These registers are accessed using a single cycle or write. This scheme allows the CPU to interact efficiently with the 8500 peripherals during normal use. The slower, clumsier initialization procedure is used much less frequently and protects the user from altering the operation mode accidentally.

All the members of the Am8500 Family are controlled and configured by software. The host CPU initializes the Am8500 operating modes by writing to the internal mode/options registers. The internal mode registers are not directly addressable by the CPU like the Control and some data registers. Instead, a two cycle process is needed to write to them. First, the address of the mode/options register being modified is written to the Control register; next, the data is written to the mode/options register via the Control register. The Am8500 peripheral has an internal state machine to keep track of whether address or data is being written to the Control register. Reading the value of the mode/options register is accomplished by first, writing an address to the Control register, and second, reading the mode/options data from the Control register.

Design Requirements

There are several problems associated with interfacing a general-purpose peripheral device to a CPU. One major problem involves the various control signals each chip uses. Unless the two families are designed to be pin-for-pin compatible (e.g., the AMD/Intel 8086/8087/8089), there generally is going to be minor variations between them; the same problem exists when interfacing the 8500 peripherals to the 68000. Part of the pin incompatibility involves genuine signal differences while other pins only require name changes.

The data pins (D_0 - D_7) on the 8500 parts are connected directly to the lower 8 data lines on the 68000 bus. The register select pins (A_0 , A_1 , A/B , $D/\overline{C}$)* can be directly connected to A_1 and A_2 of the 68000 address bus. The $\overline{RD}$ and $\overline{WR}$ lines have to be generated from the 68000's $\overline{R/\overline{W}}$ and $\overline{AS/\overline{UDS/LDS}}$ signals. The 8500 clock (PCLK) is generated by dividing down the 68000 clock.

* Note: The register select/control pins have different names on each of the 8500 peripherals.

The Interrupt Request line ($\overline{\text{INT}}$) can be wire-ORed together and connected to one of the $\text{IPL}_0\text{--IPL}_2$ inputs on the 68000, giving all the peripherals a common interrupt priority level. An alternate method might be to give each of the peripherals a separate priority level (which would require priority encoding). The interrupt acknowledge line must be generated from the CPU status lines ($\text{FC}_0\text{--FC}_2$) by the PAL. Whenever an interrupt acknowledge cycle is started, $\text{FC}_0\text{--FC}_2$ equal all ones. The interrupt daisy chain control pins (IEI and IEO on each 8500 device) are tied together in a standard priority daisy chain arrangement (see Figure 3-85). When implementing the daisy chain, arbitration delay down the chain must be accounted for in the PAL signal generation logic. The chip enable pins for each of the 8500 devices must come from the system memory mapping logic. The system designer must also provide an 8500 PAL enable line to select the PAL controller whenever any one of the 8500 devices has been selected. The DTACK

signal back to the CPU will be generated by the PAL logic using an internally implemented state counter to generate the correct timing. The output is implemented as a simulated open collector output so that other non-Am8500 peripherals in the system can use the DTACK line.

Another problem with interfacing general-purpose peripherals to the 68000 is timing. Most peripherals run at speeds considerably slower than the 8, 10, 12, and 16-MHz CPUs being produced today. This means using either a slower clock or dividing down the CPU clock. In the case of the 8500 family, this generally means dividing the CPU clock in half and using a CPU operating at less than or equal to 12 MHz. Aside from just speed problems, system integrators frequently have to tackle subtle timing differences between signals or from devised signal equivalents, e.g., deriving the Am8500 RD , WR , and DTACK from the 68000's LDS and R/W ; or guaranteeing data setup and hold times.

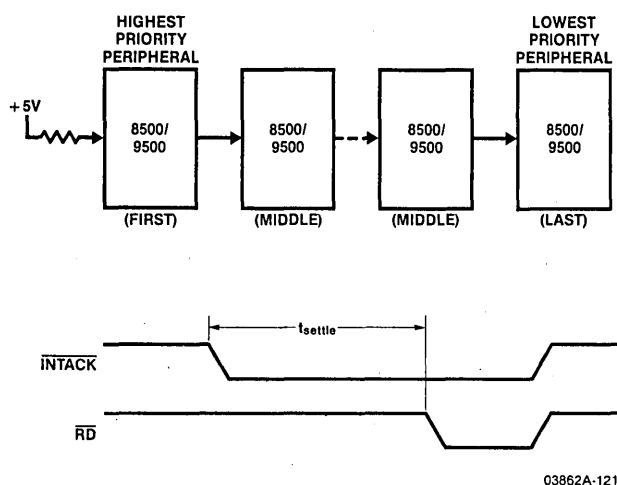


Figure 3-85.

TABLE 3-13. INTERRUPT DAISY CHAIN/PROPAGATION DELAY

Chain Position (ns)				
Peripheral		First	Middle	Last
8536	CIO	350	150	100
8038	FIO	350	150	100
8530	SCC	250	120	120

Note: First position timing is INTACK to IEO .
 Middle position timing is IEI to IEO .
 Last position timing is IEI to data strobe set-up.

03862A-122

The 68000 has two ways of interfacing to peripherals such as the 8500 family. The first uses the special VPA (Valid Peripheral Address) input pin on the 68000. The VPA pin can be activated by the Am8500 device select logic at the start of a cycle to tell the 68000 that a peripheral is being accessed. This interface was designed to allow the slow, synchronous bus 6800 peripherals to talk to the 68000's asynchronous bus until the new 68000 peripherals could be produced. Also, the VPA interface has a slow access rate (a minimum peripheral access time of over 1000 ns not including recovery time) which would slow down the CPU considerably. And, since all the 68000 peripherals are being designed to use the asynchronous method, this interface will not be discussed.

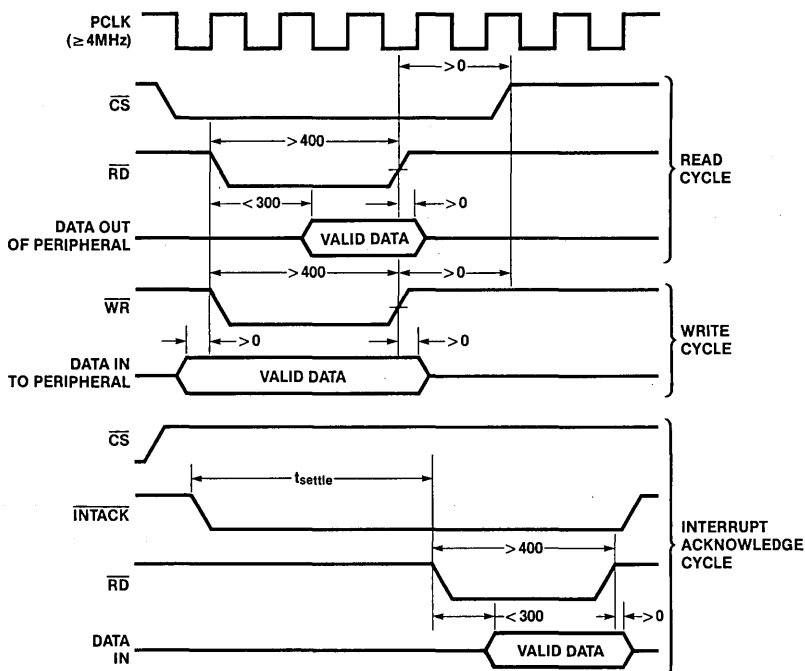
When writing data, the 68000 puts address and data onto their respective buses and uses the $\overline{\text{DTACK}}$ line as a "got data successfully" handshake from the selected device. When the $\overline{\text{DTACK}}$ line is recognized, the 68000 removes address and data one CPU clock later. This method allows the user to take advantage of the asynchronous bus of the 68000. The major difference between the 8500 family and the 68000 $\overline{\text{DTACK}}$ timing is the way data is strobed in and out of the 8500 chips. The 8500 devices sample the data on the falling edge of $\overline{\text{WR}}$. The 68000 asserts an address (when reading) onto the bus and then uses the $\overline{\text{DTACK}}$ signal from the selected peripheral (memory included) to indicate valid data and then samples on the next falling edge of the CPU clock. The other method of interfacing the Am8500 family to the 68000 uses the Data Transfer Acknowledge ($\overline{\text{DTACK}}$) cycle.

Design Approach

Two different methods of interfacing 8500 devices to the 68000 bus will be presented. One method allows the user to obtain fast access to all the 8500 devices. However, some minimum software requirements are imposed. The other interface slows down the access rate by the CPU but guarantees all 8500 minimum timing specifications and imposes no software overhead.

There are several timing requirements imposed by the 8500 family. The first involves read/write access to the parts. The 8500 (4 MHz) peripherals have a read/write/interrupt acknowledge timing as shown in Figure 3-86. The minimum read/write access time is 400 ns. This means the PAL interface must guarantee a valid access cycle of greater than 400 ns (by forcing the 68000 to execute several wait states).

The basic read or write cycle generated by the PAL interface looks like Figure 3-88. The 68000 R/W and LDS lines have been converted into 8500 $\overline{\text{RD}}$ and $\overline{\text{WR}}$ control signals and with a state timing generator, produce the 68000 data valid signal $\overline{\text{DTACK}}$. While the 8500 peripherals latch the data internally on the falling edge of $\overline{\text{WR}}$, all 9500 (Intel-type) peripherals use the rising edge of $\overline{\text{WR}}$ to strobe in data. So, the timing used is designed to guarantee proper setup and hold time for both Am8500 and Am9500 devices.



03862A-123

Figure 3-86. Am8500 Interface Timing (4 MHz)

The $\overline{DTACK}$ control logic is the only control line which employs any sort of special timing in both of the PAL interfaces. In order to guarantee proper setup and hold time for write operations to the 9500 parts, it was necessary to start the $\overline{DTACK}$ cycle in the middle of a PCLK cycle. Hence, it was necessary to use the CPU clock to condition the assertion of $\overline{DTACK}$. Using the C_0 (PCLK) and the C_1 - C_3 inputs only, would have allowed a potential setup time violation during a write operation under worst case conditions for an 8 MHz 68000.

The interrupt acknowledge cycle is very similar to the read/write cycle; only two differences exist. First, the interrupt acknowledge cycle involves only a read operation (see

interrupt acknowledge timing in Figure 3-88). Secondly, the read cycle needs to be stretched out to allow time for the interrupt daisy chain to resolve priority. If a parallel priority resolution scheme is used, then only the priority decode time delay and peripheral response time is added on to the interrupt cycle. The interrupt time delay varies, based on the number of 8500 devices in the daisy chain. The time delay is based on the 8500's position in the chain: first, somewhere in the middle, and the last device in the daisy chain (see Table 3-13). Both of the current PAL interfaces assume there are three 8500 peripherals in the interrupt daisy chain.

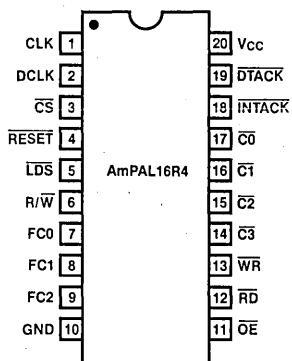
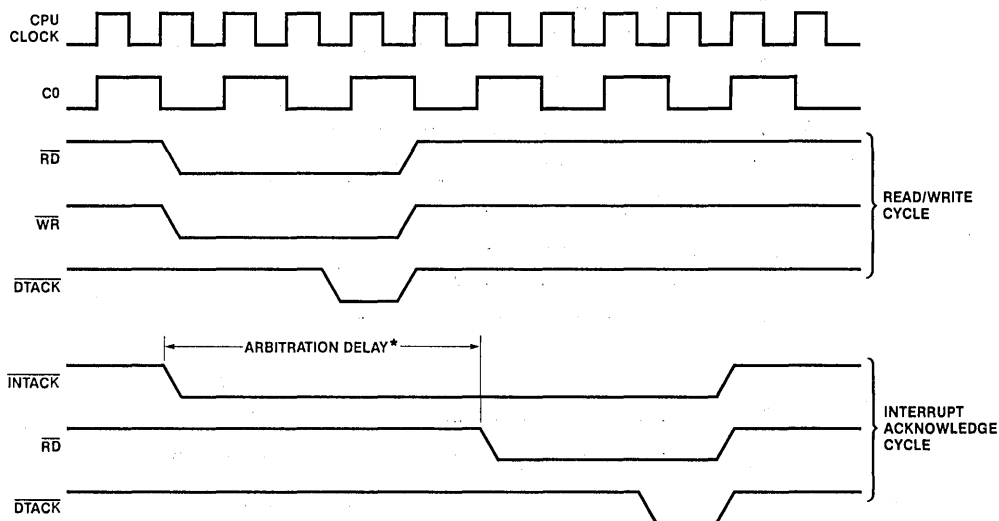


Figure 3-87.

03862A-124



03862A-125

*Delay time assumes three Am8500 devices in the daisy chain.

Note: $\overline{RD}$ and $\overline{WR}$ may not be asserted LOW simultaneously.

Figure 3-88. PAL-Generated Interface Signals

The PAL interfaces offered are designed to give the system designer maximum flexibility in integrating Am8500 peripherals with 68000-based systems. The first version is designed to allow maximum access to the 8500 devices (see Figure 3-89.a). It does this by delegating the read/write recovery time into software. All 8500 peripherals have a minimum post access recovery time; i.e., they can't be accessed for a minimum period of time after being read or written (see Table 3-14). Generally, this restriction manifests itself only if the CPU has to make repeated accesses to the same peripheral part rapidly. While the instruction fetch time of the CPU allows for some recovery time, it doesn't guarantee enough time (since the average recovery time is approximately 1000 ns and a 68000 instruction fetch requires a minimum of 500 ns @ 8 MHz). Hence, the first design requires the user to implement minimum software recovery time.

The software recovery routines in this case generally take the form of executing 1-2 instructions (depending on execution and fetch time) in between accessing the *same* 8500 device. For most systems, these instruction executions can be used to process the data just received. Another method of insuring the minimum peripheral recovery time is to juggle the accessing of the 8500 devices in the system so the recovery time requirement is not violated.

The second design (see slow PAL timing in Figure 3-89.b) relieves the user of all software considerations when using the Am8500 parts. The recovery time is built into the PAL design. This is done by delaying access on the read/write and then taking advantage of the 68000 next instruction fetch to guarantee that the minimum recovery time is given. Also, a minor change was required in the interrupt acknowledge timing; i.e., stretching out the INTACK timing slightly to avoid a potential glitch on the $\overline{RD}$ line after an interrupt acknowledge cycle.

The advantage of software-independent hardware is offset by longer read/write cycles to the peripherals, even for single accesses. Also, the user is denied access to another 8500 peripheral until the minimum recovery time has been met for the previous one. However, having software-independent hardware is sometimes an important feature in a system; and slowing down the peripheral access rate slightly is a small price to pay for it. Note, the interrupt acknowledge cycles for both designs are virtually the same. This occurred because the normal interrupt processing by the 68000 guarantees that another access to the 8500 parts cannot occur in time to violate their access recovery times. Hence no software delay is needed for the fast access interface.

The interrupt acknowledge delay (for the daisy-chained priority resolution scheme) in this example has been chosen by using an assumption of three 8500 peripherals in the chain. Larger or smaller numbers of parts in the daisy chain would increase or decrease this result with minor changes to the PAL logic equations. The design is flexible enough to support the addition of at least 3 more peripherals in the daisy chain.

The PAL equations and logic diagram for both designs are shown in Figures 3-91, 3-92, 3-93, 3-94, 3-95, and Tables 3-15 and 3-16. The equations were derived directly from their respective timing diagrams (Figures 3-89.a and 3-89.b). Some obvious logic simplification was done on the initial equations to reduce the number of terms. The integration of the Am8500 peripherals and the PAL timing generator are shown in a sample configuration in Figure 3-90.

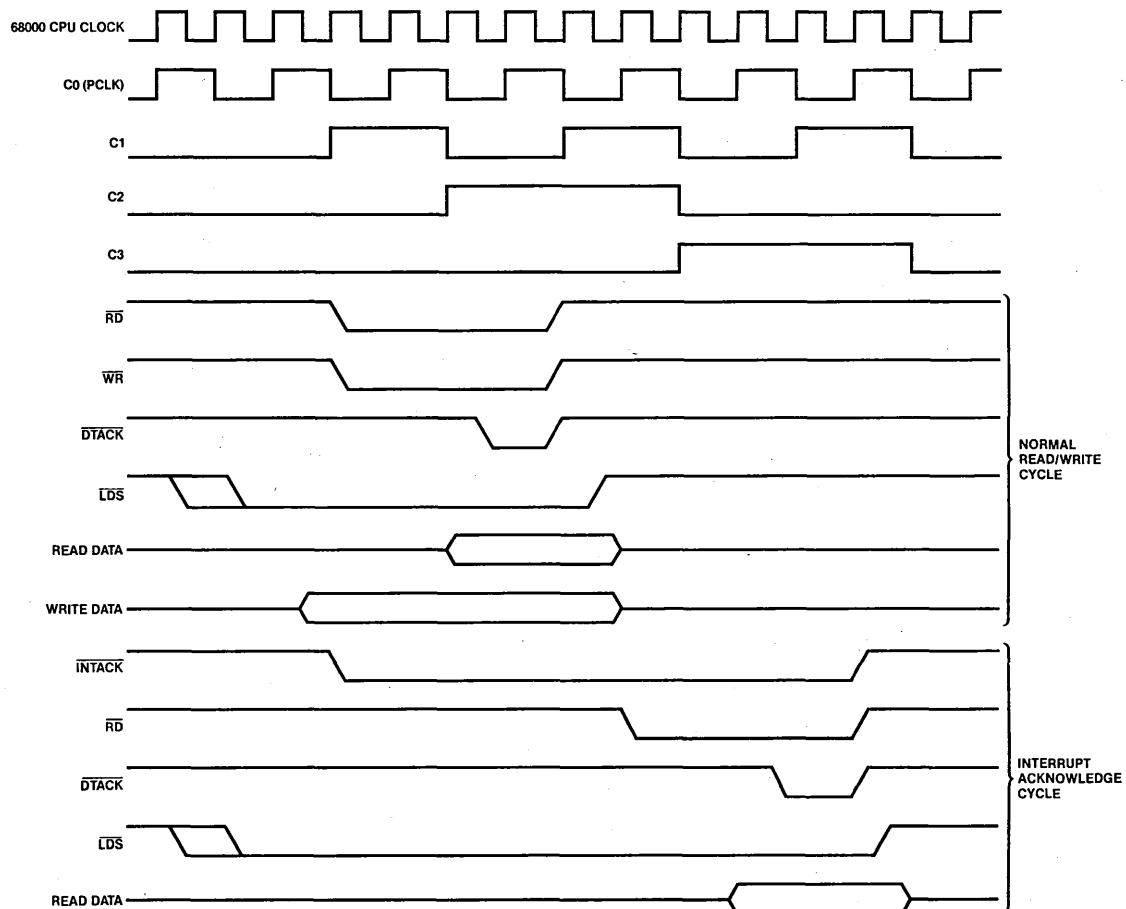
Finally, the design presented was optimized for an 8-MHz 68000 system and 4-MHz 8500 parts. The timing/state counter (C_0 - C_4) only counts as far as it is needed. Higher performance CPUs, up to 12 MHz, can be used with this interface, but 6-MHz 8500 parts will have to be used.

3

TABLE 3-14. PERIPHERAL ACCESS RECOVERY TIME

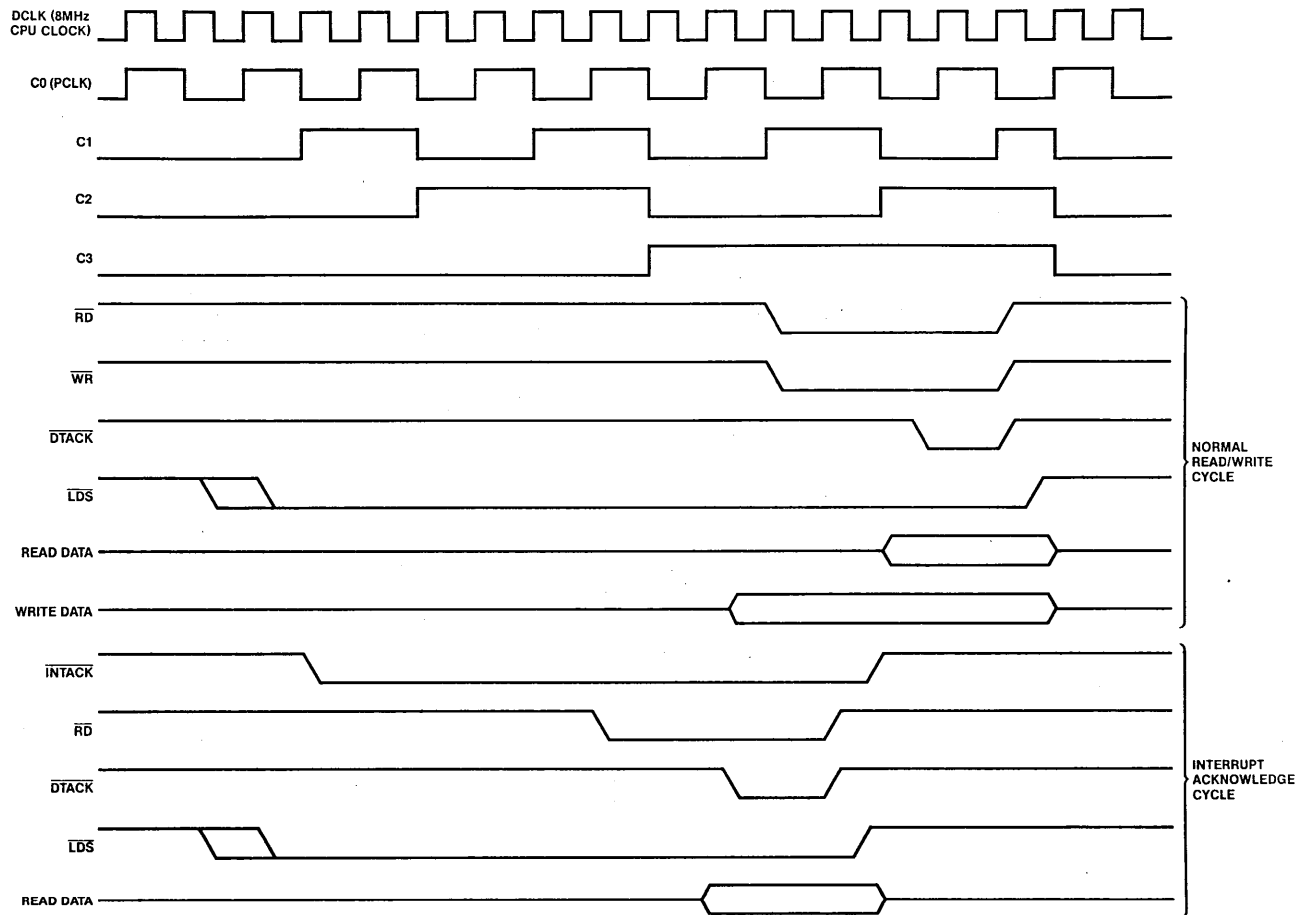
Peripheral (4MHz)	Recovery Time
8530 SCC	Greater than 6 PCLK cycles + 200 ns
8536 CIO	Greater than 3 PCLK cycles or 1000 ns
8038 FIO	Greater than 1000 ns

03862A-126



03862A-127

Figure 3-89.a "Fast" PAL Am8500 to MC68000



03862A-128

Figure 3-89.b "Slow" PAL Am8500 to MC68000

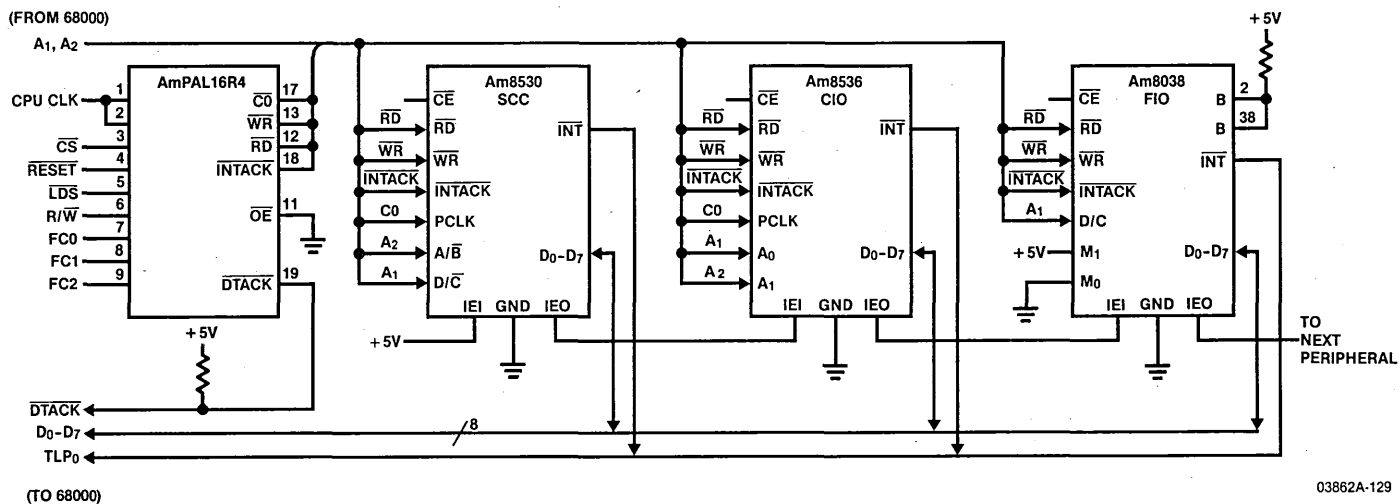


Figure 3-90. PAL Am8500 to MC68000 Hookup

```

PAL16R4
PATO50
FAST AM8500 TO MOTOROLA 68000 PAL
ADVANCED MICRO DEVICES
CLK DCLK /CS /RESET /LDS RW FCO FC1 FC2 GND
/OE /RD /WR /C3 /C2 /C1 /CO /INTACK /DTACK VCC
;
; STATE MACHINE COUNTER CO - C3
CO := /CO ; 8500 CLK

C1 := CO*/C1*CS*LDS*/RESET
+ /CO*C1*CS*LDS*/RESET

C2 := CO*C1*/C2*CS*/RESET
+ /C1*C2*CS*/RESET
+ /CO*C1*C2*CS*/RESET

C3 := CO*C1*C2*/C3*CS*/RESET
+ /C1*/C2*C3*CS*/RESET
+ /CO*C1*/C2*C3*CS*/RESET

RD = C1*/C2*/C3*RW*/INTACK*CS*/RESET ; NORMAL READ
+ /C1*C2*/C3*RW*/INTACK*CS*/RESET ; NORMAL READ
+ CO*C1*C2*/C3*INTACK*CS*/RESET ; INTERRUPT ACKNOWLEDGE
+ RD*INTACK*CS*/RESET ; INTERRUPT ACKNOWLEDGE

WR = C1*/C2*/C3*/RW*CS*/INTACK*/RESET ; WRITE OPERATION
+ /C1*C2*/C3*/RW*CS*/INTACK*/RESET ; WRITE OPERATION

; DATA ACKNOWLEDGE
IF (CS) DTACK = /DCLK*/CO*/C1*C2*/C3*/INTACK*/RESET
+ DTACK*RD*/RESET
+ DTACK*WR*/RESET
+ /DCLK*CO*/C1*/C2*C3*INTACK*/RESET

; INTERRUPT ACKNOWLEDGE
INTACK = FCO*FC1*FC2*C1*/C3*LDS*/RESET
+ C2*FC0*FC1*FC2*/RESET
+ /C1*C3*FC0*FC1*FC2*/RESET
+ /CO*C1*C3*FC0*FC1*FC2*/RESET

```

Figure 3-91. Source Listing for the Example of Figure 3-89.a

TABLE 3-15.

FUNCTION TABLE:

; NOTE: FOR THE SIMULATION, ALL THE SIGNALS USED ARE AT THE																			
; PIN LEVEL (I.E. WHAT THE CHIP SEES AND PUTS OUT). THE																			
; ONE EXCEPTION ARE THE CO - C3 PINS. THESE ARE DEFINED																			
; AT THE REGISTER OUTPUT LEVEL (NON-INVERTED) BECAUSE																			
; THEY WERE DIRECTLY DERIVED FROM THE TIMING DIAGRAMS																			
; AND THIS MAKES IT EASIER TO RELATE TO THE TIMING																			
; DIAGRAM.																			
; CLK DCLK /CS /RESET /LDS RW FCO FC1 FC2																			
/OE /RD /WR CO C1 C2 C3 /INTACK /DTACK																			
; /																			
; I																			
; N																			
; T																			
; A																			
; C																			
; K																			
; RESET SEQUENCE																			
C	H	H	L	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	
L	L	H	L	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	
C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	
L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	
; WRITE OPERATION (RW=L)																			
C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	L	H	H	
C	H	L	H	L	L	X	X	X	L	H	H	H	L	L	L	L	H	H	
L	L	L	H	L	L	X	X	X	L	H	H	H	L	L	L	L	H	H	
C	H	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	H	
L	L	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	H	
C	H	L	H	L	L	X	X	X	L	H	L	H	H	L	L	L	H	H	
L	L	L	H	L	L	X	X	X	L	H	L	H	H	L	L	L	H	H	
C	H	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	L	
L	L	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	L	
C	H	L	H	L	L	X	X	X	L	H	L	H	L	L	L	L	H	L	
L	L	L	H	L	L	X	X	X	L	H	L	H	L	L	L	L	H	L	
C	H	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	L	
L	L	L	H	L	L	X	X	X	L	H	L	L	L	L	L	L	H	L	
; INTACK CYCLE																			
C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	L	H	Z	
C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	L	H	Z	

3

DESCRIPTION:

3-107

PAL16R4
 PATO50
 FAST AM8500 TO MOTOROLA 68000 PAL
 ADVANCED MICRO DEVICES
 *D9724
 FQ

PAL DESIGN SPECIFICATION
 MARK YOUNG 1/21/83

```

L0000 1111 1011 1111 1111 1111 1111 1111 1111 *
L0032 1011 1101 0101 1101 1110 1101 1111 1111 *
L0064 1110 1111 0111 1111 1111 1111 1111 1110 *
L0096 1110 1111 0111 1111 1111 1111 1110 1111 *
L0128 1011 1110 0110 1101 1101 1110 1111 1111 *
L0256 1111 1111 1111 1111 1111 1111 1111 1111 *
L0288 1111 1111 0111 1010 1111 0101 0111 0111 *
L0320 1111 1111 0111 1111 1110 0111 0111 0111 *
L0352 1111 1111 0111 1101 1111 0110 0111 0111 *
L0384 1111 1111 0101 1110 1111 0110 0111 0111 *
L0512 1111 1111 1101 1111 1111 1111 1111 1111 *
L0768 1111 1011 0110 1001 1111 1111 1111 1111 *
L0800 1111 1011 0101 1010 1111 1111 1111 1111 *
L1024 1111 1011 0110 1110 1101 1111 1111 1111 *
L1056 1111 1011 0111 1101 1110 1111 1111 1111 *
L1088 1111 1011 0101 1110 1110 1111 1111 1111 *
L1280 1111 1011 0110 1110 1110 1101 1111 1111 *
L1312 1111 1011 0111 1101 1101 1110 1111 1111 *
L1344 1111 1011 0101 1110 1101 1110 1111 1111 *
L1536 1111 1111 1111 1111 1111 1111 1111 1111 *
L1568 1111 1001 0111 1110 1001 1101 1111 1111 *
L1600 1111 1001 0111 1101 1010 1101 1111 1111 *
L1792 1111 1111 1111 1111 1111 1111 1111 1111 *
L1824 1111 1001 0111 1110 0101 1101 1111 1111 *
L1856 1111 1001 0111 1101 0110 1101 1111 1111 *
L1888 1111 1010 0110 1110 1110 1101 1111 1111 *
L1920 1111 1010 0111 1111 1111 1111 1111 1110 *
C58C0*
V0001 C110XXXXX00HHHHHLHZ1 *
V0002 0010XXXXX00HHHHHLHZ1 *
V0003 C111XXXXX00HHHHHHHZ1 *
V0004 0011XXXXX00HHHHHHHZ1 *
V0005 C111XXXXX00HHHHHLHZ1 *
V0006 0011XXXXX00HHHHHLHZ1 *
V0007 C111XXXXX00HHHHHHHZ1 *
V0008 0011XXXXX00HHHHHHHZ1 *
V0009 C111XXXXX00HHHHHLHZ1 *
V0010 0011XXXXX00HHHHHLHZ1 *
V0011 C111XXXXX00HHHHHHHZ1 *
V0012 000100XXX00HHHHHHH1 *
V0013 C10100XXX00HHHHHLH1 *
V0014 000100XXX00HHHHHLH1 *
V0015 C10100XXX00HLHLHHH1 *
V0016 000100XXX00HLHLHHH1 *
V0017 C10100XXX00HLHLHLH1 *
V0018 000100XXX00HLHLHLH1 *
V0019 C10100XXX00HLHLHHH1 *
V0020 000100XXX00HLHLHHH1 *
V0021 C10100XXX00HLHLHLH1 *
```

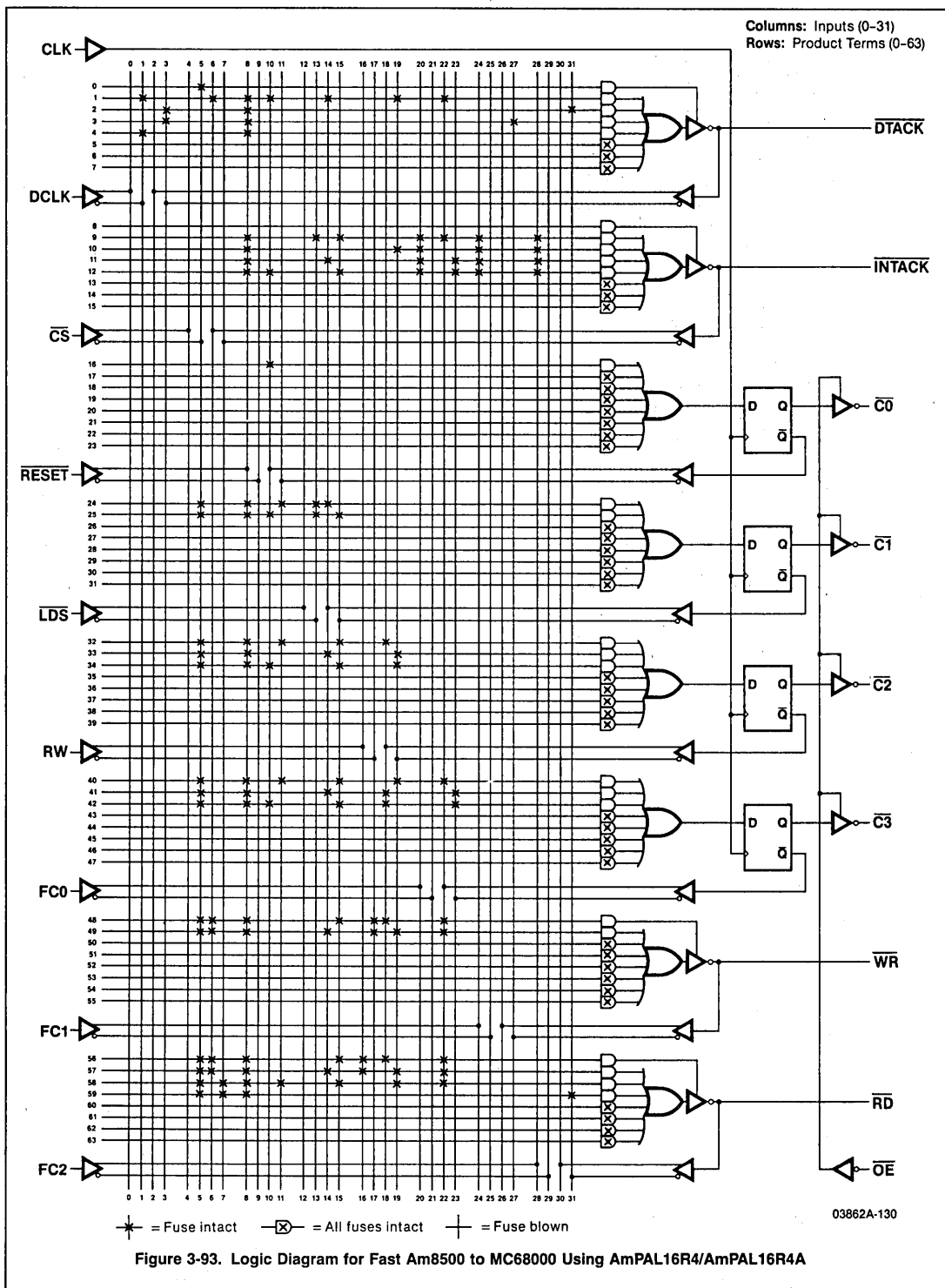
Figure 3-92. Fuse Map and Test Vectors for the Example of Figure 3-89.a

```

V0022 000100XXX00HLHLHLHL1 *
V0023 C101011XXX00HHHLLHHH1 *
V0024 0011XXXXX00HHHLLHHZ1 *
V0025 C111XXXXX00HHHHHLHZ1 *
V0026 0011XXXXX00HHHHHLHZ1 *
V0027 C111XXXXX00HHHHHHHZ1 *
V0028 0011XXXXX00HHHHHHHZ1 *
V0029 C111XXXXX00HHHHHLHZ1 *
V0030 0011XXXXX00HHHHHLHZ1 *
V0031 C10101111100HHHHLHLH1 *
V0032 000101111100HHHHLHLH1 *
V0033 C10101111100HHHLLHLH1 *
V0034 000101111100HHHLLHLH1 *
V0035 C10101111100HHHLHLHLH1 *
V0036 000101111100HHHLHLHLH1 *
V0037 C10101111100HHHLHLHLH1 *
V0038 000101111100HHHLHLHLH1 *
V0039 C10101111100HHHLHLHLH1 *
V0040 000101111100HHLLHLHLH1 *
V0041 C10101111100LHLLLLHLH1 *
V0042 000101111100LHLLLLHLH1 *
V0043 C10101111100LHLHHHLH1 *
V0044 000101111100LHLHHHLH1 *
V0045 C10101111100LHLHLLHLH1 *
V0046 000101111100LHLHLLLL1 *
V0047 C10101111100LHLHLHLL1 *
V0048 000101111100LHLHLHLL1 *
V0049 C10101111100HHLHLHLH1 *
V0050 00111XXXX00HHLHLHLHZ1 *
V0051 C1111XXXX00HHHHHHHZ1 *
V0052 00111XXXX00HHHHHHHZ1 *
7BCF

```

Figure 3-92. (Continued)



03862A-130

```

PAL16R4
PAT051
SLOW AM8500/9500 TO MOTOROLA 68000 PAL
ADVANCED MICRO DEVICES
CLK DCLK /CS /RESET /LDS RW FCO FC1 FC2 GND
/OE /RD /WR /C3 /C2 /C1 /CO /INTACK /DTACK VCC
;
; STATE MACHINE COUNTER CO - C3
;
CO := /CO ; 8500 CLK

C1 := CO*/C1*/C2*LDS*CS*/RESET
+ /CO*C1*/C2*LDS*CS*/RESET
+ CO*/C1*C2*LDS*CS*/RESET
+ /CO*C1*C2*C2*/C3*LDS*CS*/RESET

C2 := CO*C1*/C2*CS*/RESET
+ /C1*C2*CS*/RESET
+ /CO*C1*C2*/C3*CS*/RESET

C3 := CO*C1*C2*/C3*CS*/RESET
+ /C2*C3*CS*/RESET
+ /C1*C2*C3*CS*/RESET

RD = C1*/C2*C3*RW*CS*/INTACK*/RESET ; NORMAL READ
+ /C1*C2*C3*RW*CS*/INTACK*/RESET ; NORMAL READ
+ CO*C1*C2*/C3*INTACK*/RESET ; INTERRUPT ACKNOWLEDGE
+ /C1*/C2*C3*INTACK*/RESET ; INTERRUPT ACKNOWLEDGE
+ /CO*C1*/C2*C3*INTACK*/RESET ; INTERRUPT ACKNOWLEDGE

WR = C1*/C2*C3*/RW*CS*/INTACK*/RESET ; WRITE OPERATION
+ /C1*C2*C3*/RW*CS*/INTACK*/RESET ; WRITE OPERATION

; DATA ACKNOWLEDGE
IF (CS) DTACK = /DCLK*/CO*/C1*C2*C3*/INTACK*/DTACK*/RESET
+ DTACK*RD*/RESET
+ DTACK*WR*/RESET
+ /DCLK*CO*/C1*/C2*C3*INTACK*/RESET

; INTERRUPT ACKNOWLEDGE
INTACK = FCO*FC1*FC2*C1*/C3*LDS*CS*/RESET
+ C2*/C3*FC0*FC1*FC2*CS*/RESET
+ /C2*C3*FC0*FC1*FC2*CS*/RESET

```

Figure 3-94. Source Listing for the Example of Figure 3-89.b

TABLE 3-16.

FUNCTION TABLE:

;

; NOTE: FOR THE SIMULATION, ALL THE SIGNALS USED ARE AT THE

; PIN LEVEL (I.E. WHAT THE CHIP SEES AND PUTS OUT). THE

; ONE EXCEPTION ARE THE CO - C3 PINS. THESE ARE DEFINED

; AT THE REGISTER OUTPUT LEVEL (NON-INVERTED) BECAUSE

; THEY WERE DIRECTLY DERIVED FROM THE TIMING DIAGRAMS

; AND THIS MAKES IT EASIER TO RELATE TO THE TIMING

; DIAGRAM.

;

CLK DCLK /CS /RESET /LDS RW FCO FC1 FC2

/OE /RD /WR CO C1 C2 C3 /INTACK /DTACK

				/													/		/
				R	/												I		D
	C	C	/	S	L		F	F	F	/	/	/					N		T
	L	L	C	E	D	R	C	C	C	O	R	W	C	C	C	C	A		A
	K	K	S	T	S	W	O	1	2	E	D	R	O	1	2	3	K		K

; RESET SEQUENCE																			
	C	H	H	L	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	L	L	H	L	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H		Z
	L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H		Z
	C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
; WRITE OPERATION (RW=L)																			
	C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H		Z
	L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H		Z
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	C	H	L	H	L	L	X	X	X	L	H	H	H	L	L	L	H		H
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	C	H	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	C	H	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	C	H	L	H	L	L	X	X	X	L	H	H	H	L	L	L	H		H
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	C	H	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H
	L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H		H

TABLE 3-16. (Continued)

L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	H	H	H
C	H	L	H	L	L	X	X	X	L	H	H	H	L	L	H	H	H
L	L	L	H	L	L	X	X	X	L	H	H	H	L	L	H	H	H
C	H	L	H	L	L	X	X	X	L	H	L	L	H	L	H	H	H
L	L	L	H	L	L	X	X	X	L	H	L	L	H	L	H	H	H
C	H	L	H	L	L	X	X	X	L	H	L	L	L	H	H	H	H
L	L	L	H	L	L	X	X	X	L	H	L	L	L	H	H	H	L
C	H	L	H	L	L	X	X	X	L	H	L	H	L	H	H	H	L
L	L	L	H	L	L	X	X	X	L	H	H	L	H	H	H	H	H
C	H	L	H	L	L	X	X	X	L	H	H	L	H	H	H	H	H
L	L	L	H	L	L	X	X	X	L	H	H	L	L	L	L	H	Z
C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H	Z
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H	Z

; INTACK CYCLE

C	H	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H	Z
L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	L	H	Z
C	H	L	H	L	H	H	H	H	L	H	H	L	H	L	L	L	H
L	L	L	H	L	H	H	H	H	L	H	H	H	L	L	L	L	H
C	H	L	H	L	H	H	H	H	L	H	H	H	L	L	L	L	H
L	L	L	H	L	H	H	H	H	L	H	H	L	L	H	L	L	H
C	H	L	H	L	H	H	H	H	L	H	H	H	L	H	L	L	H
L	L	L	H	L	H	H	H	H	L	H	H	H	L	H	L	L	H
C	H	L	H	L	H	H	H	H	L	L	H	H	H	H	L	L	H
L	L	L	H	L	H	H	H	H	L	L	H	H	H	H	L	L	H
C	H	L	H	L	H	H	H	H	L	L	H	L	L	L	H	L	H
L	L	L	H	L	H	H	H	H	L	L	H	L	L	L	H	L	H
C	H	L	H	L	H	H	H	H	L	L	H	H	L	L	H	L	H
L	L	L	H	L	H	H	H	H	L	L	H	L	L	L	H	L	L
C	H	L	H	L	H	H	H	H	L	H	H	H	L	H	L	H	H
L	L	H	H	X	X	X	X	X	L	H	H	H	L	L	H	H	Z
C	H	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H	Z
L	L	H	H	X	X	X	X	X	L	H	H	L	L	L	L	H	Z

DESCRIPTION:

THE SLOWTOM PAL IS A SELF-CONTAINED 8500 TO 68000 INTERFACE. THERE IS NO USER SOFTWARE REQUIRED FOR THIS INTERFACE, BUT THERE IS A TRADE-OFF OF SLOWER ACCESS TIME. AGAIN, THE INTERFACE PROVIDES TOTAL SIGNAL COMPATIBILITY.

PAL16R4
 PAT051
 SLOW AM8500/9500 TO MOTOROLA 68000 PAL
 ADVANCED MICRO DEVICES
 *D9724

PAL DESIGN SPECIFICATION
 MARK YOUNG 1/21/83

F0

L0000	1111	1011	1111	1111	1111	1111	1111	1111	1111	*
L0032	1001	1101	0101	1101	1110	1110	1111	1111	1111	*
L0064	1110	1111	0111	1111	1111	1111	1111	1111	1110	*
L0096	1110	1111	0111	1111	1111	1111	1111	1110	1111	*
L0128	1011	1110	0110	1101	1101	1110	1111	1111	1111	*
L0256	1111	1111	1111	1111	1111	1111	1111	1111	1111	*
L0288	1111	1011	0111	1010	1111	0101	0111	0111	0111	*
L0320	1111	1011	0111	1111	1110	0101	0111	0111	0111	*
L0352	1111	1011	0111	1111	1101	0110	0111	0111	0111	*
L0512	1111	1111	1101	1111	1111	1111	1111	1111	1111	*
L0768	1111	1011	0110	1001	1101	1111	1111	1111	1111	*
L0800	1111	1011	0101	1010	1101	1111	1111	1111	1111	*
L0832	1111	1011	0110	1001	1110	1111	1111	1111	1111	*
L0864	1111	1011	0101	1010	1110	1101	1111	1111	1111	*
L1024	1111	1011	0110	1110	1101	1111	1111	1111	1111	*
L1056	1111	1011	0111	1101	1110	1111	1111	1111	1111	*
L1088	1111	1011	0101	1110	1110	1101	1111	1111	1111	*
L1280	1111	1011	0110	1110	1110	1101	1111	1111	1111	*
L1312	1111	1011	0111	1111	1101	1110	1111	1111	1111	*
L1344	1111	1011	0111	1101	1110	1110	1111	1111	1111	*
L1536	1111	1111	1111	1111	1111	1111	1111	1111	1111	*
L1568	1111	1001	0111	1110	1001	1110	1111	1111	1111	*
L1600	1111	1001	0111	1101	1010	1110	1111	1111	1111	*
L1792	1111	1111	1111	1111	1111	1111	1111	1111	1111	*
L1824	1111	1001	0111	1110	0101	1110	1111	1111	1111	*
L1856	1111	1001	0111	1101	0110	1110	1111	1111	1111	*
L1888	1111	1110	0110	1110	1110	1101	1111	1111	1111	*
L1920	1111	1110	0111	1101	1101	1110	1111	1111	1111	*
L1952	1111	1110	0101	1110	1101	1110	1111	1111	1111	*

C5D43*

V0001	C110XXXXX00HHHHHLHZ1	*
V0002	0010XXXXX00HHHHHLHZ1	*
V0003	C111XXXXX00HHHHHHHZ1	*
V0004	0011XXXXX00HHHHHHHZ1	*
V0005	C111XXXXX00HHHHHLHZ1	*
V0006	0011XXXXX00HHHHHLHZ1	*
V0007	C111XXXXX00HHHHHHHZ1	*
V0008	0011XXXXX00HHHHHHHZ1	*
V0009	C111XXXXX00HHHHHLHZ1	*
V0010	0011XXXXX00HHHHHLHZ1	*
V0011	C111XXXXX00HHHHHHHZ1	*
V0012	000100XXX00HHHHHHHH1	*
V0013	C10100XXX00HHHHHLHH1	*
V0014	000100XXX00HHHHHLHH1	*
V0015	C10100XXX00HHHHHLHH1	*
V0016	000100XXX00HHHHHLHH1	*
V0017	C10100XXX00HHHHHLHH1	*
V0018	000100XXX00HHHHHLHH1	*
V0019	C10100XXX00HHHLHHHH1	*

Figure 3-94. Fuse Map and Test Vectors for the Example of Figure 3-89.b

```

V0020 000100XXX00HHHLLHHHH1 *
V0021 C10100XXX00HHHLLHLHH1 *
V0022 000100XXX00HHHLLHLHH1 *
V0023 C10101XXX00HHHLLHHHH1 *
V0024 000100XXX00HHHLLHHHH1 *
V0025 C10100XXX00HHHLLHLHH1 *
V0026 000100XXX00HHHLLHLHH1 *
V0027 C10100XXX00HHHLLHHHH1 *
V0028 000100XXX00HHHLLHHHH1 *
V0029 C10100XXX00HHHLLHLHH1 *
V0030 000100XXX00HHHLLHLHH1 *
V0031 C10100XXX00HLLHLHHHH1 *
V0032 000100XXX00HLLHLHHHH1 *
V0033 C10100XXX00HLLHLHLHH1 *
V0034 000100XXX00HLLHLHLHH1 *
V0035 C10100XXX00HLLLLHHHH1 *
V0036 000100XXX00HLLLLHHHL1 *
V0037 C10100XXX00HLLLLHLHL1 *
V0038 000100XXX00HLLLLHLHL1 *
V0039 C10100XXX00HLLLLHHH1 *
V0040 000100XXX00HLLLLHHHH1 *
V0041 C111111XXX00HHHHHLLHZ1 *
V0042 0011XXXXX00HHHHHLLHZ1 *
V0043 C111XXXXX00HHHHHHHZ1 *
V0044 0011XXXXX00HHHHHHHZ1 *
V0045 C111XXXXX00HHHHHLLHZ1 *
V0046 0011XXXXX00HHHHHLLHZ1 *
V0047 C1010111100HHHHLHLH1 *
V0048 00010111100HHHHLHLH1 *
V0049 C1010111100HHHLLHLH1 *
V0050 00010111100HHHLLLLH1 *
V0051 C1010111100HHHLHLHL1 *
V0052 00010111100HHHLHLHL1 *
V0053 C1010111100HHHLHLHL1 *
V0054 00010111100HHHLHLHL1 *
V0055 C1010111100HHHLHLHL1 *
V0056 00010111100HHHLHLHL1 *
V0057 C1010111100LHHLLLLH1 *
V0058 00010111100LHHLLLLH1 *
V0059 C1010111100LHLHHHLH1 *
V0060 00010111100LHLHHHLH1 *
V0061 C1010111100LHLHHLLH1 *
V0062 00010111100LHLHLLLL1 *
V0063 C1010111100LHLHLHL1 *
V0064 00010111100LHLHLHL1 *
V0065 C1010111100HHLHLLLH1 *
V0066 0011XXXXX00HHLHLHLHZ1 *
V0067 C111XXXXX00HHHHHHHZ1 *
V0068 0011XXXXX00HHHHHHHZ1 *
F7FD

```

Figure 3-94. (Continued)

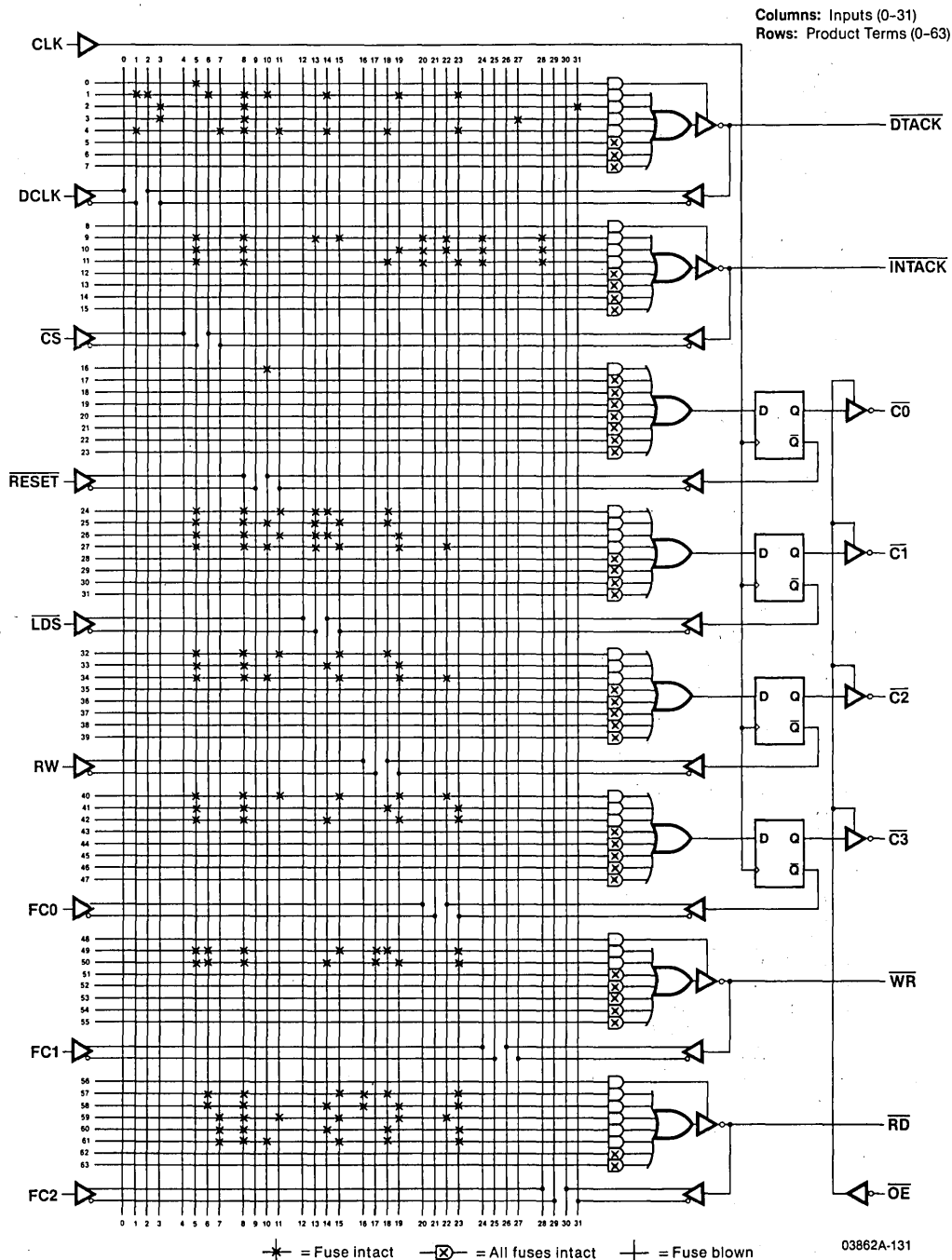


Figure 3-95. Logic Diagram for Slow Am8500 to MC68000 Using AmpAL16R4/AmpAL16R4A

Am7970A CEP Interface to the 68000 CPU

This design presents an example of how to use the CEP in a 68000 system. Though the Am7970A was designed for easy interface to the IAPX family, it can easily be adapted to the 68000.

General Discussion

The example may be a part of a workstation environment or an image storage application such as an optical disk storage device. Also note that all FAX applications (Group 3 and 4) are well served.

Figure 3-96 is the Am7970A CEP interface to the 68000 CPU. This illustration only shows how the system interface of the CEP is embedded in such a system. If very high throughput is desired, the document interface of the CEP should be connected to a large memory bank to buffer the image data. The logic for the document buffer interface is straightforward. Using memory connected to the document side as a source buffer (image data) and the system interface as the destination buffer (coded data), a whole page of image data with a resolution of 300 pixels per inch can be compressed in 1-4 seconds.

The document buffer may be loaded through the CEP system interface in transparent mode. It could also be designed as a dual port memory which is loaded directly by the CPU or by a DMA device. A third approach could load the document buffer directly by a scanner or a image processing peripheral device. The last method reduces the necessary data transfers to an absolute minimum and is therefore the preferred solution for very high performance applications.

This design assumes that the 68000 is connected to a memory bank, either onboard or via a bus interface. By setting the appropriate mode in the CEP's command register, the user determines whether this memory contains either the source or the destination buffers for the CEP, or both.

Hardware Description

A latch and two drivers are used to demultiplex the data from address bits A16-A23 of the CEP and to direct the byte-oriented data stream of the CEP to the upper and lower bytes of the data bus of the 68000. On even addresses, data is transferred through the upper half of the bus; on odd

addresses, data passes through the lower half of the 68000 data bus.

All register accesses into the CEP are performed through the upper data bus because all CEP register addresses are even. They are addressed by the signals A0-A7.

Almost all of the conversion logic for the control signals was combined into one PAL. This minimizes the hardware required for customizing the CEP to any kind of processor. The AmPAL22V10 was chosen because it provides more outputs than most other PALs and provides full freedom in choice of output characteristics (polarity, latched/unlatched function). The PAL equations are written for the PLPL PAL assembler. They can easily be changed for any other available PAL Assembler. Refer to Figure 3-97 and Figure 3-98 for the Pal Device equations.

The PAL converts the $\overline{RD}$, $\overline{WR}$ and A0 signals of the CEP to UDS, LDS, and R/W signals of the 68000. It provides the control signals for the data transceivers and transforms the two-wire bus arbitration signals of the CEP (HRQ, HLDA) to the three-wire arbitration scheme of the 68000 ($\overline{BR}$, BG and BGACK).

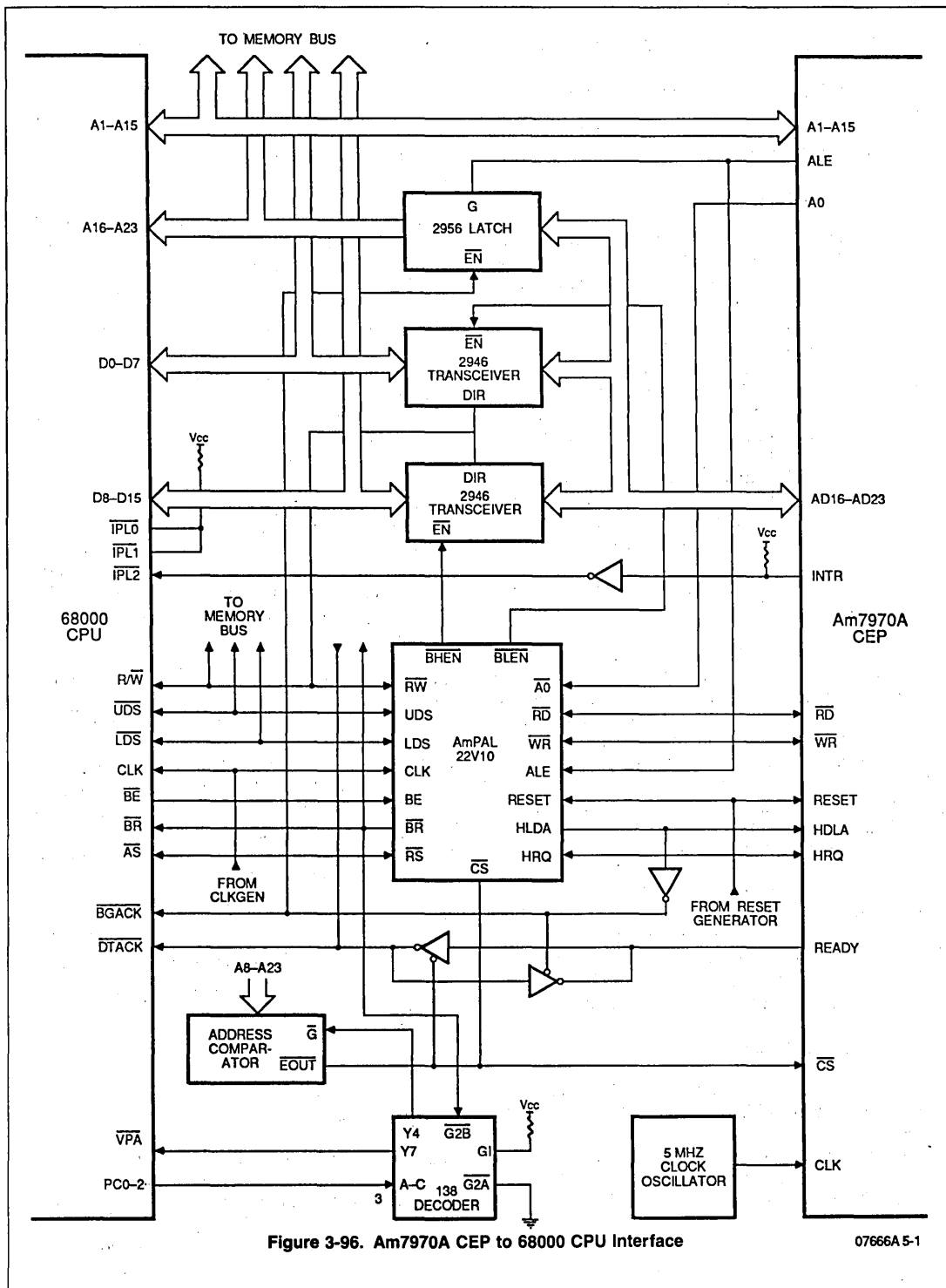
The 68000 CPU uses a memory mapped I/O address scheme. The I/O interface logic assigns a memory area to the CEP internal registers using standard address comparators. The $\overline{CS}$ output is validated by $\overline{AS}$ LOW. In sophisticated operating systems the CEP access should be reserved to supervisor level memory accesses. Here this is accomplished by an LS138 decoding this access mode from the signals FC0-2. The output is used to enable the comparators.

When designing the memory interface, care should be taken that the setup time for the READY input is met. If the environment does not provide this demand, the READY signal coming from the memory must be synchronized with a flip-flop register.

Operation

Interrupt Handling

The CEP notifies the CPU about an exception condition (e.g. end of page) by driving the INTR line HIGH. The CEP does not produce interrupt vectors by itself. If a specific application demands a user vector to be asserted by the peripheral, an



DEVICE (AMPAL22V10)
 ^7970A CEP to 68000 Interface Controller
 CEP68KPAL VERSION 1.0
 AMD Wolfgang Kemmler 9-12-85 "

PIN			
CLK	= 1	VCC	= 24
/CS	= 2	/BHEN	= 23
ALE	= 3	/BLEN	= 22
/BG	= 4	A0	= 21
HRQ	= 5	HLDA	= 20
RESET	= 6	/RD	= 21
NC	= 7	/WR	= 19
NC	= 8	/UDS	= 18
NC	= 9	/LDS	= 17
NC	= 10	/RW	= 16
NC	= 11	NC	= 15
GND	= 12	NC	= 14

BEGIN

```

IF ( RESET ) THEN ARESET() ;

IF ( HLDA ) THEN ENABLE ( RW ) ; HLDA RW = WR ;

IF ( HLDA ) THEN ENABLE( UDS ) ; UDS = RD * /AO + WR * /AO ;

IF ( HLDA ) THEN ENABLE( LDS ) ; LDS = RD * A0 + WR * A0 ;

IF ( /HLDA ) THEN ENABLE( RD ) ; RD = /RW * UDS ;

IF ( /HLDA ) THEN ENABLE( WR ) ; WR = RW * UDS ;

IF ( /HLDA ) THEN ENABLE( A0 ) ; A0 = UDS ;

BHEN = HLDA * /AO * RD + HLDA * /AO * WR + CS * UDS ;

BLEN = HLDA * A0 * RD + HLDA * A0 * WR + CS * LDS ;

BR := HRQ * BG * BR * AS + HRQ * /BG * /HLDA ;

/HLDA := /HRQ + /HRQ * /BG + /HRQ * AS + /HRQ * /HLDA
        + BG * /HLDA + AS * /HLDA ;

END
  
```

Figure 3-97. PLPL Specification for the Example of Figure 3-96

PAL16R4
VERSION 1.0
CEP68KB
AMD

CEP to 68000 Interface Controller

WOLFGANG KEMMLER 9-12-85

```
CLK /RD /WR HRQ ALE /CS /BG NC NC GND
/OE /DTACK READY /BR HLDA NC NC /AS /BGACK VCC

BR := HRQ * BG * BR * AS + HRQ * /BG * /HLDA
/HLDA := /HRQ + /HRQ * /BG + /HRQ * AS + /HRQ * /HLDA
      + BG * /HLDA + AS * /HLDA

IF ( CS ) DTACK = READY

IF ( HLDA ) READY = DTACK * RD + DTACK * WR

IF ( HLDA ) AS = ALE

BGACK = HLDA
```

Figure 3-98. CEP to 68000 Interface Controller

07666A 5-3

interrupt controller such as the Am9519A must be used.

To avoid an additional interrupt controller, this design follows an easier approach to service the interrupt request for the CPU, using the 68000 auto vector mode. The status decoder generates the interrupt acknowledge signal from the status lines F0-F2. This signal is used to drive the VPA input of the CPU. If this line instead of DTACK is asserted during an interrupt acknowledge cycle, the 68000 will use the internal auto vectors instead of an externally supplied vector.

The interrupt inputs of the 68000 are directly connected to the inverted INTR signal of the CEP without using the usual priority encoder. Assuming that the auto vector mode of the CPU is used as described above, 2 more peripherals could notify an interrupt request to the CPU by this method. With respect to all possible combinations of pending interrupt requests, the auto vector table would have to look like this:

TABLE 3-17. EXCEPTION VECTOR TABLE

Vector No.	Assignment
25	Auto Vector 1
26	Auto Vector 2
27	Auto Vector 2
28	Auto Vector 4
29	Auto Vector 4
30	Auto Vector 4
31	Auto Vector 4

The vectors are selected by the 68000 according to the the priority of the interrupt inputs IPL0-IPL2.

This schematic shows the CEP connected to IPL2 giving it the highest priority. The CEP removes INTR with the next access to a command register.

68000 Accesses to the Am7970A CEP Registers (Slave Mode)

By driving $\overline{CS}$ LOW, the address decoder notifies the CEP that the CPU wants to access the CEP internal registers. The CEP reacts by driving READY LOW and interrupting its internal microprogram. The READY signal is an output of the CEP as long it is in slave mode. Depending on the internal status of the CEP, READY is released after 4 - 50 CEP clock cycles.

The CEP provides a totally asynchronous slave interface. This keeps the logic very simple. The data hold time for a "slave write access" is 20 ns minimum which perfectly matches the 68000 up to a CPU clock frequency of 10 MHz.

Data transfers in slave mode are generally passed through the upper bus driver (D8-D15) because all registers are located at even addresses.

Am7970A CEP System Memory Access (Master Mode)

The CEP drives HRQ HIGH to gain bus control. As soon as HLDA goes HIGH it enables its system interface lines and start a memory access.

In this operating mode, READY is an input to the CEP. READY is connected to the inverted DACK of the 68000 system. The CEP samples the READY line before driving the RD or WR signals

LOW. These signals are used to provide the $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ signals which normally are asserted much earlier in typical 68000 systems. Therefore, the $\overline{\text{DTACK}}$ line which signals the completion of the memory access, cannot be asserted earlier than $\overline{\text{RD}}$ or $\overline{\text{WR}}$. This causes an automatic wait state for each CEP memory transfer.

The full performance of the CEP in a 68000 system can only be reached if the memory design is optimized not only for the 68000 but also for the specific CEP timing. If $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ are only used to enable the data driver of the memory banks and if the memories are fast enough, and if the READY line is

driven HIGH during master access all the time (disregarding $\overline{\text{DTACK}}$), then the CEP can be used without a wait state.

NOTE:

The CEP needs only 3 clock cycles for a memory transfer while the 68000 CPU takes 4. An additional wait cycle would equal the access times of both devices, assuming they are running at the same clock frequency. A CEP running at 5 MHz without a wait state, on the other hand, would match the memory access time of an 8-MHz 68000. A 5-MHz CEP does not necessarily reduce the performance of faster clocked CPUs.

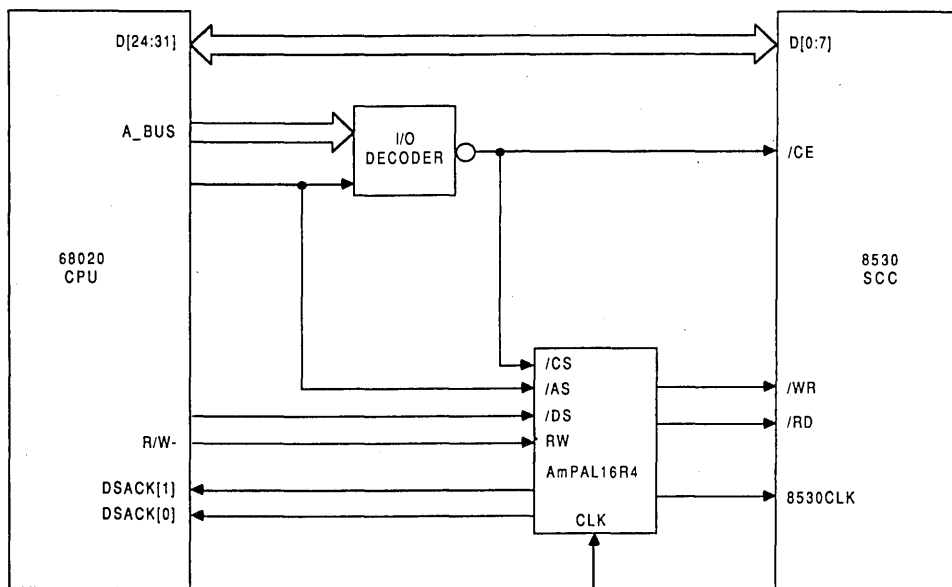
Am8530 to 68020 Interface

This design note shows the logic to interface a 6-MHz Am8530 Serial Communications Controller to the 68020 CPU running at 10 Mhz with a system clock cycle time of 100 ns. The Am8530 is a high-performance, dual-channel SCC that supports data rates up to 1.5M bps and a variety of communication protocols.

The PAL device generates the /RD and /WR control signals for the Am8530 from the 68020 /DS and R/W- control signals. It also generates the clock for the SCC by dividing the 68020 system clock. This meets the 165-ns minimum cycle time for the 6-MHz SCC clock.

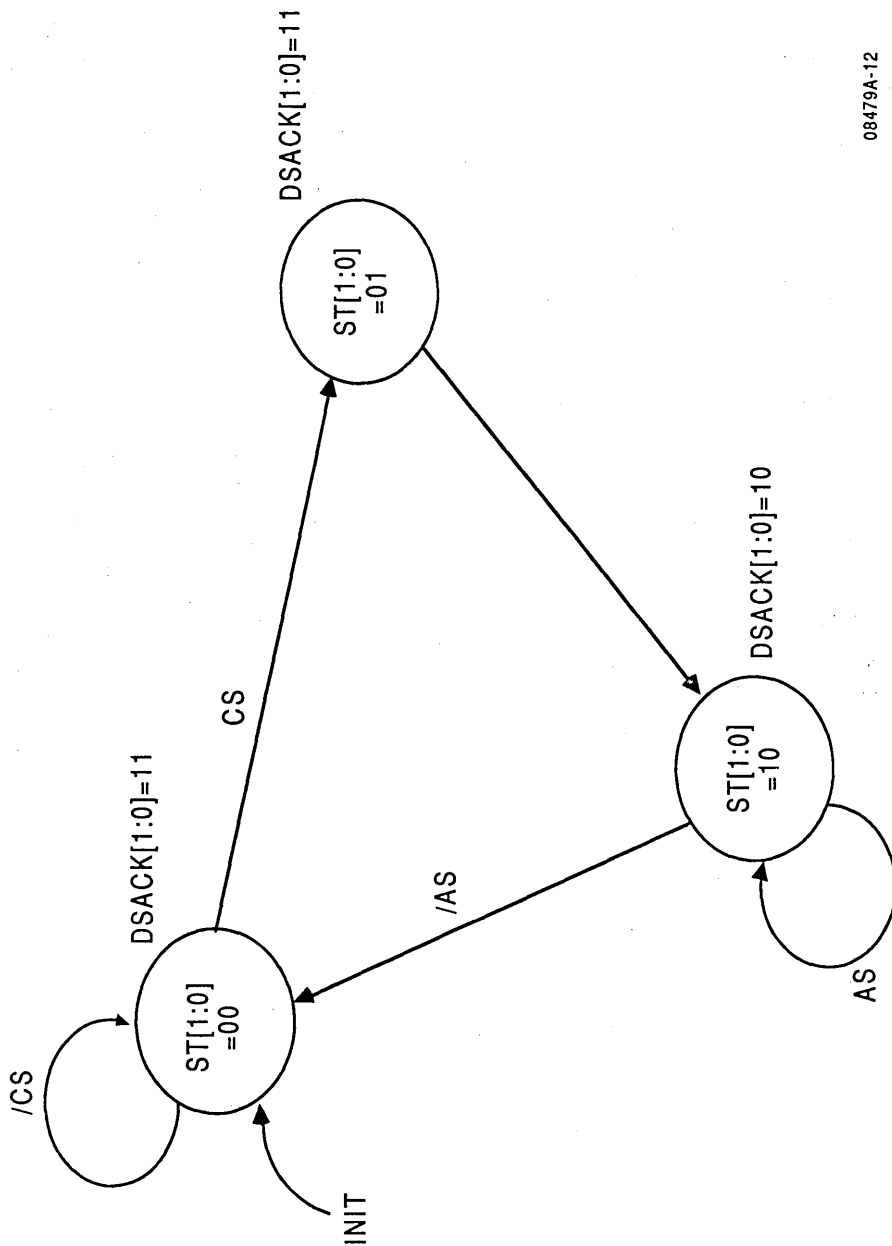
Also, a state machine is implemented to perform the handshake necessary for byte transfers on the 68020 bus. The state machine transition diagram is shown in Figure 3-100. Normally, DSACK[1:0] is inactive. When this device is selected, a wait state is inserted and then data transfer is acknowledged by asserting DSACK[1:0] for byte transfer. When the CPU deasserts the address strobe, DSACK[1:0] is negated. DSACK[1:0] will be driven by other devices on the 68020 system bus. The PAL device enables DSACK[1:0] only when the CPU performs a read or write cycle for the SCC; at other times DSACK[1:0] lines are three-stated.

Since the Am8530 is a byte-wide peripheral, it is connected to the upper byte of the 32-bit wide 68020 data bus.



08749A-11

Figure 3-99. Am8530 to 68020 Interface



08479A-12

Figure 3-100. 68020 Byte Port DSACK Handshake State Diagram

" THIS PLPL FILE IS FOR A 16R4 THAT IMPLEMENTS THE LOGIC
NECESSARY TO INTERFACE AN Am8530 (SCC) TO A 68020 SYSTEM. "

DEVICE Am8530_TO_68020 (PAL16R4)

PIN

CLK	= 1	VCC	= 20
/AS	= 2	DSACK[1]	= 19
/DS	= 3	DSACK[0]	= 18
RW	= 4	ST[1]	= 17
/CS	= 5	ST[0]	= 16
NC1	= 6	Q	= 15
NC2	= 7	NC5	= 14
NC3	= 8	WR	= 13
NC4	= 9	RD	= 12
GND	= 10	NC6	= 11 ;

BEGIN

" READ AND WRITE CONTROL SIGNALS ARE DERIVED FROM
R/W AND DS. "

WR = /(DS * /RW) ;

RD = /(DS * RW) ;

" THE INCOMING CLOCK IS DIVIDED BY 2 TO GENERATE THE
CLOCK FOR THE Am8530. "

CASE (Q)

BEGIN

0) Q := 1;

1) Q := 0;

END;

" THE FOLLOWING CODE IMPLEMENTS THE STATE MACHINE TO
PERFORM DSACK OPERATION. IT INSERTS WAIT STATES,
ASSERTS DSACK FOR BYTE TRANSFER AND REMOVES DSACK
WHEN AS IS NEGATED. "

IF (/CS) ENABLE (DSACK[1:0])

IF (ST[1:0] = #B10)

THEN DSACK[1:0] = #B10;

ELSE DSACK[1:0] = #B11;

CASE (ST[1:0])

BEGIN

#B00) BEGIN

IF (/CS)

THEN ST[1:0] := #B00 ;

ELSE ST[1:0] := #B01;

END ;

#B01) ST[1:0] := #B10;

#B10) BEGIN

IF (/AS)

THEN ST[1:0] := #B00 ;

ELSE ST[1:0] := #B10;

END ;

END ;

END.

Figure 3-101. PLPL Specification for the 8530 to a 68020 Interface

A PAL-Based Direct Memory Access Controller for the 68020

Introduction

Direct Memory Access operation on the 68020 system bus provides an interesting design challenge. There are many design issues that need to be addressed for an efficient DMA design.

The 68020 can accommodate memory blocks and peripherals of different widths (8, 16, and 32 bits) on the system bus. The 68020 imposes a connectivity requirement on slave devices wherein 8-bit devices must be connected to the uppermost byte of the data bus and 16-bit devices must be connected to the upper word of the data bus.

Issues related to device data width are: What kinds of transfers will be supported by the DMA device? How flexible should be the data funneling scheme? (Data funneling refers to packing and unpacking data when mixed width transfers—8-to-16, 8-to-32, etc.—are performed.) Data funneling can be further complicated when it becomes necessary to perform upper-to-lower and lower-to-upper byte swap (i.e., a device with a word-wide bus may write (read) the byte on the lower byte of the data bus, whereas the 68020 reads (writes) the byte on the upper byte of the upper data bus).

The 68020 system bus allows a memory read (or write) operation in three clock cycles. With a 16-MHz 68020 this translates to a memory cycle time of approximately 180 ns. In a high-performance design, it would be desirable to have a 32-bit wide, no-wait-state memory. Speed of the DMA operation in such a system should be limited only by the memory speed and not the DMA hardware itself. Incorporating an off-the-shelf DMA controller in the 68020 system must be evaluated on this basis.

Another aspect of Direct Memory Access operation is the block size of the data transfer. With a 32-bit wide address bus, the choice of block size is indeed large. Also, there should be no unreasonable restriction on the starting address of the source and destination transfer areas.

Design Objectives—Fast and Simple!

This design note shows one possible implementation of DMA logic for the 68020 system using PAL devices. This is a no-frills design for performing single-cycle, 32-bit, memory-to-memory transfers. It is possible to incorporate other features using more logic. These features may prove useful in some specific applications.

The design objectives are as follows: perform 32-bit memory-to-memory transfers without any wait states, support transfer to and from any area in the memory with a block transfer size of at least 64K bytes, and support preemptive DMA operation.

Implementation Detail

A block diagram of the DMA logic is shown in Figure 3-102.

Two address pointers, a counter, a data latch, and a control register make up the DMA logic. Two PAL devices are used to implement the DMA control logic. An AmPAL16R4 PAL device is used to perform the control-state sequencing and some control-signal generation, and an AmPAL16L8 device is used to generate the remaining control signals. The state sequencing PAL device is clocked with a 32-MHz signal ($2 \times \text{SYS CLK}$).

There are two address pointers—one for source and the other for destination area. The pointers are 30 bits wide—the upper word of each pointer is a segment register that is written by the CPU before a DMA transfer is initiated; the CPU must perform segment register maintenance (i.e., if the transfer area spans a segment boundary, the CPU must break up the transfer into two separate operations and update the segment register between the two DMA operations).

The lower word of the pointer is a 14-bit incrementor that is loaded by the CPU before the DMA operation is initiated; since the lower address word can be loaded by the CPU it can start the transfer at any long-word boundary memory addresses. This pointer is incremented by the DMA logic as each transfer is performed. Address bits A1 and A0 are stuck-at-zero since the DMA controller handles long words only.

A 32-bit data latch is used to temporarily hold the fetched data long-word from the source. During destination writes, this becomes the source of data.

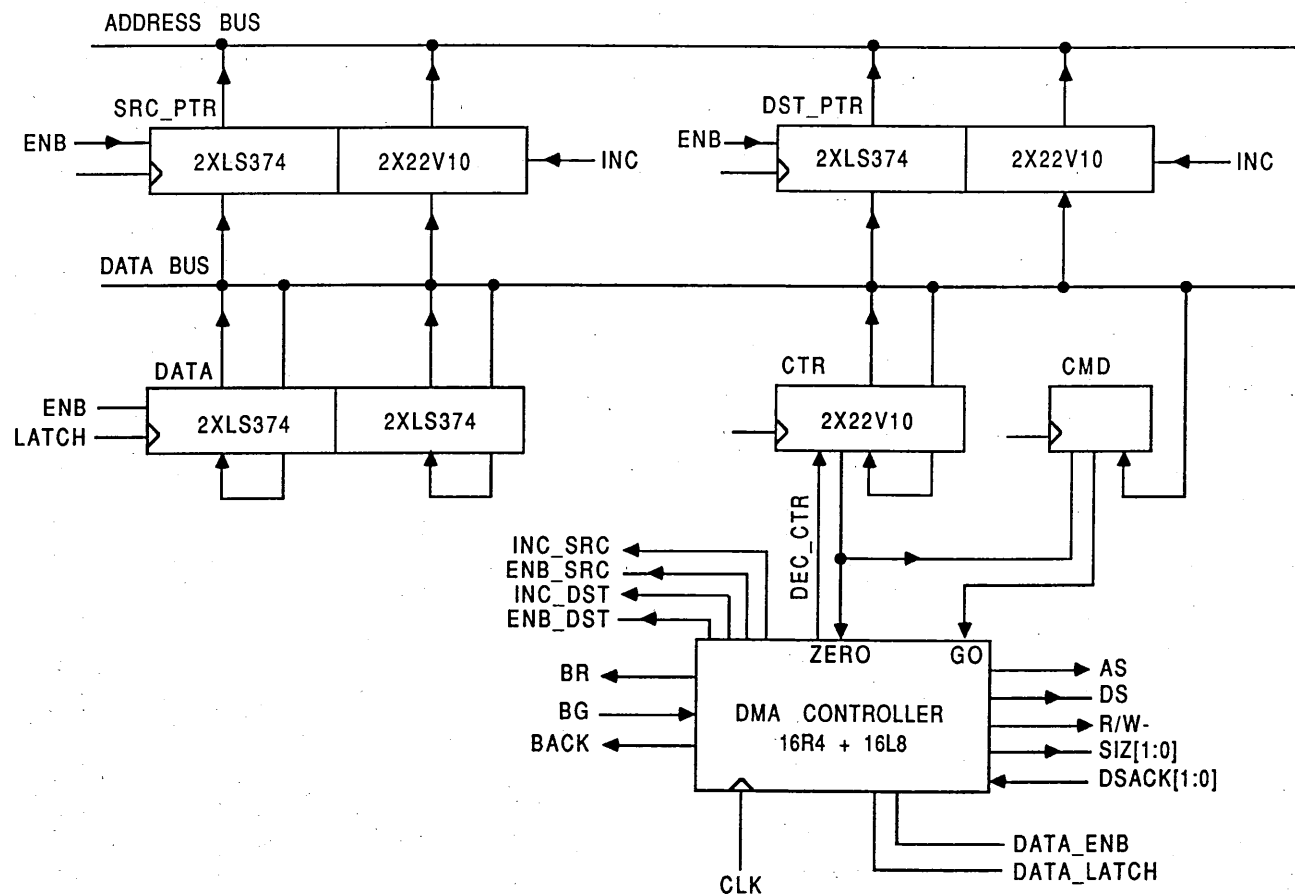
Depending on the characteristics of the RAM devices used in the system, it is possible to modify the control logic such that it is not necessary to have the external 32-bit data latch temporarily hold the long-word being transferred.

Assuming DRAMs are used in the system, here is how that scheme will work: the controller starts a read cycle from the source, after allowing for read access time, the address and RAS applied to the source are taken away while maintaining CAS to the source. This frees up the address bus for driving the destination address while maintaining valid data on the data bus with the source data. The write operation to the destination is now initiated. The address for destination is provided by the destination address pointer and the data comes directly from the source DRAM. The transfer cycle is completed by removing the RAS and CAS to the destination and the CAS to the source.

A 14-bit decrementor that is loaded by the CPU is used to track the number of long-words transferred in the current DMA cycle. This counter is loaded by the CPU before a new DMA operation is initiated. The decrementor is decremented by the DMA control logic as each transfer is performed. A count of zero indicates the end of current DMA operation.

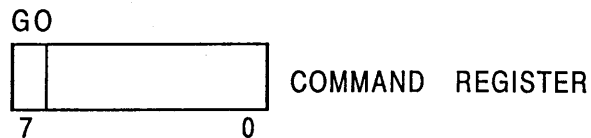
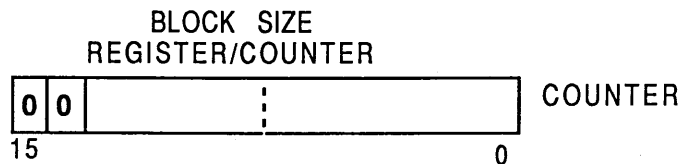
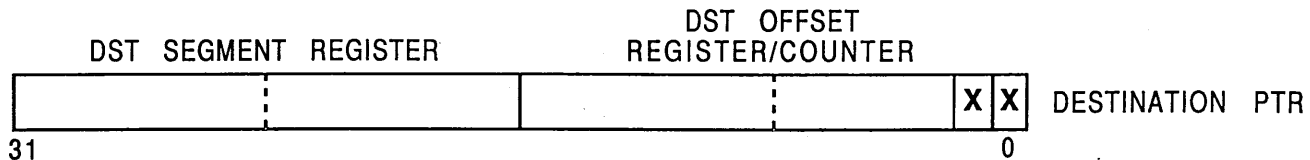
A command register is used to start the DMA operation by writing the GO bit. The zero detect signal of the counter (CTRZERO) is used to reset the GO bit.

DMA related pointers, counter and command register are shown in Figure 3-103. These locations are mapped to the 68020 memory space. They are all written by the 68020 before the DMA operation is initiated (by writing the GO bit of the



08479A-13

Figure 3-102. PAL-Based DMA Controller for the 68020



ALL MAPPED TO THE SYSTEM MEMORY SPACE.
ALL ARE INITIATED BY THE CPU.
COUNTERS ARE INCREMENTED/DEINCREMENTED BY THE DMA LOGIC.

Figure 3-103. DMA Pointers, Counters, and Command Register

command register). The address pointer count (lower word of the address pointer) is incremented by the DMA controller on every transfer cycle.

The DMA transfer cycle is a tight twelve-state sequence. It requires the destination pointer to be incremented while the read operation is in progress, and the source pointer to be incremented while the write operation is in progress. This necessitates initializing the destination pointer with a value that is one less than the desired destination transfer area start address.

It is possible to map the command and the counter locations to the same long-word address; and they can be written in the same long-word write cycle by the CPU. There should be no timing problems when count and GO is written simultaneously because the bus arbitration must take place before the DMA logic can use the count value. When the CPU regains control of the system bus, it should read the count register to determine if the transfer is complete.

DMA preemption is implemented in the control state machine. Bus Grant (BG) can be negated by an external bus arbiter at any time and the DMA controller will complete the current bus cycle and give up the system bus by deasserting the Bus Grant Acknowledge (BGACK) signal. Another device can now become the bus master; however since the transfer is not complete, the DMA logic will keep the Bus Request (BR) signal active and attempt to complete the transfer whenever the other bus master relinquishes the bus.

A state diagram for the DMA controller state machine is shown in Figure 3-104. Note that all signals are assumed active HIGH

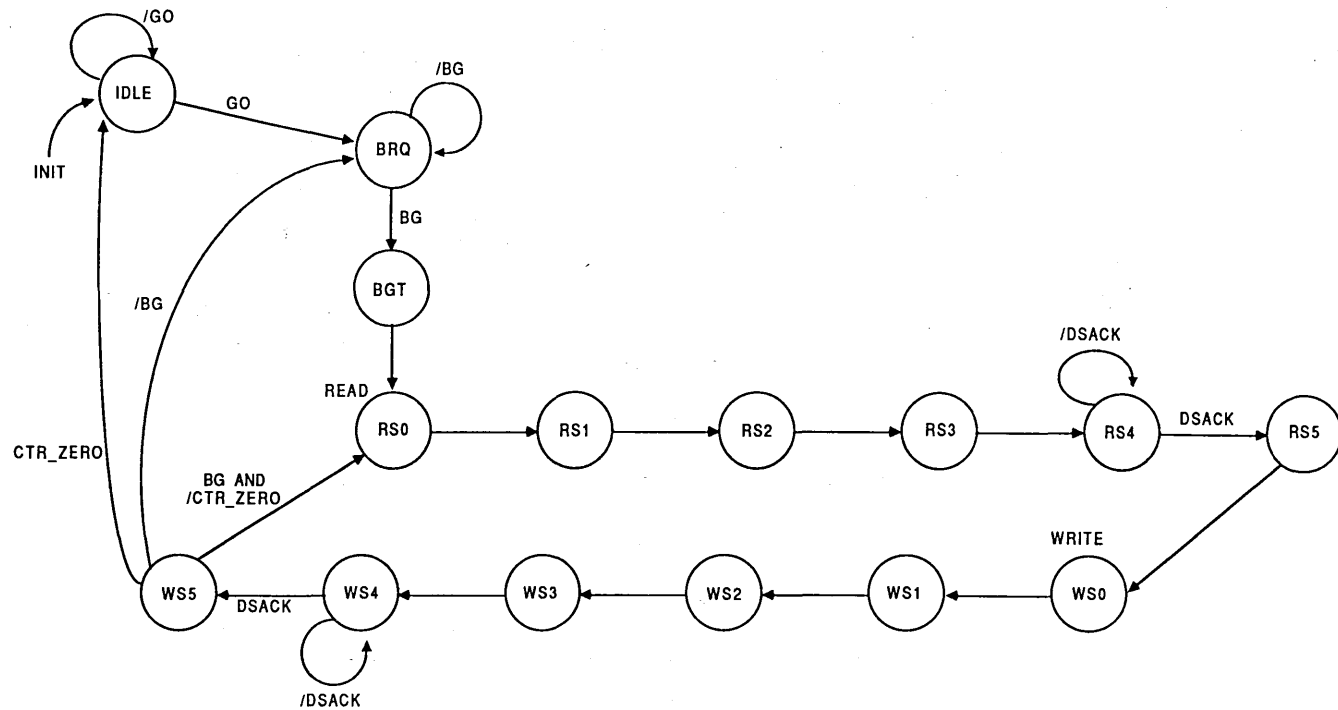
in this state transition diagram. The device stays in the IDLE state until a DMA operation is started by writing the GO bit of the command register. The controller then requests the bus (BRQ state) and after the bus is granted (BGT state), it initiates memory-to-memory transfers—read followed by write cycles; note that one memory-to-memory transfer cycle is an indivisible operation.

The DMA preemption is performed in state WS5. When the external arbiter signals that the bus is needed by another potential bus master (by negating BG), the DMA controller gives up the bus and enters the BRQ state and stays there until the bus is returned to the DMA controller.

The current DMA transfer is complete when the count value reaches zero. It causes the DMA controller to give up the system bus (BR is negated) and forces the DMA state machine into the idle state.

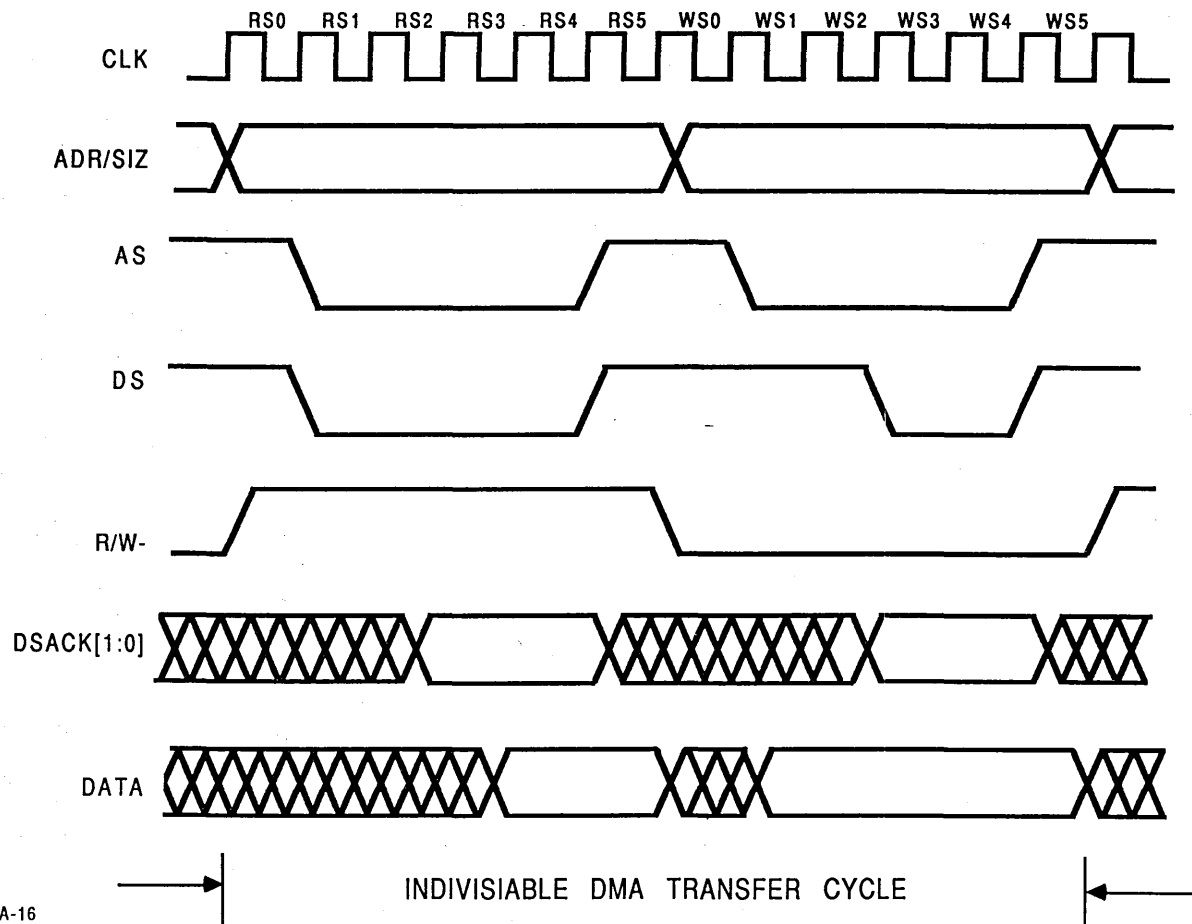
Conditions for state transitions are shown in the state transition diagram (Figure 3-104). Figure 3-106 shows the specification for a PAL device (AmPAL16R4) to implement this state diagram. Table 3-18 shows the control outputs as a function of the current state. This table can be used to derive the PAL specification for the control output generation (combinatorial only) PAL device (AmPAL16L8). It should be noted that it is necessary to three-state some of the control signals by using the BR and BGACK control inputs.

A timing diagram for major signals in a memory-to-memory transfer cycle is shown in Figure 3-105. The DMA controller essentially generates the timings of the 68020 system bus for memory read and write cycles.



08479A-15

Figure 3-104. DMA State Machine Transition Diagram



08479A-16

Figure 3-105. DMA Transfer Major Signal Timings

" THIS IS THE STATE MACHINE SPECIFICATION FOR THE 68020
DMA CONTROLLER LOGIC. - VINEET DUJARI 08/15/86 "

DEVICE 68020_DMA (PAL16R4)

PIN	CLK	= 1	VCC	= 20
	/BG	= 2	/BR	= 19
	GO	= 3	/BGACK	= 18
	CTR_ZERO	= 4	ST[3]	= 17
	/DSACK[1]	= 5	ST[2]	= 16
	/DSACK[0]	= 6	ST[1]	= 15
	INIT	= 7	ST[0]	= 14
	NC2	= 8	/AS	= 13
	NC3	= 9	/DS	= 12
	GND	= 10	NC4	= 11 ;

" STATES ARE DEFINED BELOW. "

DEFINE	IDLE	= 0 ;	BRQ	= 1 ;	BGT	= 2 ;
	RS0	= 3 ;	RS1	= 4 ;	RS2	= 5 ;
	RS3	= 6 ;	RS4	= 7 ;	RS5	= 8 ;
	WS0	= 9 ;	WS1	= 10 ;	WS2	= 11 ;
	WS3	= 12 ;	WS4	= 13 ;	WS5	= 14 ;

" THE STATE TRANSITION SPECIFICATION FOLLOWS. IT IS
DERIVED FROM THE STATE TRANSITION DIAGRAM. "

BEGIN

IF (INIT) THEN RESET (ST[3:0]) ;

IF (BR * BGACK) THEN ENABLE (AS, DS) ;

CASE (ST[3:0])

IDLE) IF (/GO) THEN ST[3:0] := IDLE ;
 ELSE ST[3:0] := BRQ ;
BRQ) IF (/BG) THEN ST[3:0] := BRQ ;
 ELSE ST[3:0] := BGT ;
BGT) ST[3:0] := RS0 ;

RS0) ST[3:0] := RS1 ;
RS1) ST[3:0] := RS2 ;
RS2) ST[3:0] := RS3 ;
RS3) ST[3:0] := RS4 ;
RS4) IF (DSACK[1:0]) THEN ST[3:0] := RS5 ;
 ELSE ST[3:0] := RS4 ;
RS5) ST[3:0] := WS0 ;

WS0) ST[3:0] := WS1 ;
WS1) ST[3:0] := WS2 ;
WS2) ST[3:0] := WS3 ;
WS3) ST[3:0] := WS4 ;
WS4) IF (DSACK[1:0]) THEN ST[3:0] := WS5 ;
 ELSE ST[3:0] := WS4 ;

WS5) BEGIN
 IF (CTR_ZERO) THEN ST[3:0] := IDLE ;
 IF (/BG) THEN ST[3:0] := BRQ ;
 IF (BG * /CTR_ZERO) THEN ST[3:0] := RS0 ;
 END;

END;

END.

Figure 3-106.a PLPL Specification for the 68020 DMA Controller Logic

" THIS IS THE PLPL SPECIFICATION FOR THE LOWER-UPPER BYTE
OF THE 68020 DMA ADDRESS POINTER.

- VINEET DUJARI 08/19/86 "

DEVICE LOWER_UPPER_ADR_PTR (AmPAL22V10)

PIN

CLK	= 1	VCC	= 24
D[8]	= 2	A[8]	= 23
D[9]	= 3	A[9]	= 22
D[10]	= 4	A[10]	= 21
D[11]	= 5	A[11]	= 20
D[12]	= 6	A[12]	= 19
D[13]	= 7	A[13]	= 18
D[14]	= 8	A[14]	= 17
D[15]	= 9	A[15]	= 16
LD	= 10	NC1	= 15
INC	= 11	CT	= 14
GND	= 12	ENB	= 13 ;

" THIS DEVICE GETS A LOAD INPUT (LD) FROM THE CPU THAT
LOADS THE VALUE FROM THE DATA BUS TO THE COUNTER
REGISTER. INCREMENT CONTROL (INC) INCREMENTS THE
COUNTER PROVIDED THE COUNT (CT) CONTROL IS ACTIVE. "

BEGIN

IF (ENB) THEN ENABLE (A[15:8]) ;

```

A[15] :=
  LD * D[15]                + " LOAD VALUE "
  /INC * A[15]              + " RECIRCULATE VALUE "
  INC * CT * (              + " COUNT UP "
    /A[15] * A[14] * A[13] *
    A[12] * A[11] * A[10] *
    A[9] * A[8]             +
    A[15] * /A[14]          +
    A[15] * /A[13]          +
    A[15] * /A[12]          +
    A[15] * /A[11]          +
    A[15] * /A[10]          +
    A[15] * /A[9]           +
    A[15] * /A[8]           +
    )                       +
  INC * /CT * A[15] ;      + " RECIRCULATE VALUE "

A[14] :=
  LD * D[14]                + " LOAD VALUE "
  /INC * A[14]              + " RECIRCULATE VALUE "
  INC * CT * (              + " COUNT UP "
    /A[14] * A[13] * A[12] *
    A[11] * A[10] * A[9] *
    A[8]                    +
    A[14] * /A[13]          +
    A[14] * /A[12]          +
    A[14] * /A[11]          +
    A[14] * /A[10]          +
    A[14] * /A[9]           +
    A[14] * /A[8]           +
    )                       +
  INC * /CT * A[14] ;      + " RECIRCULATE VALUE "

```

Figure 3-106.b PLPL Specification for the 68020 DMA Controller Logic (Continued)

```

A[13] :=
  LD * D[13]          + " LOAD VALUE "
  /INC * A[13]         + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[13] * A[12] * A[11] *
    A[10] * A[9] * A[8] +
    A[13] * /A[12] +
    A[13] * /A[11] +
    A[13] * /A[10] +
    A[13] * /A[9] +
    A[13] * /A[8]
  )
  INC * /CT * A[13] ;   " RECIRCULATE VALUE "

A[12] :=
  LD * D[12]          + " LOAD VALUE "
  /INC * A[12]         + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[12] * A[11] * A[10] *
    A[9] * A[8] +
    A[12] * /A[11] +
    A[12] * /A[10] +
    A[12] * /A[9] +
    A[12] * /A[8]
  )
  INC * /CT * A[12] ;   " RECIRCULATE VALUE "

A[11] :=
  LD * D[11]          + " LOAD VALUE "
  /INC * A[11]         + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[11] * A[10] * A[9] *
    A[8] +
    A[11] * /A[10] +
    A[11] * /A[9] +
    A[11] * /A[8]
  )
  INC * /CT * A[11] ;   " RECIRCULATE VALUE "

A[10] :=
  LD * D[10]          + " LOAD VALUE "
  /INC * A[10]         + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[10] * A[9] * A[8] +
    A[10] * /A[9] +
    A[10] * /A[8]
  )
  INC * /CT * A[10] ;   " RECIRCULATE VALUE "

A[9] :=
  LD * D[9]           + " LOAD VALUE "
  /INC * A[9]          + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[9] * A[8] +
    A[9] * /A[8]
  )
  INC * /CT * A[9] ;    " RECIRCULATE VALUE "

A[8] :=
  LD * D[8]           + " LOAD VALUE "
  /INC * A[8]          + " RECIRCULATE VALUE "
  INC * CT * (         " COUNT UP "
    /A[8]
  )
  INC * /CT * A[8] ;    " RECIRCULATE VALUE "

END.

```

Figure 3-106.b PLPL Specification for the 68020 DMA Controller Logic (Continued)

" THIS IS THE PLPL SPECIFICATION FOR THE LOWER-LOWER
 BYTE OF THE 68020 DMA ADDRESS POINTER.
 - VINEET DUJARI 08/19/86 "

DEVICE LOWER_LOWER_ADR_PTR (AmPAL22V10)

PIN

CLK	= 1	VCC	= 24
D[7]	= 2	A[7]	= 23
D[6]	= 3	A[6]	= 22
D[5]	= 4	A[5]	= 21
D[4]	= 5	A[4]	= 20
D[3]	= 6	A[3]	= 19
D[2]	= 7	A[2]	= 18
NC1	= 8	NC3	= 17
NC2	= 9	NC4	= 16
LD	= 10	NC5	= 15
INC	= 11	CT	= 14
GND	= 12	ENB	= 13 ;

" THIS DEVICE GETS A LOAD INPUT (LD) FROM THE CPU
 THAT LOADS THE VALUE FROM THE DATA BUS TO THE
 COUNTER REGISTER. INCREMENT CONTROL (INC) INCREMENTS
 THE COUNTER. COUNT (CT) CONTROL IS GENERATED FOR
 THE UPPER BYTE. "

BEGIN

IF (ENB) THEN ENABLE (A[7:2]) ;

```

A[7] :=
  LD * D[7]          + " LOAD VALUE "
  /INC * A[7]         + " RECIRCULATE VALUE "
  INC * (             " COUNT UP "
    /A[7] * A[6] * A[5] *
    A[4] * A[3] * A[2] *
    A[7] * /A[6]      +
    A[7] * /A[5]      +
    A[7] * /A[4]      +
    A[7] * /A[3]      +
    A[7] * /A[2]
  ) ;

```

```

A[6] :=
  LD * D[6]          + " LOAD VALUE "
  /INC * A[6]         + " RECIRCULATE VALUE "
  INC * (             " COUNT UP "
    /A[6] * A[5] * A[4] *
    A[3] * A[2]      +
    A[6] * /A[5]      +
    A[6] * /A[4]      +
    A[6] * /A[3]      +
    A[6] * /A[2]
  ) ;

```

```

A[5] :=
  LD * D[5]          + " LOAD VALUE "
  /INC * A[5]         + " RECIRCULATE VALUE "
  INC * (             " COUNT UP "
    /A[5] * A[4] * A[3] *
    A[2]              +
    A[5] * /A[4]      +
    A[5] * /A[3]      +
    A[5] * /A[2]
  ) ;

```

Figure 3-106.c PLPL Specification for the 68020 DMA Controller Logic (Continued)

```

A[4] :=
  LD * D[4]          + " LOAD VALUE "
  /INC * A[4]         + " RECIRCULATE VALUE "
  INC * (            + " COUNT UP "
    /A[4] * A[3] * A[2] +
    A[4] * /A[3]      +
    A[4] * /A[2]
  ) ;

A[3] :=
  LD * D[3]          + " LOAD VALUE "
  /INC * A[3]         + " RECIRCULATE VALUE "
  INC * (            + " COUNT UP "
    /A[3] * A[2]      +
    A[3] * /A[2]
  ) ;

A[2] :=
  LD * D[2]          + " LOAD VALUE "
  /INC * A[2]         + " RECIRCULATE VALUE "
  INC * (            + " COUNT UP "
    /A[2]
  ) ;

CT := A[7] * A[6] * A[5] * A[4] * A[3] * A[2] ;

END.

```

Figure 3-106.c PLPL Specification for the 68020 DMA Controller Logic (Continued)

TABLE 3-18. OUTPUT TABLE

State	Outputs													
	BR	BGACK	AS	DS	R/W	INC_SRC	ENB_SRC	INC_DST	ENB_DST	DATA_ENB	DATA_LTH	CTR_DEC	SIZ1	SIZ0
IDLE	0	0	Z	Z	Z	0	0	0	0	0	0	0	Z	Z
BRQ	1	0	Z	Z	Z	0	0	0	0	0	0	0	Z	Z
BGT	1	1	Z	Z	Z	0	0	0	0	0	0	0	Z	Z
RS0	1	1	0	0	R	0	1	1	0	0	0	0	0	0
RS1	1	1	1	1	R	0	1	1	0	0	0	0	0	0
RS2	1	1	1	1	R	0	1	0	0	0	0	0	0	0
RS3	1	1	1	1	R	0	1	0	0	0	1	0	0	0
RS4	1	1	1	1	R	0	1	0	0	0	1	1	0	0
RS5	1	1	0	0	R	0	1	0	0	0	0	1	0	0
WS0	1	1	0	0	W	1	0	0	1	0	0	0	0	0
WS1	1	1	1	0	W	1	0	0	1	0	0	0	0	0
WS2	1	1	1	0	W	0	0	0	1	1	0	0	0	0
WS3	1	1	1	1	W	0	0	0	1	1	0	0	0	0
WS4	1	1	1	1	W	0	0	0	1	1	0	0	0	0
WS5	1	1	0	0	W	0	0	0	1	1	0	0	0	0

3.4.4 INTERFACING TO THE 8088/80188

Overview

The 8088 CPU is an 8-bit processor designed around the 8086 internal structure. Most functions of the 8088 are identical to the equivalent 8086. The 8088 fetches and writes 16-bit words in two consecutive bus cycles. Both the 8086 and the 8088 handle the external bus the same way but the 8088 handles only 8-bits at a time; and both appear identical to the software engineer, with the exception of execution time.

The hardware interface of the 8088 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

A_8-A_{15}	These pins are only address outputs on the 8088. They are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
$\overline{SS}$	Provides the $\overline{SO}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only.
$DT/\overline{R}$, $IO/\overline{M}$, and $\overline{SS}$	provide the complete bus status in minimum mode.
$IO/\overline{M}$	has been inverted to be compatible with the MCS-85 bus structure.

ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

The 8088 and AMD Proprietary Peripherals

The evolution of chip design has taken the 8-bit environment into the 16-bit environment. While the new generation of peripheral devices are often 16 bits wide, the older, established 8-bit orientation of CPUs and peripherals are still significant. Interfacing a 16-bit peripheral with an 8-bit CPU often encounters data path incompatibility and involves bus control manipulation. This type of integration mainly involves separating the control and data paths from the new peripheral and the system.

The ability to mix different data path widths can improve system functionality, performance, and cost. It is less expensive to use an 8-bit bus in a new design because the memory requirements are generally cheaper. A designer can use this data path mixing to upgrade the existing system until a new system design is warranted, or the designer can simply improve on the existing design as new peripherals become available. AMD makes a number of proprietary peripherals and the following sections show users with 8-bit systems how to incorporate those AMD products into their designs.

8088 and Am8052 CRT Controller Interface

The interface technique between the Am8052 CRT Controller and an 8-bit microprocessor also applies to the 8088 and Am8052 interface.

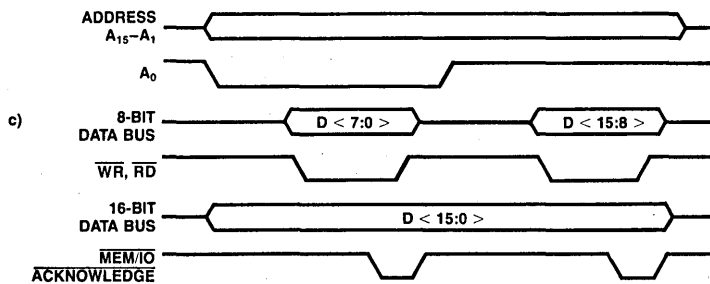
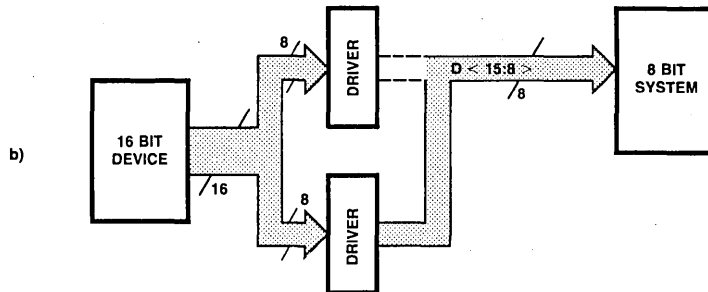
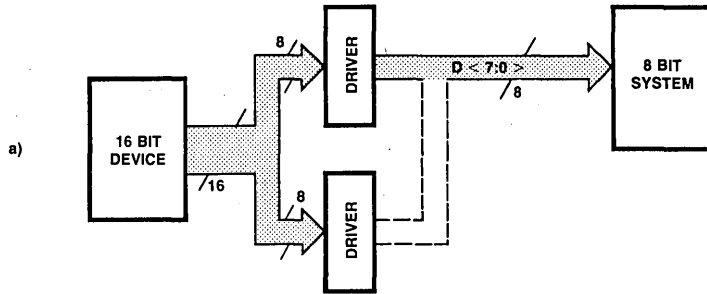
There are two fundamental issues associated with mixing devices that communicate over different-sized buses. The first problem is allowing the two devices to communicate on a "common" data bus. Consider, for example, a 16-bit system utilizing 8- and 16-bit peripherals. Overcoming the mismatched data paths requires some form of controlled multiplexing/demultiplexing of the different data paths. In addition, extra control signals for partitioning the 16-bit word into 8-, and 16-bit units may be required. Today, most of the 16-bit CPU based systems that use 8-bit peripherals usually use just the lower half of the data bus to transfer data to and from the peripheral. However, this scheme does not work when interfacing 16-bit peripherals to 8-bit CPUs, especially when these peripherals have bus master capability.

Data Funnelling

When a 16-bit peripheral attempts to transfer data over an 8-bit bus (memory write cycle or slave read cycle), the 16-bit data bus has to be broken down into two bytes and transferred sequentially. First, the lower 8-bits are transferred out on the bus (Figure 3-107.a), and then in the next transfer cycle the upper 8-bits of the 16-bit word are sent out (Figure 3-107.b). The generalized bus timing for such an operation is shown in Figure 3-107.c. Figures 3-108.a, 3-108.b, and 3-108.c show the opposite case; a bus read operation from an 8-bit bus to a 16-bit peripheral. Here, the first byte read from the system must be latched. Once the second byte has been fetched, the 16-bit peripheral reads in the assembled 16-bit (2-byte) word. Additionally, provisions may need to be made for the case when the 16-bit peripheral accesses single bytes.

Interruptions of the two cycle transfer must be analyzed very carefully. Master transfers may not be interrupted by slave accesses while being in the middle of a two-cycle transaction. Similar, slave accesses may not be interrupted by master transfers. While the interface funnels the data, the current bus cycle needs to be stretched. When the peripheral is bus master, as shown in Figures 3-107.a, 3-107.b, and 3-107.c, the 16-bit peripheral is holding its data available for what would normally be two complete bus transfer cycles. This stretch can be achieved by delaying the transfer acknowledge signal to the peripheral, causing it to wait ($\overline{WAIT}$ asserted).

In slave mode, the 8-bit CPU would have to make two consecutive read operations to examine a 16-bit peripheral status register. The peripheral must not become bus master in-between the first and second read operations since this invalidates the results of the first read operation. This function can be handled in two different ways: if the CPU has a bus lock instruction (for example, like the iAPX family of CPUs), then the programmer uses one of these before the CPU accesses the peripheral. Alternately, the CPU can disable the arbitration logic while it is performing the critical uninterruptible slave transfer.



02188A-45

Figure 3-107. Bus Master Write or Slave Read Operation

Developing the Control and Data Transfer Interface

Designing the control interface to allow mixing 8- and 16-bit peripherals requires an analysis of the data and control flow. The data flow automatically defines the data path design (see Figures 3-107 and 3-108). The bus master operation by the peripheral is relatively straightforward. During a write operation, the data is written out sequentially: the lower byte first and then the upper byte (or vice-versa). During a read operation, the data is fetched sequentially. The byte fetched first is latched, to hold the data until the peripheral can read it. In the second byte read cycle, the remaining byte is fetched, the 16-bit word is assembled from the two bytes, and the 16-bit

word is loaded into the peripheral. Similarly, $\overline{\text{WAIT}}$ is asserted until the second byte read cycle can be terminated.

The slave mode of operation works almost identically to the peripheral bus master mode. The master read cycle is similar to the slave write cycle, and the master write cycle is similar to the slave read cycle. In general, if the peripheral puts data on the narrower system bus, the peripheral can keep the data active in both sequential system bus cycles. On the other hand, if data is loaded into the peripheral, the interface logic has to latch the data of the first fetch cycle, whereas the data of the second cycle can be loaded directly into the peripheral (no latching required).

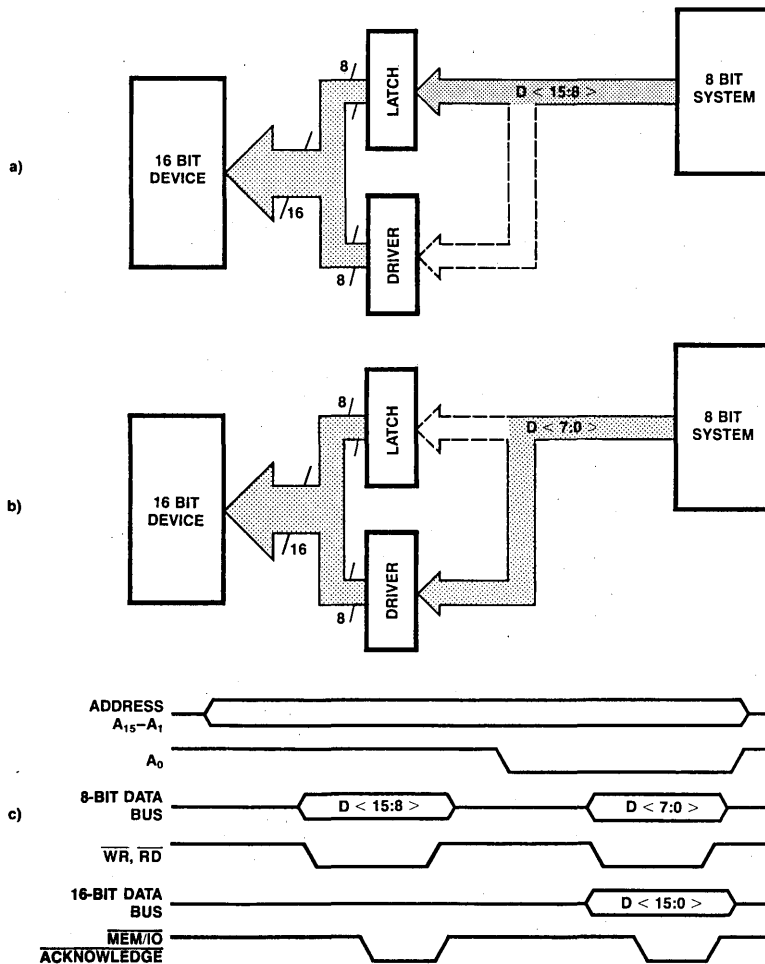


Figure 3-108. Bus Master Read or Slave Write Operation

02188A-46

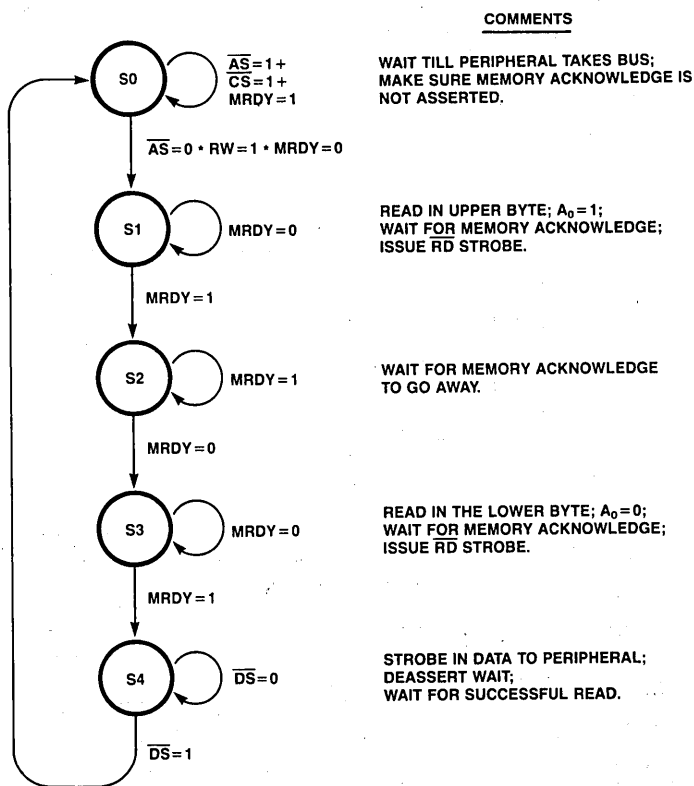
When defining the interface, the designer must make a conscious choice of which byte (upper or lower) to latch during peripheral read operations (or conversely, slave peripheral write operations). Once this decision has been made, the CPU must always access the latched data byte first (during a slave write) and then access the non-latched byte to complete the transfer. This restriction is a minor one with no extra software overhead; yet it could affect the ease of the programmer's coding if not handled properly. For example, if the programmer uses a compiler to generate the software for the system, extra care may be necessary to ensure that the compiler generates the correct addressing sequence. An alternative to this solution would be to latch both the upper and lower data bytes. In that case, the cost of the interface would be increased, as would the complexity, with no gain in performance.

The state diagram (Figure 3-109) illustrates the control sequence implemented in the 8/16-bit bus control logic. It also depicts how uninterrupted word transfers will occur and how the addresses for upper and lower bytes are generated. In

addition, the specific bus timing of the peripheral and the data bus must be examined to quantify the state control flow and provide information on data latching, read/write control strobes, and addressing to and from the peripheral. The state control flow is broken down into three parts: bus master read, slave read, and slave write operations.

The three control signals that must be generated by the 8/16-bit control unit are: Address bit 0 (A_0), peripheral hold ($\overline{WAIT}$), and bus read ($\overline{RD}$). The A_0 line is generated by the control logic to indicate which byte is to be transferred in bus master modes only. Otherwise, the A_0 generated by the system is used to indicate which byte is being accessed. The $\overline{WAIT}$ line holds up the peripheral during transfers. The $\overline{RD}$ line is required to indicate successive transfer cycles on the bus. The peripheral's control signals strobe active only once, because the two-cycle transfer must be kept hidden from the peripheral.

The slave transfer flow is almost identical, except that the CPU is generating the bus signals and the transfer directions are reversed, that is, a bus write goes into the peripheral.



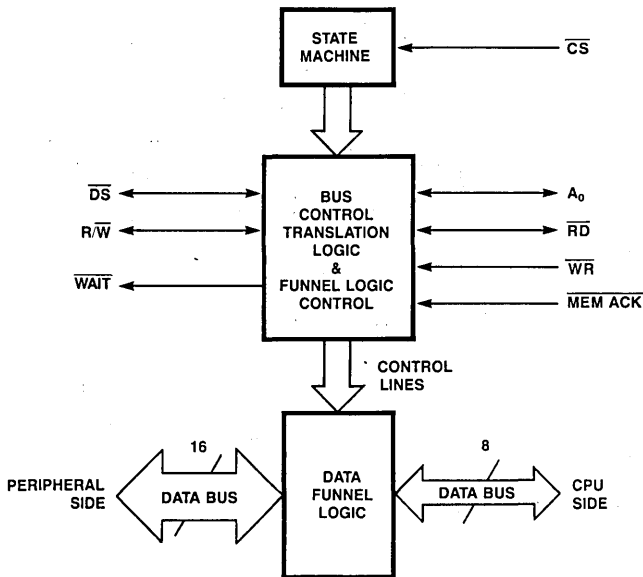
02188A-47

Figure 3-109. Bus Master Read State Flow-Control

The conceptual logic for the 16- to 8-bit data flow example is shown in Figure 3-110. The data on the upper byte is latched when data is being read (as a bus master) and read or written (as a bus slave). Although this interface must latch data coming from the 8-bit data bus into the peripheral, it also needs to act as transceiver when the peripheral is sending data out to the system. The ideal part to accomplish such an interface would be one that has a three-stated output, with an 8-bit wide latch, in one direction and a three-stated driver in the other direction. The Am2952 8-bit bidirectional I/O port combines the upper data bus latch and upper data driver chips into one IC. It provides two 8-bit clocked I/O ports, each with three-state output controls and individual clocks and clock enables. An Am2949 bidirectional bus transceiver completes the logic required to buffer the data path.

The state flow control requires logic capable of sequentially moving from state to state, holding in a particular state, and being reset or initialized back to a predefined state. This design integrates the state machine generator and the control signal logic into the same Programmable Array Logic (PAL) device.

A considerable amount of logic is required to generate the data-path flow logic and the bus control signals. This is especially true if the peripherals and CPUs have different signal conventions (for example, $\overline{AS}$, $\overline{DS}$, and R/W versus ALE , $\overline{RD}$, and $\overline{WR}$). Conversion between different signal conventions, signal polarity changes, and extra functions (such as generating A_0) requires quite a bit of logic and design effort. If the peripheral has bus master capability, additional information, such as bus arbitration controls, must be fed into the next state determination logic to decide what control sequence to follow.



02188A-48

Figure 3-110. Conceptual 16/8-Bit Conversion Logic

Figure 3-111 shows a typical 8/16-bit control interface which combines all the individual components discussed above. The state machine and the bus and latch controls have to be tightly coupled in order to transfer data between the 8-bit and 16-bit buses. The generalized machine is designed under the assumption that the peripheral has bus master capability. If this is not the case, the design can be greatly simplified.

Since the CRTC does not modify system memory, no provision for a bus master write operation is required. This is important because it eliminates the need to generate a system write control signal ($\overline{WR}$). In addition, the control and display information will always be aligned on word boundaries. This relieves the 8/16-bit control logic from worrying about funneling the bytes and performing odd/even byte transfers. It also saves control inputs from the Am8052 because all transfers are words; there is no need for upper and lower data strobes or byte high enable inputs/outputs.

The slave accesses by the CPU are either pointer writes (to select the desired control/status register) or 16-bit data read/write operations. The pointer write operation is really an 8-bit operation because only the lower 8 bits of the data form the register address. The three different transfer timings are shown in Figures 3-112, 3-113, and 3-114.

Two special conditions have been incorporated into the state flow diagrams whenever a transfer is first initiated. Before a new transfer cycle is attempted (that is, while the state machine is waiting in S0), the memory acknowledge must be inactive. This prevents any interference from the last transfer. The second special condition occurs when the Am8052 asserts the R/W line to indicate a write operation. Whenever the Am8052 updates the upper 8 bits of the 24-bit address latch, the R/W line indicates a write operation (in conjunction with $\overline{AS}$). The Am8052 is not actually performing a system data write, only an address latch update. Hence, the state flow reflects this fact by not starting a sequence if the R/W line is active Low from the Am8052.

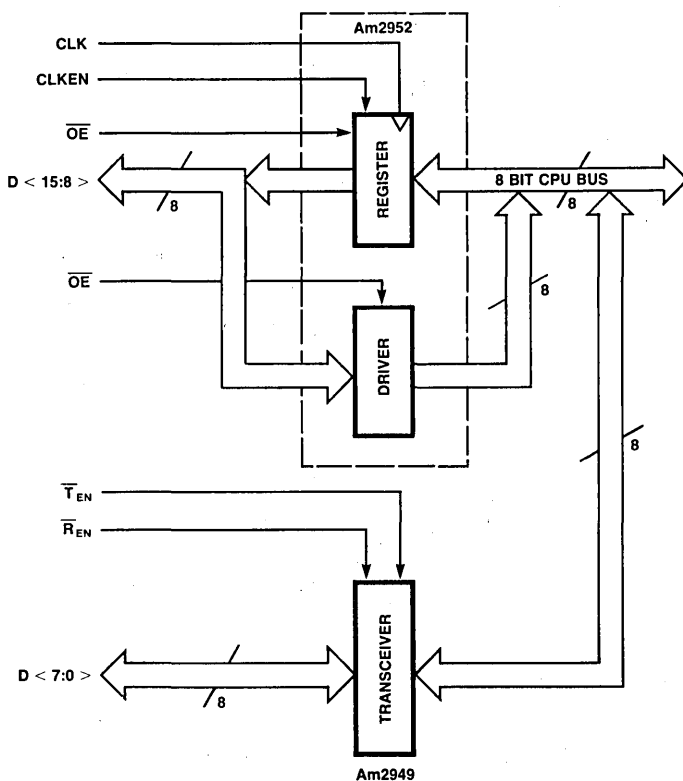


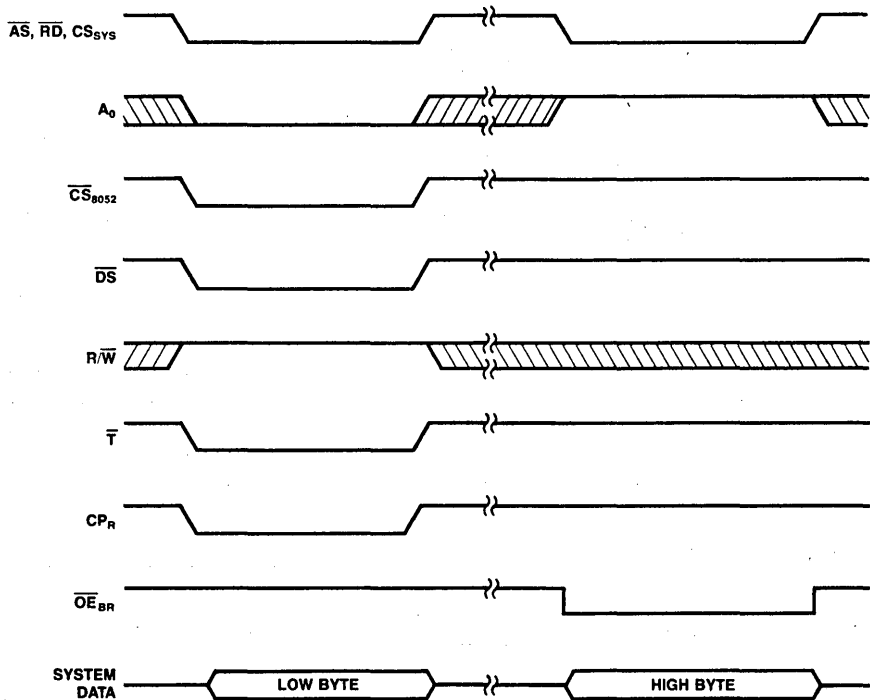
Figure 3-111. Data Funnel Logic

02188A-49

3

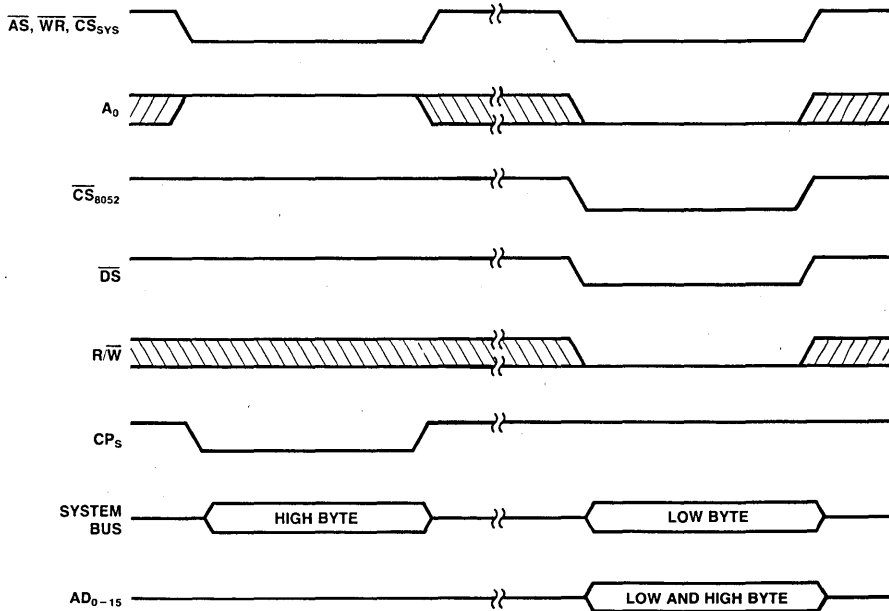


3-143



02188A-51

Figure 3-113. Slave Read Timing Diagram



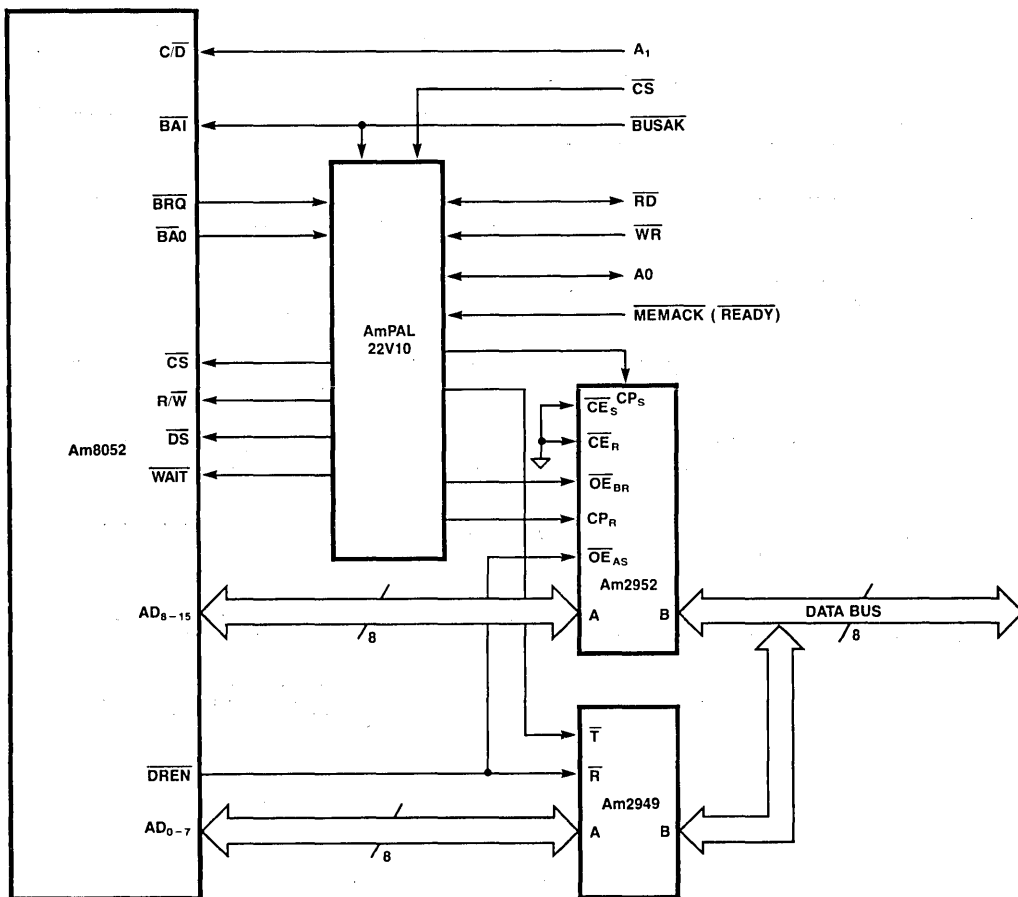
02188A-52

Figure 3-114. Slave Write Timing Diagram

These simplifications make it possible to combine the Am8052 to an 8-bit CPU control interface in a single AmpAL22V10 device (Figure 3-115) which also converts the bus control signals from $\overline{AS}$, $\overline{DS}$, and $R/\overline{W}$ to $\overline{RD}$ and $\overline{WR}$. Figure 3-115 shows the assembled control and data transfer logic for this interface. The minimum Am8052 and bus control signals that have to be generated are $\overline{RD}$, A_0 , $\overline{DS}$, $R/\overline{W}$. Although $\overline{DS}$ and $R/\overline{W}$ are used as inputs during a bus master operation by the Am8052, the AmpAL22V10 must convert the CPU $\overline{RD}$ and $\overline{WR}$ signals to $\overline{DS}$ and $R/\overline{W}$ for slave I/O operations. The signals A_0 and $\overline{RD}$ are generated by the control logic when the Am8052 is performing a read access to the system. The $WAIT$ (or not $READY$) signal to the Am8052 also needs to be generated by the control logic. Additionally, the four control signals of the bidirectional port and transceiver are generated.

Trade-Offs and Limitations

In a design dramatically affecting the I/O of the system, a number of trade-offs and limitations should be noted. The most obvious limitation in using 16-bit peripherals on an 8-bit bus is that the 16-bit peripheral will be under-utilized. The speed of all I/O operations will be cut by 50%. Consequently, the bus utilization percentage will go up if the 16-bit peripheral represents a significant factor of the bus usage. A CRT controller like the Am8052 might use 5% to 10% of the bus bandwidth for display information when using 16-bit I/O. Converting to 8-bit I/O would double bus usage to 10% or 20%.



02188A-53

Figure 3-115. Am8052 8-Bit Interface

Another factor that might affect the bus usage is the efficiency of the 8- to 16-bit conversion control logic. If the state machine designed to perform the 8/16-bit conversion (or 16/32-bit) is improperly designed, then extra transfer overhead may be introduced. This could mean that a sequential transfer of two 8-bit values takes longer than two single 16-bit transfers. The system designer must weight the cost of the extra overhead on a case-by-case basis. Most interfaces outside a system's immediate family require some kind of extra interface logic anyway. Therefore, by optimizing the control signals and incorporating them into programmable logic devices such as the AmPAL22V10, the IC count can be dramatically reduced.

8088 and Am28068 Data Ciphering Processor Interface

Figure 3-116 shows the CPU-DMA interface. The CPU is operating in Maximum Mode. The bus arbitration handshake of the DMA controller (HREQ and HACK) must be translated into the Bus Request/Grant handshake of the 8088 CPU.

If the CPU is programmed to operate in Minimum Mode, both devices have the same bus arbitration handshake. The HREQ and HACK of the DMA controller can be connected directly to the corresponding pins of the CPU (HREQ to HACK).

Master Port Read/Write is latched in the D Flip-Flop. It is clocked in an output operation with $\overline{CS_3}$ active. One of the data lines is latched in to define the status on the MR/W input. This is necessary because the DCP requires a set-up time of 100 ns of MR/W to the Data Strobe. Generation of MR/W for each cycle of a high-speed data transfer session of the DMA controller would extend each cycle and slow down the maximum throughput. This logic cannot be integrated into the PAL device because of the flip-flop's asynchronous clock.

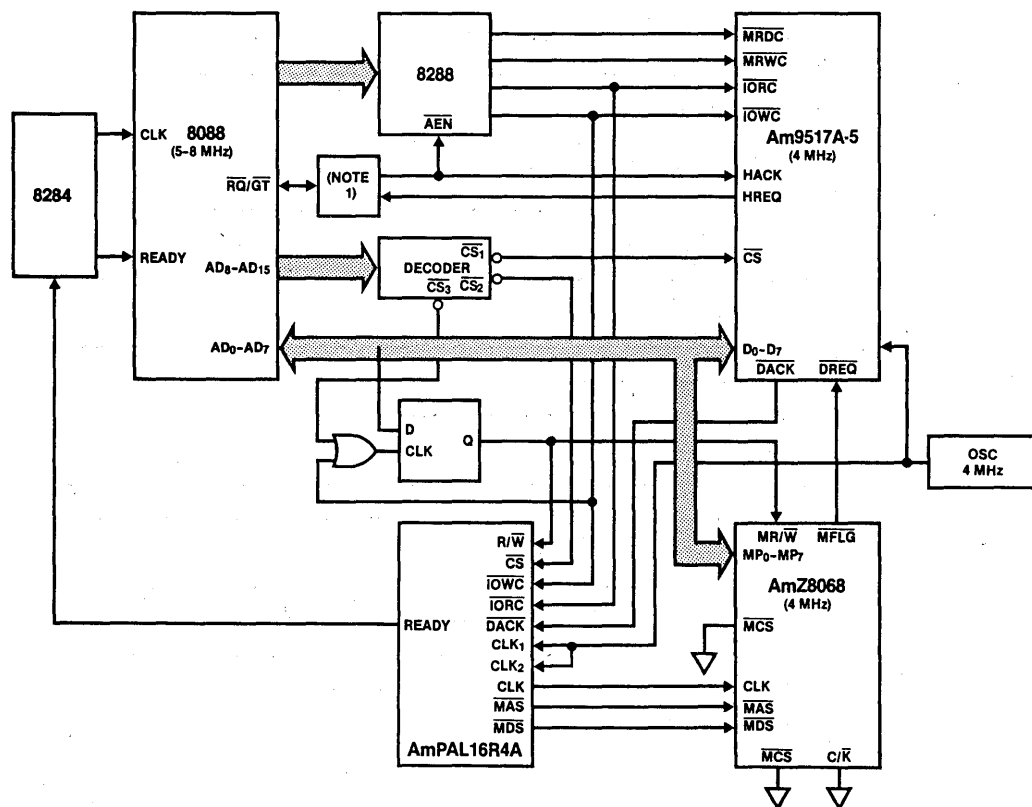


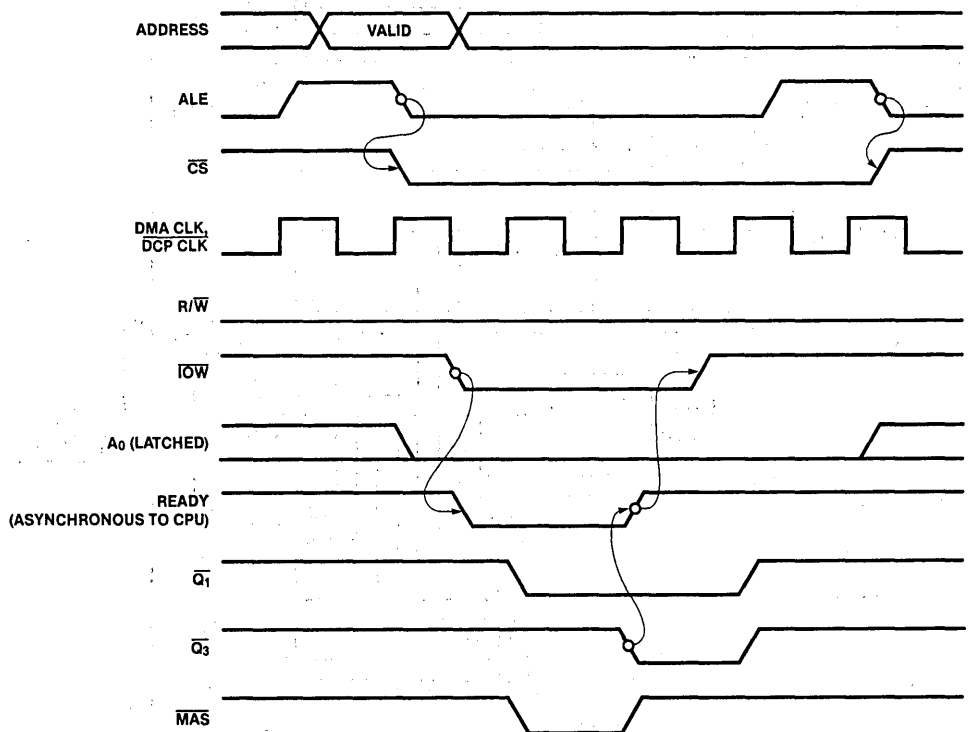
Figure 3-116. 8088-Am9517-AmZ8068 Interface

Before executing an access to the DCP, the CPU must latch the MR/W. The transfer itself is evaluated in a two-cycle operation.

Master Port Address Strobe (MAS) is only generated if the CPU executes an output instruction to a specific I/O address ($\overline{CS}_2$ active, A_0 =Low) (Figure 3-117) Address Latch Enable of the CPU (ALE) cannot be used for the generation of MAS because

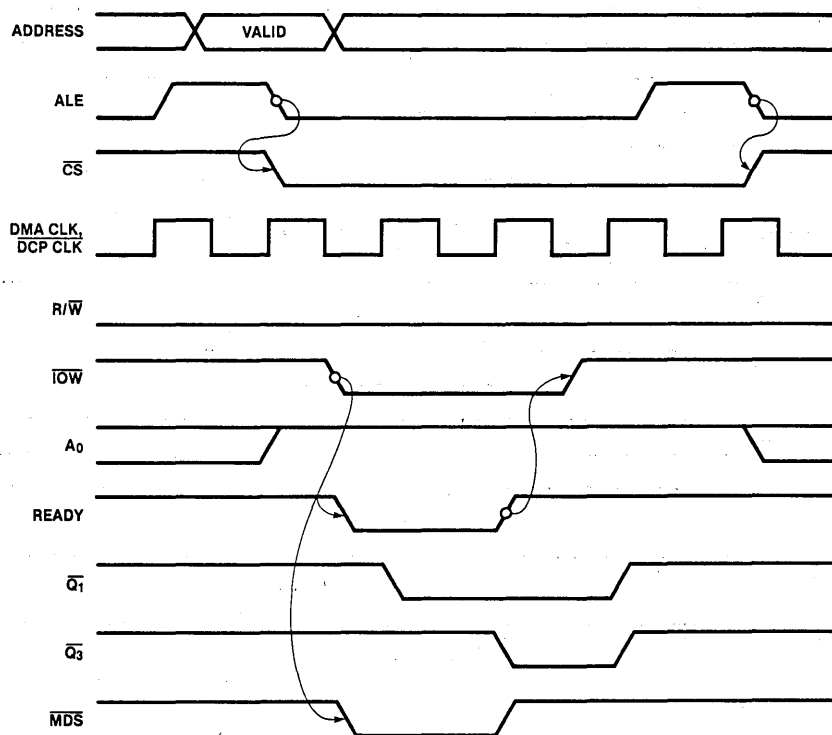
the CPU must set up the DCP for data transfer before a DMA transfer session is started. The DCP is set up by putting out a 00H (data register address) to the I/O address mentioned above.

Figures 3-118 and 3-119 show data read and write cycles. Figure 3-120 shows DMA data read and write cycles.



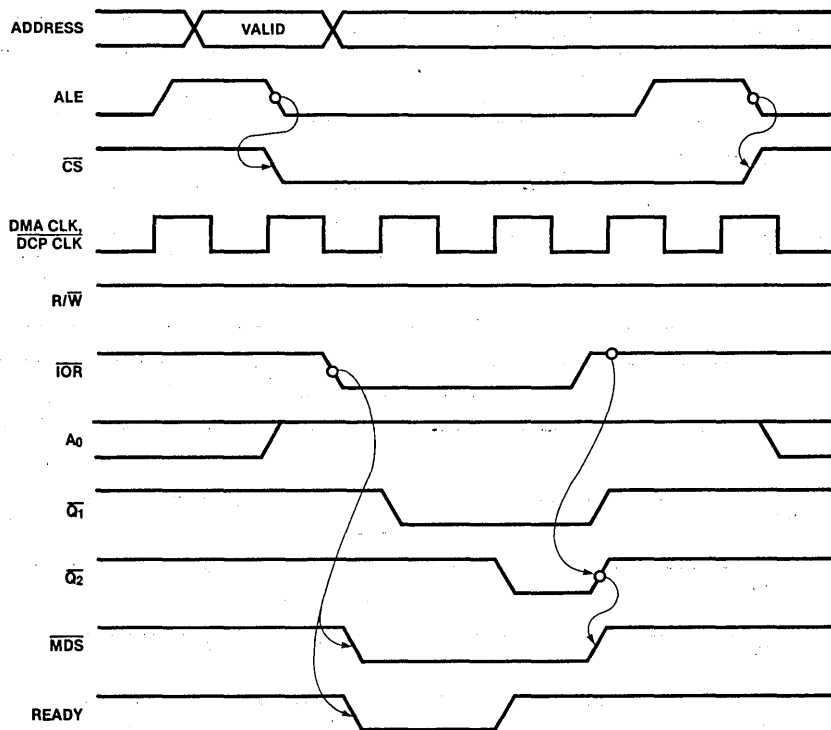
02188A-55

Figure 3-117. Address Latch Cycle Timing (CPU-DCP)



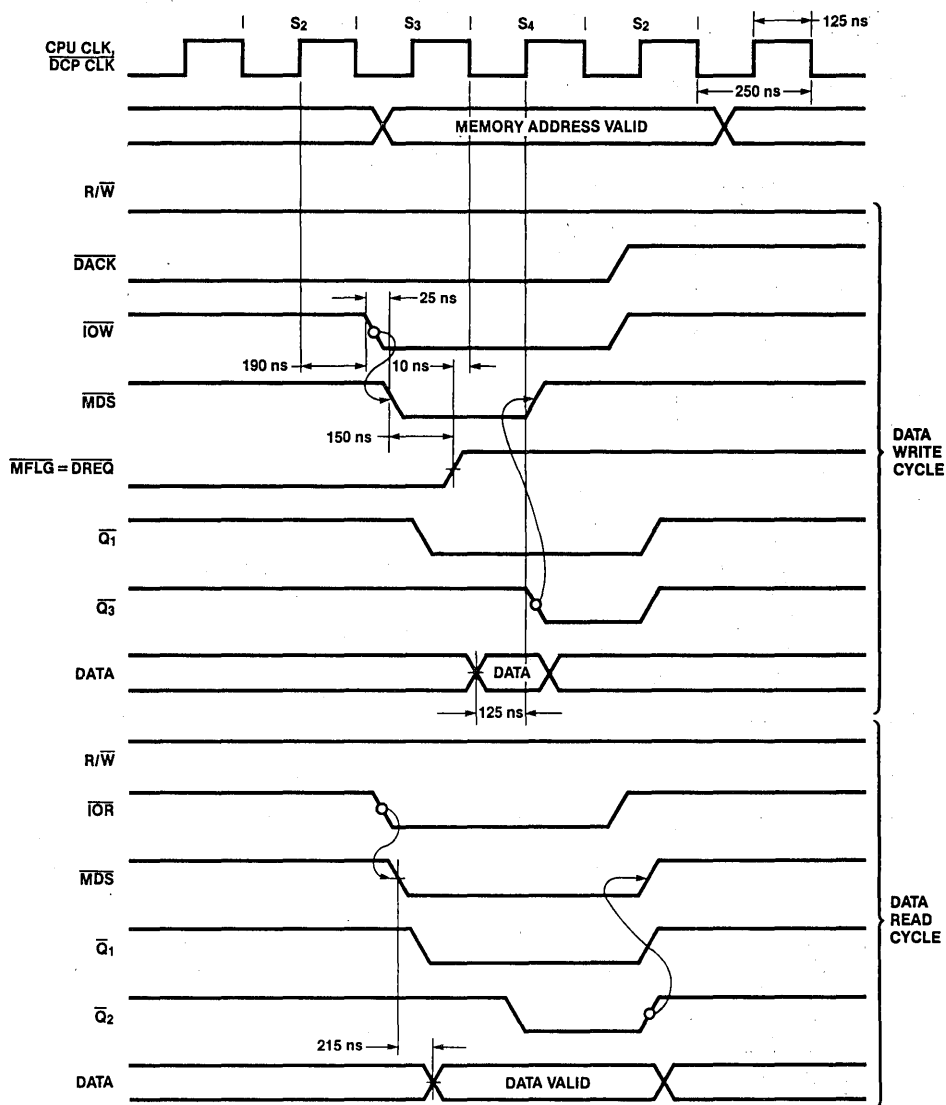
02188A-56

Figure 3-118. Data Write Cycle Timing (CPU-DCP)



02188A-57

Figure 3-119. Data Read Cycle Timing



02188A-58

Figure 3-120. DMA-DCP Timing Diagram

PAL DESIGN SPECIFICATION

JUERGEN STELBRINK 8-12-83

8088- AM9517(DMA)- AMZ8068(DCP) INTERFACE DEVICE

ADVANCED MICRO DEVICES

CLK1	CLK2	/CS	/IOR	/IOW	A0	RW	/DACK	NC	GND
/OE	/MDS	READY	/Q1	/Q2	/Q3	/MAS	NC	CLK	VCC

```
MAS      :=  IOW*/IOR*CS*/A0*/Q3*/MAS      ; MASTER PORT ADDRESS STROBE
```

```
Q1      := CS*IOR*/IOW*RW*/Q2  +
          CS*IOW*/IOR*/RW*/Q3  +
          DACK*IOR*/IOW*RW*/Q2  +
          DACK*IOW*/IOR*/RW*/Q3
```

```
Q2      := CS*IOR*/IOW*RW*Q1      +
        CS*IOR*/IOW*RW*Q2      +
        DACK*IOR*/IOW*RW*Q1     +
        DACK*IOR*/IOW*RW*Q2
```

```
Q3      := CS*IOW*/IOR*/RW*Q1      +
          CS*IOW*/IOR*/RW*Q2      +
          DACK*IOW*/IOR*/RW*Q1     +
          DACK*IOW*/IOR*/RW*Q2
```

```

MDS      = CS*A0*IOR*/IOW*RW      +      ; MASTER PORT READ
          DACK*IOR*/IOW*RW          +
          Q2*A0                     +
          CS*A0*IOW*/IOR*/RW*/Q3+      ; MASTER PORT WRITE
          DACK*IOW*/IOR*/RW*/Q3

```

```

/READY = CS*/A0*IOW*/IOR*/RW*/Q3+      ; ADDRESS LATCH CYCLE
        CS*A0*IOW*/IOR*/RW*/Q3 +      ; DATA WRITE CYCLE
        CS*A0*IOR*/IOW*RW*/Q2

```

```

/CLK = CLK2 ; DCP CLOCK

```

FUNCTION TABLE

```
CLK1 CLK2 /CS /IOR /IOW /DACK A0 RW CLK /MAS /MDS READY /Q1 /Q2 /Q3
```

										R						
C	L	L	/	/	/	D				C	M	/	E			
K	K	C	O	O	C	A	R	L	A	D	D	Q	Q	Q		
1	2	S	R	W	K	0	W	K	S	S	Y	1	2	3	COMMENT	

; CLOCK GENERATION																
X	L	X	X	X	X	X	X	X	H	X	X	X	X	X	X	
X	H	X	X	X	X	X	X	L	X	X	X	X	X	X	X	
; ADDRESS LATCH																
C	X	H	H	H	H	L	L	X	H	H	H	H	H	H	H	; CPU
X	X	L	H	L	H	L	L	X	H	H	L	H	H	H		

Figure 3-121. Source Listing for 8088 to Z8068 DCP Interface

```

C X L H L H L L X L H L L H H
C X L H L H L L X H H H L H L
C X H H H H L L X H H H H H H
; READ DATA
X X H H H H H X H H H H H H ; CPU
X X L L H H H H X H L L H H H
C X L L H H H H X H L L L H H
C X L L H H H H X H L H L L H
C X L L H H H H X H L H H L H
C X H H H H H X H H H H H H
X X H L H L X H X H L H H H H ; CYCLE S3 (DMA)
C X H L H L X H X H L H L H H
C X H L H L X H X H L H L H H ; CYCLE S4
C X H H H X H X H H H H H H ; CYCLE S2
; WRITE DATA
X X L H L H H L X H L L H H H ; CPU
C X L H L H H L X H L L L H H
C X L H L H H L X H H H L H L
C X H H H H H L X H H H H H H
X X H H L L H L X H L H H H H ; CYCLE S3 (DMA)
C X H H L L H L X H L H L H H
C X H H L L H L X H H H L H L ; CYCLE S4
C X H H H H L X H H H H H H ; CYCLE S2
; INVALID CYCLES
X X L L L H H H X H H H H H H
X X L L H H H L X H H H H H H
X X L H L H H H X H H H H H H
-----

```

DESCRIPTION:

THIS PAL GENERATES ALL NECESSARY BUS CONTROL SIGNALS, TO INTERFACE A 8088 CPU AND A AM9517 DMA CONTROLLER TO THE AMZ8068 DATA CIPHERING PROCESSOR. THE MAXIMUM SYSTEM CLOCK FOR THE DMA CONTROLLER AND THE DCP IS 4 MHZ, THE SYSTEM CLOCK OF THE CPU CAN BE UP TO 8 MHZ. THE DEVICES ARE WORKING ASYNCHRONOUSLY.

INPUT SIGNALS:

CLK1, DMA CLOCK
CLK2

/CS CHIP SELECT FOR THE DCP, GENERATED BY A DECODER LOGIC

/IOR INPUT/ OUTPUT READ

/IOW INPUT/ OUTPUT WRITE

A0 LEAST SIGNIFICANT BIT OF THE Z80 ADDRESS BUS TO SELECT THE TYPE OF OPERATION:

A0 = LOW SELECT DCP REGISTER FOR NEXT DATA CYCLES
(ADDRESS LATCH)

A0 = HIGH READ OR WRITE INTERNAL REGISTER

Figure 3-121. Source Listing for 8088 to Z8068 DCP Interface (Continued)

(DATA TRANSFER TO CONTROL, MODE, INPUT OR
OUTPUT REGISTER)

/DACK DMA ACKNOWLEDGE FROM DMA CONTROLLER, TREATED AS /CS=LOW
AND AO=HIGH

RW READ/ WRITE SIGNAL STORED IN A EXTERNAL LATCH, TO ALLOW
A DMA OPERATION WITHOUT WAIT STATES. THIS SOLVES THE
PROBLEM OF THE SETUP TIME OF MR/W OF THE MASTER PORT TO
MDS GOING ACTIVE. THE STATUS OF THIS SIGNAL MUST AGREE
WITH /IOR OR /IOW OR THE PAL GENERATES NO STROBES.

OUTPUT SIGNALS:

CLK INVERTED DMA CLOCK FOR THE DCP

/MAS MASTER PORT ADDRESS LATCH ENABLE, ACTIVE DURING ADDRESS
LATCH CYCLES TO LATCH THE REGISTER ADDRESS ON MP1 AND MP2
(2 LINES OF THE MASTER PORT BUS) AND THE STATE OF /MCS
IN. THE DCP STORES INTERNALLY THE ADDRESS AND CHIP SELECT
TO THE NEXT ADDRESS LATCH CYCLE

/MDS MASTER PORT DATA STROBE, TO TIME DCP DATA TRANSFERS

/Q1, INTERNAL USED STATE SIGNALS (DO NOT CONNECT). Q1 IS ACTIVE 2
/Q2, CLOCK CYCLES IN ALL CYCLES. IT IS USED TO GENERATE THE DELAYED
/Q3 Q2 AND Q3. Q2 IS ACTIVE IN A DATA READ CYCLE. IT ALLOWS /MDS
TO BE ACTIVE UNTIL /IOR HAS GONE INACTIVE. Q3 IS ACTIVE IN AN
ADDRESS LATCH OR DATA WRITE CYCLE. Q3 DISABLES READY AND /MDS
IN THE SECOND HALF OF THE CYCLE.

Figure 3-121. Source Listing for 8088 to Z8068 DCP Interface (Continued)

3

slave and master operations, the system must use the latched A_0 (LA₀) from the AmPAL22V10 as its sole A_0 . That is, this LA₀ must circumvent any address latching to memory and so is to be used directly.

As can be seen from the figures, the AmPAL22V10 manipulates the control signals to the transceivers and latches so that bytes can be funneled into words during Am9516 Command Chaining operations and Slave Writes, and words can be funneled into bytes during Am9516 Slave Reads. Also, LAO is alternately toggled in order to provide the dual address (16-bits) during chaining operations.

Figure 3-125 shows the PLPL equations and resulting sum-of-products equations for the design (PLPL is a higher level “C-like” language for PAL devices). For those not familiar with “C” or Pascal, the intermediate sum-of-products form shows the equations in Boolean form.



3-155

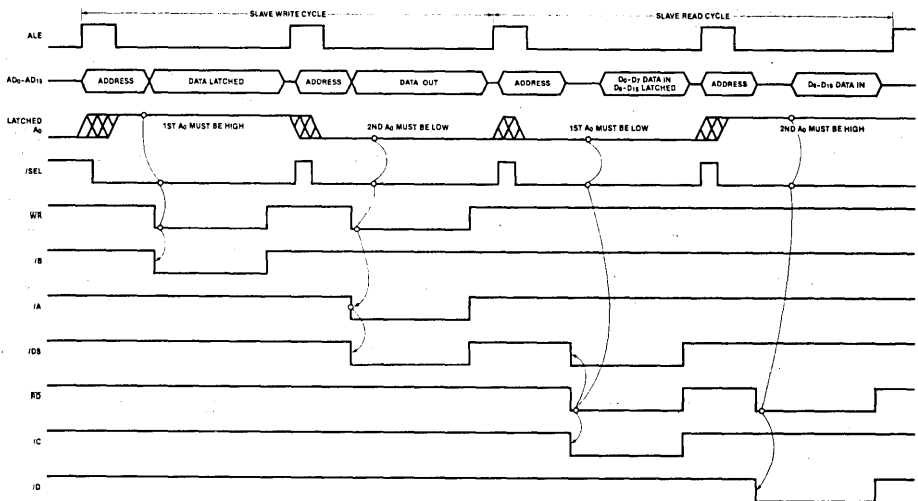


Figure 3-123. The Am9516 UDC to 8088 CPU Bus Slave Timing

02188A-61

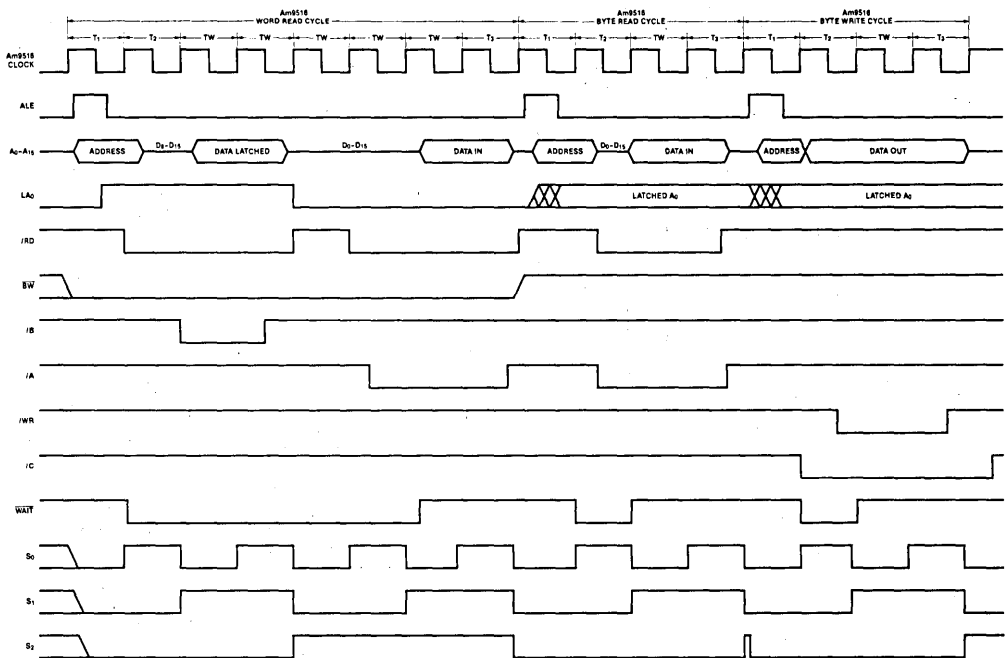


Figure 3-124. The Am9516 UDC to 8088 CPU Bus Master Timing

02188A-62

DEVICE _8088_9516 (pal22v10)

"This is a PLPL file to implement an interface between the 8-bit 8088 and the 16-bit Am9516. This design assumes that the Am9516 is programmed for byte operations and so only accesses 16-bit words during chaining. -James Williamson"

PIN

CK	= 1	/RD	= 23
S[0:2]	= 2:4	/WR	= 22
AO	= 5	LAO	= 21
/SEL	= 6	/DS	= 20
ALE	= 7	/RW	= 19
HLDA	= 8	/WAIT	= 18
/BW	= 9	/A	= 17
READY	= 10	/B	= 16
RESET	= 11	/C	= 15
		/D	= 14;

BEGIN

IF (RESET) THEN ARESET();

"=====
 "This section defines the wiggles when the Am9516 is Bus Master"
 "=====

IF (HLDA) THEN ENABLE();

IF (/S[2] * HLDA) THEN BEGIN

IF (/S[1] * /S[0]) THEN

LAO = /CK * BW + /BW * AO * ALE +
 /BW * LAO * /ALE ;

ELSE

LAO = BW + /BW * AO * ALE +
 /BW * LAO * /ALE ;

END;

IF (HLDA) THEN

CASE (S[2:0])

BEGIN

1) BEGIN

RD = /RW * DS ;

A = /BW * /RW * /CK ;

WR = /BW * RW * DS ;

C = /BW * RW ;

WAIT = 1 ;

END;

2) BEGIN

RD = /RW * DS ;

B = BW ;

A = /BW * /RW ;

WR = /BW * RW * DS ;

C = /BW * RW ;

WAIT = BW ;

END;

Figure 3-125. PLPL Specification for 8088 to Am9516 Interface

```

3) BEGIN
    RD  = /RW * DS * B      ;
    B   = BW * CK           ;
    A   = /BW * RD          ;
    WR  = /BW * RW * DS    ;
    C   = /BW * RW          ;
    WAIT = BW               ;
END;

5) BEGIN
    RD  = /RW * DS          ;
    A   = /BW * /CK         ;
    WAIT = BW               ;
END;

6) BEGIN
    RD  = /RW * DS          ;
    A   = BW                 ;
END;

7) BEGIN
    RD  = /RW * DS          ;
    A   = /RD                ;
END;

END;

"=====
"This section defines the wiggles when the 8088 is Bus Master"
"=====

BEGIN

    LAO = AO * ALE * SEL +
          LAO * /ALE * SEL      ;
    B   = LAO * WR * SEL        ;
    A   = /LAO * WR * SEL       ;
    DS  = A +
          /LAO * RD * SEL       ;
    C   = /LAO * RD * SEL       ;
    D   = LAO * RD * SEL        ;
END;

END.

```

Figure 3-125. PLPL Specification for 8088 to Am9516 Interface (Continued)

Am7970A CEP Interface to the 80188 CPU

This application note shows how to use the Am7970A CEP in a low-cost environment. The 8-bit data interface of the CEP to the 8-Bit 80188 microprocessor is simple. It reduces the number of additional drivers, latches and control logic to an absolute minimum. The 80188 also provides an interrupt controller and a chip select decoder. No additional parts are necessary to access and control the Am7970A CEP.

General Discussion

This example assumes a single board approach with an onboard memory bank which may vary from 64 kbytes to 1 Mbyte. By adding drivers to the control signals and to the address lines A8-A15 it could easily be expanded into a bus controlled system. One Mbyte of memory is sufficient for storing the image data of one page with a resolution of 300 pixels per inch. This allows the CEP to compress or expand such a picture without interruption and with a minimum of software overhead. A smaller memory bank might be chosen for cost reasons.

This design does not use the faster document buffer interface of the CEP. Both the image data and the compressed data are passed through the system interface. Therefore, bus arbitration is incurred for every single transferred byte limiting the use of this example to low throughput applications.

It is definitely sufficient for FAX applications and may even serve very well for image storage applications where speed is not the most important factor. In these cases a memory bank of 64 to 128 kbytes is sufficient because the CEP is capable of processing fractions of a whole page without producing inconsistencies in the coded image data.

An additional memory bank connected to the document buffer interface will improve the overall throughput of the compression or expansion by a factor of four. This buffer should be used only to store the image data because it requires a data rate approximately 10 times higher than the compressed data.

Hardware Description

Figure 3-126 is a diagram of the Am7970A CEP to 80188 CPU interface. It shows how the address and data lines of the CPU and the CEP are transformed into a common demultiplexed memory bus. Additional peripheral devices could either be located on this memory bus or be connected directly to the CPU interface.

The Am7970A CEP multiplexes the data on the address lines A16-A23 while the CPU multiplexes data on the address lines A0-A7. These are the only differences between the two interfaces. All control signals can be used without conversion.

The 80188 CPU provides a 50% duty cycle clock output which can be used to drive the CEP. If the CPU runs on a higher clock rate, the CEP has to be driven by either an additional clock generator or a CPU clock that has been divided down. This may be very useful because the CEP needs only three clock cycles for a memory access while the 80188 needs four cycles. Running the CEP with a slower clockrate than the CPU does not necessarily result in slower memory access.

The READY signal coming from the memory interface must meet the setup time of the CEP READY input. For Revision A of the Am7970, a wait state must also be inserted for each memory access from the CEP. Figure 3-127 shows the logic to be added to the design above.

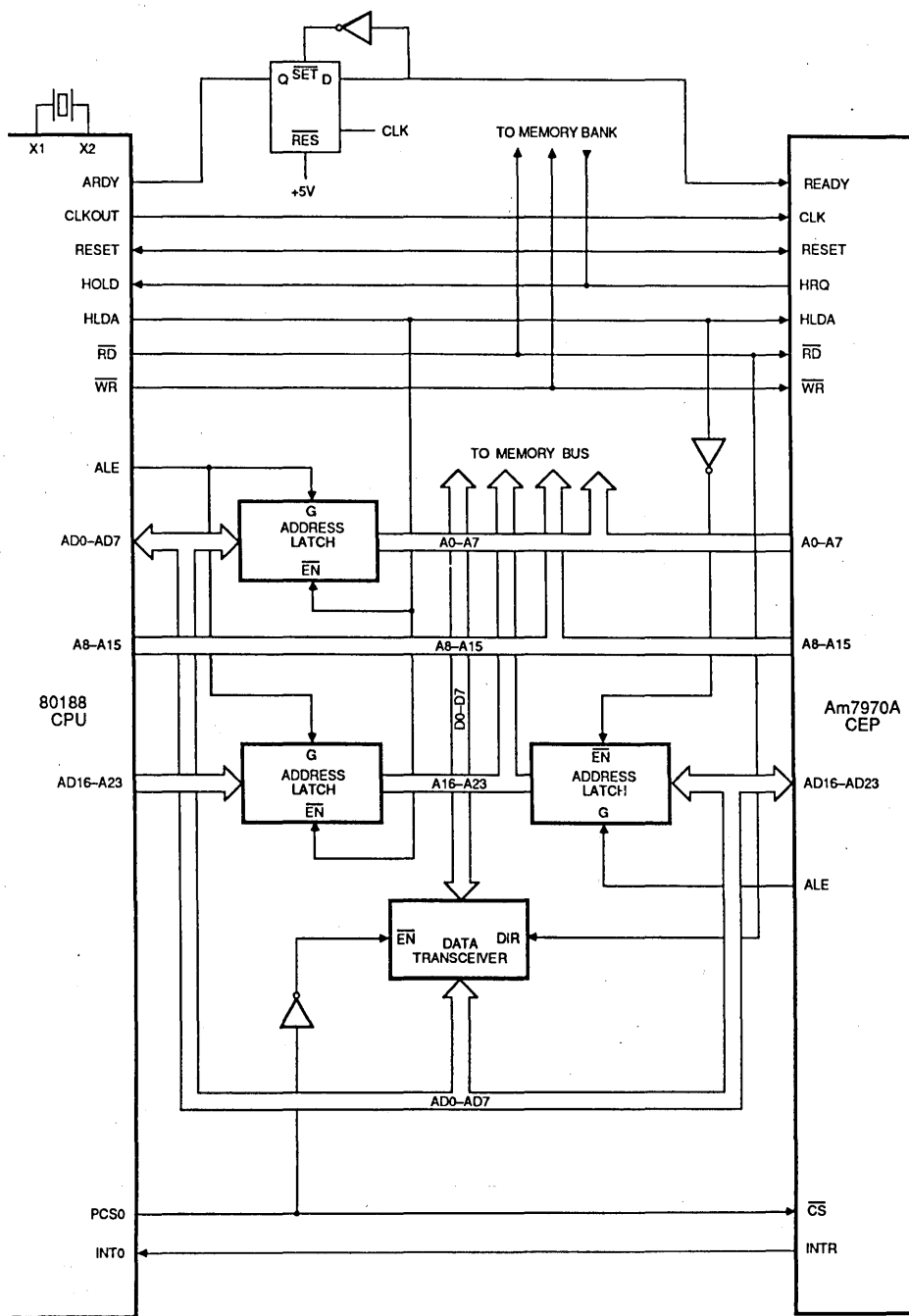
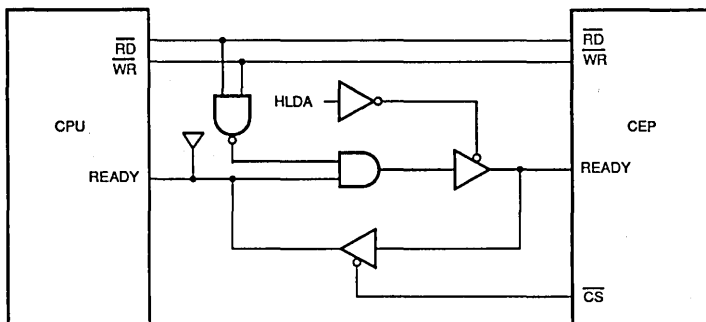


Figure 3-126. Am7970A CEP to 80188 CPU Interface

07666A 5-4



07666A 5-5

Figure 3-127. Wait State Circuit

Operation

80188 CPU Access to the Am7970A CEP

The CEP has many registers which provide programmability of many different options such as paper size, memory address control, and status information. Access to these registers is started by driving the $\overline{CS}$ signal LOW. This input is driven by the Peripheral Chip Select Output of the 80188. This signal also disables the data transceiver to the memory bus. The different registers are addressed by the address lines A0-A7 which are directly connected to the CEP. The CEP drives READY LOW as long as it needs to move the data to or from the appropriate register. This operation is called "slave access".

Am7970A CEP Access to the Memory

As soon as the CEP is started, it activates its internal DMA device to gain control of the memory buffer. It signals this to the CPU by driving HRQ HIGH. It then waits until HLDA is driven HIGH by the CPU to acknowledge that the bus is released. The CEP then starts a memory access. HLDA also enables the address latch for the CEP and disable the ones for the CPU. The CEP releases the bus after each byte transfer. This operation is called "master access".

The data transceiver is activated all the time except during a slave access. It normally drives data onto the memory bus. Only when $\overline{RD}$ goes LOW and $\overline{CS}$ is not active, the direction of the data transceiver is switched to the opposite direction.

Am7979A CEP Evaluation Board

Features

- Interface for IBM PC/XT or IBM AT on the same board.
- Automatic recognition of PC or AT environment.
- Full master mode capability in IBM AT using AT memory for system bus access.
- 1 MByte dynamic memory on board.
- Dualport arbitration allows memory access from CEP system and document side.
- PC/XT has full access into the on board memory addressing it as 16 64 kbyte blocks in page mode.
- All operating modes of CEP can be evaluated with maximum performance.
- CEP hardware reset initiated by I/O address access.
- Clock rate supplied by plug in exchangeable clock generators or from an external input.
- All I/O addresses are memory mapped.
- Jumper selectable DRQ/DACK and IRQ lines.
- Performs master access to AT memory

The CEP Evaluation Board In An IBM PC/XT

The IBM PC/XT does not allow another DMA master beside its own on-board DMA device working on the extension bus. Any attempt do so without changing the logic on the mother board will cause serious bus contention. That requires a separate memory bank dedicated to the CEP and accessible by the IBM PC/XT.

It is also quite useful to have enough memory dedicated to the document bus to hold a whole page of image data with a resolution of 300 Pixel/inch in memory. The evaluation board solves this problem by giving both sides of the CEP full access to a 1 Mbyte dynamic memory bank. That gives the user the freedom to assign as much memory to any side of the CEP as necessary.

The on-board "system bus" is shared by the CEP system interface and the CPU. So that makes it a three port memory design.

Normally the "system bus" is dedicated to the CEP system interface. If the CPU wants to access the evaluation board through this bus from the extension bus it drives "SBUSRQ" LOW by accessing an I/O address. The CPU is then kept waiting by the logic with "IOCHRDY" until the CEP releases "HRQ". Then "HLDA" is driven LOW by the interface logic to prevent the CEP from reaccessing the bus.

As long as "SBUSRQ" is LOW, the CPU has free access to the system bus. The document bus side is kept in Wait state while "SBUSRQ" is active.

"SBUSRQ" is latched and must be reset by another I/O access to a different address after completion of the read and write cycles onto the evaluation board. The CPU accesses the CEP registers by driving "CEPRQ", it accesses the memory bank by driving "MEMRQ" and accesses the page latch by driving "PAGE" LOW through different I/O addresses.

Since the IBM PC/XT I/O address layout does not support enough consecutive I/O addresses, all CEP I/O addresses are memory mapped in this design.

While the CEP is compressing or expanding a document, the CPU either polls the status register of the CEP or waits for an interrupt caused by the completion or an exception of the process.

The CEP Evaluation Board In An IBM AT

The IBM PC/XT extension bus connector is fully compatible with the new IBM AT connector. The AT introduces an additional connector to provide the extra signals needed for the increased memory size and the 16-bit data format. The AT also offers a fully compatible PC/XT mode. Thus all functions of the evaluation board designed for the IBM PC/XT will also work on the AT without any change in software and hardware.

In addition the evaluation board wants to make use of the master mode capability offered by the AT extension bus. To do so it uses the signals of the added connector to perform a proper bus arbitration on the extension bus. A ground pin on that connector will tell the board that it is connected to the AT.

For slave mode, everything said in the previous chapter will work the same except that there is now no need for driving the "SBUSRQ" signal before accessing the board because the system bus is automatically released by the CEP due to the bus arbitration. The access of the AT onto the evaluation board is still in 8-bit PC/XT compatible mode.

In master mode the system interface of the CEP will no longer access the on-board memory but will place its address and data signals onto the AT extension bus giving free access to all the memory that is provided by the IBM AT. The bytes coming out of and going into the CEP have to be divided into the upper and lower data bus of the AT according to the address being even or uneven.

Figure 3-128 is a system map of the Evaluation Board. Figure 3-129 is a block diagram. The PAL device equations are shown in Figure 3-130.

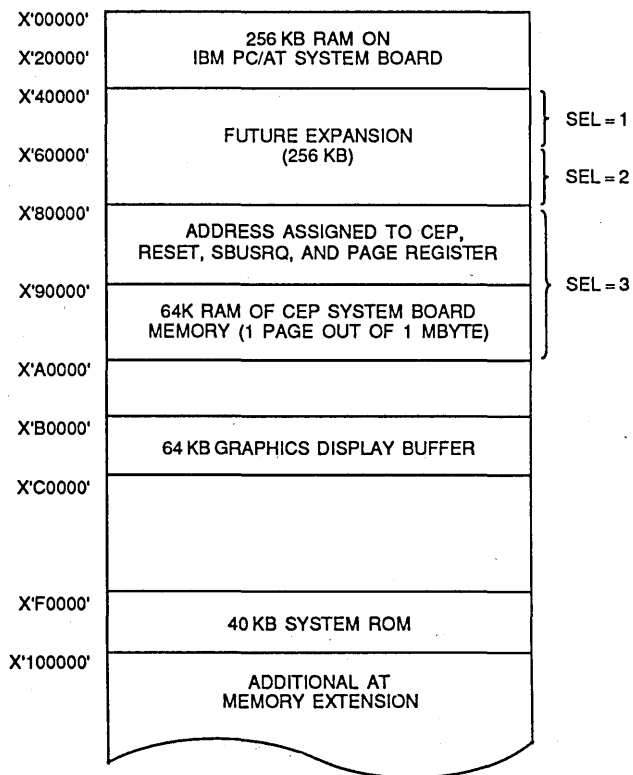


Figure 3-128. Evaluation Board System Memory Map

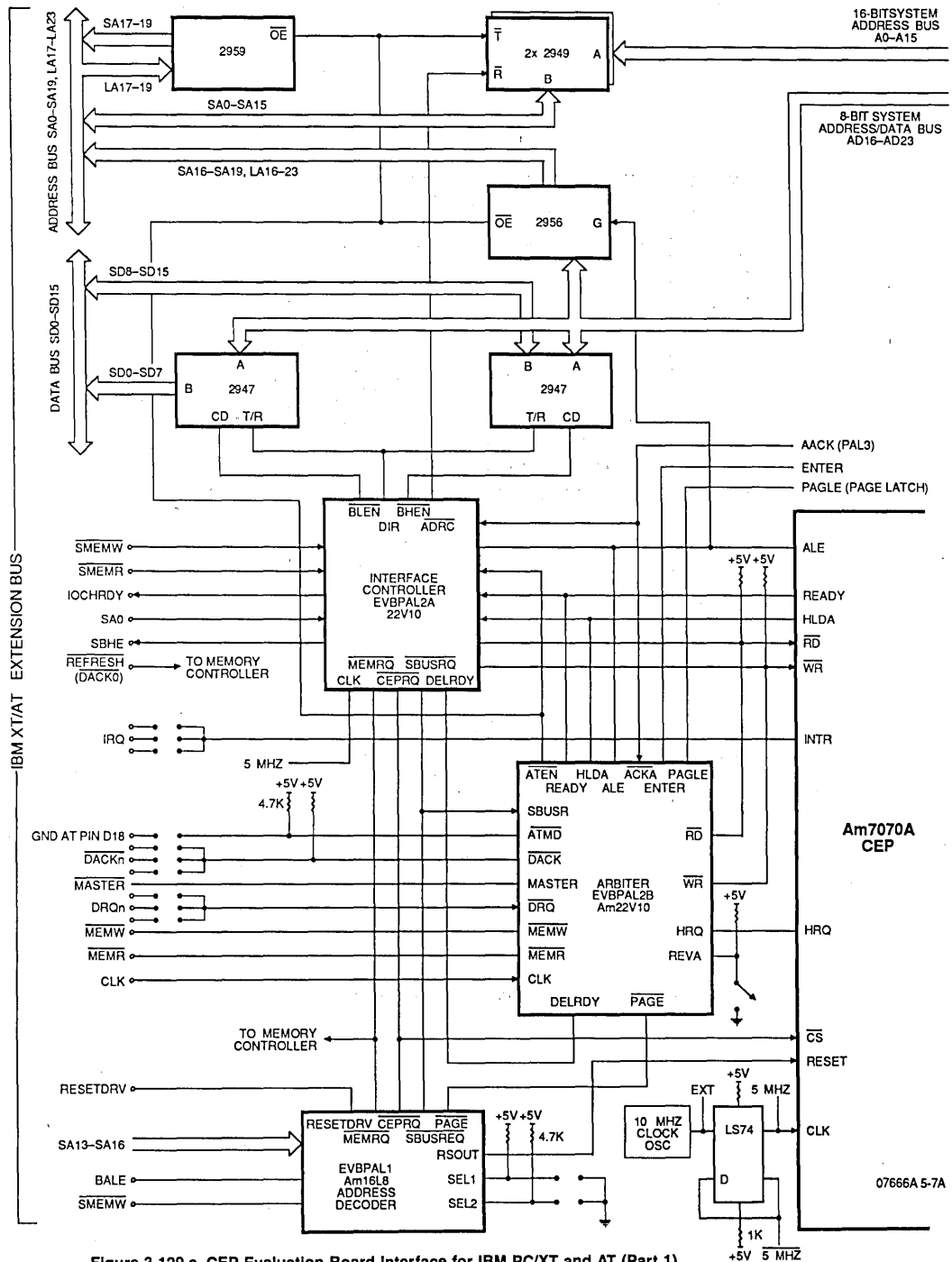


Figure 3-129.a CEP Evaluation Board Interface for IBM PC/XT and AT (Part 1)

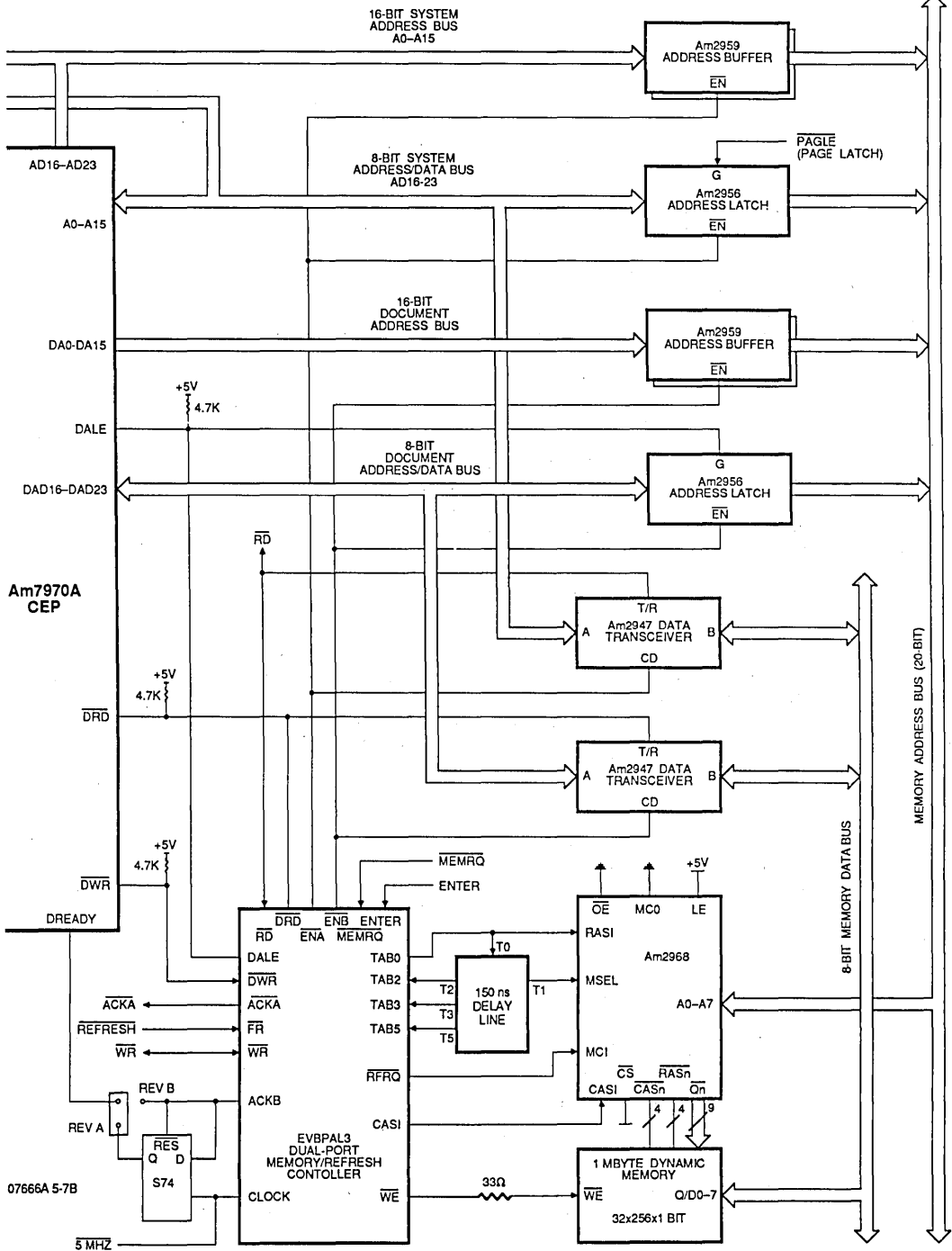


Figure 3-129.b CEP Evaluation Board Interface for IBM PC/XT and AT (Part 2)

```

PARTNO      NONE;
NAME        EVBPAL1;
DATE        11-6-85;
REV         1.1;
DESIGNER    Wolfgang Kemmler;
COMPANY     Advanced Micro Devices;
ASSEMBLY    CEP Evaluation Board;
LOCATION      U61;

/* Pal Assembler: CUPL (Assisted Technology)*/
/*****
/*                                     */
/*      Address Decoder for          */
/*      Am7970 CEP Evaluation Board  */
/*                                     */
*****/

/* Target Device Type : PAL16L8      */

/* Outputs :                          */

PIN 19 = !PAGE; /* Latch enable foe Page Latch */
PIN 17 = !CEPRQ; /* CEP Chip Select */
PIN 16 = !SBUSRQ; /* On Board System Bus Request */
PIN 15 = !ADDR; /* Interm. Signal for Addr. Decoding */
PIN 14 = RSOUT; /* RESET */
PIN 12 = !MEMRQ; /* On Board Memory Request */

/* Inputs :                          */
PIN [1..7] = [A13..19]; /* Addresses from Extension Bus */
PIN [8..9] = [S0..1]; /* Address range select inputs */
PIN 10 = GND;
PIN 11 = !SMEMW; /* Write Signal from Ext. Bus */
PIN 13 = RSIN ; /* RESET Input from Ext. Bus */
PIN 18 = BALE; /* Address latch enable " " */
PIN 20 = VCC;

/* Declarations and Intermediate Variable Definitions : */
FIELD UADDR = [A17..19]; /* Upper Address Range */
FIELD LADDR = [A13..15]; /* Lower Address Range */
FIELD SEL = [S0,S1]; /* Address range select field */

/* Address range definitions: */

/* Upper Address range : */
USEL3 = UADDR:[80000..9ffff];
USEL2 = UADDR:[60000..7ffff];
USEL1 = UADDR:[40000..6ffff];

/* Lower Address range: */
ADR0 = !A16&ADDR&LADDR:[0000..1fff]; /*CEP register access */
ADR1 = !A16&ADDR&LADDR:[2000..3fff]; /*ask for S-Bus */
ADR2 = !A16&ADDR&LADDR:[4000..5fff]; /*release S-Bus */
ADR3 = !A16&ADDR&LADDR:[6000..7fff]; /*Pageaddr. into Pageregister */
ADR4 = !A16&ADDR&LADDR:[8000..9fff]; /*CEP hardware RESET aktiv */
ADR5 = !A16&ADDR&LADDR:[a000..bfff]; /*CEP hardware RESET inaktiv */

```

Figure 3-130. Evaluation Board PAL Device Equations

```

/*LOGIC EQUATIONS : */
ADDR = SEL:3 & USEL3 & !BALE /*SEL selects 3 allowed */
      # SEL:2 & USEL2 & !BALE /* "addressranges (blocks of*/
      # SEL:1 & USEL1 & !BALE ; /* "128kBytes )*/

MEMRQ = A16 & ADDR & !BALE ; /*Access on board memory access*/

SBUSRQ = ADR1 & SMEMW /*Request the S-Bus */
      # SBUSRQ & !(ADR2 & SMEMW) ;

PAGE = ADR3 & SMEMW; /*Load Page address into */
      /*Pageregister*/

RSOUT = ADR4 & SMEMW & RSIN /*Software controlled */
      # RSOUT & !(ADR5 & SMEMW) ; /* Hard Reset */

CEPRQ = ADR0 & !RSOUT ; /*CEP Register Access */
      /* (Memory mapped )*/

PARTNO NONE;
NAME EVBPAL2A;
DATE 11-6-85;
REV 1.2;
DESIGNER Wolfgang Kemmler;
COMPANY Advanced Micro Devices;
ASSEMBLY CEP Evaluation Board;
LOCATION U31;

/* Pal Assembler: CUPL (Assisted Technology) */
/*****
/*
/* IBM XT/AT Extension Bus Interface Controller */
/* for Am7970 CEP Evaluation Board */
/*
/*
/*****
/* Target Device Type : AmPal 22V10 */
/*
/* Outputs: */
PIN 23 = IOCHRDY; /* Open Coll. Extension Bus Signal */
PIN 22 = !RD; /* CEP Signal */
PIN 21 = !WR; /* " */
PIN 20 = !BLEN; /* Data transceiver low enable */
PIN 19 = !BHEN; /* Data transceiver high enable */
PIN 18 = !ADDRC; /* Address receiver enable */
PIN 17 = DIR; /* Direction controll of data transc. */
PIN 16 = !SBHE; /* Extension Bus Signal */
PIN 15 = !DELRDY; /* Signal helps generating two wait states */
/* for Rev.A CEP - IBM AT interface combination */
PIN 14 = !IONRDY; /* Interm. I/O Not Ready Signal */

/* Input: */
PIN 1 = CLK; /* CEP clock (5 MHz) */
PIN 2 = !SBUSRQ; /* On board system bus access request */
PIN 3 = !CEPRQ; /* CEP register access request */
PIN 4 = !MEMRQ; /* On board memory access request */
PIN 5 = !ATEN; /* AT Mode enable */
PIN 6 = !HLDA; /* CEP signal */
PIN 7 = READY; /* CEP signal */
PIN 8 = ACKA; /* Acknowledge from on board memory */
PIN 9 = !SMEMW; /* Extension bus signal */
PIN 10 = !SMEMR; /* " */
PIN 11 = SA0; /* " */
PIN 12 = GND;
PIN 13 = !PAGE; /* Page latch access request */
PIN 24 = VCC;

```

Figure 3-130. Evaluation Board PAL Device Equations (Continued)

```

/* Logic Equations : */

WR.OE = !HLDA ; WR = SMEMW & (CEPRQ # MEMRQ) ;

RD.OE = !HLDA ; RD = SMEMR & (CEPRQ # MEMRQ) ;

BHEN = ATEN & SAO & (RD # WR) ;

BLEN = ATEN & !SAO & (RD # WR)
      # CEPRQ & !HLDA & (SMEMR # SMEMW)
      # MEMRQ & !HLDA & (SMEMR # SMEMW)
      # PAGE & !HLDA ;

IOCHRDY.OE = IONRDY ; IOCHRDY = !IONRDY ;

IONRDY = HLDA & SBUSRQ
      # !HLDA & (SMEMR # SMEMW)
      & (MEMRQ & !ACKA # CEPRQ & !READY) ;

ADDRC = !HLDA & (CEPRQ # MEMRQ) ;

DIR = HLDA & WR # !HLDA & SMEMR ;

SBHE.OE = ATEN ; SBHE = !SAO ;

DELRDY.D = RD # WR ;
DELRDY.AR = 'b'0 ; DELRDY.SP = 'b'0. ;


PARTNO      NONE;
NAME        EVBPAL2B;
DATE        1-13-85;
REV         1.3;
DESIGNER     Wolfgang Kemmler;
COMPANY      Advanced Micro Devices;
ASSEMBLY     CEP Evaluation Board;
LOCATION      U30;
/*Pal Assembler: CUPL (Assisted Technology) */
/*****
/*
/*      IBM XT/AT Extension Bus Arbiter      */
/*      for the CEP Evaluation Board          */
/*
/*
/*****

/* Target Device Type : AmPal22V10 */

/* Inputs : */
PIN 1 = CLK; /* Extension bus clock */
PIN 2 = !RD; /* CEP signal */
PIN 3 = !WR; /* " */
PIN 4 = HRQ; /* " */
PIN 5 = ACKA; /* On board memory acknowledge */
PIN 6 = !SBUSRQ; /* Onboard system bus request */
PIN 7 = !DACK; /* Extension bus arbitration signal */
PIN 8 = !PAGE; /* Page Latch access request */
PIN 9 = !ATMD; /* AT Mode Indicator */
PIN 10 = REVA; /* CEP Revision A Indicator (adds 1 wait state)*/
PIN 11 = ALE; /* CEP signal */
PIN 13 = DELRDY; /* Delay Ready */
/* Outputs : */
PIN 23 = !MASTER; /* Extension bus arbitration signal */

```

Figure 3-130. Evaluation Board PAL Device Equations (Continued)

```

PIN 22 = DREQ;      /*          "          */
PIN 21 = !MEMW;     /* Extension bus signal */
PIN 20 = !MEMR;     /*          "          */
PIN 19 = READY;     /* CEP signal */
PIN 18 = HLDA;      /*          "          */
PIN 17 = !ATEN;     /* Enable master interface to AT ext. bus */
PIN 16 = ENTER;     /* Onboard memory access request */
PIN 15 = EARLY;     /* Intermediate signal for "ENTER" */
PIN 14 = PAGLE;     /* PAGE LATCH ENABLE signal */
/* Logic Equations : */

ATEN.D = MASTER & ATMD;
ATEN.AR = 'b'0 ; ATEN.SP = 'b'0 ;

MASTER.OE = HRQ & HLDA & ATMD;
MASTER.D = HRQ & HLDA & ATMD;
MASTER.AR = 'b'0 ; MASTER.SP = 'b'0 ;

DREQ.D = HRQ & ATMD & !DACK
        # DREQ & HRQ & ATMD ;

DREQ.AR = 'b'0 ; DREQ.SP = 'b'0 ;

MEMW.OE = ATEN ; MEMW = WR ;

MEMR.OE = ATEN ; MEMR = RD ;

HLDA = !ATMD & HRQ & !SBUSRQ
        # !ATMD & HLDA & HRQ
        # ATMD & DACK & DREQ
        # ATMD & HLDA & DREQ ;

EARLY = ALE # EARLY & !RD & !WR ;

ENTER = (EARLY & !ALE # RD # WR) & !ATMD & HLDA ;

PAGLE = ALE & HLDA # PAGE & !HLDA ;

READY.OE = HLDA ;
READY = HLDA & ATMD & (RD # WR) & (DELRDY & REVA # !REVA)
        # ACKA & !ATMD & (REVA & (RD # WR) # !REVA) ;

PARTNO      NONE;
NAME        EVBPAL3;
DATE        11-13-85;
REV         1.4;
DESIGNER    Wolfgang Kemmler;
COMPANY     Advanced Micro Devices;
ASSEMBLY    Am7970 CEP Evaluation Board;
LOCATION      U64;
/* Pal Assembler: CUPL (Assisted Technology) */
/*****
/*
/* Dual Port Dynamic Memory Access/Refresh Controller */
/*   for Am7970 CEP Evaluation Board */
/*
/*
*****/
/* Target Device Type : PAL22V10 */

/* Inputs : */
PIN 1 = clock ; /* 5MHz clock synchr. & inverted */

```

Figure 3-130. Evaluation Board PAL Device Equations (Continued)


```

/* to CEP clock */
PIN 2 = tab3 ; /* 60% Tab of 150ns delay line */
PIN 3 = tab5 ; /* 100% " */
PIN 4 = dale ; /* CEP signal document side */
PIN 5 = !drd ; /* " " */
PIN 6 = !dwr ; /* " " */
PIN 7 = tab2 ; /* 40% Tab of delay line */
PIN 8 = !wr ; /* CEP signal system side */
PIN 9 = !rd ; /* " " */
PIN 10 = !memrq ; /* On board memory request */
PIN 11 = fr ; /* REFRESH from IBM AT; DACK0 from IBM XT */
PIN 12 = GND ;
PIN 13 = enter ; /* CEP system interface access request */

/* Outputs : */
PIN 23 = tab0 ; /* Input of delay line */

PIN 22 = !rfrq ; /* Interm. signal for refresh arb. */
PIN 21 = !we ; /* Write enable for on board memory */
PIN 20 = endcyc ; /* Intermediate signal for document
/* side arbitration */
PIN 19 = fh ; /* Interm. signal for refresh arb. */
PIN 18 = !enb ; /* Enable document side to memory */
PIN 17 = ackb ; /* Acknowledge document side access */
PIN 16 = acka ; /* " system " " */
PIN 15 = !ena ; /* Enable system side to memory */
PIN 14 = casi ; /* CASI for 2968 */

/* Logic Equations : */
ena = enter & !rfrq & !enb & !(fh & fr) & !tab3
    # (rd & wr) & !rfrq & !enb & memrq & !tab3
    # ena & (enter & rd & wr) & !rfrq ;

enb = (!dale & !endcyc & drd & dwr) & !(fh & fr)
    & !enter & !rfrq & !ena & !tab3
    # enb & (!dale & !endcyc & drd & dwr) & !rfrq & !ena ;

endcyc = drd & dwr & endcyc & !dale ;

tab0 = ena & enb & rfrq & (!tab3 & !tab5) ;

acka.d = ena & !rfrq & !enb ;

ackb.d = enb & !rfrq & !ena ;

fh = fr & rfrq & tab0 & fh & fr ;

rfrq = fr & !fh & !ena & !enb & !tab3
    # rfrq & (tab0 & tab2) ;

we = ena & !enb & !rfrq & wr & enb & !ena & !rfrq & dwr ;

casi = tab2 & (we & ena & rd & enb & drd) ;

acka.ar = 'b'0 ;
acka.sp = 'b'0 ;
ackb.ar = 'b'0 ;
ackb.sp = 'b'0 ;

```

Figure 3-130. Evaluation Board PAL Device Equations (Continued)

3.4.5 INTERFACING TO THE Z80 AND Z8000

Z80 and Am9568 Data Ciphering Processor with the Am9517 DMA Controller

This application design shows how to operate the ciphering throughput up to 890 Kbyte/s by using the advanced 8-bit DMA Controller Am9517A-5 (also called the 8237-5). The host CPU is a Z80A. (Figure 3-131)

The CPU sets up a data block in memory and programs the DMA controller to transfer this data block to the DCP via the Master Port. The DCP encrypts the data. A high-speed peripheral device can read out the ciphered data from the Slave Port. This dual-port configuration allows data input and output simultaneously and increases the throughput, compared to a single-port configuration, by a factor of two. In the single-port configuration, only the Master Port is used for data transfer; it handles both the clear and ciphered data.

The multiplexed address/data bus of the DCP is simulated in a two-cycle operation. For output operation to an even address, the PAL interface timing controller generates a Master Port Address Strobe (MAS) to select one of the internal registers. Subsequent I/O operations to an odd address (A_0 =HIGH) transfer data to or from the preselected DCP register. During I/O operations to an odd address, the PAL device generates Master Port Data Strobes ($\overline{MRD}$ or $\overline{MWR}$). Before the DMA block transfer starts, the CPU must preselect the DCP data register. The register address of the data register is $00H$.

The DMA controller operates in "flyby" mode. Data is transferred on the system data bus one byte at a time from memory to the DCP, or vice versa, without going through a DMA register. An I/O Read ($\overline{IOR}$) and Memory Write ($\overline{MEMW}$) or I/O Write ($\overline{IOW}$) and Memory Read ($\overline{MEMR}$) are active at the same time. The DCP is selected by DMA Acknowledge ($\overline{DACK}$). The PAL device treats $\overline{DACK}$ as $\overline{CS}$ active and A_0 =HIGH. In this design the DMA controller can only execute data transfer cycles; it is not able to change the internal register address of the DCP.

The DMA controller is set up for Demand Transfer Mode. It releases the bus when the data request input goes inactive. The Master Port Flag (MFLG) is wired to the data request input. The flag output goes active when the DCP is ready to accept data, or the output data is ready to be read out. After transferring one block of data (8 bytes), this flag goes inactive until a new block can be put in or read out. The inactive time depends on the response time of the peripheral logic at the Slave Port. This flag is inactive for a minimum of five clocks.

Speed

The DMA controller needs three clock cycles to transfer one byte. After each block transfer (8 bytes), the DMA controller releases the bus and requests it back if MFLG goes active again. This time is assumed to be 12 clocks. The ciphering of one block is done concurrently with the input of the next block; the internal operation is pipelined. The maximum throughput can be calculated as:

$$T = 8 / (8 \cdot 3 + 12) \cdot 4 \text{ MHz} = 0.89 \text{ Mbyte/s}$$

The Compressed Transfer mode of the DMA controller cannot be used, because the PAL synchronization logic needs normal timing to synchronize the Data Strokes to the DCP clock.

Initialization

The Multiplexed Control Mode ($C/\overline{K} = \text{LOW}$) of the DCP is selected to enable access to the internal registers. The CPU first programs the Mode Register to reset the DCP and to set up the port configuration and ciphering mode. After that, the keys and initial vectors can be loaded. To initialize the DCP for DMA transfer, the CPU executes one Address Latch Cycle, to pre-select the data register.

The DMA controller must be programmed such that $\overline{\text{DREQ}}$ and $\overline{\text{DACK}}$ are active LOW.

Timing

The PAL device simulates the multiplexed address/data bus of the DCP assuming a two-cycle operation mode. In the first cycle the CPU latches the address of the internal register into the DCP; subsequent cycles transfer data to or from the selected register. Address A_0 distinguishes the two cycles (Figure 3-132). An I/O instruction with $A_0 = \text{LOW}$ generates an address latch cycle; an I/O instruction with $A_0 = \text{HIGH}$ generates a data transfer cycle.

The DMA controller must be initialized for "extended" I/O write in order to have a similar I/O bus timing to the Z80A CPU. A "late" I/O write delays the Master Port Write Strobe ($\overline{\text{MWR}}$) to the DCP by one clock cycle. If a late write is used, the data bus will not be valid at the time data is latched.

To execute a DCP-to-memory transfer, the DMA does an I/O read and memory write. The DMA controller can be programmed for an "extended" or "late" write, depending on the memory design.

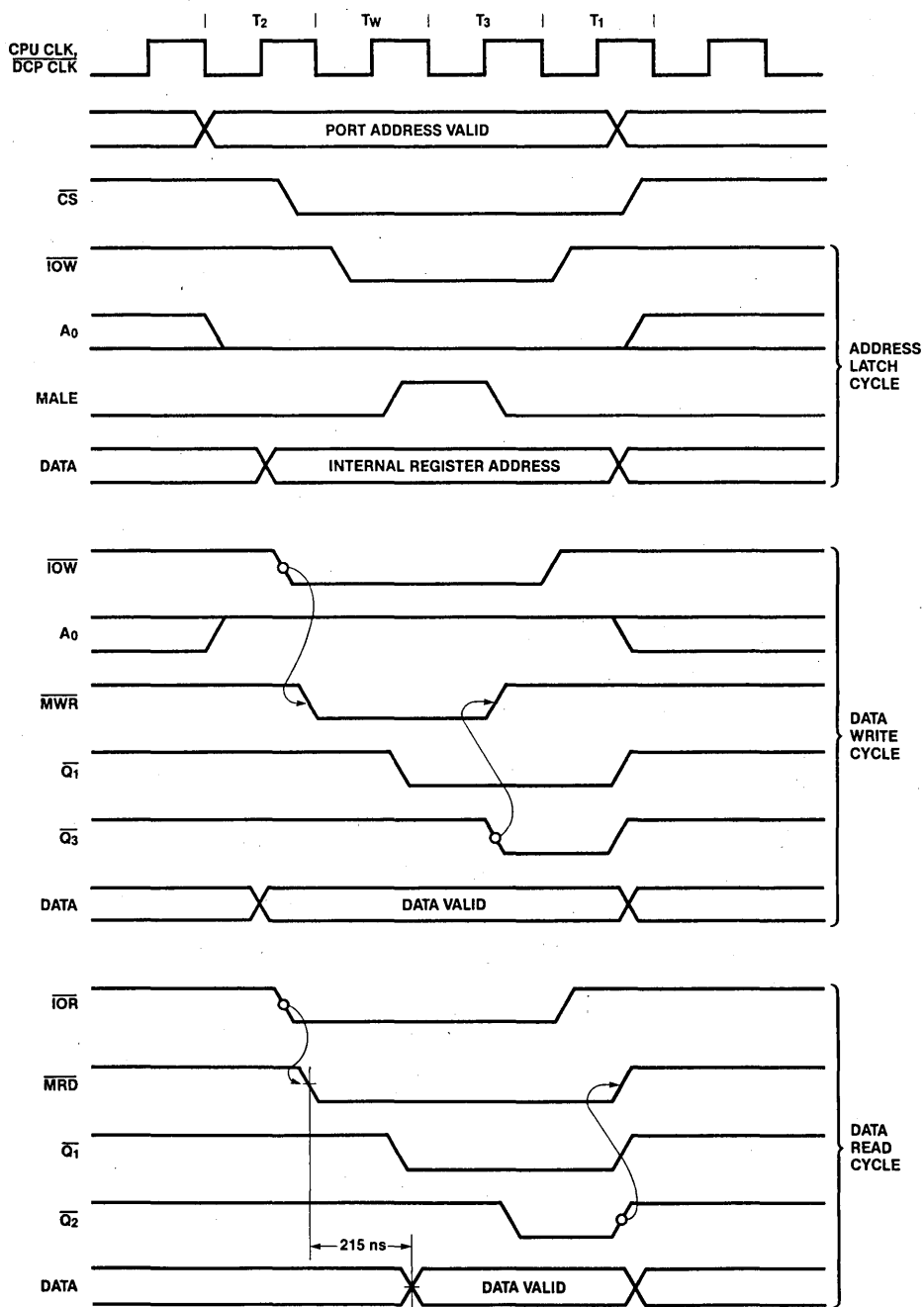
In "flyby" mode the DMA controller generates no I/O address, so the CPU has to preselect the data Input or Output Register. A DMA Acknowledge ($\overline{\text{DACK}}$) enables MRD or MWR to control the data transfer.

Figure 3-132 shows the DMA-DCP data transfer timing. When the DMA Controller has transferred one block of data, the data transfer has to be stopped until the DCP is ready for the next block transfer. The DCP makes the DMA Controller stop the transfer by deactivating MFLG. If MFLG is LOW, data may be transferred; if MFLG is HIGH, the DCP does not accept data transferred. The timing of the MFLG to DREQ path is the most critical in this application. If MFLG is deactivated too late, the DMA Controller will issue another data transfer which will be disregarded by the DCP. The critical signal path will be analyzed below.

To prevent the DMA from issuing another cycle the Data Request input has to go inactive by the falling edge of the DMA clock at the end of cycle S3. The DMA controller samples the input at this time and instigates another cycle if the request is still active. The setup time of DREQ is 0 ns. The Master Port Flag which is connected to the DREQ input goes inactive in the eighth cycle with a maximum delay time of 150 ns after the Data Strokes. The Data Strobe itself has a maximum delay time of 190 ns (Am9517A-5) after the rising edge of the clock in cycle S2. That gives a time window of 375 ns of which 340 ns are already used for the two delays (190 ns + 150 ns). The propagation delay of a fast PAL device is 25 ns. This leaves 10 ns for other delays in the signal path.

The PAL design assumes that the system memory needs no Wait States.

The peripheral logic at the Slave Port can use the Slave Port Flag ($\overline{\text{SFLG}}$) to time the transfer. If $\overline{\text{SFLG}}$ is active LOW, data can be written to or read from the data register. Figure 3-133 shows the PAL device equations for this interface.



02188A-96

Figure 3-132. CPU-DCP Timing Diagram

PAL DESIGN SPECIFICATION

JUERGEN STELBRINK 8-9-83

Z80A- AM9517(DMA)- AM9568(DCP) INTERFACE DEVICE

ADVANCED MICRO DEVICES

```

CLK1      CLK2      /CS      /IOR      /IOW      A0      /MFLG      /DACK      NC      GND
/OE      /MWR      /MRD      /Q1      /Q2      /Q3      MALE      NC      CLK      VCC

/MALE := /IOW+IOR+/CS+A0+MALE ; MASTER PORT ADDRESS STROBE

Q1      := CS*AO*IOR*/IOW*/Q2      +
           CS*AO*IOW*/IOR*/Q3      +
           DACK*IOR*/IOW*/Q2      +
           DACK*IOW*/IOR*/Q3

Q2      := CS*AO*IOR*/IOW*Q1      +
           CS*AO*IOR*/IOW*Q2      +
           DACK*IOR*/IOW*Q1      +
           DACK*IOW*/IOW*Q2

Q3      := CS*AO*IOW*/IOR*Q1      +
           CS*AO*IOW*/IOR*Q2      +
           DACK*IOW*/IOR*Q1      +
           DACK*IOW*/IOR*Q2

MRD      = CS*AO*IOR*/IOW      + ; MASTER PORT READ
           DACK*IOR*/IOW      +
           Q2

MWR      = CS*AO*IOW*/IOR*/Q3      + ; MASTER PORT WRITE
           DACK*IOW*/IOR*/Q3

/CLK      = CLK2 ; DCP CLOCK

```

FUNCTION TABLE

```

CLK1 CLK2 /CS /IOR /IOW /DACK A0   CLK MALE /MRD /MWR /Q1 /Q2 /Q3
;
; C C / / D M / /
; L L / I I A C M / /
; K K C O O C C A C L L R W Q Q /
; 1 2 S R W K 0 K E D R 1 2 3 COMMENT
-----
; CLOCK GENERATION
X L X X X X X H X X X X X X
X H X X X X X L X X X X X X
; ADDRESS LATCH
C X H H H H L X L H H H H H ; CYCLE T2 (CPU)
C X L H L H L X H H H H H H ; CYCLE TW
C X L H L H L X L H H H H H ; CYCLE T3
C X H H H H L X L H H H H H ; CYCLE T1
; READ DATA
X X H H H H H X L H H H H H ; CYCLE TW (CPU)
X X L L H H H X L L H H H H
C X L L H H H X L L H L H H
C X L L H H H X L L H L L H ; CYCLE TW (EXTRA WAIT STATE)
C X H H H H X L H H H H ; CYCLE T3
C X H H H H X L H H H H ; CYCLE T1
X X H L H L X X L L H H H H ; CYCLE S3 (DMA)
C X H L H L X X L L H L H H
C X H L H L X X L L H L H H ; CYCLE S4
C X H H H H X X L H H H H H ; CYCLE S2

```

Figure 3-133. Source Listing for the Z80 to Am9568 DCP Interface

```

; WRITE DATA
X X L H L H H X L H L H H H ; CYCLE TW (CPU)
C X L H L H H X L H L L H H
C X L H L H H X L H H L H L ; CYCLE T3
C X H H H H H X L H H H H H ; CYCLE T1
X X H H L L H X L H L H H H ; CYCLE S3 (DMA)
C X H H L L H X L H L L H H
C X H H L L H X L H H L H L ; CYCLE S4
C X H H H H H X L H H H H H ; CYCLE S2
;
-----

```

DESCRIPTION:

THIS PAL GENERATES ALL NECESSARY BUS CONTROL SIGNALS, TO INTERFACE A Z80A CPU AND A AM9517 DMA CONTROLLER TO THE AM9568 DATA CIPHERING PROCESSOR. THE MAXIMUM SYSTEM CLOCK FOR ALL PARTS IS 4 MHZ.

1 INPUT AND 3 INPUT/ OUTPUT PINS ARE NOT USED.

INPUT SIGNALS:

CLK1, Z80 SYSTEM CLOCK
CLK2

/CS CHIP SELECT FOR THE DCP, GENERATED BY A DECODER LOGIC

/IOR INPUT/OUTPUT READ

/IOW INPUT/OUTPUT WRITE

A0 LEAST SIGNIFICANT BIT OF THE Z80 ADDRESS BUS TO SELECT THE TYPE OF OPERATION:

A0 = LOW SELECT DCP REGISTER FOR NEXT DATA CYCLES (ADDRESS LATCH)

A0 = HIGH READ OR WRITE INTERNAL REGISTER (DATA TRANSFER TO CONTROL, MODE, INPUT OR OUTPUT REGISTER)

/DACK DMA ACKNOWLEDGE FROM DMA CONTROLLER, TREATED AS /CS=LOW AND AO=HIGH

OUTPUT SIGNALS:

CLK INVERTED SYSTEM CLOCK FOR THE DCP

MALE MASTER PORT ADDRESS LATCH ENABLE, ACTIVE DURING ADDRESS LATCH CYCLES TO LATCH THE REGISTER ADDRESS ON MP1 AND MP2 (2 LINES OF THE MASTER PORT BUS) AND THE STATE OF /MCS IN. THE DCP STORES INTERNALLY THE ADDRESS AND CHIP SELECT TO THE NEXT ADDRESS LATCH CYCLE

/MRD MASTER PORT READ, TO ENABLE REGISTER READ OPERATIONS

/MWR MASTER PORT WRITE, TO ENABLE REGISTER WRITE OPERATIONS

/Q1, INTERNAL USED STATE SIGNALS (DO NOT CONNECT). Q1 IS ACTIVE 2
/Q2, CLOCK CYCLES IN EACH DATA TRANSFER OR DMA ACKNOWLEDGE CYCLE.
/Q3, IT IS USED TO GENERATE THE DELAYED Q2 AND Q3. Q2 IS USED TO HOLD /MRD ACTIVE UNTIL /IOR IS GONE INACTIVE. Q3 MASKS /MWR OFF.

Figure 3-133. Source Listing for the Z80 to Am9568 DCP Interface (Continued)

Z80 to Am9518/Am9568 DCP

This chapter shows, in two examples, how the Data Ciphering Processor (DCP) can be interfaced to a Z80 (Z80A, Z80B) CPU (Figure 3-134). All interface control signals are generated by one PAL device.

In CPU transfer mode, ciphering speed can reach up to 280 Kbyte/s. A Z80A DMA controller can double this value, and a Z80-DMA-DCP hookup can increase the speed to 1.1 Mbyte/s.

The multiplexed address/data bus of the DCP is simulated using a two-cycle operation mode. An output instruction to an even address (A_0 =LOW) selects one of the internal registers of the DCP. In all subsequent I/O operations with A_0 =HIGH, the CPU can transfer data to or from DCP registers. The register address stays latched in the chip until the next Address Strobe latches in a new address. The Address Latch Cycle does not represent significant overhead in an encryption or decryption session because, once the DCP is initialized and the data register is selected, no further Address Latch Cycle is needed.

I/O addresses: XXXX XXX0 - Address Latch Cycle
 XXXX XXX1 - Data Transfer Cycle
 X - User definable

The AmPAL16R4 device controls the interface timing. It generates the synchronized strobe signals for the DCP and the Wait for the CPU to extend the cycles.

The PAL device is programmed to allow two operation modes. In Mode A the DCP works with the same clock rate as the CPU. Mode B increases the ciphering speed by allowing higher than 4-MHz system clock rates for the CPU. In this mode, the PAL device provides half the system clock rate for the DCP.

A system with a Z80B at 6 MHz and an AmZ8068 at 3 MHz increases the ciphering speed compared to a system where both the CPU and the DCP clock are 4 MHz; the limiting factor is the data transfer capability of the CPU.

The key requirement in interfacing the DCP to a Z80 CPU is to meet the timing relationship between the Master Port Data Strobe ($\overline{MDS}$) and the DCP clock. The rising edge of $\overline{MDS}$ must be synchronous to the falling edge of the clock.

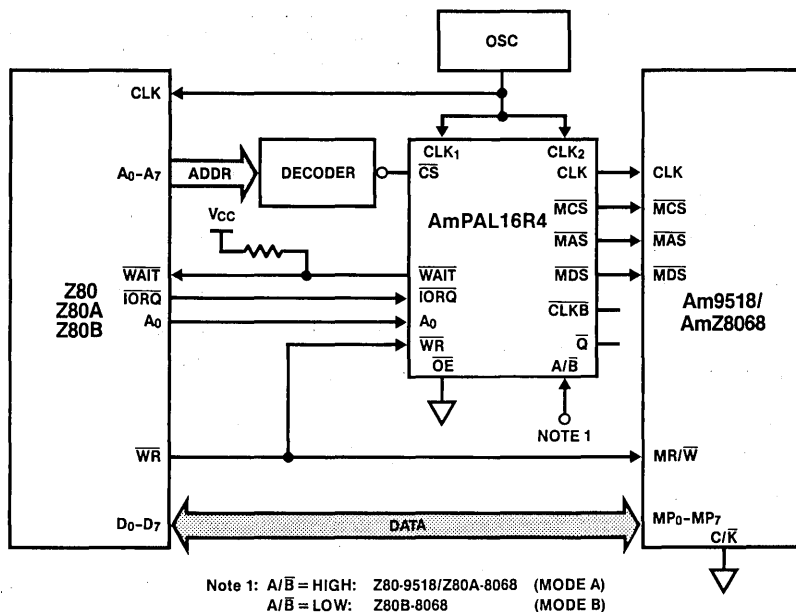


Figure 3-134. Z80-DCP Interface

02188A-97

The Operation Modes

Mode A: Both the Z80 CPU and the DCP are operating synchronously at the same frequency. The DCP clock is inverted. This mode can be used with system clocks up to 4 MHz. No extra Wait States are inserted.

Mode B: To get higher ciphering throughput, the data transfer speed of the Z80 bus should be increased by using a higher system clock rate. In Mode B the PAL device divides the system clock by two to generate the DCP clock. The DCP clock is synchronized to the $\overline{\text{MDS}}$ by delaying the clock one half cycle if they are not in phase (Figures 3-137 and 3-138). During a Data Write Cycle, one extra Wait State is inserted. An AmZ8068 must be used in this mode, even at a DCP clock rate of 3 MHz, because of its faster register access time.

Figure 3-134 shows the interface. The $\text{A}/\overline{\text{B}}$ input of the PAL device is wired HIGH to select Mode A or LOW to select Mode B.

The Interface Timing

Address Latch Cycle: (Figures 3-135 and 3-136)

Master Port Chip Select ($\overline{\text{MCS}}$) is active when $\overline{\text{IORQ}}$ and $\overline{\text{CS}}$ are active LOW and $\text{A}_0=\text{LOW}$ (even address). Master Port Address Strobe ($\overline{\text{MAS}}$) is strobed LOW for one system clock cycle during the Z80's automatically inserted Wait cycle T_{W} to meet the hold time requirement of $\overline{\text{MAS}}$ HIGH to $\overline{\text{MCS}}$ HIGH (parameter 35).

Data Read Cycle: (Figures 3-135 and 3-137)

A Data Read Cycle reads the register whose address was latched in the previous Address Latch Cycle. $\overline{\text{MCS}}$ and $\overline{\text{MAS}}$ are inactive the whole cycle. $\overline{\text{MDS}}$ is active during the last two clock cycles, T_{W} and T_3 . In both A and B Modes, no Wait State is inserted. $\overline{\text{WR}}$ and A_0 must be HIGH. In Mode B the DCP clock is set HIGH in the beginning of T_3 using an internal signal $\overline{\text{Q}}$ to synchronize the falling edge of the DCP clock to the rising edge of $\overline{\text{MDS}}$. $\overline{\text{Q}}$ is only active in Mode B during Wait State T_{W} . This interface meets the data hold time of the Z80, because the data is stable to the beginning of T_1 of the next machine cycle.

Data Write Cycle

In this cycle, the CPU can write one byte into the addressed register. $\overline{\text{MCS}}$ and $\overline{\text{MAS}}$ are inactive. $\overline{\text{WR}}$ is active and A_0 is HIGH.

Mode A (Figure 3-135)

$\overline{\text{MDS}}$ is strobed LOW for T_{W} . The DCP reads the data in at the beginning of T_3 . No Wait State is inserted.

Mode B (Figure 3-138)

$\overline{\text{MDS}}$ is strobed LOW for the Wait cycle T_{W} and the additional Wait cycle T_{W} to meet the minimum data strobe active time (parameter 44) of the DCP. The DCP reads the data in at the begin of T_3 .

Data Ciphering Speed

The byte transfer capability of the Z80 system bus limits the data ciphering throughput of the DCP. A Z80 DMA controller doubles the maximum throughput compared to a CPU-controlled transfer as indicated in the following table:

TABLE 3-19.

System Clk	DCP Clk	CPU	DCP	Mode	N	T
6 MHz	3 MHz	Z80B	AmZ8068	B	168/176	0.28/0.27
4 MHz	4 MHz	Z80A	AmZ8068	A	168	0.19
2.5 MHz	2.5 MHz	Z80	Am9518	A	168	0.14

N — Number of DCP clock cycles to transfer and cipher 8 bytes of data. In CPU-controlled modes the use of the Z80 block transfer commands like INIR, INDR, OTIR or OTDR is assumed.

T — Throughput in Mbyte/s

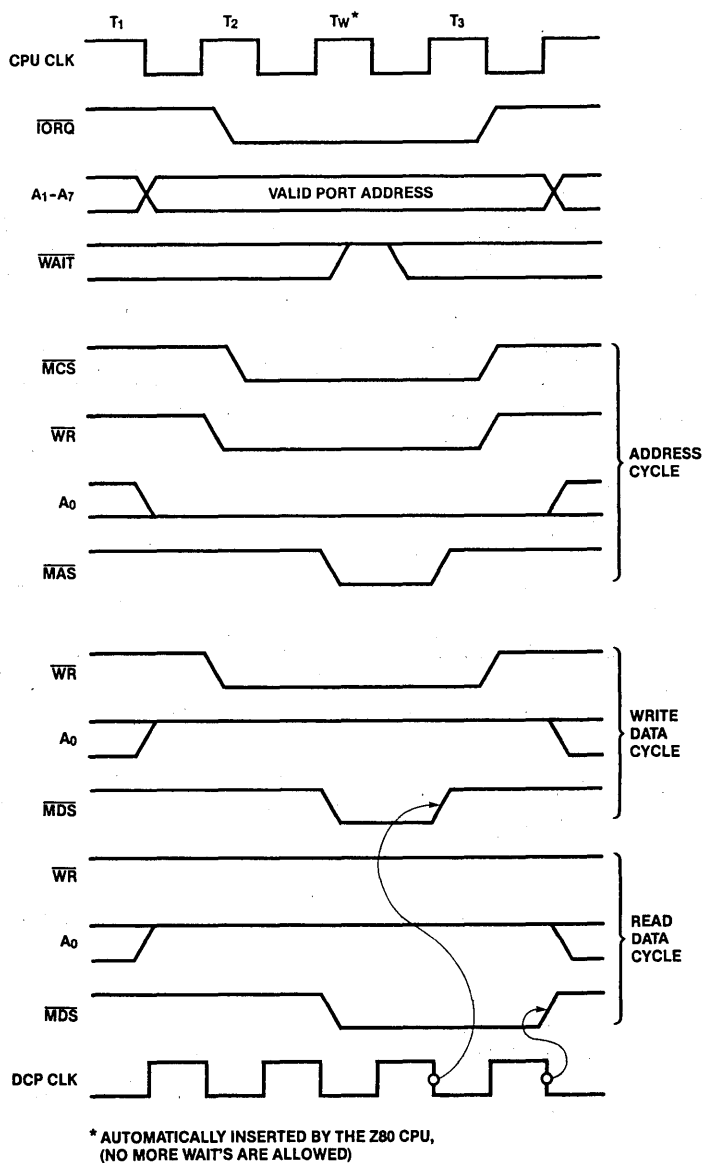
The formula for calculating the throughput is:

$$T = (8 * f) / (N + m) \text{ Mbyte/s}$$

f — DCP clock in MHz

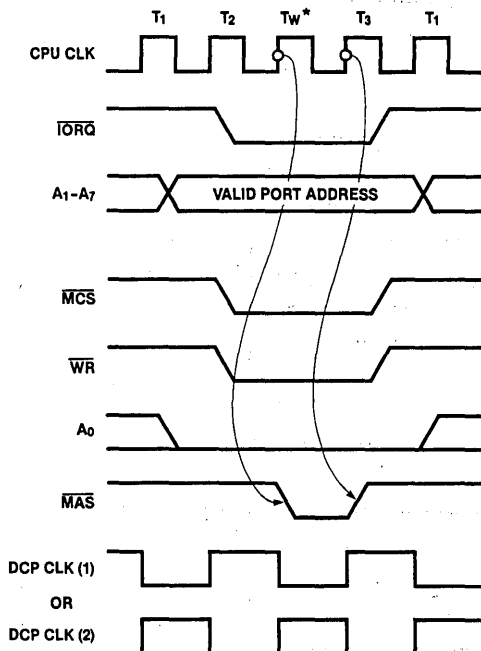
8 — 8 bytes per block

m — Number of extra DCP clock cycles to get a minimum delay time of five clocks between transferring the last byte of one block and the first byte of the next block. In CPU controlled transfers $m=0$ can be assumed, because the CPU has to evaluate instruction fetches and memory data transfers between two I/O accesses. MFLG indicates if the DCP accepts data transfer.



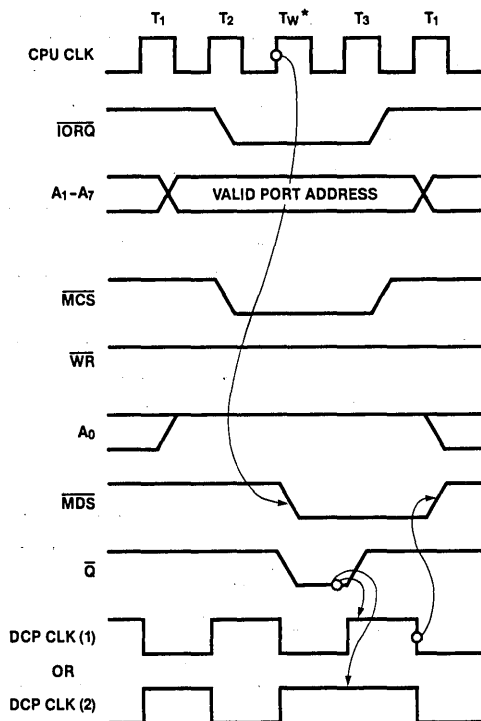
02188A-98

Figure 3-135. Z80-Am9518/Z80A-AmZ8068 Timing Diagram (Mode A)



02188A-99

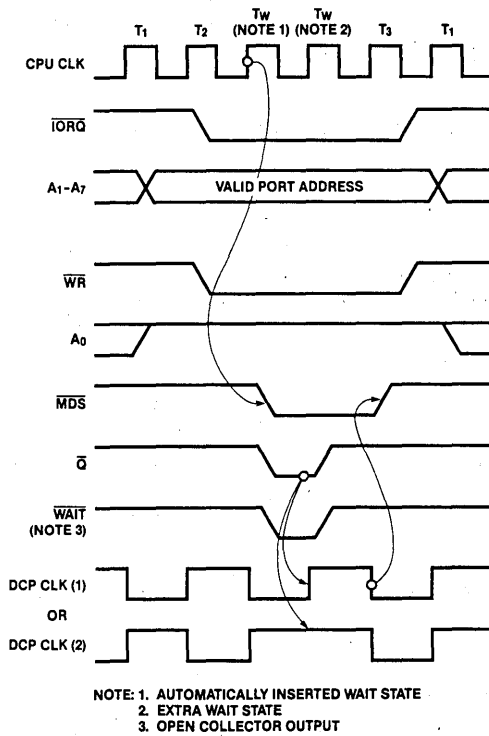
Figure 3-136. Address Latch Cycle (Mode B)
(No Clock Synchronization)



* AUTOMATICALLY INSERTED WAIT STATE

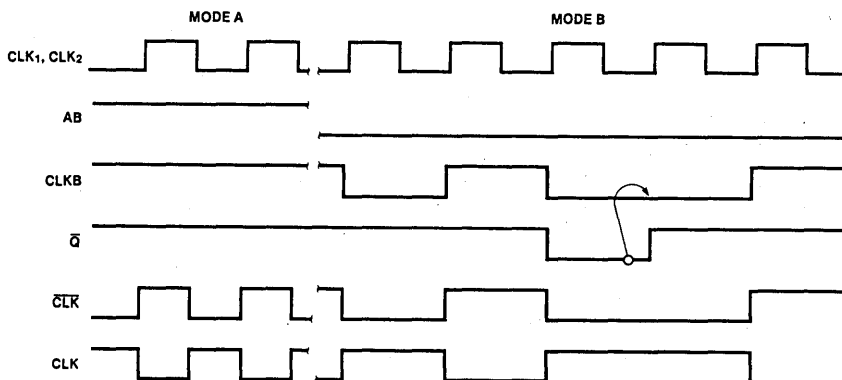
02188A-100

Figure 3-137. Data Read Cycle (Mode B)



02188A-101

Figure 3-138. Data Write Cycle (Mode B)



02188A-102

Figure 3-139. Clock Timing Diagram (Mode A and B)

PAL16R4

DCP046

Z80- AM9518/AMZ8068 INTERFACE CONTROLLER
ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION

JUERGEN STELBRINK 5/2/83

CLK1	CLK2	/CS	/IORQ	A0	/WR	AB	NC	NC	GND
/OE	NC	/WAIT	/CLKB	/Q	/MDS	/MAS	/MCS	CLK	VCC

MCS = IORQ*CS*/A0 ; MASTER PORT CHIP SELECT

MAS := IORQ*CS*/A0*WR*/MAS ; MASTER PORT ADDRESS STROBE

MDS := IORQ*CS*WR*/MDS*A0*AB + ; WRITE DATA STROBE (MODE A)
 IORQ*CS*WR*A0*/MDS*/Q*/AB + ; WRITE DATA STROBE (MODE B)
 IORQ*CS*WR*A0*MDS*Q*/AB + ; WRITE DATA STROBE (MODE B)
 IORQ*CS*/WR*A0 ; READ DATA STROBE (MODE A+B)

CLKB := /CLKB*/Q*/AB ; CLOCK FOR MODE B

/CLK = CLK2*AB + ; (MODE A)
 CLKB ; (MODE B)

Q := IORQ*CS*/MDS*/Q*A0*/AB ; USED TO GENERATE MDS AND WAIT

IF (Q*WR) WAIT = Q*WR ; WAIT TO Z80

FUNCTION TABLE

CLK1	CLK2	AB	/CS	/IORQ	A0	/WR	CLK	/MCS	/MAS	/MDS	/WAIT	/Q	/CLKB
/	/	/	/	/	/	/	/	/	/	/	/	/	/
C	C		I				W						C
L	L	/	O	/	C	M	M	M	A				L
K	K	A	C	R	A	W	L	C	A	D	I	/	K
1	2	B	S	Q	0	R	K	S	S	S	T	Q	B

COMMENT

; MODE A: Z80- AM9518 OR Z80A- AMZ8068 INTERFACE
 ; (DCP CLOCK = CPU CLOCK)

; CLOCK GENERATION

X	L	H	X	X	X	X	H	X	X	X	Z	H	H
X	H	H	X	X	X	X	L	X	X	X	Z	H	H

; ADDRESS LATCH

H	X	H	H	H	X	H	X	H	H	H	Z	H	H	; MACHINE CYCLE T1
L	X	H	L	H	L	H	X	H	H	H	Z	H	H	
C	X	H	L	H	L	H	X	H	H	H	Z	H	H	
H	X	H	L	L	L	L	X	L	H	H	Z	H	H	; CYCLE T2
C	X	H	L	L	L	L	X	L	L	H	Z	H	H	
C	X	H	L	L	L	L	X	L	H	H	Z	H	H	; CYCLE TW
H	X	H	L	H	L	H	X	H	H	H	Z	H	H	; CYCLE T3

Figure 3-140. Source Listing for the Z80 to Am9518/Am9568 Interface

```

C X H L H L H X H H H Z H H
;
; WRITE DATA OPERATION
;
C X H L H H H X H H H Z H H ; CYCLE T1
C X H L L H L X H H L Z H H ; CYCLE T2
C X H L L H L X H H H Z H H ; CYCLE TW
C X H L H H H X H H H Z H H ; CYCLE T3
;
; READ DATA OPERATION
;
C X H L H H H X H H H Z H H ; CYCLE T1
C X H L L H H X H H L Z H H ; CYCLE T2
C X H L L H H X H H L Z H H ; CYCLE TW
C X H L H H H X H H H Z H H ; CYCLE T3
;
; INVALID OPERATION (READ IN ADDRESS LATCH)
;
C X H L L L H X L H H Z H H ; NO /MAS !
;
;
; C C / I / / / W C
; L L / O / C M M M A L
; K K A C R A W L C A D I / K
; 1 2 B S Q 0 R K S S S T Q B COMMENT
-----
;
; MODE B: Z80B- AMZ8068 INTERFACE
; (DCP CLOCK = CPU CLOCK/2)
;
; WRITE DATA OPERATION
;
C X L H H H H L H H H Z H L ; CYCLE T1
C X L H H H H H H H H Z H H ; CYCLE T2
C X L L L H L L H H L L L L ; FIRST WAIT CYCLE (CLK=L)
C X L L L H L H H H L Z H H ; SECOND WAIT CYCLE
C X L L L H L L H H H Z H L ; CYCLE T3
;
C X L L L H L H H H L L L H ; FIRST WAIT CYCLE (CLK=H)
C X L L L H L H H H L Z H H ; SECOND WAIT CYCLE (SYNC !)
C X L L L H L L H H H Z H L ; CYCLE T3
;
; READ DATA OPERATION
;
C X L H H H H H H H H Z H H ; CYCLE T1
C X L H H H H L H H H Z H L ; CYCLE T2
C X L L L H H H H H L Z L H ; WAIT CYCLE
C X L L L H H H H H L Z H H ; CYCLE T3 (SYNC!)
C X L L H H H L H H H Z H L ; NEXT CYCLE
-----

```

Figure 3-140. Source Listing for the Z80 to Am9518/Am9568 Interface (Continued)

DESCRIPTION:

THIS PAL GENERATES ALL NECESSARY BUS CONTROL SIGNALS, TO INTERFACE THE AM9518 OR AMZ8068 TO THE Z80 CPU WITH A SYSTEM CLOCK UP TO 6 MHZ.

2 INPUT AND 1 INPUT/ OUTPUT PINS ARE NOT USED, SO THAT FOR EXAMPLE A DATA BUS TRANSCEIVER CONTROL LOGIC CAN BE ADDED.

IN SYSTEMS WITH A CLOCK UP TO 4 MHZ, THE DCP RUNS DIRECTLY AT THIS FREQUENCY (MODE A, INPUT AB = HIGH). IF THE FREQUENCY IS HIGHER, THE DCP IS DIVIDED BY TWO FROM THE SYSTEM CLOCK (MODE B, AB = LOW).

INPUT PINS:

CLK1, CLK2 CLK1 IS THE CLOCK INPUT FOR THE FOUR INTERNAL D-FLIP-FLOPS. THEY ARE CLOCKED BY THE RISING EDGE OF CLK1. THE DCP DATA STROBE MUST BE SYNCHRONOUS TO THE FALLING EDGE OF THE CLOCK; THE INVERTED CLK2 IS THEREFORE SENT TO THE OUTPUT CLK. IN MODE B CLK2 IS SYNCHRONIZED BEFORE IT APPEARS ON THE CLK OUTPUT. BOTH INPUTS ARE CONNECTED TO THE Z80 SYSTEM CLOCK.

/CS CHIP SELECT GENERATED BY AN ADDRESS DECODER LOGIC (ACTIVE LOW). IF /CS IS ONLY ACTIVE IN I/O CYCLES, THE /IORQ INPUT CAN BE WIRED LOW.

/IORQ INPUT/ OUTPUT REQUEST OF THE Z80 (LOW ACTIVE)

A0 LEAST SIGNIFICANT BIT OF THE Z80 ADDRESS BUS TO SELECT TYPE OF OPERATION:
A0= LOW SELECT REGISTER FOR NEXT DATA CYCLES (ADDRESS LATCH)
A0= HIGH READ OR WRITE INTERNAL REGISTER (DATA TRANSFER TO CONTROL, MODE, INPUT OR OUTPUT REGISTER)

/WR WRITE SIGNAL OF THE Z80, DEFINES DATA TRANSFER DIRECTION

AB AB= HIGH MODE A
 AB= LOW MODE B

OUTPUT SIGNALS:

/WAIT ACTIVE LOW DURING FIRST WAIT CYCLE IN WRITE DATA OPERATION IN MODE B, TO GENERATE AN EXTRA WAIT STATE. THE OTHER TIME /WAIT IS IN THREE STATE.

/MCS MASTER PORT CHIP SELECT, ONLY ACTIVE IN ADDRESS LATCH CYCLES

/MAS MASTER PORT ADDRESS STROBE, ACTIVE IN ADDRESS CYCLES TO LATCH THE REGISTER ADDRESS AND /MCS IN. THE DCP STORES INTERNALLY THE ADDRESS AND THE CHIP SELECT TO THE NEXT ADDRESS LATCH CYCLE

/MDS MASTER PORT DATA STROBE TO ENABLE DATA TRANSFER TO THE INTERNAL REGISTERS OF THE DCP

CLK DCP CLOCK, IN MODE B SYNCHRONIZED TO THE MASTER PORT DATA STROBE (/MDS)

/CLKB DCP CLOCK OUTPUT INTERNALLY USED FOR MODE B (NOT CONNECT)

/Q INTERNAL STATUS SIGNAL (NOT CONNECT)

Figure 3-140. Source Listing for the Z80 to Am9518/Am9568 Interface (Continued)

Z8001/8002 Overview

This CPU has a 16-bit data multiplexed address/data bus (the Z8001 has 7 additional non-multiplexed address outputs, called segment bits, extending its memory address range to 8 Mbyte).

Z8001/8002 addresses are specified as bytes. In a 16-bit word the least significant byte has the higher address, the most significant byte has the lower address. This differs from the 8080, 8085, 8086, and Z80. Moreover, 16-bit and 32-bit words must be aligned with an even address.

The 8-bit peripherals should be connected to the lower half of the Z8001/2 data bus and must be addressed with odd addresses (non-contiguous addressing).

The data bus is "asynchronous", i.e. the CPU machine cycle can be stretched without clock manipulation by inserting Wait States between T2 and T3 of a read or write cycle to accommodate slower memory or peripherals.

Separate address space for I/O (64 kbytes) defined by Status Code 0010 on the four Status outputs and by the R/W output from the Z8001/2. Addresses are guaranteed valid before the rising edge of $\overline{AS}$.

Data from I/O must be valid before the rising edge of $\overline{DS}$. Data to I/O is valid during $\overline{DS}$.

DMA: The bus is requested by activating the $\overline{BUSRQ}$ input to the Z8001/2.

Bus Grant is confirmed by the $\overline{BUSAK}$ output from the Z8001/2.

Interrupt is requested by activating either the $\overline{NMI}$ (non-maskable), $\overline{NVI}$ (non-vectored interrupt), $\overline{VI}$ (vectored interrupt), or $\overline{SEGT}$ (segment trap from the memory management unit).

Interrupt is acknowledged by the STATUS code output from the Z8001/2. During the Interrupt Acknowledge Cycle the Z8001/2 reads a 16-bit word containing an 8-bit jump vector for vectored interrupt.

Z8001 to Am7990 LANCE Interface

Z8001 interface to the LANCE is easily accomplished since the Z8000 also has a multiplexed bus and most of the control signals can directly be connected (Figures 3-141 and 3-142). This design also uses the PAL (AmPAL16L8) to reduce parts count. $\overline{INTR}$ pin of the LANCE is connected to $\overline{NVI}$ pin of the Z8001, since LANCE does not return a vector during the interrupt acknowledge cycle. The PAL uses the status lines ST_3 - ST_0 (when Z8000 is the bus master), or $\overline{HLDA}$ from LANCE (when LANCE is in bus master mode), to generate $\overline{M}$, the memory request signal. The user should program the $\overline{ACON}$, $\overline{BCON}$, and $\overline{BSWP}$ to 1, prior to initializing the LANCE. When the LANCE is the bus master, $\overline{DALI}$ and $\overline{DALO}$ control the transceiver. When the CPU is the bus master, $\overline{T}$ and $\overline{R}$ are generated from R/W and $\overline{DS}$ to control the transceiver.

AMPAL16L8
PAT001
File: Z8K90.PAL

RASOUL M. OSKOUY
MARCH 13, 1984

Z8001 TO LANCE INTERFACE
ADVANCED MICRO DEVICES

```
/AS /HLDA READ /CS /DS /READY ST3 ST2 ST1 GND
ST0 /T /M /WAIT LE /R NC NC NC VCC

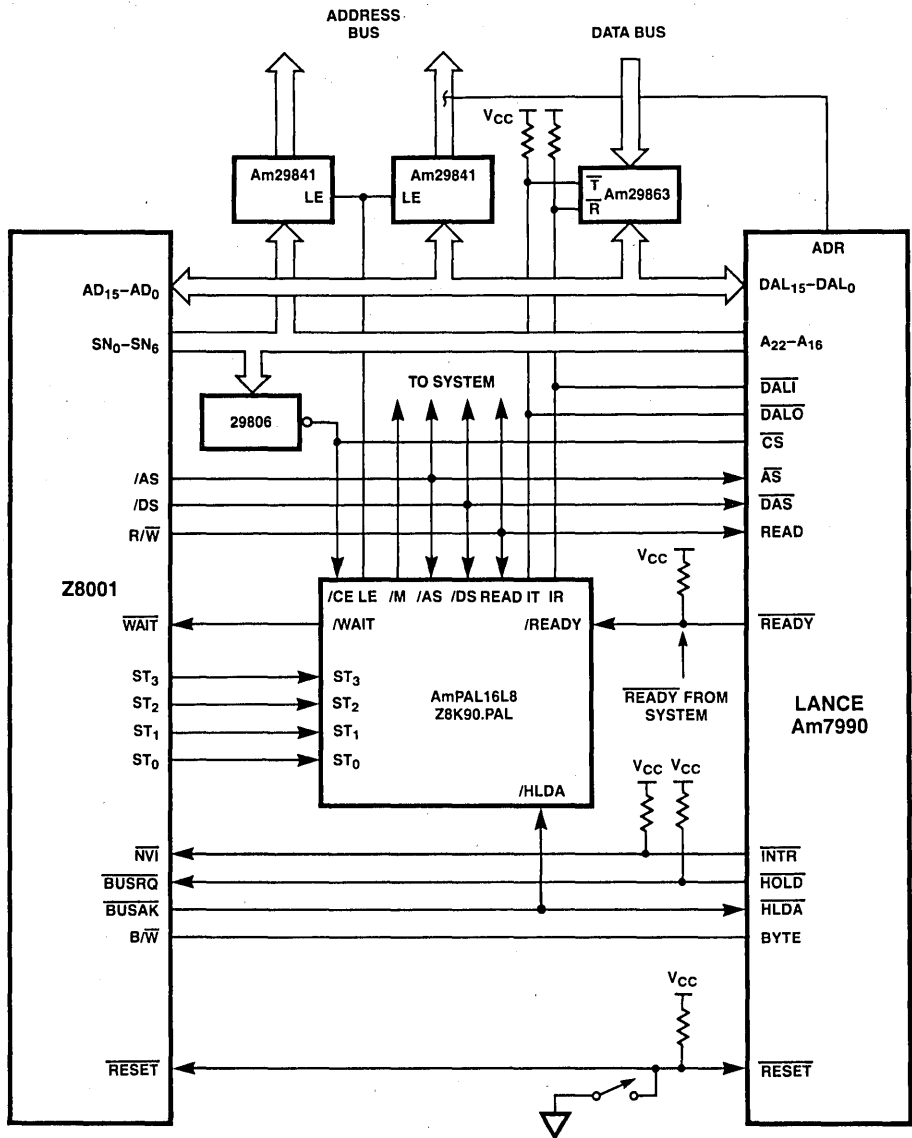
IF /(HLDA) T = /READ*/CS
IF (/HLDA) R = READ*DS*/CS

M = /HLDA*ST3*/ST2*/ST1*/ST0 +
    /HLDA*/ST2*/ST1*ST0 +
    /HLDA*ST3*ST2*/ST1 +
    HLDA

WAIT = /READY

/LE = AS
```

Figure 3-141. Source Listing for the Example of Figure 3-142



02188A-121

Figure 3-142. Z8000-AmZ8068 Interface

3.5 BUS INTERFACE LOGIC

3.5.1 MULTIBUS TO Am9516 INTERFACE

This interface shows the Am9516 connected to the MULTIBUS (Figure 3-143). This is accomplished by two PAL device designs. The equations for the PAL devices are shown in Figures 3-144 and 3-145. The first designated Am9516MBC does the MULTIBUS arbitration as defined by the MULTIBUS specification. Common bus request ($\overline{\text{CBRQ}}$) was not implemented in this design. Additionally this design holds the bus as long as $\overline{\text{BREQ}}$ is active. If the user wishes to release the bus after each transaction the Am9516 should be programmed for CPU interleave.

The second PAL device in this interface designated 9516MBC, converts the Am9516 signals into MULTIBUS control signals. It also generates $\overline{\text{RD}}$ and $\overline{\text{WR}}$ for the 8530 so that flyby trans-

fers can be done. When not doing flyby, this part of the PAL device should be changed. There are several considerations when using the SCC with DMA not addressed here. These are covered in a separate application. This example will work for moderate serial data rates. To operate the SCC at maximum speed, a local RAM should be used as bus arbitration overhead could cause problems. The main purpose here was to illustrate the versatility of PAL devices and how easy it is to interface apparently incompatible devices to the MULTIBUS.

The two PAL device shown are similar in function to the 8289 and 8288 shown with the 8086 CPU. A similar design could be done for processors such as the 68000 or other bus masters such as the 8052 CRT controller which has its own DMA.

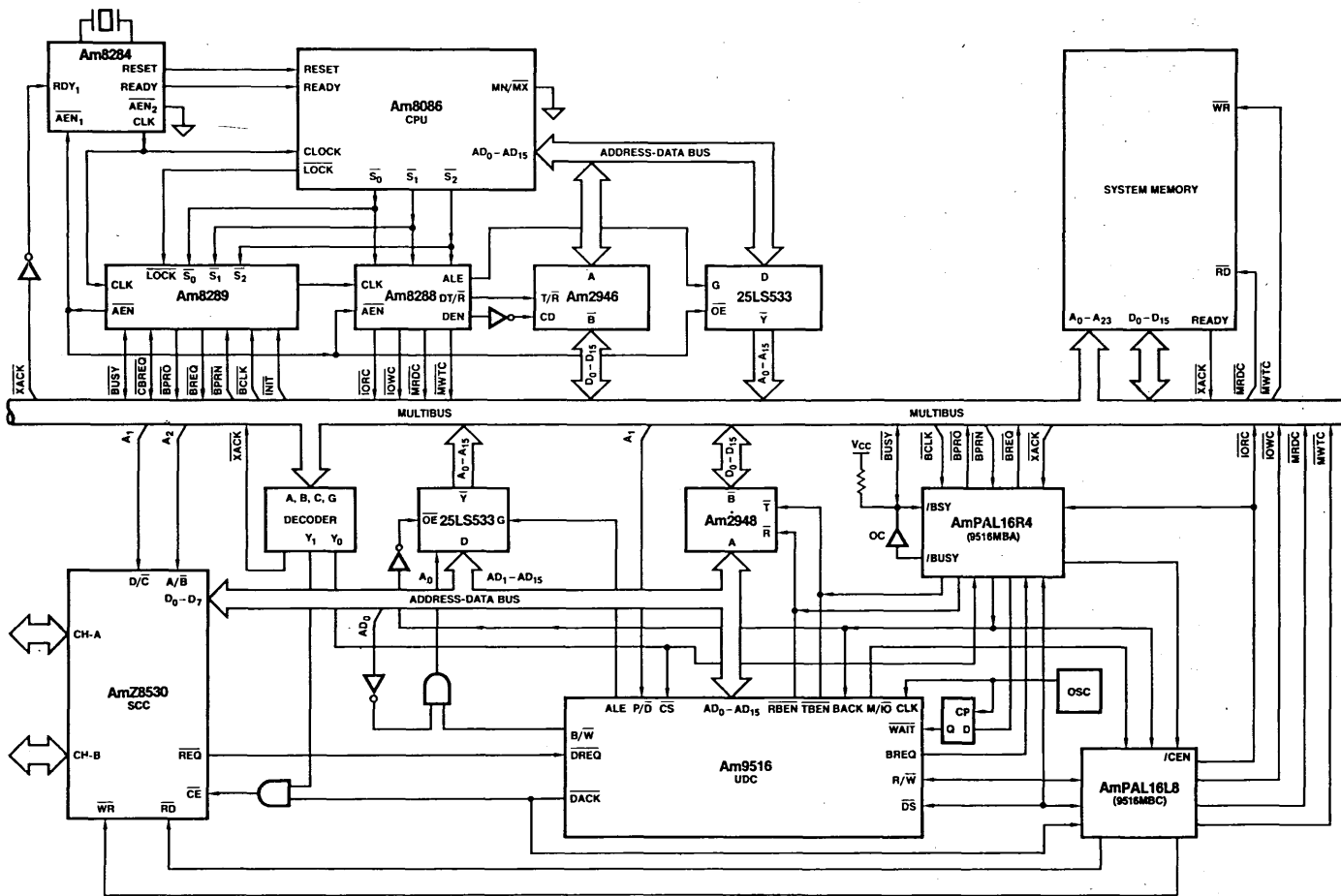


Figure 3-143. Am9516 to 8086 in a MULTIBUS® Environment

PAL16L8
PAT 002
MULTIBUS CONTROL FOR 9516
ADVANCE MICRO DEVICES

PAL DESIGN SPECIFICATION
JOE BRCICH 9 SEPT 82

BACK MIO NC NC /DACK NC NC NC /CEN GND
NC /RD /IORC /DS /MWTC /MRDC /IOWC /RW /WR VCC

IF (BACK) IORC = /MIO*DS*/RW*CEN

IF (BACK) IOWC = /MIO*DS*RW*CEN

IF (BACK) MRDC = MIO*DS*/RW*CEN

IF (BACK) MWTC = MIO*DS*RW*CEN

RD = DACK*RW*BACK + IORC*/BACK

WR = DACK*, RW*BACK + IOWC */BACK

IF (/BACK) DS = IORC + IOWC

IF (/BACK) RW = IOWC

DESCRIPTION

THIS PAL CONVERTS MULTIBUS SIGNALS INTO 9516 COMPATIBLE SIGNALS AND VICE VERSA. IT ALSO SUPPORTS THE 8530 IN FLYBY MODE.

Figure 3-144. Source Listing for MULTIBUS to Am9516 Interface

PAL16R4
PAT 004
MULTIBUS ARBITER FOR 9516
ADVANCE MICRO DEVICES

PAL DESIGN SPECIFICATION
JOE BRCICH 30 JULY 84

/BCLK /XACK BRQ /BSY /BPRN /DS NC /IORC /CS GND
/OE /RBEN /TBEN BACK /CEN /BREQ /BUSY /BPRO /WAIT VCC

IF (/BACK) TBEN = IORC * CS

IF (/BACK) RBEN = /IORC * CS

WAIT = /XACK * BACK

BREQ := BRQ

BPRO = /BRQ * BPRN

/BACK := /BUSY

BUSY := BREQ * BPRN * /BSY * /BUSY +
BREQ * BUSY * BPRN + DS * BUSY

CEN := BACK

DESCRIPTION

/CEN DELAYS THE COMMANDS TO MEET THE MULTIBUS REQUIREMENT THAT ADDRESS AND DATA BE VALID AT LEAST 50 NS PRIOR TO CONTROL ACTIVE. IF WE DO NOT ALLOW PREEMPTION; REMOVE BPRN FROM THE SECOND EXPRESSION IN THE BUSY EQUATION AND ELIMINATE THE THIRD EXPRESSION.

Figure 3-145. Source Listing for MULTIBUS to Am9516 Interface

3.5.2 Z-BUS AND 8088/8086 INTERFACE

This application note describes how replacing two 8086 support chips with a Z8000 support chip and an AmPAL16R8A allows the 8086 CPU to interface directly to the Z-BUS. Since the timing of the signals used is the same for the 8088 CPU, this circuit will work equally well in those applications.

Interfacing the 8086 CPU to the Z-BUS allows 8086 users to take advantage of the very powerful Z8000 peripheral and memory support circuits that are available. The Z8000 peripheral circuits in particular offer the user higher throughput rates, simpler control software and less system overhead requirements than any previous generation peripheral family for any CPU.

Design Requirements

The 8086 CPU can operate in two different modes. In minimum mode, it generates all the bus control and timing signals for the 8086 (8085, 8088) buses directly on-chip. In maximum mode, the CPU puts out status information early in each bus cycle and relies on an external bus controller chip, the 8288, to generate timing and control signals. This implementation uses the CPU in maximum mode and replaces the 8288 with a programmable array logic element

(AmPAL16R8A) that generates the Z-BUS timing and control signals from the status signals provided by the CPU. It also makes use of the AmZ8127 clock generator to allow precise timing resolution by providing an oscillator signal at 3 times the CPU clock frequency. The AmZ8127 provides all the clock generation functions of an 8284A as well as several additional functions. Either clock chip will work in this system.

The bus controller provides the following functions:

- Generates $\overline{AS}$, $\overline{DS}$, $\overline{INTACK}$ and R/W with proper timing relative to address and data.
- Provides simultaneous assertion of $\overline{AS}$ and $\overline{DS}$ during reset.
- Automatic insertion of 1 wait state for all I/O cycles.
- Synthesizes a single Z-BUS interrupt acknowledge cycle from the 8086 IACK cycles.

Figure 3-146 shows the circuit interconnection diagram. The system uses a high-speed AmPAL16R8A to generate $\overline{AS}$, $\overline{DS}$, R/W and $\overline{INTACK}$ of the Z-BUS, RDY for wait state generation, and three internal state variables. The registers are clocked with the 15-MHz OSC signal from the 8127. The five input signals to the PAL are 5-MHz CPU Clock Signal (CLK), System Reset (RESET), and the three CPU Status States ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$).

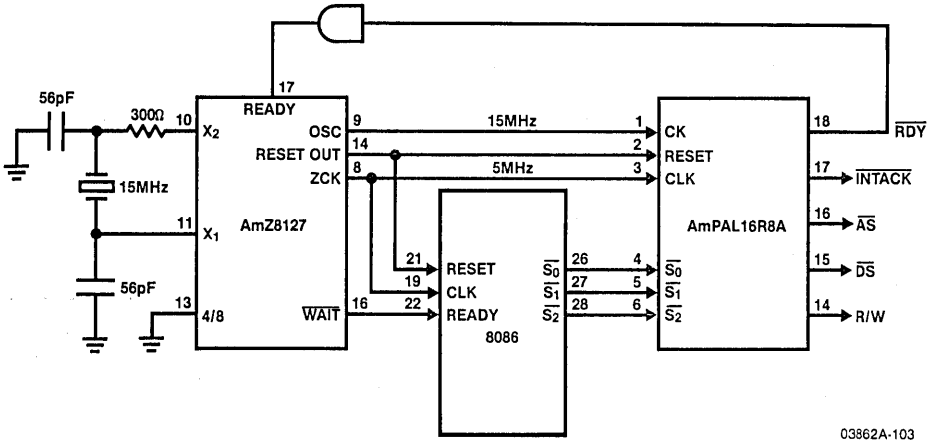
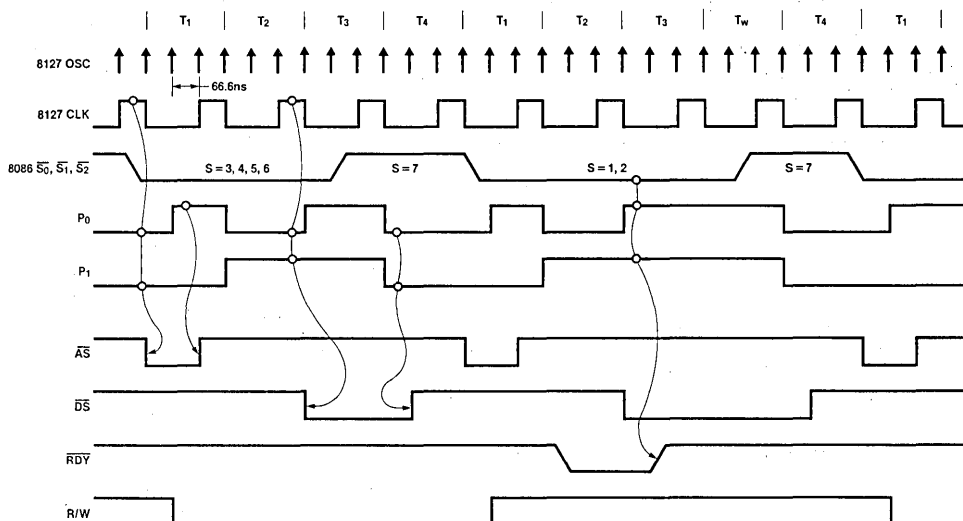


Figure 3-146. Circuit Interconnection Diagram

The CPU indicates the start of a bus cycle by bringing at least one of the status lines low from the idle high state (see Figure 3-147). This starts an internal timing sequence within the PAL which corresponds closely to the various T states of a bus cycle. $\overline{AS}$ is asserted during the time CLK is LOW during T_1 . $\overline{DS}$ is asserted at the start of T_3 . If it is an I/O cycle, then RDY would be disabled for one CLK period straddling T_2 and T_3 causing the 8127 to request 1 wait state after T_3 . In either case, $\overline{DS}$ remains asserted until after the first 1/3 of T_4 , which is identified by the status lines returning to the idle state during the previous cycle. R/W is generated by sampling $\overline{S_0}$ and $\overline{S_1}$ during $\overline{AS}$.

It is in the realm of interrupts where the Z8000 peripherals shine over other peripherals. Each peripheral can identify many different exception conditions during its operation. The occurrence of one or more of these conditions causes activation of a single interrupt request line. The peripheral wants the CPU to respond with a single interrupt acknowledge cycle, during which the peripheral resolves priority and provides the CPU with enough status and vector information to allow it to respond to the exception without any further interrogation of the peripheral. This allows interrupt driven systems to achieve very high data throughput rates.



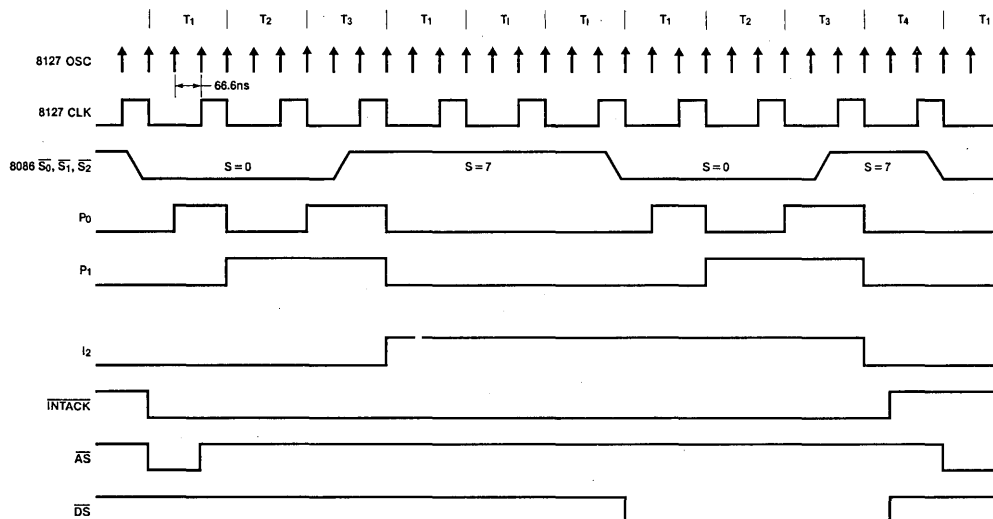
03862A-104

Figure 3-147. Memory and I/O Timing Diagram

The 8086 CPU responds to an interrupt request with a sequence of two interrupt acknowledge cycles, and only in the second is any data read off the bus. As stated before, the Z-BUS peripherals require only one acknowledge cycle. The timing of this has to be such that there is enough delay between $\overline{AS}$ going HIGH and $\overline{DS}$ going LOW to allow any prioritizing daisy chains to settle, and $\overline{DS}$ has to be wide enough to allow the peripheral time to place vector or status information on the bus. Figure 3-148 shows how these two requirements are accomplished by turning the two acknowledge cycles into one. The first cycle allows only $\overline{AS}$ and the second asserts only $\overline{DS}$ and does so for the complete cycle. This appears to the peripheral as one very long bus cycle which is identified as an interrupt acknowledge cycle by the assertion of INTACK.

Design Approach

To implement this design in an AmPAL16R8A requires recognizing the Z-BUS timing characteristics in Figures 3-147 and 3-148. The major characteristic to consider is counting the phases of a bus cycle. Internal state variables P_0 and P_1 are the result (see Figure 3-149). An additional internal state variable (I_2) is necessary to count the second bus cycle of an interrupt acknowledge sequence. As shown in Figure 3-150, I_2 in conjunction with INTACK allows $\overline{AS}$ to be asserted only in the first interrupt acknowledge cycle and $\overline{DS}$ only in the second. The RESET input is used to initialize the internal variables and assert $\overline{AS}$ and $\overline{DS}$. Note also that S_0 and S_1 are included in the $\overline{DS}$ equation to prevent $\overline{DS}$ from being asserted during a halt cycle.



03862A-105

Figure 3-148. Interrupt Acknowledge Timing Diagram

The fact that AMD PALs are user programmable allows a great deal of flexibility for the designer. Minor timing changes are easily implemented by simply adding or changing a term in the logic equations and reprogramming the device. In this system, we have timing resolution to 67 ns. This same configuration can be used with a 24-MHz crystal for 8-MHz CPU chips. The logic equations would change because the OSC period would be 42 ns. The only hardware change would be the crystal.

An additional PAL could also perform chip select decoding based on both address and status signals.

Conclusion

We have seen how a properly programmed PAL can be used to replace a specialized bus controller chip and allow an 8086 CPU to interface directly to Z-BUS peripheral(s) and/or memory systems. This brings all the advantages of the superior Z8000 peripheral family in terms of both throughput and ease of use to 8086 users with no increase in chip count while still allowing a wide range of design flexibility. The logic diagram and PALASM equations for the AmpAL16R8A 8086 to Z-BUS interface chip are shown in Figures 3-151, 3-152, 3-153, and Table 3-20.

P ₁	P ₀	PHASE	CPU T STATES
0	0	IDLE	T ₄ , T ₁
0	1	$\overline{A\overline{S}}$ TIME	T ₁
1	0	$\overline{A\overline{S}}$ TO $\overline{D\overline{S}}$ DELAY	T ₂
1	1	$\overline{D\overline{S}}$ TIME	T ₃ , T _w

03862A-106

Figure 3-149.

I ₂	INTACK	$\overline{A\overline{S}}$	$\overline{D\overline{S}}$
NO	NO	YES	YES
NO	YES	YES	NO
YES	YES	NO	YES

03862A-107

Figure 3-150.

PAL16R8

PAL DESIGN SPECIFICATION

PAT004

NICK ZWICK

6/21/82

8086 TO Z-BUS INTERFACE CHIP

ADVANCED MICRO DEVICES

CK RESET CLK /SO /S1 /S2 NC NC NC GND
/E /PO /P1 /RW /DS /AS /INTACK RDY /I2 VCC

;INTERNAL STATE VARIABLES

PO := /RESET* SO*/PO*/P1*/CLK +
/RESET* S1*/PO*/P1*/CLK +
/RESET* S2*/PO*/P1*/CLK +
/RESET*PO*/CLK +
/RESET*P1*CLK* SO +
/RESET*P1*CLK* S1 +
/RESET*P1*CLK* S2

P1 := /RESET*PO*/P1*CLK +
/RESET*P1*/CLK +
/RESET*P1*CLK* SO +
/RESET*P1*CLK* S1 +
/RESET*P1*CLK* S2

I2 := /RESET*INTACK*/I2*CLK*PO*P1 +
/RESET*I2*/P1 +
/RESET*I2*/PO +
/RESET*I2*PO*P1*/CLK

;Z-BUS OUTPUT SIGNALS

AS := RESET +
/PO*/P1*CLK*/I2 +
AS*/PO*/I2*/DS

DS := RESET +
/INTACK*/PO*P1*CLK*SO +
/INTACK*/PO*P1*CLK*S1 +
I2* SO* S1* S2 +
DS*PO*P1

RW := AS*SO*/S1 +
RW*/AS

INTACK := /RESET* SO* S1* S2 +
/RESET*INTACK*/I2*P1 +
/RESET*I2

/RDY := /RESET*SO*/S1* S2*/PO*P1 + ;DISABLE READY ON I/O OP
/RESET*/SO*S1* S2*/PO*P1

Figure 3-151. Source Listing for the Z-Bus to 8088/8086 Interface

TABLE 3-20. FUNCTION TABLE

FUNCTION TABLE

CK RESET CLK /S2 /S1 /S0 P0 P1 I2 /AS /DS /RW /INTACK RDY

[illegible]

TABLE 3-20. FUNCTION TABLE (Continued)

C	L	L	L	L	L	L	H	H	H	L	H	L	H
C	L	L	L	L	L	L	H	H	H	L	H	L	H
C	L	H	L	L	L	H	H	H	H	L	H	L	H
C	L	L	L	L	L	H	H	H	H	L	H	L	H
C	L	L	H	H	H	H	H	H	H	L	H	L	H
C	L	H	H	H	H	L	L	L	H	L	H	L	H
C	L	L	H	H	H	L	L	L	H	H	H	H	H

DESCRIPTION

THIS DEVICE IS USED FOR INTERFACING THE 8086 CPU DIRECTLY TO THE Z-BUS ALLOWING INTEGRATION OF THE VERY POWERFUL Z8000 PERIPHERAL AND MEMORY SUPPORT CIRCUITS INTO 8086 SYSTEMS. THE DEVICE IS DESIGNED TO WORK IN CONJUNCTION WITH THE Z8127 CLOCK GENERATOR FOR PRECISE TIMING RESOLUTION.

PAL16R8
 PAT004
 8086 TO Z-BUS INTERFACE CHIP
 ADVANCED MICRO DEVICES

PAL DESIGN SPECIFICATION
 NICK ZWICK 6/21/82

*D9724

FO

```

L0000 1001 0111 1110 1111 1111 1111 1110 1110 *
L0032 1010 1111 1111 1111 1111 1111 1101 1111 *
L0064 1010 1111 1111 1111 1111 1111 1111 1101 *
L0096 1010 1011 1111 1111 1111 1111 1110 1110 *
L0256 1011 1111 1011 0111 1011 1111 1110 1101 *
L0288 1011 1111 0111 1011 1011 1111 1110 1101 *
L0512 1011 1111 1011 1011 1011 1111 1111 1111 *
L0544 1001 1111 1110 1111 1111 1111 1110 1111 *
L0576 1010 1111 1111 1111 1111 1111 1111 1111 *
L0768 0111 1111 1111 1111 1111 1111 1111 1111 *
L0800 1101 0111 1111 1111 1111 1111 1101 1101 *
L0832 1101 1111 1111 1110 1101 1111 1111 1101 *
L1024 0111 1111 1111 1111 1111 1111 1111 1111 *
L1056 1111 0111 1001 1111 1111 1111 1110 1101 *
L1088 1111 0111 1101 1011 1111 1111 1110 1101 *
L1120 1110 1111 1011 1011 1011 1111 1111 1111 *
L1152 1111 1111 1111 1111 1110 1111 1110 1110 *
L1280 1111 1111 1011 0110 1111 1111 1111 1111 *
L1312 1111 1111 1111 1101 1111 1110 1111 1111 *
L1536 1011 0111 1111 1111 1111 1111 1101 1110 *
L1568 1011 1011 1111 1111 1111 1111 1110 1111 *
L1600 1011 0111 1011 1111 1111 1111 1110 1111 *
L1632 1011 0111 1111 1011 1111 1111 1110 1111 *
L1664 1011 0111 1111 1111 1011 1111 1110 1111 *
L1792 1011 1011 1011 1111 1111 1111 1101 1101 *
L1824 1011 1011 1111 1011 1111 1111 1101 1101 *
L1856 1011 1011 1111 1111 1011 1111 1101 1101 *
L1888 1011 1011 1111 1111 1111 1111 1111 1110 *
L1920 1011 0111 1011 1111 1111 1111 1110 1111 *
L1952 1011 0111 1111 1011 1111 1111 1110 1111 *
L1984 1011 0111 1111 1111 1011 1111 1110 1111 *

```

C711D*

```

V0001 C1XXXXXXXXXHHXLLHHH1 *
V0002 C00111XXXOXHHHHHHHH1 *
V0003 C01011XXXOXHHHHLHHH1 *
V0004 C00011XXXOXLHLHLHHH1 *
V0005 C00011XXXOXLHLHHHHH1 *
V0006 C01011XXXOXHLLHHHHH1 *
V0007 C00011XXXOXHLLHHHHH1 *
V0008 C00011XXXOXHLLHHHHH1 *
V0009 C01011XXXOXLLLHHHHH1 *
V0010 C00011XXXOXLLLHHHHH1 *
V0011 C00111XXXOXLLLHHHHH1 *
V0012 C01111XXXOXHHLHHHHH1 *
V0013 C00111XXXOXHHLHHHHH1 *
V0014 C00111XXXOXHHLHHHHH1 *
V0015 C01100XXXOXHHLHLHHH1 *
V0016 C00100XXXOXLHHHLHHH1 *
V0017 C00100XXXOXLHHHHHHH1 *

```

Figure 3-152. Fuse Plot and Test Vectors for the Z-Bus to 8088/8086 Interface

```

V0018 C01100XXXOXHLHHHHHH1 *
V0019 C00100XXXOXHLHHHHLH1 *
V0020 C00100XXXOXHLHHHHLH1 *
V0021 C01100XXXOXLLHLHHLH1 *
V0022 C00100XXXOXLLHLHHHH1 *
V0023 C00100XXXOXLLHLHHHH1 *
V0024 C01100XXXOXLLHLHHHH1 *
V0025 C00100XXXOXLLHLHHHH1 *
V0026 C00111XXXOXLLHLHHHH1 *
V0027 C01111XXXOXHHHLHHHH1 *
V0028 C00111XXXOXHHHHHHHH1 *
V0029 C01000XXXOXHHHHLHH1 *
V0030 C00000XXXOXLHHHLLHH1 *
V0031 C00000XXXOXLHHHHLHH1 *
V0032 C01000XXXOXHLHHHLHH1 *
V0033 C00000XXXOXHLHHHLHH1 *
V0034 C00000XXXOXHLHHHLHH1 *
V0035 C01000XXXOXLLHHHLHH1 *
V0036 C00000XXXOXLLHHHLHH1 *
V0037 C00111XXXOXLLHHHLHH1 *
V0038 C01111XXXOXHHHHHLHL1 *
V0039 C00111XXXOXHHHHHLHL1 *
V0040 C00111XXXOXHHHHHLHL1 *
V0041 C01000XXXOXHHHLHLHL1 *
V0042 C00000XXXOXLHHHLHLHL1 *
V0043 C00000XXXOXLHHHLHLHL1 *
V0044 C01000XXXOXHLHLHLHL1 *
V0045 C00000XXXOXHLHLHLHL1 *
V0046 C00000XXXOXHLHLHLHL1 *
V0047 C01000XXXOXLLHLHLHL1 *
V0048 C00000XXXOXLLHLHLHL1 *
V0049 C00111XXXOXLLHLHLHL1 *
V0050 C01111XXXOXHHHLHLHH1 *
V0051 C00111XXXOXHHHHHHHH1 *
A334

```

Figure 3-152. Fuse Plot and Test Vectors for the Z-Bus to 8088/8086 Interface (Continued)

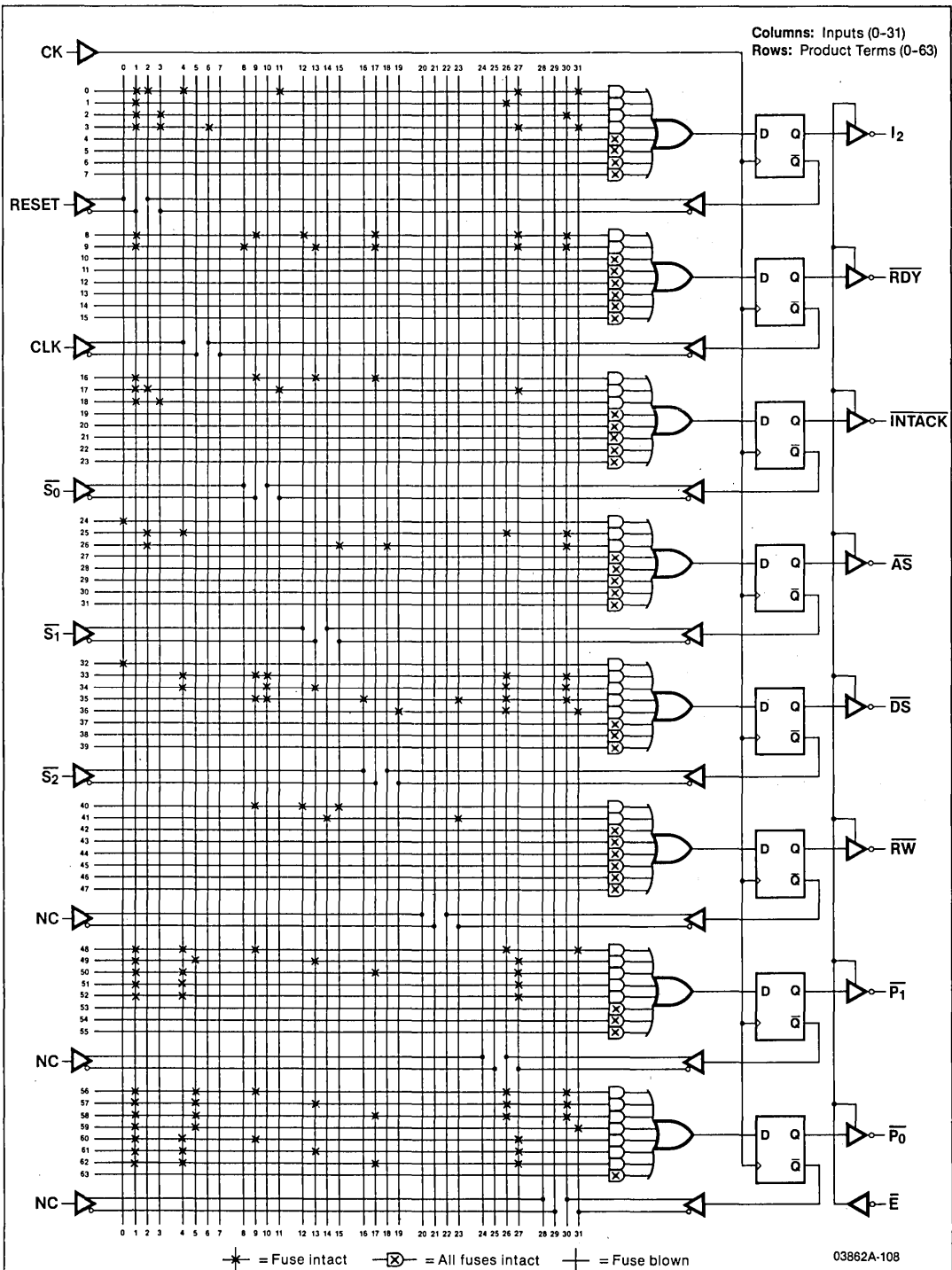


Figure 3-153. Logic Diagram for 8086 to Z-Bus Interface Using AmPAL16R8

3.5.3 AN AMD PAL MULTIBUS ARBITER

The popularity of bus oriented systems can be traced to their low cost, flexibility, and expandability. Expansion is easy because a well defined standard enforces compatibility and simplifies interfacing. The MULTIBUS is a good example of a popular and easily used bus standard. However, all of these bus systems require some way of interacting with the other devices on the bus. This interaction is generally controlled by an arbitration unit. This application note will describe how a PAL can be used to construct a custom MULTIBUS arbiter which is higher performance and more economical than the LSI alternatives.

Design Requirements

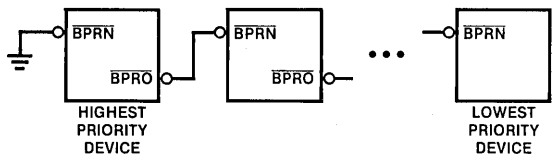
Since the MULTIBUS can have more than one master trying to use the bus at the same time, an arbitration scheme is required to ensure correct operation of the system. Prioritization is accomplished by assigning different access priorities to the different bus masters. The two implementations available to assign priority are serial and parallel. The serial method involves a daisy chain of bus grant ins (BPRN) and bus grant outs (BPRO) with higher priority devices occupying the positions closer to the beginning of the chain (Figure 3-154). The parallel method prioritizes the bus requests (BREQ) of all the masters and generates a bus grant in (BPRN) to the master of highest priority (Figure 3-155). The priority decoder of Figure 3-155 is easily implemented in a single AmPAL16L8 (Figure 3-156).

The timing requirements for the MULTIBUS are very easy to implement and are designed to handle the usual timing problems that appear in many systems. Control signals are all active LOW so that unconnected signals don't interfere with the

normal operation of the bus. All bus arbitration is synchronous and all data transfers are asynchronous (see Figure 3-157). The only requirement is that an address be valid 50 ns before any control signals (read or write) become active. This prevents subtle timing problems caused by slow buffer/driver turn-on times.

The arbitration and grant timing (illustrated in Figure 3-158) is also straightforward. Transfer requests are sent to the arbiters which then decide who gets the bus. If a master device is currently using the bus and is ordered off, it is allowed to complete its current bus transfer cycle (i.e., the current word or byte only). If the master, which was overruled, still needs the bus, it must wait until its priority is high enough to regain the bus.

A typical sequence is initiated when an external request is received (SREQ) from a master device. This signal is synchronized to insure a valid bus request, minimizing the possibility of a metastable state occurring. After synchronization, the bus priority out (BPRO) line is disabled to signal lower priority masters in the chain that a higher priority master wants the bus (serial method). In the parallel method, bus request and common bus request (BREQ and CBREQ) are asserted to let the other MULTIBUS arbiters know a master wants the bus. Now, if the bus is not busy (BUSY is inactive), then the arbiter grants the requesting master access to the bus and asserts BUSY. The address buffers are enabled at this time, one cycle ahead of the read and write signals. When the master receives a bus acknowledge (XACK) from the slave device, a single transfer cycle has occurred. The bus master then releases the bus and if it needs to do more transfers, another arbitration/grant cycle must take place.



03862A-109

Figure 3-154. Serial Priority Resolution

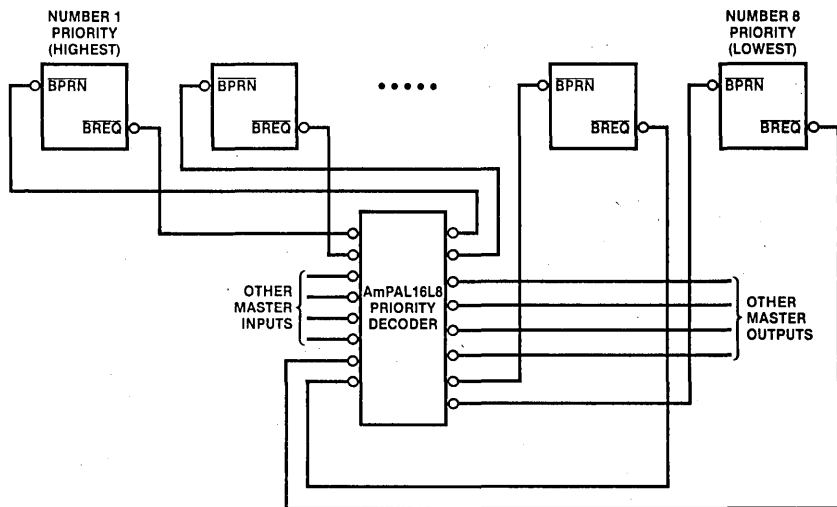
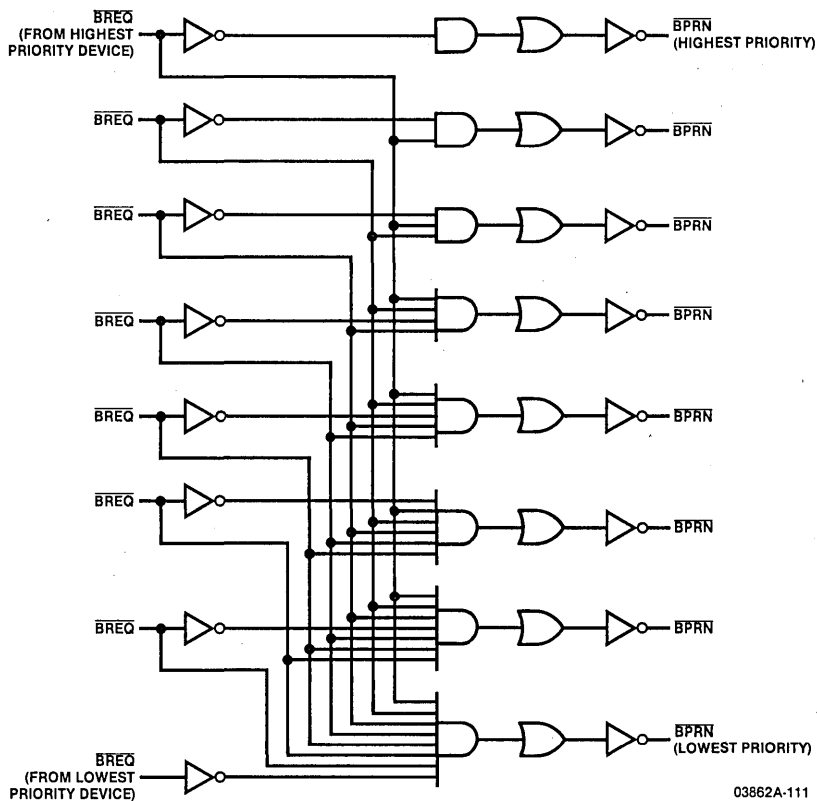


Figure 3-155. Parallel Priority Technique

03862A-110



03862A-111

Figure 3-156. PAL Implementation of Parallel Priority Resolution for MULTIBUS

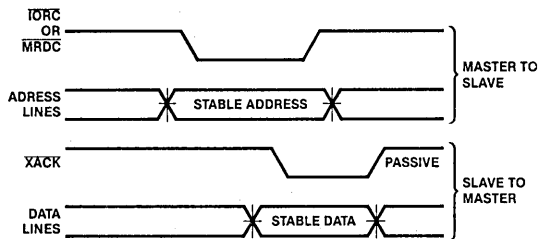


Figure 3-157.a Read AC Timing

03862A-112

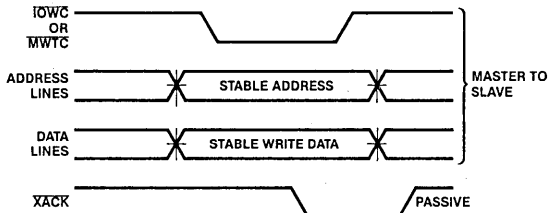


Figure 3-157.b Write AC Timing

03862A-113

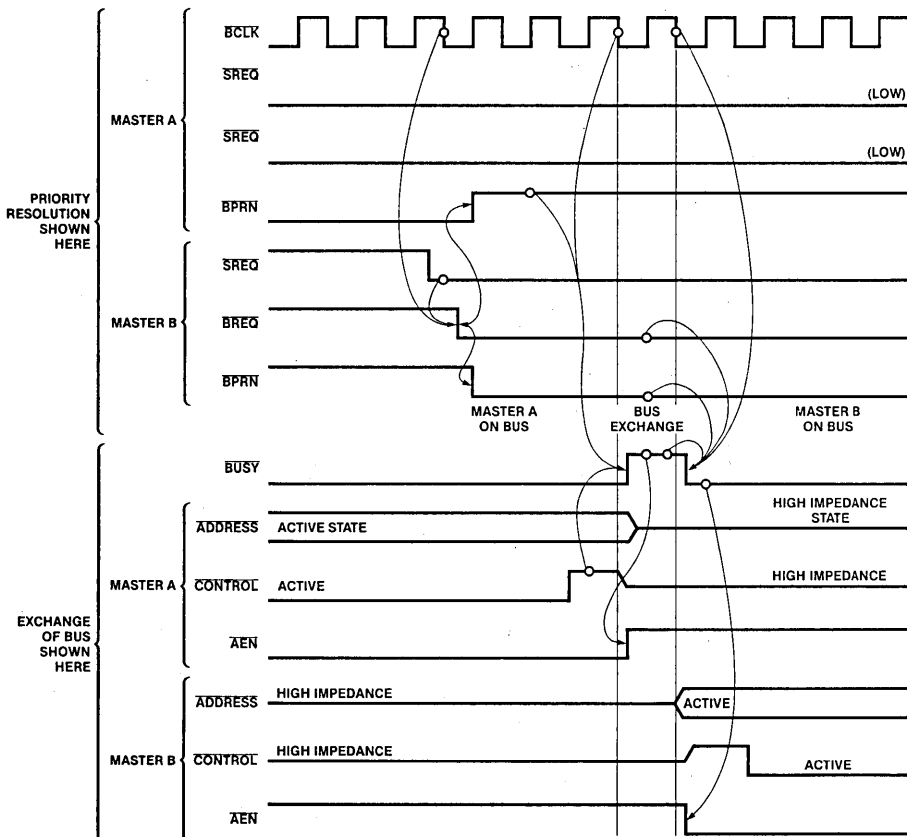


Figure 3-158. Bus Control Exchange Operation

03862A-114

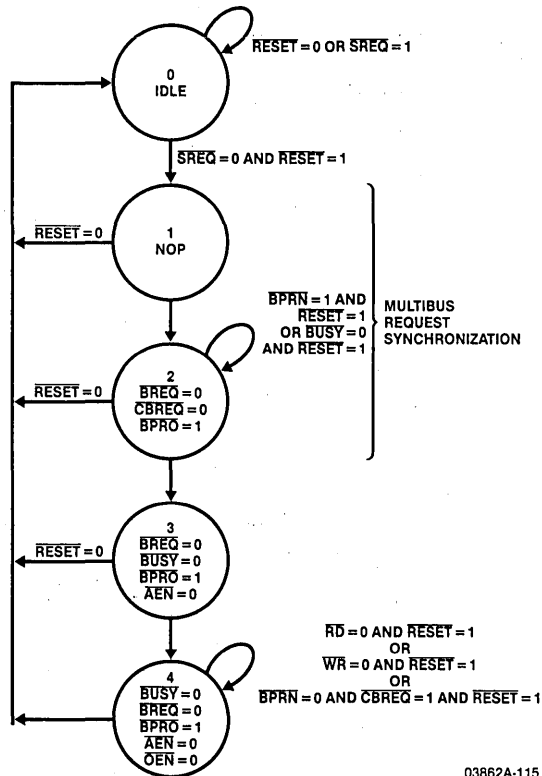
Design Approach

The first step in actually designing the arbiter was to convert the arbitration/grant and control signal sequence into a simple state transition diagram (see Figure 3-159). The state diagram was then partitioned into its three basic components:

- request/synchronization
- grant/access logic
- control signals

The bus request logic decides when to issue a bus request using $\overline{SREQ}$ along with a qualifying read or write request ($\overline{RD}$ or $\overline{WR}$). This signal is then fed through a double synchronizer (states 1 and 2). This creates an internally stable bus request

signal for the arbiter state machine (see Figure 3-160). Next, this request is fed into a bus grant flip-flop through some intervening logic. The intervening logic uses current bus status lines to determine whether to acquire the bus, give it up after the current transfer cycle is complete, or hold the bus (see grant/access logic in Figure 3-161). The final major function is the bus control logic. After successful acquisition of the bus, the appropriate control signals ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$) and address buffer enables must be asserted. In this case, the address buffer enable/grant ($\overline{AEN}$) line is run through a flip-flop to become a delayed read/write control signal enable ($\overline{OEN}$). This gives the drivers enough turn-on and setup time (100 ns minimum) for the address to stabilize on the MULTIBUS. The bus transfer control signal logic is illustrated in Figure 3-162.



03862A-115

Figure 3-159. PAL MULTIBUS Arbiter State Transition Diagram

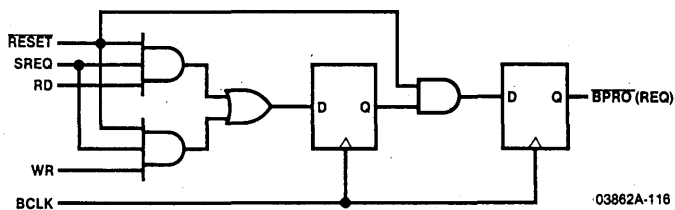


Figure 3-160. Request Synchronizer

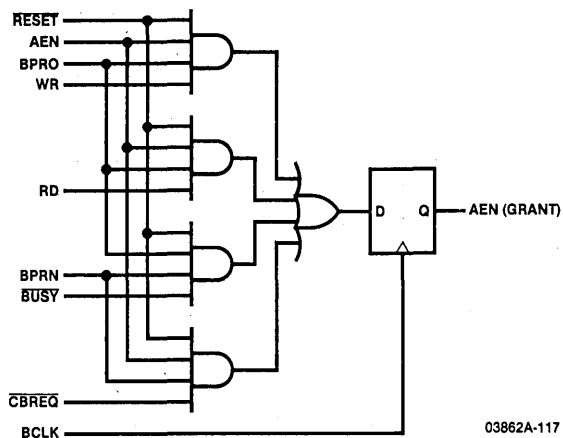


Figure 3-161. Grant/Access Logic

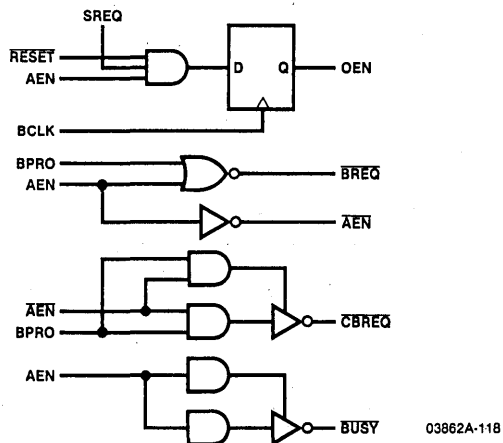


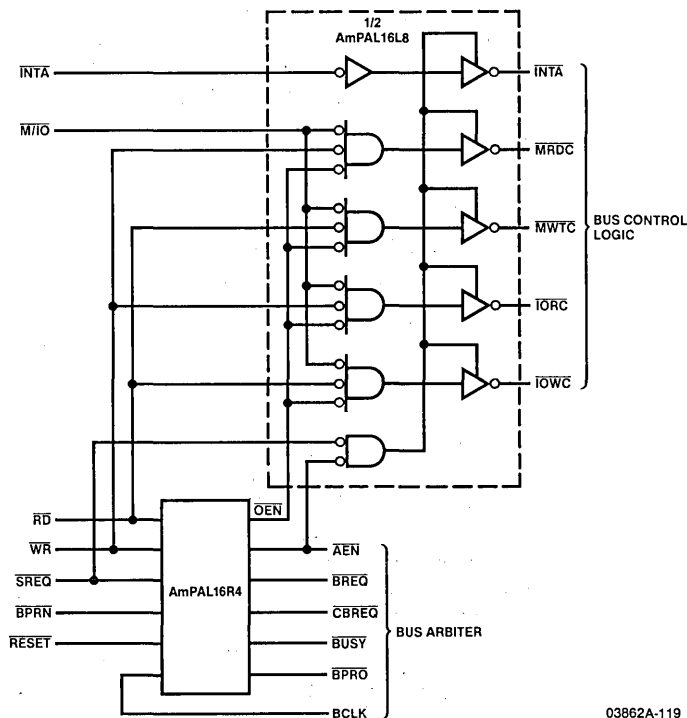
Figure 3-162. Bus Transfer Control

The **RESET** line is run to all the registers to provide a synchronous reset for arbiter initialization. The full MULTIBUS standard requires a control signal drive capability of 32 mA for all of its control lines. The PAL 24 mA drive specification can drive up to 7 cards on the bus, which is more than adequate for almost all applications.

A couple of interesting features are available which show the flexibility and functionality of PALs in a custom logic design such as this. The available MULTIBUS arbiters, like most LSI devices, have been optimized for one processor family. Any microprocessor (from the Z80 to the 8086, Z8000 or 68000) can be interfaced to the MULTIBUS by tailoring the request logic of the PAL arbiter. This can result in a significant parts count reduction. The MULTIBUS uses open collector drivers for several signal lines (**BUSY** and **CBREQ**). While PALs don't have open collector drivers, that condition can easily be

simulated by enabling the output drivers only when the output is active. In addition, most MULTIBUS arbiters force each master to re-arbitrate for access to the bus at the end of each data transfer. However, in the PAL arbiter design, if a master is executing multiple data transfers and if no master of higher priority is requesting the bus then the current master can retain the bus and execute more transfers. This reduces bus arbitration overhead and increases bus bandwidth. Finally, at the completion of all transfers, the current bus master normally releases the bus. In this design, if no one wants the bus, the last master to use it holds onto the bus on the assumption that it is going to be the next user. If it is, arbitration time is saved. If not, no time is lost.

Figure 3-163 shows a block diagram of the arbiter and bus control logic (which fits into 1/2 of an AmPAL16L8). A complete logic diagram and PALASM equations of the AmPAL16R4 are shown in Figures 3-164, 3-165, 3-166, and Table 3-21.



03862A-119

Figure 3-163. PAL MULTIBUS Arbiter with Bus Control Logic

PAL16R4

PAL DESIGN SPECIFICATION

PAT005

MARK S. YOUNG 6/22/82

MULTIBUS ARBITER

ADVANCED MICRO DEVICES

BCLK /RD /WR /SREQ /RESET /BPRN NC NC NC GND

/E /CBREQ /BUSY /SYNC /BPRO /AEN /OEN /BREQ NC VCC

SYNC := /RESET*SREQ*RD +
/RESET*SREQ*WR

BPRO := /RESET*SYNC

AEN := /RESET* AEN*BPRO*WR +
/RESET* AEN*BPRO*RD +
/RESET*BPRO*BPRN*/BUSY +
/RESET* AEN*BPRN*/CBREQ

OEN := /RESET*SREQ*AEN

IF(BPRO*/AEN) CBREQ = BPRO*/AEN

IF(AEN) BUSY = AEN

BREQ = BPRO +
AEN

Figure 3-164. Source Listing for the MULTIBUS Arbiter

TABLE 3-21. FUNCTION TABLE

FUNCTION TABLE

BCLK /E /RESET /RD /WR /SREQ /BPRN SYNC BPRO AEN OEN CBREQ BUSY BREQ

;INITIALIZE													
C	L	L	X	X	X	X	L	L	L	L	L	L	L
;WRITE OPERATION													
C	L	H	H	H	L	L	L	L	L	L	L	L	L
C	L	H	H	L	L	L	H	L	L	L	L	L	L
C	L	H	H	L	L	L	H	H	L	L	H	L	H
C	L	H	H	L	L	L	H	H	H	L	L	X	H
C	L	H	H	L	L	L	H	H	H	H	L	H	H
C	L	H	H	L	L	L	H	H	H	H	L	H	H
C	L	H	H	L	L	L	H	H	H	H	L	H	H
C	L	H	H	H	L	L	L	L	H	L	L	H	H
;REMOVE THE ARBITER FROM THE BUS													
C	L	H	H	H	H	H	L	L	L	L	L	L	L
;READ OPERATION (WITH BUS CONTENTION)													
C	L	H	H	H	L	L	L	L	L	L	L	H	L
C	L	H	L	H	L	L	H	L	L	L	L	H	L
C	L	H	L	H	L	L	H	H	L	L	H	H	H
C	L	H	L	H	L	L	H	H	L	L	H	H	H
C	L	H	L	H	L	L	H	H	H	L	L	X	H
C	L	H	L	H	L	L	H	H	H	H	L	H	H
C	L	H	L	H	L	L	H	H	H	H	L	H	H
C	L	H	L	H	L	L	H	H	H	H	L	H	H
C	L	H	H	H	L	L	L	H	H	H	L	H	H
C	L	H	H	H	H	L	L	L	H	L	L	H	H

DESCRIPTION

THIS DEVICE IS A MULTIBUS ARBITER. IT SUPPORTS BOTH SERIAL AND PARALLEL BUS ARBITRATION SCHEMES. INTERNAL SYNCHRONIZATION LOGIC MINIMIZES THE POSSIBILITY OF METASTABLE CONDITIONS. THE GRANT/ACCESS LOGIC HAS BEEN DESIGNED TO MINIMIZE BUS ARBITRATION OVERHEAD. THE TRANSFER CONTROL LOGIC HAS BEEN DESIGNED TO ALLOW INTERFACING A WIDE VARIETY OF PROCESSORS TO THE MULTIBUS.

PAL16R4

PAT005

MULTIBUS ARBITER

ADVANCED MICRO DEVICES

*D9724

F0

L0256 1111 1111 1111 1111 1111 1111 1111 1111 *
L0288 1111 1111 1111 1111 1110 1111 1111 1111 *
L0320 1111 1111 1111 1110 1111 1111 1111 1111 *
L0512 1111 1111 1011 0110 1111 1111 1111 1111 *
L0768 1111 1011 1111 0110 1110 1111 1111 1111 *
L0800 1011 1111 1111 0110 1110 1111 1111 1111 *
L0832 1111 1111 1111 0111 1010 1111 1101 1111 *
L0864 1111 1111 1111 0110 1011 1111 1111 1101 *
L1024 1111 1111 1111 0111 1111 1110 1111 1111 *
L1280 1011 1111 1011 0111 1111 1111 1111 1111 *
L1312 1111 1011 1011 0111 1111 1111 1111 1111 *
L1536 1111 1111 1111 1110 1111 1111 1111 1111 *
L1568 1111 1111 1111 1110 1111 1111 1111 1111 *
L1792 1111 1111 1111 1101 1110 1111 1111 1111 *
L1824 1111 1111 1111 1101 1110 1111 1111 1111 *

C3602*

V0001 CXXXOXXXX0011HHHHHX1 *
V0002 C1101OXXXX0011HHHHHX1 *
V0003 C1001OXXXX0011LHHHHX1 *
V0004 C1001OXXXX001L1LHHLX1 *
V0005 C1001OXXXX001XLLLHLX1 *
V0006 C1001OXXXX001LLLLLLX1 *
V0007 C1001OXXXX001LLLLLLX1 *
V0008 C1001OXXXX001LLLLLLX1 *
V0009 C1101OXXXX001LHLLLLX1 *
V0010 C1111OXXXX001LHHLHLX1 *
V0011 C11111XXX0011HHHHHX1 *
V0012 C1101OXXXX0010HHHHHX1 *
V0013 C0101OXXXX0010LHHHHX1 *
V0014 C0101OXXXX00LOLLHHLX1 *
V0015 C0101OXXXX00LOLLHHLX1 *
V0016 C0101OXXXX00LOLLHHLX1 *
V0017 C0101OXXXX001XLLLHLX1 *
V0018 C0101OXXXX001LLLLLLX1 *
V0019 C0101OXXXX001LLLLLLX1 *
V0020 C0101OXXXX001LLLLLLX1 *
V0021 C0101OXXXX001LLLLLLX1 *
V0022 C1101OXXXX001LHLLLLX1 *
V0023 C1111OXXXX001LHHLHLX1 *

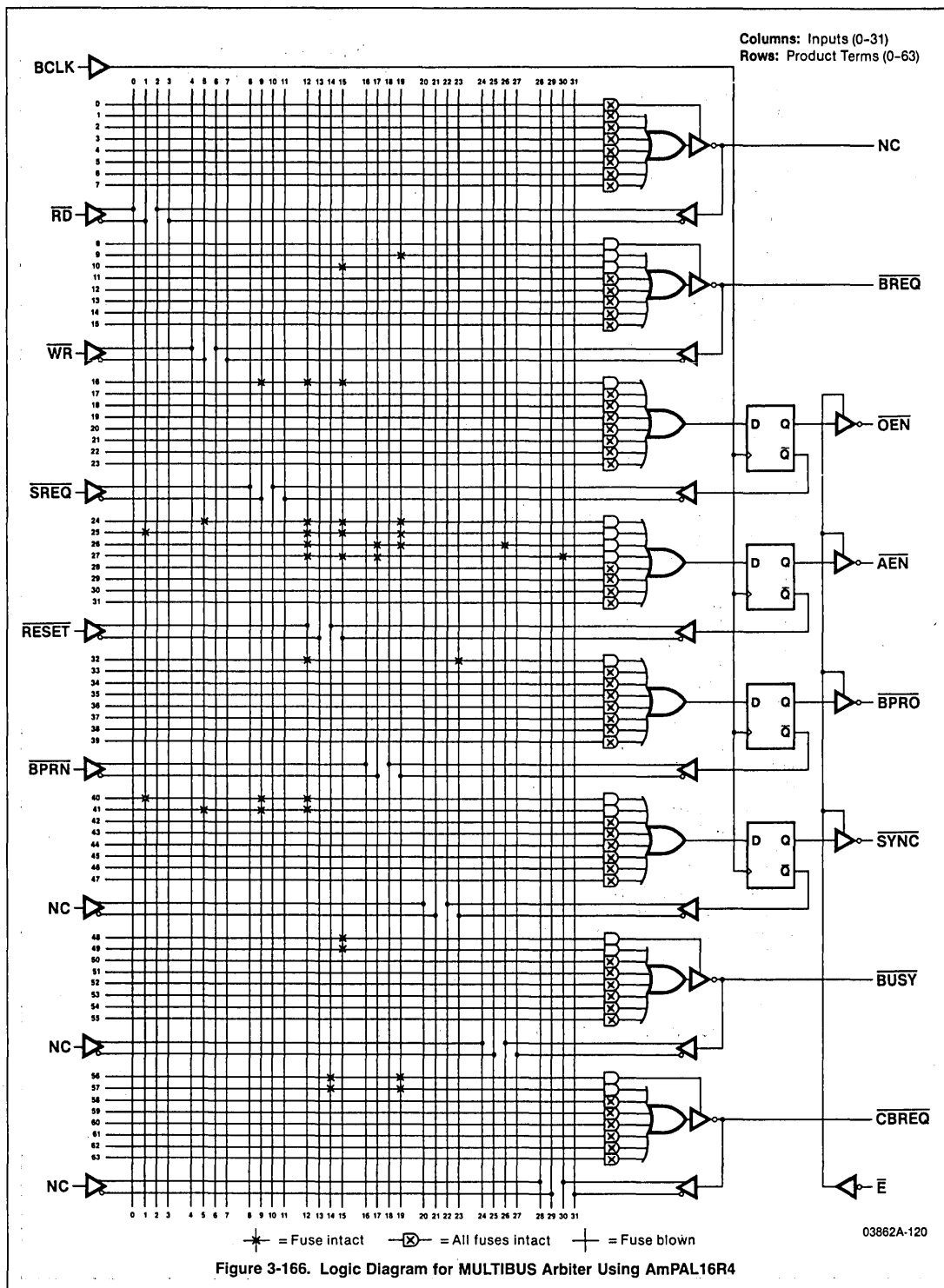
5484

PAL DESIGN SPECIFICATION

MARK S. YOUNG 6/22/82

3

Figure 3-165. Fuse Plot and Test Vectors for the MULTIBUS Arbiter



3.5.4 VME BUS CONTROL SIMPLIFIED WITH PLDS

A High-Performance Bus

The VMEbus is a high-performance asynchronous bus capable of supporting multiple 16- and 32-bit processors. Its asynchronous nature permits easy implementation of hardware controllers that reduce the time required for bus control tasks such as bus arbitration. By offloading these tasks to hardware, bus arbitration time is minimized which consequently maximizes the overall bus transfer rate.

This article describes PLD implementations of two such controllers in a typical VMEbus system: the bus arbiter and the interrupter that comply with the VMEbus protocol. These state machines use to require multi-chip designs consisting of se-

quencers, microprogrammed control stores, and other MSI/LSI chips. By using the AmPAL22V10 programmable logic device (PLD), you can reduce substantially the number of chips needed for these state-machine implementations.

Functional Modules

A typical VMEbus-based computer system contains a bus arbiter and an interrupt handler board. A processor/master initiates a data transfer to a slave by requesting control of the data bus from the bus arbiter. Once bus control has been granted to a master, control signals to and from the master and slave are exchanged which follow predefined protocols. These protocols guarantee an orderly transfer of data between the communicating modules. Interrupt servicing capability is provided by the interrupt handler board (see Figure 3-167)

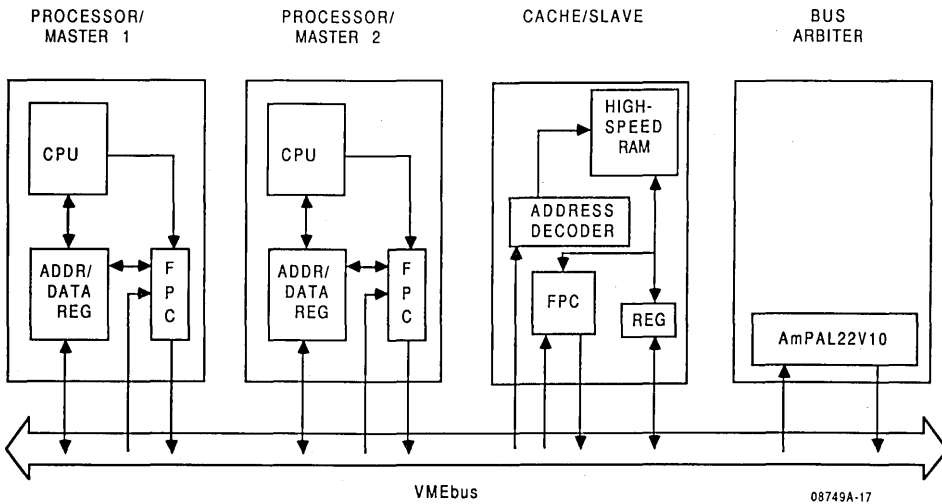


Figure 3-167. Intermodule Communication Through Am29PL141 (FPCs) and PLDs in a VMEbus System

Steps in Designing the Bus Controllers

VMEbus protocols outline the steps to perform any bus-related operation, such as transferring data over the bus between two modules. These protocols are described using flow diagrams with interactions between the communicating modules shown through various interface signals. For example, the priority bus arbiter flow diagram (Figure 3-168) shows how bus requests between two modules using the same request line are resolved.

The flow diagrams are analyzed to pick out the functions that can be incorporated into PLDs, and then state machines are created for these functions. These state machines can be described logically, or with flowchart symbols such as rectangular blocks (to symbolize events or control signals) and decision diamonds (to determine control program flow). Logic equations are then written for these state machines.

This methodical process is used in designing the two PLD-based VMEbus controllers: the bus arbiter and interrupt handler controller.

A PLD Bus Arbiter

Before any module can perform a data transfer, it must request control of the data transfer bus from the bus arbiter. This design uses a priority option bus arbiter implemented on a single AmPAL22V10 that monitors the four bus request lines BR0-3(L), with BR3(L) assigned the highest priority.

Based on the flow diagram (Figure 3-168), the arbiter grants the bus to the requesting module using the highest active request line, in this case BR1(L). A bus grant signal is daisy-chained to all the devices using this request line to resolve simultaneous bus requests from two or more modules. This dictates that the arbiter be in the first slot of the VMEbus system, and that the module physically closest to the arbiter through the daisy-chain will get the bus grant.

All bus requests are processed in parallel by the AND-OR array of the PLD. This allows priority arbitration to occur in a single clock cycle, which minimizes arbitration time and maximizes the data transfer rate on the VMEbus.

When the bus is free, the module using the highest active bus request line is granted the bus. This is described with Boolean logic notation where registered outputs are defined with ':=':

<pre> IF (/BBSY*(BR0+BR1+BR2+ BR3)) THEN BEGIN IF (BR3) THEN BG3IN := 1 ; IF (/BR3*BR2) THEN BG2IN := 1 ; IF (/BR3*/BR2*BR1) THEN BG1IN := 1 ; IF (/BR3*/BR2*/BR1*BR0) THEN BG0IN := 1 ; END ; </pre>	if bus not busy and a request line is active if BR3 is active, grant bus to device on BR3 activate bus grant daisy chain 2 if BR1 is active and BR3 and BR2 are not, then grant bus to device using request line 1 BR0-3 and BG0IN to BG3IN are active low
--	---

The AmPAL22V10 can be programmed with normal or inverted outputs and the logic software can support active high or low input polarities.

BG1IN(L) is asserted until the requesting module asserts BBSY(L). This is defined in logic equation form as:

<pre> IF (/BBSY*BG1IN) THEN BG1IN := 1; </pre>	continue asserting BG1IN until BBSY becomes active
--	---

Priority Arbitration Options

It may be necessary for the arbiter to force the current bus master to relinquish bus control for certain conditions, such as if a higher priority operation must be attended to. This is done by asserting the bus clear signal BCLR(L) based on the priority of the bus current bus master and the bus clear conditions.

The PLD arbiter keeps track of the current bus master's priority by recording which bus request line was used to gain control of the bus. For example, two output registers called BUS_MASTER will be set to the binary value "2" if the current bus master used BR2(L) to gain control of the bus.

In this design, BCLR(L) will be activated under two conditions:

1. If BUS_MASTER is 2,1 or 0 and the active bus request line is 3 or 2
2. If BUS_MASTER is 0, and any bus request line is active.

When BUS_MASTER is 3, then the arbiter will not honor any bus requests until the bus busy line BBSY(L) is low. The above conditions are expressed logically as:

```

IF (BBSY*(BR0+BR1+BR2+BR3)) THEN
  BEGIN IF (BR3*(/(MASTER[1:0] = 3) )) THEN
    BCLR := 1; "assert BCLR if MASTER <> 3"
  IF (/BR3*BR2*(/(MASTER[1:0] = 3) )) THEN
    BCLR := 1;
  IF (MASTER[1:0] = 0) THEN
    BCLR := 1;
  END;

```

Once the BCLR(L) line is active, then it should be held until the current bus master releases BBSY(L). This is logically expressed as $BCLR = BCLR * BBSY$, or in a high-level syntax,

```

IF (BCLR*BBSY) THEN
  BCLR := 1;

```

The BCLR(L) signal is defined such that uninterruptible devices use BR3(L). Devices that can be temporarily suspended to accommodate interrupts and higher-priority operations are assigned to bus request line 0. The BCLR conditions will vary with your application, but any modifications will only require redefinitions of the high-level logic expressions.

The AmPAL22V10 is ideal as the bus arbiter because of the input/output signal requirements and the large number of product terms needed to logically define the bus grant and bus clear signals.

Processing Interrupts

When real-time or urgent response is needed by a processor, it generates an interrupt request signal and waits for the interrupt acknowledge signal IACK(L) and IACKIN(L) daisy-chain signal. A 3-bit value is then read from the VMEbus when the data strobes are active; this value indicates which interrupt request line was acknowledged. These 3-bits are decoded to determine if it matches the processor's request level.

If the interrupt is acknowledged, then the interrupting processor puts its status or ID byte on the data bus for the interrupt handler. This data is used by the interrupt handler as an interrupt vector. This processor can then wait in a loop until the IACK(L) signal from the interrupt handler is driven HIGH to signify interrupt service completion.

Interrupt Handling: The Interrupt Handler Preprocessor (IHP)

To handle external I/O or special system events (e.g., timeout, overflow), interrupts are supported on the VMEbus through an interrupt handler (IH) module.

The logic complexity of the IH (See VMEbus Architecture) is reduced by offloading some of the initial interrupt recognition tasks to a PLD. An AmPAL22V10 can be programmed as an IHP to preprocess interrupt requests, obtain the bus and handle data transfers (such as interrupt acknowledge signals). Control is passed to the interrupt handler only when the IHP latches the interrupt vector.

```
IF (BBSY) THEN
  BEGIN
    (1) IF (IR7) THEN
      INTR[2:0] := 7 ;
    IF (/IR7*IR6) THEN
      INTR[2:0] := 6 ;
    IF (/IR7*/IR6*IR5) THEN
      INTR[2:0] := 5 ;
      :
      :
      :
```

All interrupt request lines are monitored according to the following logic equation:

If any of the interrupt lines are active, then BR3(L) is asserted. This initiates the bus acquisition phase.

```

IF (BR3*/BG3IN) THEN      —|————— [A]
  BR3 := 1 ;
IF (BR3*BG3IN + BBSY*/SERVICE_DONE)
                                THEN ——— [B]
  BBSY := 1 ;

```

```

(2) IF (/IR7*/IR6*/IR5*/IR4*/IR3*/IR2*IR1) THEN
    INTR[2:0] := 1 ;      I acknowledge interrupt line I
END:

```

The remaining preprocessing involves completing the interrupt vector byte transfer. This is logically expressed as:

```

IF (BBSY) THEN
  BEGIN
    IACK := 1;
  (X) IF (DTACK) THEN
    LATCH_STATUS := 1;
  END;
  IF (IACK) THEN
    AS := 1;

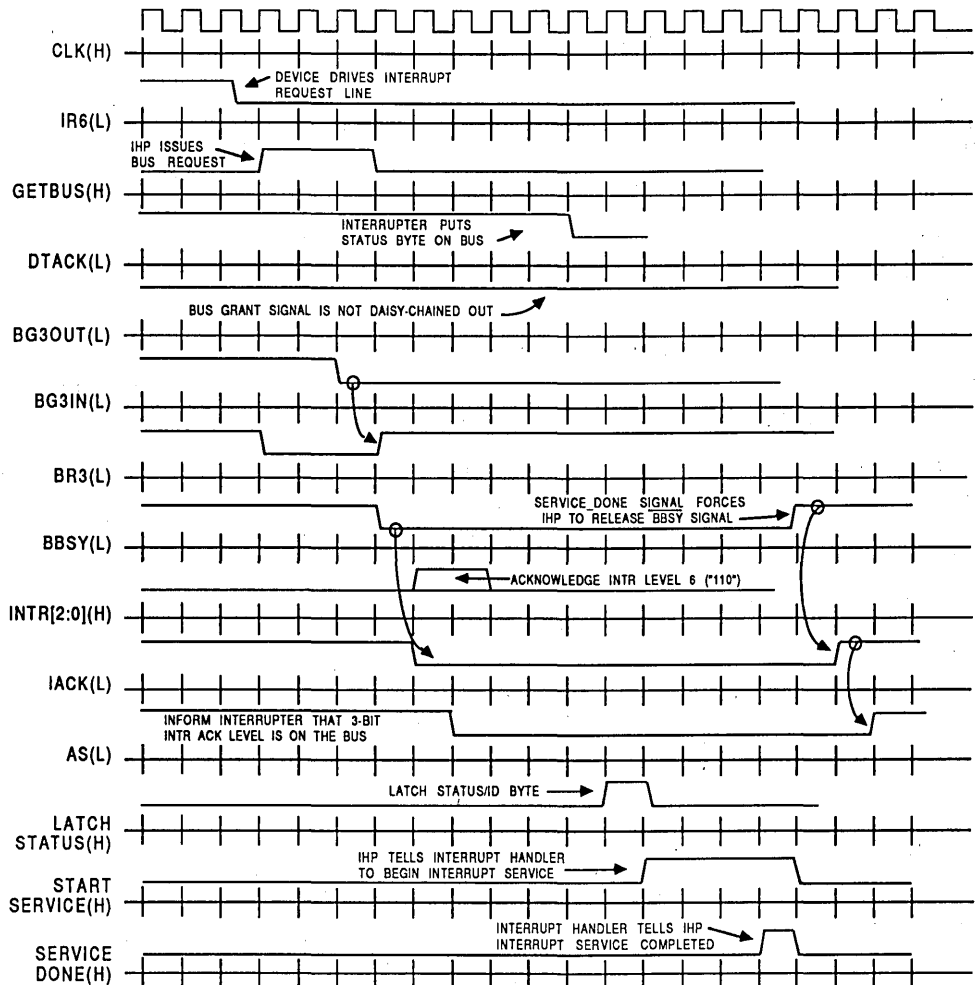
```

begin interrupt acknowledge
daisy chain; if device sends
data transfer acknowledge
(DTACK) signal, then latch
the status
assert the address strobe signal to inform
the interrupter that the interrupt
acknowledge level is ready

Once the 8-bit status or ID byte is transferred successfully via the DTACK(L) signal (X), the IHP PAL device informs the interrupt handler module to begin the interrupt service routine:

```
IF (BBSY*LATCH_STATUS +BBSY*START_SERVICE*
    /SERVICE_DONE) THEN
    START_SERVICE := 1;
```

In this design, the `START_SERVICE` signal is constantly asserted until the interrupt handler generates a `SERVICE_DONE` signal, at which point `START_SERVICE` is brought LOW. The above equations will generate the timing diagram in Figure 3-169.



08749A-19

Figure 3-169. Timing Diagram

Controller Design Simplified with Development Tools

The task of programming FPCs and PLDs as VMEbus controllers is simplified with the development tools. The designer

need only analyze the bus protocols, convert these into state machines and then write assembly language programs or high-level logic equations to describe these state machines. The assembler and logic software will then process these programs and equations to fit into the FPC or PLD.

DEVICE ARBITER (AMPAL22V10) " VMEbus Priority Arbiter Option
 For this application, if the bus is busy, then bus requests will be deferred until the BBSY(L) (bus busy line active LOW) goes HIGH, unless the bus request is of a higher priority. In this case, the bus clear signal BCLR(L) is asserted and the current bus master relinquishes control. If the bus is free and 2 or more masters request service, then the bus is granted to the master with the higher priority request line. "

```
PIN CLK      = 1
/BBSY        = 2
/BRO         = 3
/BR1         = 4
/BR2         = 5
/BR3         = 6

/BCLR        = 23
/BGOIN       = 22
/BG1IN       = 21
/BG2IN       = 20
/BG3IN       = 19
MASTER[1:0] = 18,17;
```

BEGIN

"if a bus request occurred and bus = free, then activate the bus grant in line corresponding to the highest priority bus request line "

```
IF (/BBSY*(BRO+BR1+BR2+BR3)) THEN
  BEGIN
    IF (BR3) THEN
      BG3IN := 1 ;
    IF (/BR3*BR2) THEN
      BG2IN := 1 ;
    IF (/BR3*/BR2*BR1) THEN
      BG1IN := 1 ;
    IF (/BR3*/BR2*/BR1*BRO) THEN
      BGOIN := 1 ;
  END ;
```

" while the bus grant in line is active and the bus is still busy, then latch the bus grant signals until BBSY(L) becomes active"

```
IF (BG3IN*/BBSY) THEN
  BG3IN := 1 ;
IF (BG2IN*/BBSY) THEN
  BG2IN := 1 ;
IF (BG1IN*/BBSY) THEN
  BG1IN := 1 ;
IF (BGOIN*/BBSY) THEN
  BGOIN := 1 ;
```

```
IF (BG3IN*BBSY) THEN " when requester responds with BBSY active, then "
  MASTER[1:0] := 3 ; " MASTER is set to the line currently controlling"
IF (BG2IN*BBSY) THEN " the bus; this is used internally for future "
  MASTER[1:0] := 2 ; " bus priority resolution "
IF (BG1IN*BBSY) THEN
  MASTER[1:0] := 1 ;
IF (BGOIN*BBSY) THEN
  MASTER[1:0] := 0 ;
```

```
IF (BBSY) THEN " when bus busy, then remember current bus master line "
  MASTER[1:0] := MASTER[1:0] ;
```

"if a higher priority bus request occurs and the bus is busy, then begin bus resolution (by using bus clear BCLR) under the following conditions :

- if BR3 and present bus master is not using bus line 3; i.e., once a device uses BR3, then no one can force it to relinquish the bus or
- if BR2 and present bus master is not 3; i.e., BR2 can assert bus clear except when the present bus master obtained the bus using bus line 3 or
- if BR1 and the bus master obtained the bus using bus request line 0, then assert BCLR

==> if BRO is activated when the bus is busy, then do not assert BCLR; BRO devices cannot force anyone off the bus. This is one possible priority implementation.

```
"
IF (BBSY*(BRO+BR1+BR2+BR3)) THEN "if a bus request occurred and bus = busy"
  BEGIN
    IF (BR3*(/MASTER[1]+/MASTER[0])) THEN "if BR3 and present master is not "
      BEGIN
        BCLR := 1 ; " using line 3, then assert BCLR "
        IF (BCLR*BBSY) THEN
          BCLR := 1;
        END;
```

Figure 3-170. PLPL Specifications for the Example of Figure 3-167 (1 of 4)


```

IF (/BR3*BR2*(/MASTER[1]+/MASTER[0])) THEN
  BEGIN
    BCLR := 1 ;
    IF (BCLR*BBSY) THEN
      BCLR := 1;
    END;
  IF (/BR3*/BR2*BR1*/MASTER[1]*/MASTER[0]) THEN
    BEGIN
      BCLR := 1 ;
      IF (BCLR*BBSY) THEN
        BCLR := 1;
      END;
    END ;
  END.
"Test vectors used for simulation and testing"
TEST_VECTORS
IN  CLK , /BBSY , /BR0 , /BR1 , /BR2 , /BR3 ;
OUT /BCLR , /BG0IN , /BG1IN , /BG2IN , /BG3IN , MASTER[1:0] ;
BEGIN
  "  / / / / /  / / / / /  "
  " c b b b b b  b b b b b m "
  " l s r r r r  c g g g g s "
  " k y 0 1 2 3  r 0 1 2 3 1 0 "
  "-----"
  C 1 0 1 1 1  X X X X X X X ;
  C 0 1 1 1 1  X X X X X X X ;
  C 1 1 1 1 1  X X X X X X X ;
  C 1 1 0 0 1  X X X X X X X ;
  C 0 1 1 1 1  X X X X X X X ;
  C 0 1 1 1 0  X X X X X X X ;
  C 0 1 1 1 0  X X X X X X X ;
  C 0 1 1 1 0  X X X X X X X ;
  C 1 1 1 1 0  X X X X X X X ;
  C 1 1 1 1 0  X X X X X X X ;
  C 1 1 1 1 0  X X X X X X X ;
  C 0 1 1 1 1  X X X X X X X ;
  C 0 0 1 1 1  X X X X X X X ;
  C 0 0 1 1 1  X X X X X X X ;
  C 0 1 0 1 1  X X X X X X X ;
  C 0 1 1 0 1  X X X X X X X ;
  C 0 1 1 1 0  X X X X X X X ;
END.

```

Figure 3-170. PLPL Specifications for the Example of Figure 3-167 (2 of 4)

DEVICE INTR_HND_CTRLR (AMPAL22V10) " VMEbus interrupt handler preprocessor

This PAL serves as an interface between the interrupt servicing logic in the interrupt handler and the bus arbiter: when an interrupt is detected, the PAL requests the bus from the arbiter. When the bus is granted, the bus busy BBSY(L) is asserted. The 3 bit code corresponding to the highest priority interrupt request is put on the bus, and both IACK(L) and AS(L) are also asserted. The PAL waits for DTACK(L) LOW and then latches the status/ID byte. The START_SERVICE signal informs the interrupt handler to begin the interrupt service sequence with the status/ID byte latched in.

Note : this interrupt handler interface PAL handles all interrupts using bus request line BR3; this is application-dependent "

PIN

```
CLK      = 1
/IR1     = 2      "interrupt request lines 1 to highest priority 7"
/IR2     = 3
/IR3     = 4
/IR4     = 5
/IR5     = 6
/IR6     = 7
/IR7     = 8
/DTACK   = 9      "data transfer ack; used by interrupter to indicate
                  status/ID data byte ready on bus "
/BG3IN   = 10     "bus grant in signal from arbiter"
SERVICE_DONE = 13 "signal from interrupt handler indicating
                  service complete"
LATCH_STATUS = 23 "latches data from bus for interrupt service routine"
/BBSY    = 22     "bus busy"
/IACK    = 21     "interrupt acknowledge"
/AS      = 20     "address strobe"
/BR3     = 19     "bus request signal to bus arbiter"
GET_BUS  = 18     "used internally by PAL "
INTR[2:0] = 15:17 "3-bit level code put on bus "
START_SERVICE = 14; "sent from PAL to intr handler "
```

BEGIN

"if any interrupt request line is active, then try getting the bus. All the interrupts handled by this PAL will use the bus request line BR3. A request can occur only when no other interrupt is being serviced (i.e., bus should not be busy and bus grant in should not be active) "

```
IF (/BBSY*/BG3IN*(IR1 + IR2 + IR3 + IR4 + IR5 + IR6 + IR7)) THEN
BEGIN
GET_BUS := 1 ;
BR3 := 1 ;      "subject to application"
END ;
```

"if interrupt handler wants the bus, the bus grant line is still inactive, and the interrupt request is still active, then continue requesting the bus "

```
IF (GET_BUS*BR3*/BG3IN*(IR1 + IR2 + IR3 + IR4 + IR5 + IR6 + IR7)) THEN
BEGIN
BR3 := 1 ;
GET_BUS := GET_BUS ;
END;
```

"if bus grant in line is active, then assert the bus busy signal"

```
IF (GET_BUS*BG3IN + BBSY*/SERVICE_DONE) THEN
BBSY := 1 ;
```

"if interrupt handler received bus, then acknowledge the interrupt"

```
IF (BBSY) THEN
BEGIN
IF (/START_SERVICE) THEN
IACK := 1 ;
IF (DTACK) THEN      "when the interrupter responds with DTACK(L), then "
LATCH_STATUS := 1 ;  "latch the status/ID byte into the interrupt handler"
```

```
IF.(IR7) THEN
INTR[2:0] := 7 ;      "IR7 line is highest priority"
IF (/IR7*IR6) THEN
INTR[2:0] := 6 ;
IF (/IR7*/IR6*IR5) THEN
INTR[2:0] := 5 ;
IF (/IR7*/IR6*/IR5*IR4) THEN
INTR[2:0] := 4 ;
IF (/IR7*/IR6*/IR5*/IR4*IR3) THEN
INTR[2:0] := 3 ;
IF (/IR7*/IR6*/IR5*/IR4*/IR3*IR2) THEN
INTR[2:0] := 2 ;
IF (/IR7*/IR6*/IR5*/IR4*/IR3*/IR2*IR1) THEN
INTR[2:0] := 1 ;      "IR1 line is lowest priority"
END;
```

Figure 3-170. PLPL Specifications for the Example of Figure 3-167 (3 of 4)

```

IF (IACK*/AS + IACK*/DTACK) THEN "hold the interrupt acknowledge level until"
  INTR[2:0] := INTR[2:0] ; "strobed by address strobe signal or until
                           the interrupter acknowledges the 3-bit
                           level with DTACK "

```

```

    "if the IACK line is active and interrupt servicing has not yet begun
    activate the address strobe line AS "

```

```

IF (IACK*/START_SERVICE) THEN
  AS := 1 ;

```

```

"send the start service signal until service is done"

```

```

IF (BBSY*LATCH_STATUS + BBSY*START_SERVICE*/SERVICE_DONE) THEN
  START_SERVICE := 1 ;

```

```

END.

```

```

TEST_VECTORS

```

```

IN  CLK , /IR1 , /IR2 , /IR3 , /IR4 , /IR5 , /IR6 , /IR7 ,
    /DTACK , /BG3IN , SERVICE_DONE;

```

```

OUT START_SERVICE , GET_BUS , INTR[2:0] , LATCH_STATUS ,
    /AS , /IACK , /BBSY , /BR3 ;

```

```

BEGIN

```

```

" C / / / / / / / / S | S G I I I L / / / / "
" L I I I I I I I I D B R | R E N N N T A I B B "
" K R R R R R R R T G V | V T T T T C S A B R "
" 1 2 3 4 5 6 7 A 3 D | C B R R R R H C S 3 "
"           C I N | E U 2 1 0   K Y "
"           K N E | S "

```

```

C 1 1 1 1 0 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 0 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 0 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 0 1   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 0   X X X X X X X X X X ;
C 1 1 1 1 1 1 1 1 1 1   X X X X X X X X X X ;

```

```

END.

```

Figure 3-170. PLPL Specifications for the Example of Figure 3-167 (4 of 4)

3.6 MISCELLANEOUS LOGIC FUNCTIONS

3.6.1 8088 TO Am2968 INTERFACE

Introduction

This application note describes the implementation of a timing generator which interfaces between the 8088 and the Am2968 Dynamic Memory Controller (DMC) using programmable logic and a delay line. The implementation does not account for any error detection and correction (EDC) circuitry.

The timing generator, like the Am2970, is needed in memory systems utilizing a dynamic memory controller. It serves as an interface between the processor and the controller and generates the necessary control signals for the controller.

A dynamic memory controller, in brief, interfaces between a processor and the dynamic memory array. It steers the appropriate address inputs and Row and Column Address Strokes (RAS and CAS) required in the selected memory operations (i.e. refreshing, read/write).

The following sections will introduce and illustrate the implementation of the timing generator using AMD's 20-pin PAL devices, and also show its interface to AMD's Am2968 (DMC) and Intel's 8088 processor. Figure 3-171 shows the block diagram of the interface. Programmable logic was chosen because it is readily available, simple to use and reduces the number of devices required to implement specific functions.

Interface Overview

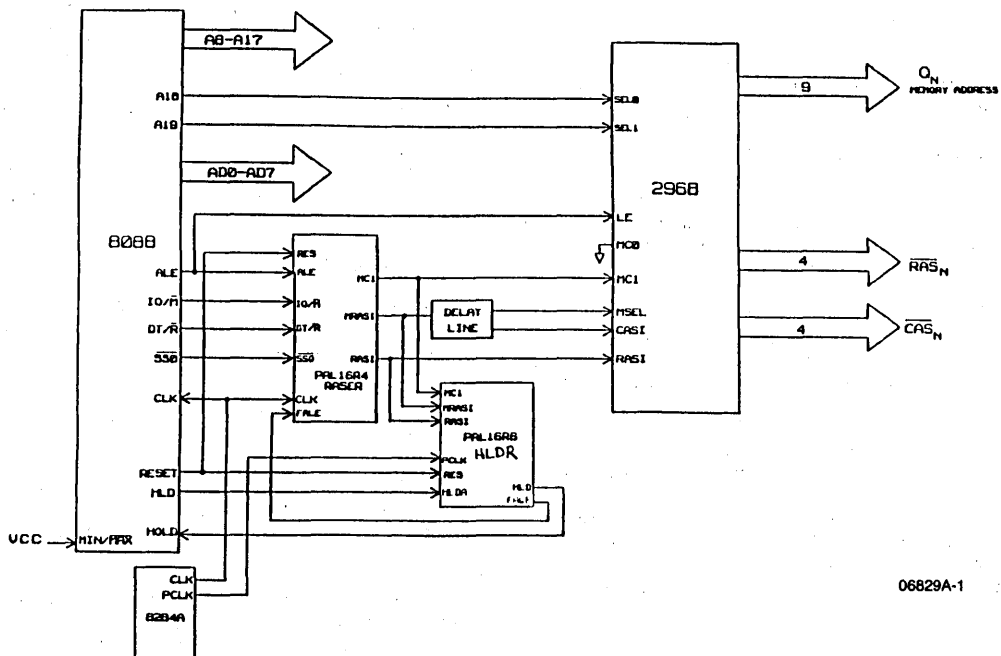
The implementation of this timing generator is not a general purpose application. It is dedicated specifically to a particular processor, the Intel 8088, and is limited to accessing only four

banks of memory. The range of the four banks of memory is therefore 1 Mbyte—with 256K in each bank. This limitation is set by the fact that no additional decoding circuitry has been added. Decoding is done with the two address lines (A18-A19) of the 8088, thus allowing only four banks to be selected.

The clock to the 8088 and the two PAL devices is provided by the 8284A. The CLK signal from the 8284A is connected directly to the CLK input of the 8088 and also to CLK of RASER (PAL16R4). The PCLK of the 8284A connects to the PCLK input of HLDR (PAL16R6) to operate the 6-bit counter. Note that the clock operates on a 1/3 and 2/3 duty cycle.

Description of PAL Device Function

The function of RASER (PAL16R4) is to create RASI (Row Address Strobe) and CASI (Column Address Strobe) at the appropriate time. RASI is generated directly from the device, whereas CASI is generated from MRASI via a delay line. Figure 3-172 shows the timing delays between the controller and processor interface signals. The two modes of operation which the timing is focused on is Refresh w/o Scrubbing and Read/Write. By toggling the state of the mode control pin MC1 to either a LOW or HIGH respectively, and with MC0 tied LOW (at the Am2968), the desired mode is generated. The mode control pin MC1 is generated based on the state of the processor's interface pins: $\overline{IO/\overline{M}}$, $\overline{DT/\overline{R}}$ and $\overline{SS0}$. The combination of these signals decodes the processor's current bus cycle and indicates the ongoing memory activity (for more detail see Intel's Microprocessor Handbook). The MUX Select pin (MSEL) is also generated from the delay line. It determines whether a Row or Column address is sent to the memory address input based on the mode control inputs, MC0 and MC1.



06829A-1

Figure 3-171. Am2968 to 8088 Interface Block Diagram

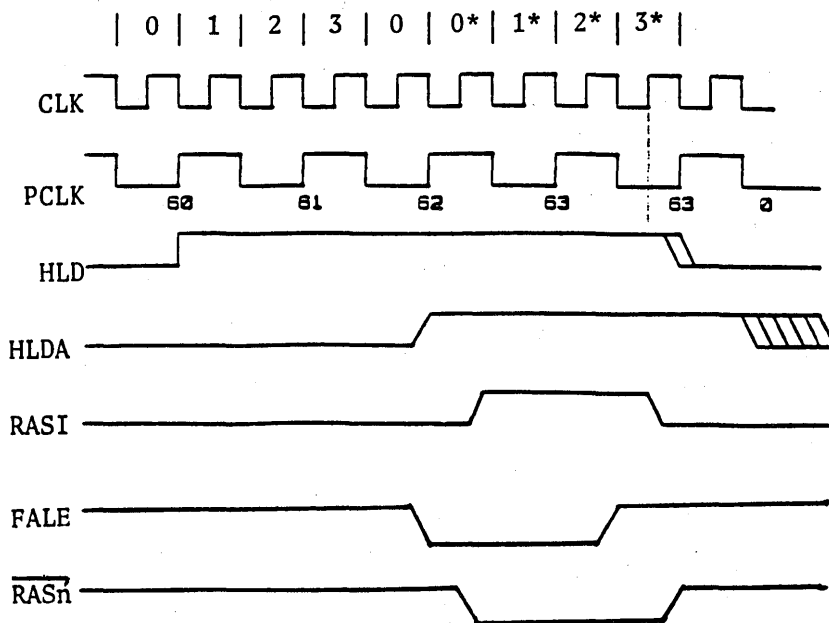


Figure 3-172. Interface Timing

06829A-2

A two-bit counter implemented internal to the (PAL16R4) RASER is used to keep track of the internal state of RASi during the Read/Write or Refresh operations. In a normal memory Read/Write operation, the appropriate $\overline{\text{RASn}}$ outputs from the DMC will be activated in response to RASi going HIGH, as shown in the memory timing in Figure 3-175, during this time ALE initiates the cycle. In Refresh mode, receiving RASi will force all the $\overline{\text{RASn}}$ outputs of the DMC to go Low. FALE initiates the refresh cycle, since no ALE occurs during this time, as shown in refresh timing of Figure 3-175. The counter is also configured such that, during memory operations, sufficient time has been allotted within the cycle for $\overline{\text{RAS}}$ precharge (required in DRAMS) to occur.

b) HLDR (PAL16R6)

The function of the (PAL16R6) HLDR is to generate the two signals—FALE (False Address Latch Enable) and HLD (Hold). FALE controls the internal latches (ALS and ALR) of the (PAL16R4) RASER and initiates the refresh cycle independent of the processor's ALE (Address Latch Enable) signal.

The (PAL16R6) HLDR is essentially a 6-bit counter that generates 64 T-Clock cycles for the Intel 8088 processor. When a hold request (HLD) is made and a hold acknowledge (HLDA) is received from the processor, FALE will be generated for the (PAL16R4) RASER to initiate the refresh cycle. Refresh is performed every 8 μs which is double the required frequency. This time is derived in the Refresh Calculations section of this application note. The 8088 processor allows request only at the end of the bus cycle. A bus cycle duration is 4 T-Clocks, thus refresh is performed at cycles "60-63" as noted in the function table in the HLDR design specification. Figure 3-172 refresh timing shows this critical portion of the timing activity when RASi is generated for refresh. During CLK (0 - 0*), (this corresponds to PCLK count value 62), the processor acknowledge is received and RASi becomes active. While RASi is HIGH, the internal 2-bit counter will cycle through its count sequence (0* - 3*); at the end of the sequence as indicated by PCLK count value 63 (i.e., CLK 3*), RASi will become inactive.

PAL16R4
RASER
VER .001

2968 TO 8088 I/F
G.R. SPEARS 7/6/84
AMD BELLEVUE.WA

CLK RES ALE FALE IOM DTR SS0 CLK2 NU2 GND
TRI MRASI RASI /MC1 NC /B /A /ALR /ALS VCC

ALS = FALE*/ALE*/ALR + RES

ALR = /A*B*/ALS + A*/ALS

/RASI = /A*/B*/RES + RES*CLK2

/MRASI = IOM*/DTR + IOM*DTR + /IOM*DTR*SS0 +
/A*/B

A := /A*/RES*/ALS + /A*/RES*ALE

B := A*/B*/RES*/ALS + /A*B*/RES*/ALS

MC1 := IOM*/DTR*ALE*/RES + DTR*SS0*ALE*/RES +
IOM*DTR*/SS0*ALE*/RES + MC1*/ALE*/RES

FUNCTION TABLE

CLK	CLK2	ALE	FALE	IOM	DTR	SS0	MRASI	RASI	/MC1	/ALR	/ALS	/B	/A	RES
;C	C	A	F	I	D	S	M	R	/	/	/	/	/	R
;L	L	L	A	O	T	S	R	A	M	A	A	B	A	E
;K	K	E	L	M	R	O	A	S	C	L	L			S
;2	2	E	E				S	I	1	R	S			
							I							

C	X	X	X	L	L	L	X	X	H	X	X	H	H	H ;M
C	X	X	X	L	L	L	X	X	H	X	X	H	H	L ;
H	X	L	H	L	L	L	L	L	H	H	L	H	H	L ;E
L	X	L	H	L	L	L	L	L	H	H	L	H	H	L ;M
L	X	H	H	L	L	L	L	L	H	H	H	H	H	L ;
C	X	H	H	L	L	L	H	H	H	L	H	H	L	L ;A
H	X	L	H	L	L	L	H	H	H	L	H	H	L	L ;C
L	X	L	H	L	L	L	H	H	H	L	H	H	L	L ;C
C	X	L	H	L	L	L	H	H	H	L	H	L	H	L ;F
H	X	L	H	L	L	L	H	H	H	L	H	L	H	L ;S
L	X	L	H	L	L	L	H	H	H	L	H	L	H	L ;S
C	X	L	H	L	L	L	H	H	H	L	H	L	L	L ;
H	X	L	H	L	L	L	H	H	H	L	H	L	L	L ;
L	X	L	H	L	L	L	H	H	H	L	H	L	L	L ;
C	X	L	H	H	H	L	L	L	H	H	L	H	H	L ;
H	X	L	H	H	H	L	L	L	H	H	L	H	H	L ;
L	X	L	H	H	H	L	L	L	H	H	L	H	H	L ;
C	X	L	H	H	H	L	L	L	H	L	H	H	H	L ;
H	X	L	H	H	H	L	L	L	L	L	H	H	H	L ;
L	X	L	H	H	H	L	L	L	L	L	H	H	H	L ;
C	X	L	H	H	H	L	L	L	L	L	H	L	H	L ;
H	X	L	H	H	H	L	L	L	L	L	H	L	H	L ;

Figure 3-173. Source Listing for the 8088 to Am2968 Interface (AmPAL16R4)

L	X	L	H	H	H	L	L	H	L	L	H	L	H	L	S
C	X	L	H	H	H	L	L	H	L	L	H	L	L	L	;
L	X	L	H	H	H	L	L	H	L	L	H	L	L	L	;
C	X	L	H	H	H	L	L	L	L	L	H	L	L	L	;
H	X	L	H	H	H	X	X	X	X	X	L	H	H	L	;
L	X	H	H	X	X	X	X	X	X	H	H	H	H	L	;
C	X	H	H	X	X	X	X	X	X	L	H	H	L	L	;
H	X	L	H	X	X	X	X	X	X	L	H	H	L	L	;
L	X	L	H	X	X	X	X	X	X	L	H	H	L	L	;

DESCRIPTION

END

Figure 3-173. Source Listing for the 8088 to Am2968 Interface (AmPAL16R4) (Continued)

PAL16R6
HLDR
VER .000

2968 TO 8088 I/F
G.R. SPEARS 7/6/84
AMD BELLEVUE.WA

PCLK RASI MRASI HLDA RES CLK MC1 NU2 NU3 GND
TRI /FALE /F /E /D /C /B /A HLD VCC

A: = /A*/RES*MC1

B: = A*/B*/RES*MC1 + /A*B*/RES*MC1

C: = A*B*/C*/RES*MC1 + /B*C*/RES*MC1 + /A*C*/RES*MC1

D: = A*B*C*/D*/RES*MC1 + /C*D*/RES*MC1 + /B*D*/RES*MC1 +
/A*D*/RES*MC1

E: = A*B*C*D*/E*/RES*MC1 + /D*E*/RES*MC1 + /A*E*/RES*MC1 +
/B*E*/RES*MC1 + /C*E*/RES*MC1

F: = /A*F*/RES*MC1 + /C*F*/RES*MC1 + /B*F*/RES*MC1 +
/D*F*/RES*MC1 + A*B*C*D*E*/F*/RES*MC1 + /E*F*/RES*MC1

/HLD = /C + /D + /E + /F

FALE = HLDA*CLK*RASI*/MRASI

FUNCTION TABLE

PCLK	HLDA	RES	MC1	/F	/E	/D	/C	/B	/A	HLDA	CLK	RASI	MRASI	/FALE
/														
C	X	H	H	H	H	H	H	H	H	L	X	L	L	H ; 0
H	L	L	H	H	H	H	H	H	H	L	X	L	L	H
C	L	L	H	H	H	H	H	H	L	L	X	L	L	H
C	L	L	H	H	H	H	H	L	H	L	X	L	L	H
C	H	L	H	L	L	L	L	H	H	L	X	L	L	H ; 60
C	H	L	H	L	L	L	L	H	L	L	X	L	L	H ; 61
C	H	L	H	L	L	L	L	L	H	H	H	H	L	L ; 62
C	H	L	H	L	L	L	L	L	L	H	L	L	L	H ; 63
C	L	L	H	H	H	H	H	H	H	L	X	L	L	H ; 0

DESCRIPTION

Figure 3-174. Source Listing for the 8088 to Am2968 Interface (AmPAL16R6)

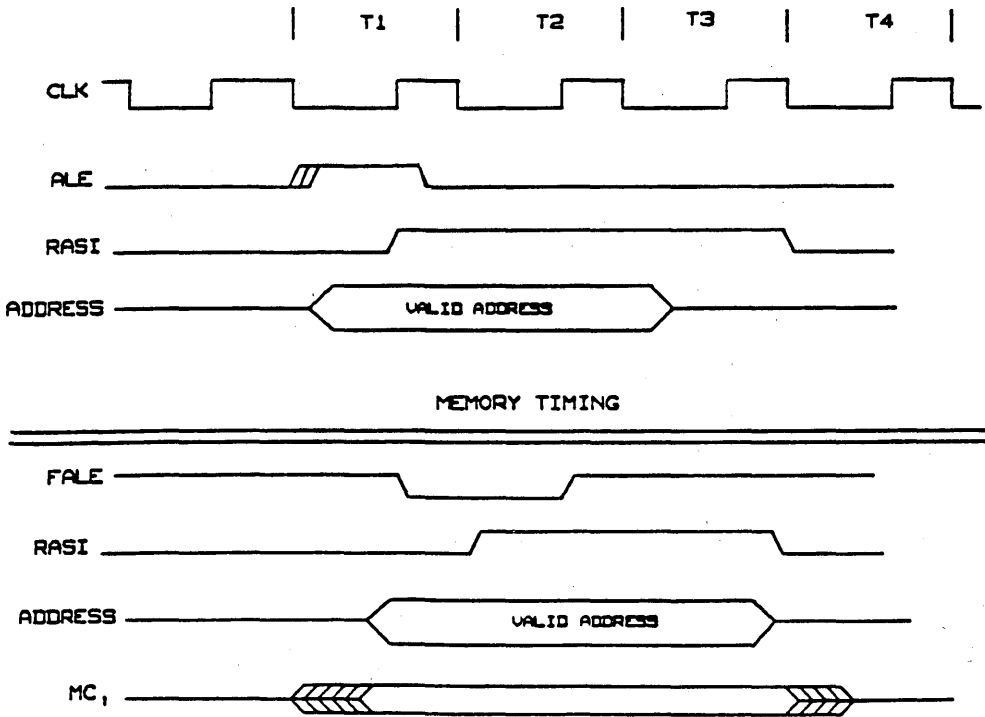


Figure 3-175. Refresh Timing

06829A-3

Refresh Timing Calculations

The timing calculations are based on Intel's 8088 processor being in minimum mode (8 MHz) and a cell array of 128 rows.

The 8088 clock period is 8 MHz, thus

$$t = 1/f = 1/8 \text{ M} = 125 \text{ ns.}$$

Refreshing for DRAMs are performed on 128 rows every 2 ms. The calculations for the above memory array is determined to be:

$$\text{Required refresh time} = 2 \text{ ms} / 128 = 15.6 \text{ } \mu\text{s.}$$

Thus the required time for 256 row cell array will be:

$$\text{Required refresh time} = 4 \text{ ms} / 256 = 15.6 \text{ } \mu\text{s.}$$

From the above calculations, refreshing needs to be performed at least every 15 μs . But for this application, memory is being refreshed every 8 μs as shown below:

$$\text{Refresh time} = 64 \text{ T-clock cycles} \times 125 \text{ ns} = 8 \text{ } \mu\text{s.}$$

Delay Line Taps Computation

The calculations for the assignment of the delay line taps is based on the parameters CASI and MSEL relative to RAS making a transition to the active state. See the timing in Figure 3-175.

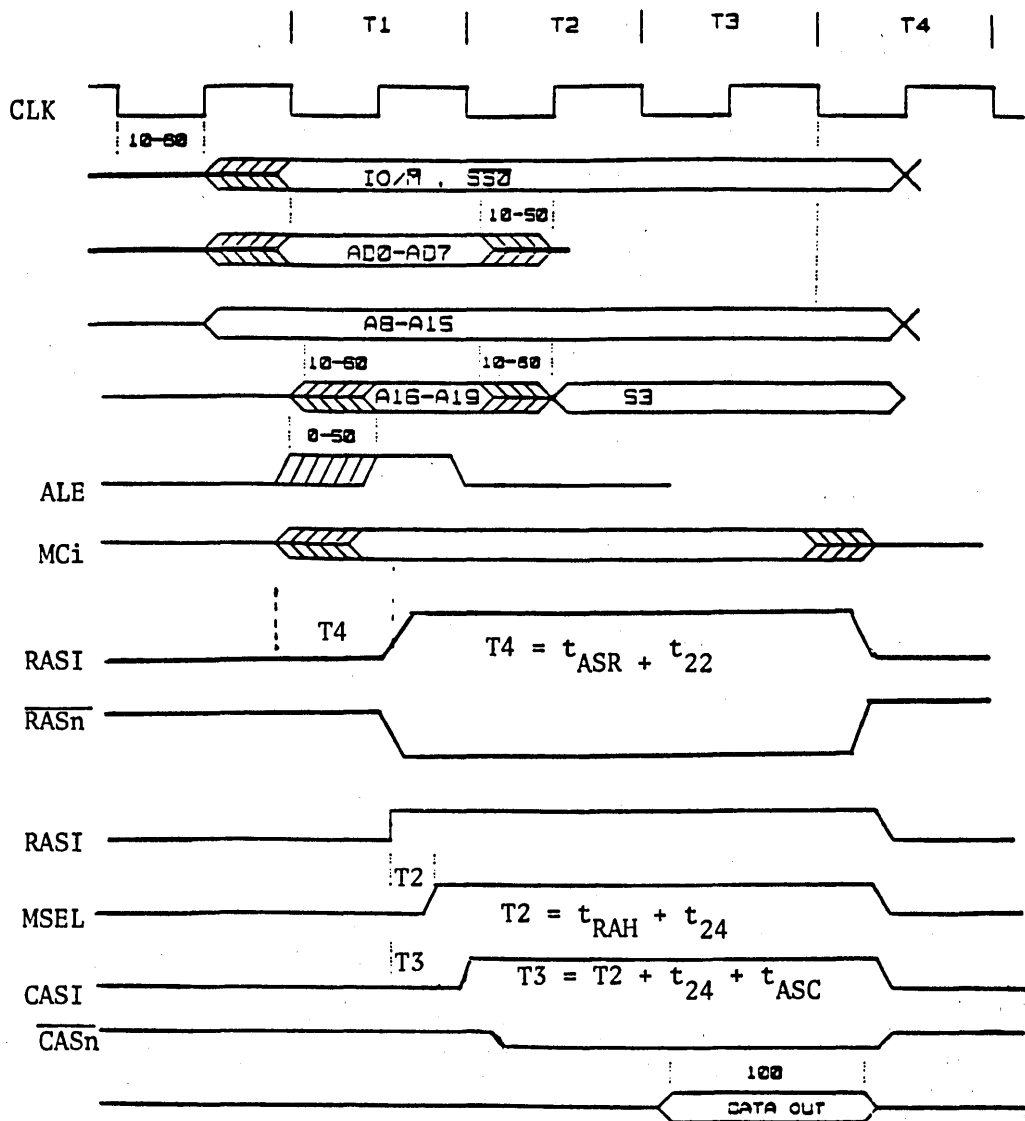


Figure 3-176. Delay Line Timing Diagram

MSEL may switch some time after T_2 . T_2 is calculated to be as follows:

$$T_2 = t_{RAH} + t_{24} = 20 + 11 = 31 \text{ ns.}$$

where T_2 = delay time from RASI to MSEL

t_{RAH} = Row Address Hold Time of the DRAM

t_{24} = skew time for Q_n to $\overline{RAS}_n$ of the DMC

Thus, after 31 ns MSEL should initiate its active transition.

CASI is calculated to make its change; T_3 ns relative to RASI. The calculated value is as follows:

$$T_3 = T_2 + t_{25} + t_{ASC} = 31 + 33 + 0 = 64.$$

where T_3 = delay from RASI to CASI

t_{25} = skew time for Q_n to $\overline{CAS}_n$ of the DMC

t_{ASC} = column setup time of the DRAM

Thus, 64 ns after RASI initiates its transition, CASI may begin to go active.

For simplicity, delay line taps may be assigned as follows:

RASI ----- MSEL ~ 30 ns

RASI ----- CASI ~ 60 ns

Counter and Mode Function Table

The following function tables describe the activity of the RASI/MRASI during the specific modes of the processor's interface signals ($\overline{IO/M}$, $\overline{DT/R}$ and $\overline{SS0}$) and the state of the internal counter which controls RASI/MRASI. Note: RASI will be active during the counter sequences between 1 to 3 as shown below.

3.6.2 A GENERAL-PURPOSE INTERFACE FOR THE Am2968

Introduction

This application note describes a general-purpose timing generator that simulates the asynchronous operation of the Am2970 and illustrates the interface of the Am2968 (successor to the Am2964) Dynamic Memory Controller (DMC) to various processors. The timing generator interfaces between the DMC and the processor by generating signals such as RASI (Row Address Strobe), CASI (Column Address Strobe), and WE (Write Enable). It consists of a PAL device, a timer, and a delay line.

The Am2968 (DMC) is a controller which interfaces between a processor and the dynamic memory array. It utilizes a timing generator to initiate refresh or memory cycles. When the DMC receives the correct control signal from the timing generator, it generates the appropriate address outputs (Q_n), and Row and Column Address Strobes ($\overline{RAS}_n$ and $\overline{CAS}_n$) based on the input control signals from the processor. These signals are needed to perform operations such as memory read/write and refresh.

The first section illustrates the general implementation of the timing generator using AMD's 20-pin PAL device (AmPAL16L8) and a commercially available delay line and timer (Figure 3-177). Programmable logic was selected because it is readily available, simple to use, and requires only a single device to implement the desired functions. The succeeding application note illustrates the interface of this timing generator to Motorola's MC68000 processor.

TABLE 3-22.

TWO BIT COUNTER

A	B	RASI
0	0	0
.	.	.
0	0	0
1	0	1
0	1	1
1	1	1

TABLE 3-23.

TWO BIT COUNTER

Sequence	$\overline{IO/M}$	$\overline{DT/R}$	$\overline{SS0}$	MRASI	COMMENTS
0	1	0	0 -----	0 -----	IACK
0	1	0	1 -----	0 -----	IOR
0	1	1	0 -----	0 -----	IOW
0	1	1	1 -----	0 -----	HALT
0	0	1	1 -----	0 -----	PASSIVE
1 3 -----	0	0	0 -----	1 -----	CODE ACCESS
1 3 -----	0	0	1 -----	1 -----	RD
1 3 -----	0	1	0 -----	1 -----	WR

Note: This design does not allow the user to insert Wait States during memory operations! Wait States may be inserted in interrupt or I/O cycles.



06829A-5

Interface Overview

The implementation of this timing generator is a general-purpose solution. The timing generator can be interfaced to various processors by simply asserting the proper control signals from the particular processor. The required interface signals are CS (Chip Select), R/W (Read/Write), B/W (Byte/Word), DS (Data Strobe) and CYCREQ (Cycle Request); see Figure 3-178. As an example, for the MC68000, (see MC68000 Interface), AS (address strobe) from the 68000 replaces CYCREQ, and UDS (Upper Data Strobe) & LDS (Lower Data Strobe) replaces DS and B/W.

The design emulates a subset of the Am2970 (Memory Timing Controller) operating in asynchronous mode, asserting no Wait States for memory accesses, and the 20-pin programmable logic device (*AmPAL16L8*) has $t_{pd} = 15 \text{ ns}$ (B-speed). B-speed devices were chosen to guarantee that the required interface signals would meet the setup time required by the memory device. The application note is based on three main cycles: memory, refresh, and memory/refresh. The memory/refresh cycle allows priority servicing of refresh over memory operation if both memory and refresh requests are generated simultaneously.

The refresh timer, implemented with a 555 timer, needs to generate a $\overline{FR}$ (Forced Refresh) signal which is negative-edge-triggered every 15 μs . This means that a refresh will

occur when a low-going edge on the $\overline{\text{FR}}$ signal is detected. As shown in Figure 3-179, with $\overline{\text{FR}}$ HIGH, $\overline{\text{FRH}}$ (Forced Refresh High) will be set. When $\overline{\text{FR}}$ detects a LOW-going edge, this will cause $\overline{\text{RRFQ}}$ (refresh request) to go LOW. With $\overline{\text{RRFQ}}$ active, $\overline{\text{RAS}}$ will also become active. The 15 μs value is derived from the dynamic memory refresh timing requirements; refresh is normally required over 256 rows every 4 ms. This equates to a required refresh cycle every 15.6 μs . In this application, the refresh timer generates $\overline{\text{FR}}$ approximately every 9.8 μs . For details see Refresh Timer Calculations.

The AmPAL16L8 generates the RAS_I output based on either a cycle request from the processor or a refresh request from the timer. The RAS_I is also an input to the delay line from which MSEL (MUX Select) and CAS_I are generated.

The dynamic memory array to be accessed is limited to only four banks. This is equivalent to one megaword of memory, assuming 256K DRAMs and only one controller is used. Extra banks of memory may be added to expand the amount of memory required for particular applications. To do this, additional controllers must be used. The selection of byte or word for memory write operations is provided by the two signals **WEH** (Write Enable HIGH) and **WEL** (Write Enable LOW). The upper and/or lower bytes may be written by activating either **WEH** or **WEL** as appropriate.

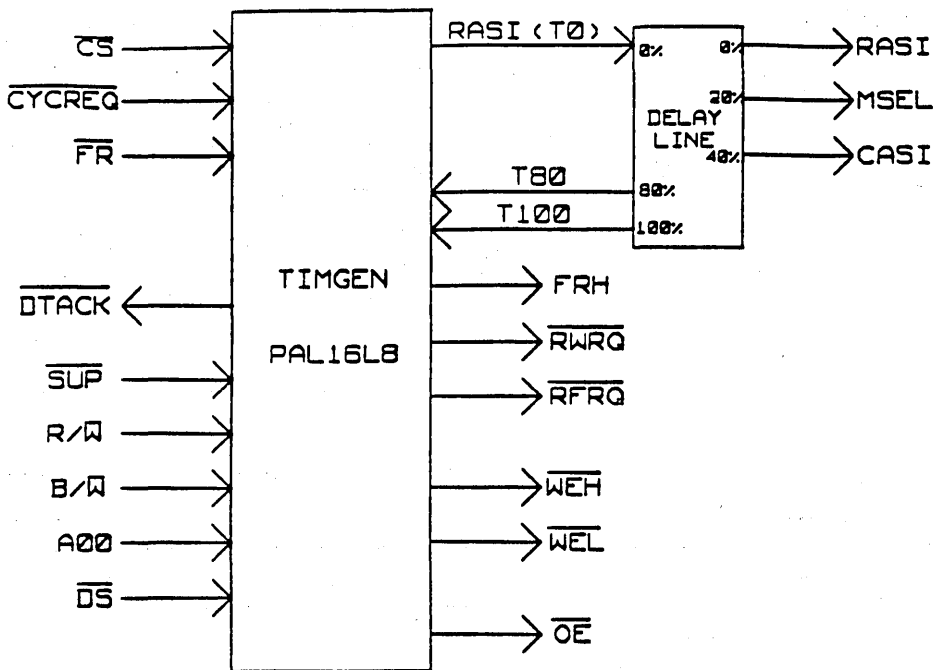


Figure 3-178. Interface Diagram

06829A-6

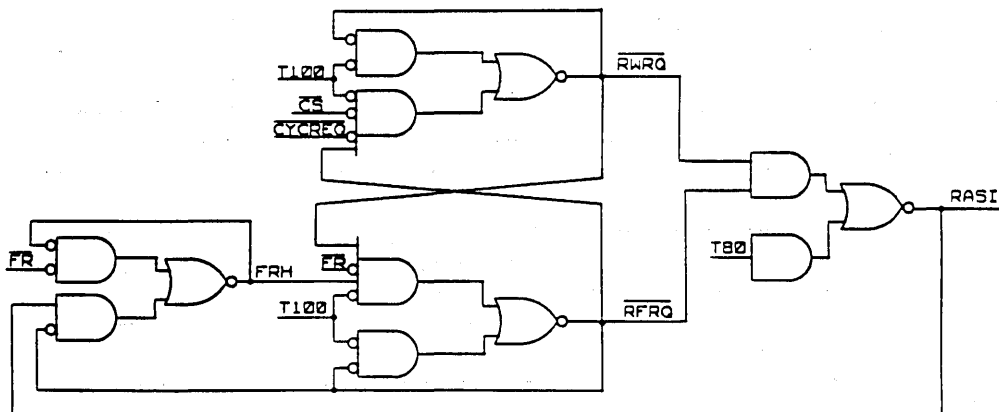


Figure 3-179. Arbitration Logic

06829A-7

TimGen (PAL16L8)

The function of TimGen (PAL16L8) is to generate RASI (Row Address Strobe), and also arbitrate between memory cycle and refresh cycle requests. The generation and arbitration logic shown in Figure 3-179 implements an S-R latch. With either a Memory Cycle Request (RWRQ) or a Refresh Cycle Request (RFRQ), RASI will be generated for the DMC so the appropriate memory control signals may be generated accordingly. The memory and Refresh timing diagram in Figure 3-183 illustrates the states of the interface signals for each of the cycles. But, if either one of the two cycle requests occurs in the middle of an existing cycle, the current cycle will be allowed to finish before the next request is acknowledged. Meanwhile, the latch illustrated in Figure 3-179, holds the new request in its

feedback loop. In the event that simultaneous requests are made by both the memory and refresh cycles, this latch arbitrates the request by allowing refresh to have higher priority.

The Arbitrated Refresh Cycle Timing 1 in Figure 3-180 notes the following: since the S-R latch is effectively edge-triggered, it waits for a HIGH on FR to set FRH (Force Refresh HIGH). Upon detecting a LOW-going level on the FR signal, refresh will occur and RASI will be asserted.

Because the design must allow for both memory byte and word operations, the two signals, WEH and WEL, are generated as selectors for writing either a byte or word during write operations. They are direct inputs to memory's WE (Write Enable) signal.

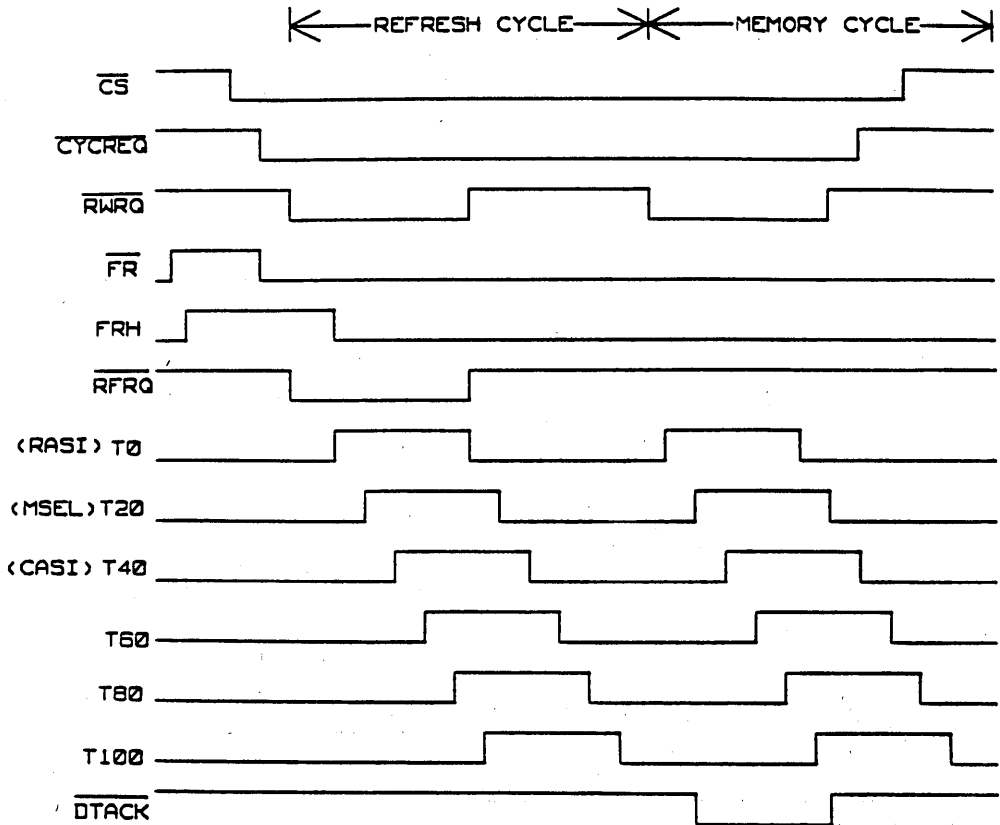


Figure 3-179. Arbitrated Refresh Cycle Timing 1

PAL16L8
TIMGEN
VER 1.00

GEN INTERFACE TO 2968
LEE/YEE 12/19/84
AMD

/CS /CYCREQ /FR /SUP RW BW A00 /DS T100 GND
T80 /DTACK TO FRH /RFRQ /RWRQ /WEL /WEH /OE VCC

/FRH = FR*/FRH + T0*RFRQ

RWRQ = CS*CYCREQ*/RFRQ*/T100 + RWRQ*/T100

RFRQ = FR*FRH*/RWRQ*/T100 + RFRQ*/T100

/T0 = /RWRQ*/RFRQ + T80

IF (CS*RWRQ*/RFRQ*T80) DTACK = CS*RWRQ*/RFRQ*T80

WEH = /SUP*RWRQ*CS*/BW*/RW + /SUP*RWRQ*CS*BW*A00*/RW

WEL = /SUP*RWRQ*CS*/BW*/RW + /SUP*RWRQ*CS*BW*/A00*/RW

OE = CS*RW*DS

FUNCTION TABLE

/CS /CYCREQ /FR T100 /DTACK /RWRQ FRH /RFRQ TO T80 A00 /WEH /WEL
/OE /SUP RW BW /DS

```

;
;
;C C F T / / F / T T A / / / / R B
;S Y R O D R R R O 8 0 W W O S W W
; C 0 T W H F 0 0 E E E U P
; R A R Q Q
; E C Q K
; Q

```

```

H H L L H H L H L L L H H H L X X
H H H L H H H H L L L H H H L X X
H H H L H H H H L L L H H H L X X
L H H L H H H H L L L H H H L X X
;BOTH MEMORY AND REFRESH CYCLES MAKE REQUEST--ACKNOWLEDGE REFRESH CYCLE
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X
L L L L H L L L L H L H H H L X X

```

Figure 3-181. Source Listing for a General-Purpose Interface for the Am2968 (AmPAL16L8)


```

L   L   L   H   H   H   L   H   L   L   L   H   H   H   L   X   X
L   L   L   H   H   H   L   H   L   L   L   H   H   H   L   X   X
;END REFRESH -- ACKNOWLEDGE MEMORY CYCLE -- (WRITE)
L   L   H   L   H   L   H   H   L   L   L   L   L   H   H   X   L
L   L   H   L   H   L   H   H   L   L   L   L   L   H   H   L   L
L   L   H   L   L   L   H   H   L   H   L   L   L   H   H   L   L
L   L   H   L   L   L   H   H   L   H   L   L   L   H   H   L   L
L   L   H   L   L   L   H   H   L   H   L   L   L   H   H   L   L
;READ OPERATION
L   L   H   L   L   L   H   H   H   L   L   H   H   L   L   H   L
L   L   H   L   L   L   H   H   H   L   L   H   H   L   L   H   L

```

DESCRIPTION

Figure 3-181. Source Listing for a General-Purpose Interface for the Am2968 (AmPAL16L8) (Continued)

Refresh Timer Calculations

Figure 3-182 shows the setup for the forced refresh ($\overline{FR}$) timer. The timer used is a commercially-available 555 timer.

The general timer equation is as follows, it must be solved for R_1 , R_2 and C values. This will satisfy the refresh requirement of less than or equal to 15.6 μs

$$T = 0.693 (R_1 + 2R_2)C < 4 \text{ ms}/256 = 15.6 \mu s.$$

For the following values of R_1 , R_2 and C

$$R_1 = R_2 = 10K \text{ and } C = 470 \text{ pF}$$

the timer will generate a refresh every 9.8 μs which satisfies the refresh requirement. The specified values for R_1 , R_2 and C allow for 30% tolerance.

Delay Line Assignment

The calculations for the assignment of delay line taps are based on the MSEL and CASI parameters relative to RASi being active. The calculations below indicate the relative minimum times. MSEL (T_{20}) is calculated to be:

$$T_{20} = t_{RAH} + t_{24} = 15 + 11 = 26 \text{ ns.}$$

where T_{20} = delay from RASi to MSEL

t_{RAH} = row address hold time of the DRAM

t_{24} = skew time for Q_n to $\overline{RAS}_n$ of the DMC

Thus, after a minimum of 26 ns from the time RASi becomes active, MSEL may be asserted. CASI (T_{40}) calculated relative to RASi is:

$$T_{40} = T_{20} + t_{ASC} + t_{25} = 26 + 0 + 33 = 59 \text{ ns.}$$

where T_{40} = delay from RASi to CASi

t_{ASC} = column setup time of the DRAM

t_{25} = skew time for Q_n to $\overline{CAS}_n$ of the DMC

Thus, after a minimum of 59 ns from the time RASi becomes active, CASi may make its active transition. The delay line assignment is shown in Figure 3-178. For convenience, the values calculated above have been approximated and are shown below:

RASi ----- MSEL ~ 30 ns

RASi ----- CASi ~ 60 ns

Two other significant delay taps which are not interface outputs but deserve some mention are T80 and T100. The values for these two signals are, respectively, 80% and 100% of the delay line. The T80 output delay is used as a hold time at the end of the refresh cycle so that RFRQ (MC1) remains valid for some time. This allows the row address counter and precharge to be updated before another cycle begins. T100 prevents the memory cycle from beginning too soon, it ensures that the current cycle has ended before a new cycle will begin (see Memory and Refresh timing diagram, Figure 3-183).

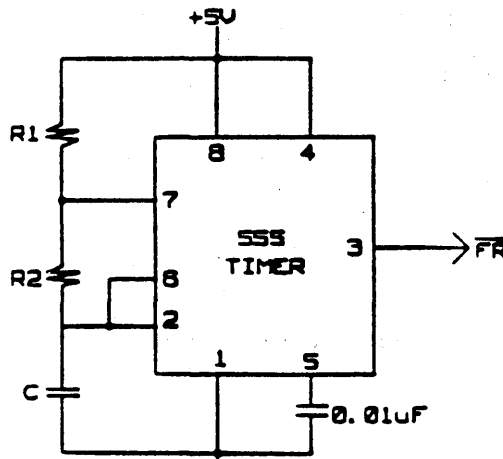


Figure 3-182. Refresh Timer

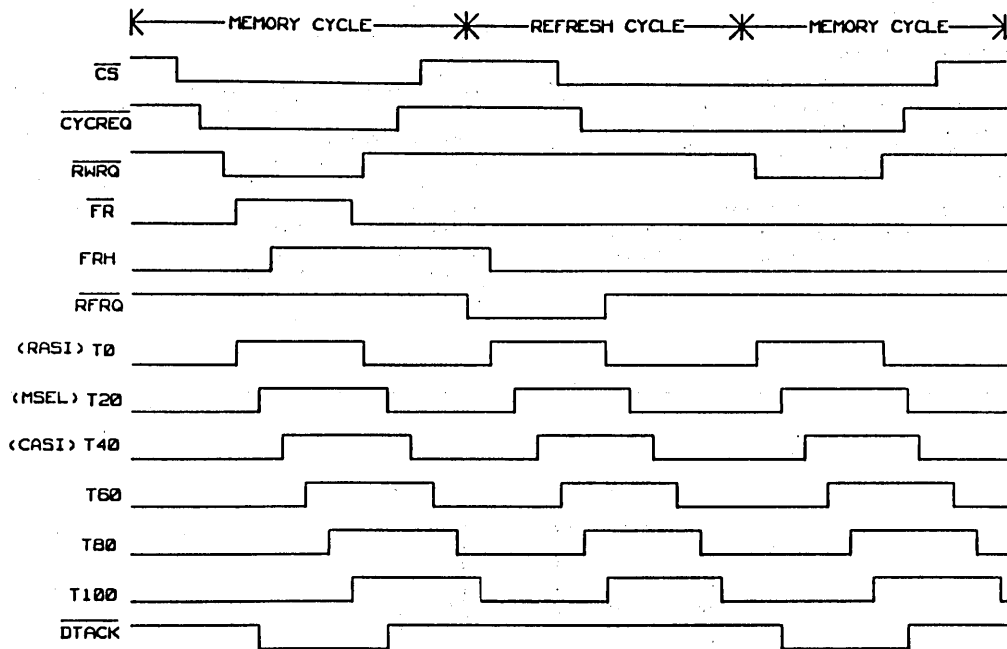


Figure 3-183. Memory And Refresh Timing

3.6.3 MC68000 TO Am2968 INTERFACE

This application note shows how the general-purpose timing generator can be configured to interface between Motorola's MC68000 and the Am2968. Figure 3-184 shows the block diagram of the interface. The correct interface signals must be adapted from the processor to the TimGen (PAL16L8) and the Am2968. Note, from Figure 3-185, that $\overline{AS}$ (Address Strobe) replaces CYCREQ, and $\overline{UDS}$ (Upper Data Strobe) and $\overline{LDS}$ (Lower Data Strobe) replace $\overline{DS}$ and $B/\overline{W}$.

Asserting the Address Strobe signal ($\overline{AS}$), which is connected to the LE input, latches the valid address into the Am2968. $\overline{UDS}$ and $\overline{LDS}$ are controls indicating the flow of data on the data bus (D00-D15) during memory read/write operations. By asserting the correct polarity on these two signals ($\overline{UDS}$ & $\overline{LDS}$), either a byte or a word may be accessed.

The following sections will show the timing requirements which are being considered—that is, the desired processor operating frequency and the appropriate DRAM. Timing diagrams are also included to show the status of the interface signals during read/write operations for various processor frequencies.

MC68000 Timing Requirements

In generating the hardware for the timing interface for the MC68000 to the Am2968, the overall system timing requirements must be considered to guarantee that the desired memory access time can be met.

The following general equation formulates the parameters which must be considered in generating the read cycle time for the 68000 processor. The write cycle time may be similarly generated.

$$t_{\text{READ}} = 2t_{\text{CYC}} + t_{\text{CH}} - t_{\text{CHSLX}} - t_{\text{DIDL}} - 2t_{\text{PAL}} - t_2 - t_{\text{LATCH}} + Nt_{\text{CYC}}$$

t_{READ} = read cycle time

t_{CYC} = clock period of the processor

t_{CH} = clock width High of the processor

t_{CHSLX} = clock High to $\overline{AS}$, $\overline{DS}$ Low of the processor

t_{DIDL} = data in to clock Low (setup time) of the processor

t_{PAL} = programmable logic access time

t_2 = delay from RAS to $\overline{RAS}$ of the DMC

t_{LATCH} = delay through latch

N = number of inserted Wait States.

For simplicity, zero Wait States have been assumed in selecting the appropriate DRAMs. If Wait States had been asserted, the appropriate DRAM must be selected to meet the overall t_{READ} cycle time. Note that the access time of the DRAM (t_{ACC}) must be less than or equal to t_{READ} cycle time or else access to memory will be missed. The following evaluations show the calculated t_{ACC} based on the various processor operating frequencies: 8 MHz, 10 MHz and 12.5 MHz.

MC68000 W/WAIT STATES

8 MHz

$$\begin{aligned} t_{\text{READ}} &= (250 + 55 - 60 - 15 - 30 - 23 - 13 + N \cdot 125) \text{ ns} \\ &= (164 + 125N) \text{ ns} \end{aligned}$$

$$\text{DRAM } t_{\text{ACC}} = 150 \text{ ns}$$

10 MHz

$$\begin{aligned} t_{\text{READ}} &= (200 + 45 - 55 - 10 - 30 - 23 - 13 + N \cdot 100) \text{ ns} \\ &= (114 + 100N) \text{ ns} \end{aligned}$$

$$\text{DRAM } t_{\text{ACC}} = 100 \text{ ns}$$

12.5 MHz[®]

$$\begin{aligned} t_{\text{READ}} &= (160 + 35 - 55 - 10 - 30 - 23 - 13 + N \cdot 80) \text{ ns} \\ &= (64 + 80N) \text{ ns} \end{aligned}$$



The diagram illustrates the timing relationships for the PAL16L8 device. On the left, input signals TS, AS, and FR are shown as active-low pulses. Below them, DTACK is an active-low output, while SUP, R/Q, ODS, and COS are active-low inputs. The central block represents the PAL16L8, with internal signals TIMGEN and PAL16L8. On the right, output signals RASI, MSEL, and CASI are shown as active-low pulses. Below these, FRH, RWRQ, RPRQ, WEH, WEL, and OE are shown as active-low signals. A 'DELAY LINE' block is connected to the RASI, MSEL, and CASI signals, with inputs at 0ns, 30ns, and 40ns, and outputs at 0ns, 30ns, and 40ns. A 'T0' signal is shown as an active-low pulse, and a 'T100' signal is shown as an active-low pulse.

Figure 3-185. Interface Diagram

06829A-12

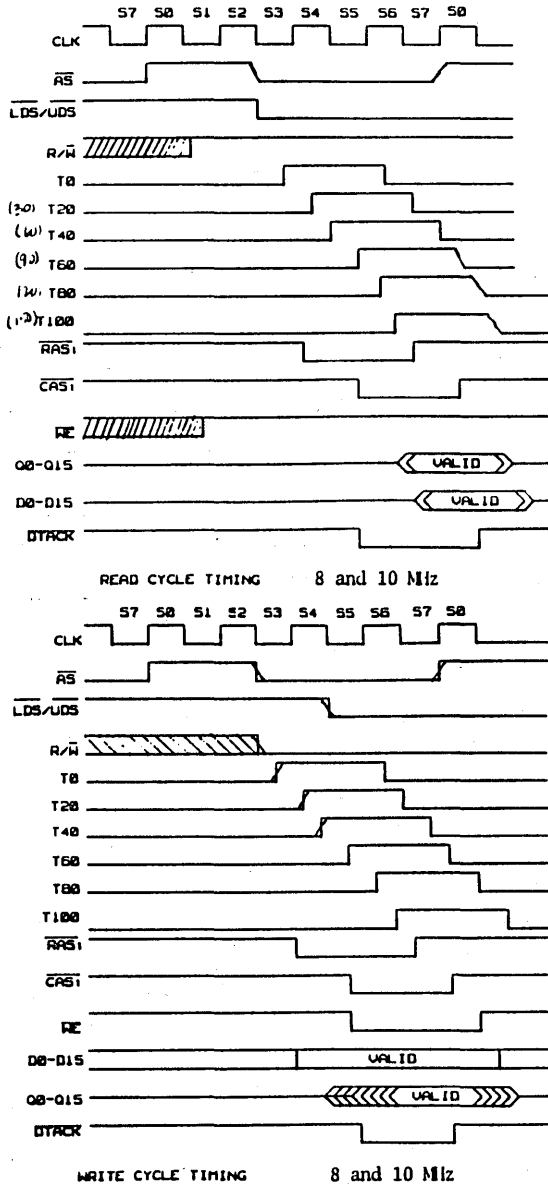


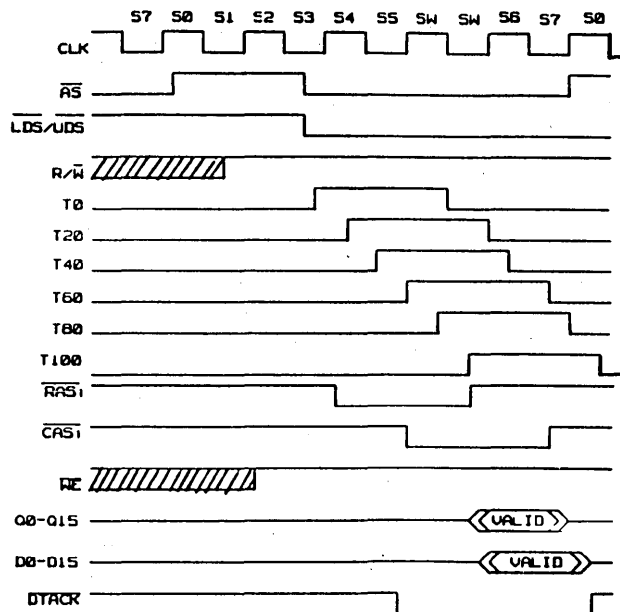
Figure 3-186. Read and Write Cycle Timing Diagram

06829A-13

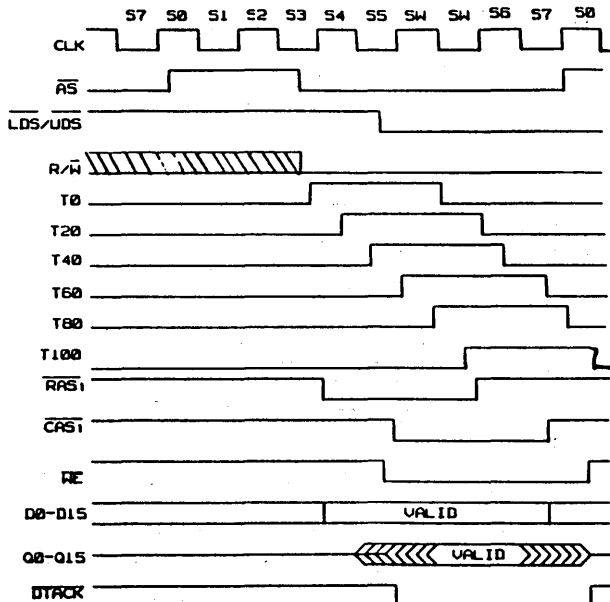
@ for this particular frequency, asserting at least one Wait State will allow $t_{\text{READ}} = 144 \text{ ns}$, thus $\text{DRAM } t_{\text{ACC}} = 120 \text{ ns}$

Figure 3-186 shows the read and write cycle timing diagrams relative to the processor's cycle time. The 8

MHz and 10 MHz are shown on the same diagram because of their similarity. Figure 3-187 shows the read and write timing for the processor operating at 12.5 MHz with Wait States asserted during these operations.



READ CYCLE TIMING
12.5MHz



WRITE CYCLE TIMING
12.5MHz

Figure 3-187. Read and Write Cycle Timing Diagram

06829A-14

PAL16L8
TIMGEN
VER 1.00

GEN INTERFACE for 68K
LEE/YEE 1-18-85
AMD

/CS /AS /FR /SUP RW T80 /UDS /LDS T100 GND
NC /DTACK TO FRH /RFRQ /RWRQ /WEL /WEH /OE VCC

/FRH = FR*/FRH + T0*RFRQ

RWRQ = CS*AS*/RFRQ*/T100 + RWRQ*/T100

RFRQ = FR*FRH*/RWRQ*/T100 + RFRQ*/T100

/TO = /RWRQ*/RFRQ + T80

IF (CS*RWRQ*/RFRQ*T80) DTACK = CS*RWRQ*/RFRQ*T80

WEH = /SUP*RWRQ*CS*UDS*/RW

WEL = /SUP*RWRQ*CS*LDS*/RW

OE = CS*RW*UDS + RW*CS*LDS

FUNCTION TABLE

/CS /AS /FR T100 /DTACK /RWRQ FRH /RFRQ TO T80 /WEH /WEL
/OE /SUP RW UDS LDS

```

;
;
;/ / / T / / F / T T / / / / R U L
;C A F 1 D R R R 0 8 W W O S W D D
;S S R 0 T W H F 0 0 E E E U S S
;
; 0 A R R H L P
;
; C Q R
;
; K
;

```

```

-----
H H L L H H L H L L H H H L X X X
H H H L H H H H L L H H H L X X X
H H H L H H H H L L H H H L X X X
L H H L H H H H L L H H H L X X X
L H H L H H H H L L H H H L X X X
;BOTH MEMORY AND REFRESH CYCLES MAKE REQUEST--ACKNOWLEDGE REFRESH CYCLE
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X
L L L L H L L L L H H H L X X X

```

Figure 3-188. Source Listing for the 68000 Interface


```

L   L   L   H   H   H   L   H   L   L   H   H   H   L   X   X   X
L   L   L   H   H   H   L   H   L   L   H   H   H   L   X   X   X
;END REFRESH -- ACKNOWLEDGE MEMORY CYCLE -- (WRITE)
L   L   H   L   H   L   H   H   H   L   L   L   H   H   X   H   H
L   L   H   L   H   L   H   H   H   L   L   L   H   H   L   H   H
L   L   H   L   L   L   H   H   L   H   L   L   H   H   L   H   H
L   L   H   L   L   L   H   H   L   H   L   L   H   H   L   H   H
L   L   H   L   L   L   H   H   L   H   L   L   H   H   L   H   H
;READ OPERATION
L   L   H   L   L   L   H   H   H   L   H   H   L   L   H   H   H
L   L   H   L   L   L   H   H   H   L   H   H   L   L   H   H   H
-----

```

DESCRIPTION

THIS PAL IMPLEMENTS THE TIMING INTERFACE BETWEEN THE AM2968 DMC AND MOTOROLA'S MC68000

Figure 3-188. Source Listing for the 68000 Interface (Continued)

Interface Overview

The dual-port arbiter can be interfaced to various processors, using the proper control signals from the particular processor. Typically, the processor interface signals are $\overline{CS}$ (Chip Select), $\overline{CYCREQ}$ (Cycle Request), RD (Read), and WR (Write), see Figure 3-189.

The two PAL devices, TIMGEN2A and TIMGEN2B are designed to operate in the following modes:

- (1) generate proper control signals when cycle request is made
- (2) arbitrate between two processor requests
- (3) arbitrate between a refresh request and processor request
- (4) arbitrate between a refresh request and two cycle requests

Requests by the processor are made when $\overline{CS}$ and $\overline{CYCREQ}$ are valid and the memory address is present. During dual-port arbitration, the occurrence of simultaneous requests, for instance, CSA (Chip Select Port A), $\overline{CYCREQA}$ (Cycle Request Port A) and FR (Forced Refresh), will result in the granting of a refresh cycle and the appropriate address being generated accordingly by the DMC.

TIMGEN2A generates the appropriate control signals to the data and address multiplexers, the delay line, and the memory controller. These control signals are: RASI (Row Address Strobe Input), SEL (Processor Address Mux Select), and MC1 (Mode Control Input). RASI is generated for all read/write and refresh cycles. It is also used as the delay line input from which MSEL (MUX Select for the DMC) and CASI (Column Address Strobe Input) are generated. SEL allows the correct port address, A or B, to be the input to the DMC depending upon which port is acknowledged. MC1 specifies the operating mode of the DMC, read/write or refresh (see Am2968A data sheet).

The arbitration logic prioritizes and grants the requests for read/write cycles from either port, A or B, and refresh cycles. The Mode Table (Table 3-24) shows the order of precedence. When simultaneous request for a read/write or refresh cycle occurs, the refresh cycle will be given priority. The connection of $\overline{RFRQ}$ (Refresh Request) from TIMGEN2A to MC1 of the DMC, as shown in the block diagram, implicitly gives refresh the precedence. When simultaneous read/write requests are made by processor A and processor B, processor A will be given priority over B. Again, the order of precedence is set implicitly by the connection of $\overline{RWRQA}$ (Read/Write Request Port A) to the select input signal of the processor address MUX.

The second programmable logic device, TIMGEN2B generates the $\overline{OE}$ (Output Enable) signal for the data bus latches and the $\overline{WE}$ (Write Enable) signal for memory. The inputs to TIMGEN2B are the control signals—CSA, CSB, RDA (Read A), RDB (Read B), WRA (Write A) and WRB (Write B). Signals such as BHEN (Byte High Enable) and A00, the least significant bit of the address, have been included to show their function (if available). These two signals are provided by the processor (i.e., iAPX-type processors) and together they determine whether a word or byte transfer is to be performed during memory operations.

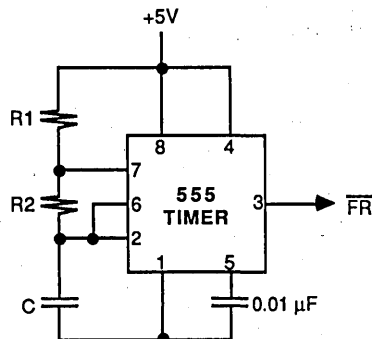


Figure 3-190. Refresh Timer Diagram

TABLE 3-24. MODE SELECT TABLE

$\overline{FR}$	$\overline{CYCREQ}$		GRANT
	Port A	Port B	
L	L	L	$\overline{FR}$ (refresh)
L	L	H	$\overline{FR}$ (refresh)
L	H	L	$\overline{FR}$ (refresh)
L	H	H	FR (refresh)
H	L	L	PORT A (μP Req)
H	L	H	PORT A (μP Req)
H	H	L	PORT B (μP Req)
H	H	H	no activity

08479A 3-189T

Refresh Timer and Calculations

The refresh timer below shows one method of implementing an external refresh clock. This section is optional if an alternate source is used. The refresh timer, implemented with a 555 timer, needs to generate an active-LOW edge-triggered signal at least once every 15 μs . This means that a refresh occurs when a LOW-going edge on the FR signal is detected. The 15 μs value is derived from the dynamic memory refresh timing requirements; refresh is normally required, over 256 rows, every 4 ms. This equates to a required refresh cycle every 15.6 μs . In this application, the refresh timer generates FR approximately once every 9.8 μs . Figure 3-190 shows the circuit for the forced refresh ($\overline{FR}$) timer. The timer is a commercially-available 555 timer. The delay equation shown below must be solved for R1, R2, and C values. This satisfies the refresh requirement of less than, or equal to, 15.6 μs

$$T = 0.693 (R1 + 2R2) C < 4 \text{ ms}/256 = 15.6 \mu s$$

For the following values of R1, R2 and C

$$R1 = R2 = 10K \text{ and } C = 470 \text{ pF}$$

the timer generates a refresh every 9.8 μs which allows some margin in the refresh requirement. The specified values for R1, R2, and C allow for 30% total discrete component tolerance.

TIMGEN2A

The function of TIMGEN2A is to generate $\overline{\text{RASI}}$ (Row Address Strobe Input) and arbitrate between processor and refresh cycles. Based on the arbitration, TIMGEN2A also generates $\overline{\text{RWRQ}}$ (Read/Write Request), or $\overline{\text{RFRQ}}$ (Refresh Request), that determines the type of cycle to be performed. TIMGEN2A can arbitrate between refresh and processor requests and also arbitrate between Port A and Port B processor requests.

When simultaneous refresh and processor requests are generated, refresh ($\overline{\text{RFRQ}}$) will be given priority. The priority is implicit by design because the $\overline{\text{RFRQ}}$ output of TIMGEN2A is connected to MC1 of the Am2968A DMC. When $\overline{\text{RFRQ}}$ becomes active (LOW), MC1 forces the DMC into refresh mode.

When simultaneous processor requests are made, $\overline{\text{RWRQA}}$ (Read/Write Request of Port A) will be granted before $\overline{\text{RWRQB}}$ (Read/Write Request of Port B). This priority is also implicit by design because the $\overline{\text{RWRQA}}$ output is connected to SEL of the processor's address multiplexer. When $\overline{\text{RWRQA}}$ becomes active, address from Port A will be selected for input to the Row and Column Address latches of the Am2968A, also, the data from Data Bus A will be allowed to flow to the inputs of the DRAMs. In either case, $\overline{\text{RASI}}$ must be generated to initiate any of the cycles. Once $\overline{\text{RASI}}$ is generated, the control signals, $\overline{\text{MSEL}}$ and $\overline{\text{CASI}}$, will be generated from the delay line relative to $\overline{\text{RASI}}$.

TIMGEN2B

The function of TIMGEN2B is to generate $\overline{\text{WE}}$ (Write Enable) for memory and also $\overline{\text{OE}}$ (Output Enable) to control the flow of data through the data bus latches. The inputs to TIMGEN2B are, $\overline{\text{CSA}}$, $\overline{\text{RDA}}$ (Read Strobe A), $\overline{\text{WRA}}$ (Write Strobe A), $\overline{\text{CSB}}$, $\overline{\text{RDB}}$ (Read Strobe B), and $\overline{\text{WRB}}$ (Write Strobe B). In addition, there are $\overline{\text{BHENA}}$ (Byte High Enable A) and $\overline{\text{BHENB}}$ (Byte High Enable B), A00A (least significant address bit for Port A) and A00B (least significant address bit for Port B). These signals are generated by the processor and are used to specify either byte or word transfers.

The following description outlines the operation of TIMGEN2B. If $\overline{\text{CSA}}$ and $\overline{\text{RDA}}$ become active (LOW), TIMGEN2B will cause $\overline{\text{OEA}}$ to become active (LOW), enabling the data bus latches and allowing data corresponding to address A to flow onto the system data bus (read operation). The identical procedure occurs for Port B when the control inputs for Port B is active. The block diagram shown in Figure 3-189 shows a multiplexer for the system data bus. This multiplexer controls the data flow into memory from either processor A or processor B, during write operations. The select to the MUX is controlled by $\overline{\text{RWRQA}}$, the same signal that controls the select to the processor address MUX, allowing address Port A to implicitly have higher priority during processor memory access.

The following sections provide general information about interfacing the arbiter to various major types of processors; they are: the iAPX-type, the AmZ8000, and the MC68000-type processors. The PAL devices for the three types of processors will be identified as follows: TIMGEN2A and TIMGEN2B are for

the iAPX-type; TIMGEN3A, TIMGEN3B for the AmZ8000; and TIMGEN4A and TIMGEN4B are for the MC68000-type. When mixing different processor in an interface, some of the control signals mentioned for the particular groups may need to be modified to be applicable to the processor under consideration. The designer can tailor the design to meet specific needs with simple modification to the PAL logic equations provided.

Interfacing the iAPX-Type Processors

The control signals required for TIMGEN2A are shown in Figure 3-191.a. These signals are: $\overline{\text{CS}}$, $\overline{\text{CYCREQ}}$, $\overline{\text{FR}}$ and the delay line outputs (TAP1–TAP5). $\overline{\text{CS}}$ and $\overline{\text{CYCREQ}}$ are provided by the processor. $\overline{\text{FR}}$ is the refresh request from the refresh timer. Sources to this input can be provided either by the timer described in this application note or other appropriate sources. TAP1–TAP5 are the timing delay outputs used to regulate many of the signal generation. The outputs generated from this PAL device are TAP0 ($\overline{\text{RASI}}$), $\overline{\text{XACKA}}$, $\overline{\text{XACKB}}$, and the grants for processor and refresh requests.

The signals for TIMGEN2B are $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{BHE}}$, and A00. For byte/word transfers, the designer should consult the appropriate processor data sheet. In general, this processor group uses $\overline{\text{BHE}}$ in conjunction with A00 to define the transfer function of the data bus. The outputs from this PAL device are: Write Enables, Latch Enables, and Outputs Enables.

Interfacing the AmZ8000

The control signals required for this interface are shown in Figure 3-191.b. For TIMGEN3A, the general controls are $\overline{\text{CS}}$ and $\overline{\text{CYCREQ}}$, $\overline{\text{FR}}$ and the delay line outputs (TAP1–TAP5). $\overline{\text{CS}}$ and $\overline{\text{CYCREQ}}$ are generated by the processor. $\overline{\text{FR}}$ is the refresh request from the refresh timer. TAP1–TAP5 are the delay line outputs used to regulate the signal generation. The outputs from this PAL device are $\overline{\text{WAITA}}$, $\overline{\text{WAITB}}$, TAP0 ($\overline{\text{RASI}}$), and the grants for processor and refresh requests.

The control signals for TIMGEN3B are $\overline{\text{R/W}}$, $\overline{\text{DS}}$, A00 and $\overline{\text{B/W}}$. For byte/word transfers, the designer should consult the processor data sheet for correct signal generation. The outputs from this PAL device are: Write Enables, Latch Enables, and Outputs Enables.

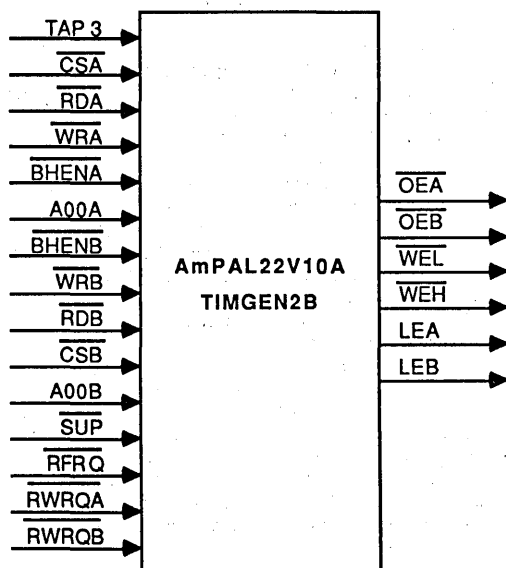
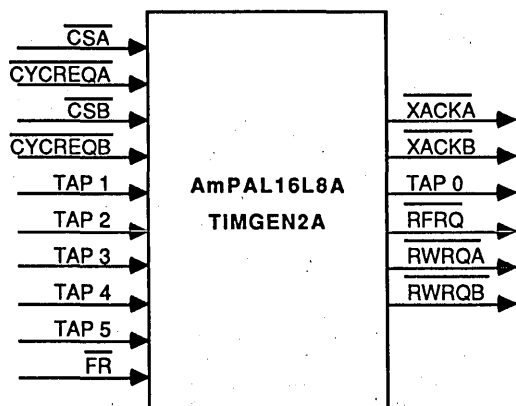
Interfacing the MC68000-Type Processors

The control signals required for this interface are shown in Figure 3-191.c. For TIMGEN4A, the general controls are $\overline{\text{CS}}$ and $\overline{\text{CYCREQ}}$, $\overline{\text{FR}}$ and the delay line outputs (TAP1–TAP5). $\overline{\text{CS}}$ and $\overline{\text{CYCREQ}}$ are generated by the processor. $\overline{\text{FR}}$ is the refresh request from the refresh timer. TAP1–TAP5 are the delay line outputs used to regulate signal generation. The outputs generated from this PAL device are: $\overline{\text{DTACKA}}$, $\overline{\text{DTACKB}}$, TAP0 ($\overline{\text{RASI}}$) and the grants for processor and refresh request.

The control signals for TIMGEN4B are $\overline{\text{R/W}}$, $\overline{\text{UDS}}$, and $\overline{\text{LDS}}$. For byte/word transfers, the designer should consult the respective processor data sheet for correct signal generation. For example, to obtain either byte or word transfers with the MC68000, three signals, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$ and $\overline{\text{R/W}}$, must provide the correct levels to guarantee the correct data transfers, see

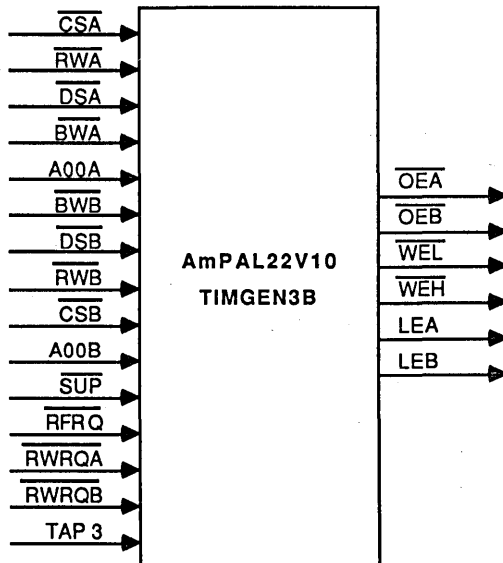
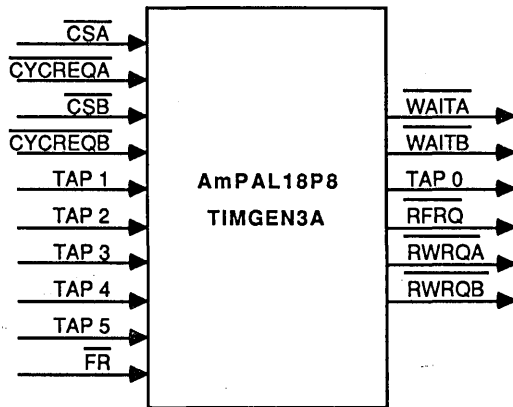
MC68000 Data Bus Control Table in the User's Manual. The outputs generated from this PAL device are: Write Enables, Latch Enables and Output Enables.

The programmable logic device equations for all three interfaces are shown in Figures 3-192 and 3-193.



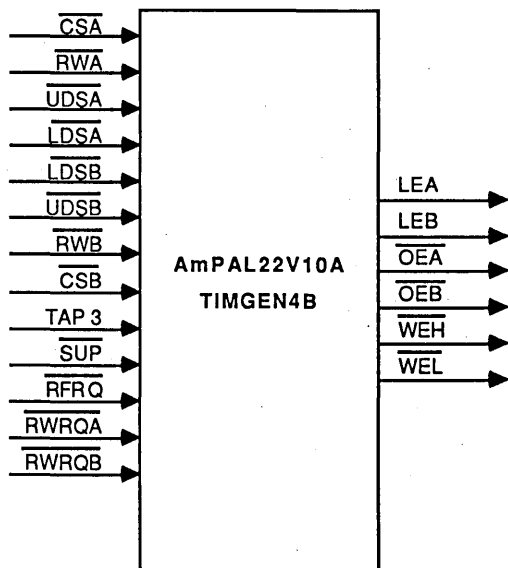
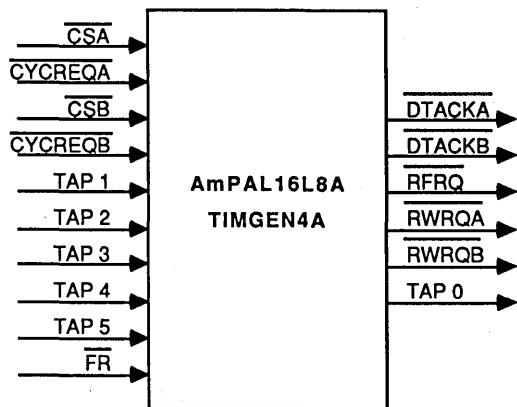
08479A 3-191

Figure 3-191.a Interface for 8086, 80186, and 80286-Type Processors



08479A 3-192

Figure 3-191.b Interface for Z8000-Type Processors



08479A 3-193

Figure 3-191.c Interface for MC68000-Type Processors

ADVANCED MICRO DEVICES

GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #1
TIMGEN2A VERSION 1.0

/CSA /CYCREQA /CSB /CYCREQB TAP1 TAP2 TAP3 TAP4 TAP5 GND
/FR /XACKA /XACKB TAP0 FRH /RFRQ /RWRQA /RWRQB NC VCC

/FRH = FR*/FRH + TAP0*RFRQ

RFRQ = FR*FRH*/RWRQA*/RWRQB*/TAP5 + RFRQ*/TAP5

RWRQA = CSA*CYCREQA*/RFRQ*/RWRQB*/TAP5 + RWRQA*/TAP5

RWRQB = CSB*CYCREQB*/RFRQ*/RWRQA*/TAP5 + RWRQB*/TAP5

/TAP0 = /RFRQ*/RWRQA*/RWRQB + TAP4

IF (CSA*RWRQA*/RFRQ*TAP4) XACKA = CSA*RWRQA*/RFRQ*TAP4

IF (CSB*RWRQB*/RFRQ*/RWRQA*TAP4) XACKB = CSB*RWRQB*/RFRQ*/RWRQA*TAP4

DESCRIPTION

PAL DEVICE THAT ARBITRATES AND GENERATES DUAL-PORT DRAM TIMING INTERFACE
CONTROL SIGNALS FOR AM8086, AM80186, AM80286-TYPE MICROPROCESSORS

DEVICE

(PAL22V10)

"ADVANCED MICRO DEVICES"

"GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #2"

"TIMGEN2B VERSION 1.0"

PIN /CSA /RDA /WRA /BHENA A00A /BHENB /WRB /RDB /CSB
A00B /OEA /OEB /SUP /RFRQ /RWRQA /RWRQB /WEL /WEH LEB LEA

BEGIN

WEH = /SUP*RWRQA*/RFRQ*CSA*WRA*BHENA
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*WRB*BHENB

WEL = /SUP*RWRQA*/RFRQ*CSA*WRA*/A00A
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*WRB*/A00B

OEA = CSA*RWRQA*/RFRQ*RDA

OEB = CSB*RWRQB*/RFRQ*RDB

LEA = CSA*RWRQA*/RFRQ*RDA*TAP3

LEB = CSB*RWRQB*/RFRQ*RDB*TAP3

END.

"DESCRIPTION"

"PAL DEVICE THAT GENERATES DUAL-PORT DRAM INTERFACE CONTROL SIGNALS FROM"

"AM8086, AM80186, AM80286 TYPE CONTROL SIGNALS (/WR, /RD, /BHE)."

Figure 3-192. PLPL Specification for the Example of Figure 3-191.a

ADVANCED MICRO DEVICES

GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #1

TIMGEN3A VERSION 1.0

/CSA /CYCREQA /CSB /CYCREQB TAP1 TAP2 TAP3 TAP4 TAP5 GND
/FR /WAITA /WAITB TAPO FRH /RFRQ /RWRQA /RWRQB NC VCC

$$/FRH = FR*/FRH + TAPO*RFRQ$$

$$RFRQ = FR*FRH*/RWRQA*/RWRQB*/TAP5 + RFRQ*/TAP5$$

$$RWRQA = CSA*CYCREQA*/RFRQ*/RWRQB*/TAP5 + RWRQA*/TAP5$$

$$RWRQB = CSB*CYCREQB*/RFRQ*/RWRQA*/TAP5 + RWRQB*/TAP5$$

$$/TAPO = /RFRQ*/RWRQA*/RWRQB + TAP4$$

$$IF (CSA*RWRQA*/RFRQ*TAP4) /WAITA = CSA*RWRQA*/RFRQ*TAP4$$

$$IF (CSB*RWRQB*/RFRQ*/RWRQA*TAP4) /WAITB = CSB*RWRQB*/RFRQ*/RWRQA*TAP4$$

DESCRIPTION

PAL DEVICE THAT ARBITRATES AND GENERATES DUAL-PORT DRAM INTERFACE
CONTROL SIGNALS FOR Z8000-TYPE MICROPROCESSORS

DEVICE (PAL22V10)

"ADVANCED MICRO DEVICES"

"GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #2"

"TIMGEN3B VERSION 1.0"

PIN /CSA RWA /DSA BWA A00A BWB /DSB RWB /CSB TAP3
A00B /OEA /OEB /SUP /RFRQ /RWRQA /RWRQB /WEL /WEH LEA LEB

BEGIN

WEH = /SUP*RWRQA*/RFRQ*CSA*/RWA*/BWA
+ /SUP*RWRQA*/RFRQ*CSA*/RWA*BWA*A00A
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*/BWB
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*BWB*A00B

WEL = /SUP*RWRQA*/RFRQ*CSA*/RWA*/BWA
+ /SUP*RWRQA*/RFRQ*CSA*/RWA*BWA*/A00A
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*/BWB
+ /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*BWB*/A00B

OEA = CSA*RWRQA*/RFRQ*RWA*DSA

OEB = CSB*RWRQB*/RFRQ*RWB*DSB

LEA = CSA*RWRQA*/RFRQ*RWA*TAP3

LEB = CSB*RWRQB*/RFRQ*RWB*TAP3

END.

"DESCRIPTION"

"PAL DEVICE THAT GENERATES DUAL-PORT DRAM INTERFACE CONTROL SIGNALS FROM"

"Z8000 TYPE CONTROL SIGNALS (B/W, R/W, /DS)."

Figure 3-193.a PLPL Specification for the Example of Figure 3-191.b

ADVANCED MICRO DEVICES
 GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #1
 TIMGEN4A VERSION 1.0

/CSA /CYCREQA /CSB /CYCREQB TAP1 TAP2 TAP3 TAP4 TAP5 GND
 /FR /DTACKA /DTACKB TAP0 FRH /RFRQ /RWRQA /RWRQB NC VCC

/FRH = FR*/FRH + TAP0*RFRQ

RFRQ = FR*FRH*/RWRQA*/RWRQB*/TAP5 + RFRQ*/TAP5

RWRQA = CSA*CYCREQA*/RFRQ*/RWRQB*/TAP5 + RWRQA*/TAP5

RWRQB = CSB*CYCREQB*/RFRQ*/RWRQA*/TAP5 + RWRQB*/TAP5

/TAP0 = /RFRQ*/RWRQA*/RWRQB + TAP4

IF (CSA*RWRQA*/RFRQ*TAP4) DTACKA = CSA*RWRQA*/RFRQ*TAP4

IF (CSB*RWRQB*/RFRQ*/RWRQA*TAP4) DTACKB = CSB*RWRQB*/RFRQ*/RWRQA*TAP4

DESCRIPTION

PAL DEVICE THAT ARBITRATES AND GENERATES DUAL-PORT DRAM TIMING INTERFACE
 CONTROL SIGNALS FOR MC68000-TYPE MICROPROCESSORS

DEVICE (PAL22V10)

"ADVANCED MICRO DEVICES"

"GENERAL DUAL-PORT DRAM INTERFACE TIMING PAL DEVICE #2"

"TIMGEN4B VERSION 1.0"

PIN /CSA RWA /UDSA /LDSA NC5 /LDSB /UDSB RWB /CSB TAP3
 NC10 /OEA /OEB /SUP /RFRQ /RWRQA /RWRQB /WEL /WEH LEA LEB

BEGIN

WEH = /SUP*RWRQA*/RFRQ*CSA*/RWA*UDSA
 + /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*UDSB

WEL = /SUP*RWRQA*/RFRQ*CSA*/RWA*LDSA
 + /SUP*RWRQB*/RFRQ*/RWRQA*CSB*/RWB*LDSB

OEA = CSA*RWRQA*/RFRQ*RWA*(UDSA + LDSA)

OEB = CSB*RWRQB*/RFRQ*RWB*(UDSB + LDSB)

LEA = CSA*RWRQA*/RFRQ*RWA*TAP3

LEB = CSB*RWRQB*/RFRQ*RWB*TAP3

END.

"DESCRIPTION"

"PAL DEVICE THAT GENERATES DUAL-PORT DRAM INTERFACE CONTROL SIGNALS FROM"
 "MC68000 TYPE CONTROL SIGNALS (R/W, UDS, LDS)."

Figure 3-193.b PLPL Specification for the Example of Figure 3-191.c

3.6.5 CUSTOMIZE A FLEXIBLE DRAM CONTROLLER USING SECOND-GENERATION PAL DEVICES

Ever increasingly dense DRAMs and their interface to the emerging 32-bit sophisticated microprocessors provide increasing number of system options to the designer. In order to utilize these advanced features available on the DRAMs and to satisfy the individual system design requirements of different systems, often customizable DRAM controllers are required. PAL-based sequencer devices such as AmPAL23S8 provide effective control sequencing power, flexibility/customizability of design along with ease of use, and offer a desirable alternative to the dedicated functionality of VLSI DRAM controllers.

The following megabit DRAM controller design, based on AmPAL23S8, illustrates the ease by which a PAL device can be easily customized to fulfill system requirements. It supports data bus sizing and data alignment features, which although required for emerging 32-bit microprocessors, are not yet easily and adequately supported by VLSI controllers.

A DRAM controller needs to support, at the minimum, the following functions:

1. Flexible refresh generation mechanism
2. Flexible arbitration scheme
3. Processor handshake protocol
4. Processor cycle timing and control signals
5. Refresh cycle timing and control signals.

Flexible Refresh Generation Mechanisms

For refresh generation, different system designs may require either burst mode, intermittent mode, or both kinds of refresh. In the burst mode, refresh request would occur once every four milliseconds to meet the dynamic RAM's speed. For a megabit RAM with a 512 refresh cycle requirement, all 512 refresh cycles will be performed consecutively. The major disadvantage of the burst refresh approach is that memory becomes out of service for a long time. Intermittent refresh minimizes the out of service time by stealing single cycles between two processor cycles. This cycle stealing is performed by a predefined arbitration mechanism that prioritizes different cycle requests. This design illustrates both intermittent and burst mode refreshes.

Flexible Arbitration Scheme

The arbitration in most designs involves prioritizing memory accesses for: the read cycle, write cycle, read-modify-write cycle, and refresh cycle.

The DRAM controller must allow both asynchronous memory and refresh cycle requests. It should decide whether a memory cycle is an access cycle or a refresh cycle. Typically for intermittent refresh, the cycles are derived from the oscillator, which operates asynchronously with the system clock. Its arbiter grants a request when the refresh request is made and no memory cycle is either occurring or pending. If a refresh cycle is in progress and a memory request is made, then the memory

access is inhibited until the refresh is complete. If a memory cycle is in progress, the arbiter inhibits the refresh cycle until the current cycle is completed. However it needs to remember that a refresh request was made and grant that request on completion of the processor cycle. Of course when refresh and processor access are requested simultaneously, the refresh is given priority. The AmPAL23S8's buried registers are very useful for performing such arbitration functions between equal asynchronous events. These registers can also be used for remembering refresh cycle requests and for later initiating refresh cycle execution.

However, some designs may require additional functions such as prioritizing competing processor requests, or requiring a specific kind of handshake mechanism between the processor and DRAM controller. All such functions can be achieved with a PAL sequencer by using chip resources such as I/O pins, buried registers, and product terms. On the other hand, some systems may not require such functions as read-modify-write cycles, and the PAL sequencer device's resources can be conserved for other system functions.

Flexible Handshake Protocol

PAL sequencer devices provide customizing capabilities to interface to system processors. The logic required for interface protocols for the iAPX family or the 68000 family microprocessors are easily built using the PAL device's internal resources. The design example described here is for a 68020 processor interface signal.

Processor Cycle Execution

Different DRAMs can have different processor cycles. Some just support the basic read/write cycles, while others support extended functions such as page-mode, ripple-mode, or static-column-mode cycles. Some DRAMs also support additional functions, such as the transfer cycle in VRAMs (DRAMs optimized for video applications). PAL sequencer devices allow only user-definable functionality to be implemented in the design, which optimizes resources and improves system performance. Another advantage offered by such designs is the ability to customize the design to DRAM timing requirements, which can vary significantly between -12, -15, -20 parts. All timing functions are easily implemented and modified using buried state registers.

Our design implements the basic read, write, and read-modify-write cycles required in most designs. The timings required assume a -12 memory part. The processor-cycle execution involves address multiplexing, and timing control signal generation. For address multiplexing, Am29841 10-bit buffers are used. The multiplexing of the row or column address is under the control of the DRAM controller.

Generation of timing and control signals, RAS, CAS, WE, and DACK, is also the core function of the controller. The controller switches the address for multiplexing and produces RAS, CAS, WE in a sequence that is understandable by particular DRAMs. It also generates a handshake signal "DACK" for the CPU, to indicate whether or not the memory is ready to be accessed.

Refresh Cycle Execution

DRAMs support different types of refresh cycles. Some DRAMs support CAS-before-RAS-refresh, which embeds the refresh-address counter along with the memory and eliminates the need for an external counter. Some designs may require the Hidden refresh or the standard RAS-only refresh. Depending upon system requirements the designer can optimize the design and resources. For this example, the PAL device has been designed to execute a RAS-only refresh. However, it can be easily modified to support CAS before RAS refresh, or hidden refresh also, if needed.

MBIT DRAM Controller with Data Alignment and Dynamic Bus Sizing

In many cases, currently available DRAM controllers do not support the esoteric requirements of different system designs. The designer is left to make his/her own DRAM controller. The AmPAL23S8 with its embedded extra buried state registers provides an excellent tool for designing such a flexible, fast DRAM controller. It also offers the right speed to interface to the emerging 8/16-MHz processors allowing superior synchronous designs with few or no wait states.

Some of the emerging 32-bit microprocessors, such as 68020 and 80386, support the concept of dynamic bus sizing and misaligned data transfers. Currently, there is no DRAM controller that uses the advantages offered by these two features effectively.

Dynamic Data Bus Sizing

Dynamic bus sizing allows the processor to handle variable-width data buses (8, 16, or 32 bits) on a cycle-by-cycle basis. Dynamic bus sizing makes the size of R/W port/channel transparent to the software designer.

The M68020 allows operand transfers to or from 8-, 16-, and 32-bit ports by dynamically determining the port size during each bus cycle. Similarly, iAPX80386 allows operand transfers to or from 16-, and 32-bit ports by determining the port size during each bus cycle. During an operand transfer, the slave device signals its port size and transfer status (complete or not) to the processor using DSACKY inputs for 68020 (Ready & BS16 for 80386).

DSACK1 DSACK0

H	H	Insert Wait States
H	L	Complete Cycle— Data bus port size 8 bits
L	H	Complete Cycle— Data bus port size 16 bits
L	L	Complete Cycle— Data bus port size 32 bits

The design implemented shows a 68020 interface, but it can be easily modified to accommodate the iAPX386. M68020 always attempts to transfer the maximum data (32 bits) in a single cycle. If the port responds with a size smaller than 32-

bits, the processor repeats the cycles with the remaining bytes of data. For each cycle, M68020 informs the slave of the number of bytes it is attempting to transfer by two signals SIZ0, SIZ1, as follows:

SIZ0,	SIZ1	SIZE
0	1	Byte
1	0	Word
1	1	3 Byte
0	0	Long Word

To accommodate various system tradeoffs, system designs with heterogeneous memory organization sizes such as 32-bits, 16-bits or 8-bits are common. For example, it is often preferable to have smaller external bus sizes, but retain 32-bit internal memory access on a board. As the DMA-bus width is 16/8, it may be preferable to have part 8/16-bit organization of system memory for allowing easy DMA transfers. Similarly, in a graphics system, because of stripe architecture, the video-memory-frame buffer-access may be 32-bit with four pixel stripe of 8-bit planes, but the scratch-pad memory of the lookup table memory may be of smaller width. PAL devices allow customizable, dynamic memory, data-bus sizing based on various system requirements, such as address range (address signals), or jumper-selectable static control for increasing memory width.

Misaligned Transfers

Dynamic sizing is used also in conjunction with address bits A0, A1 to support misaligned data transfers. An example of data-bus size with misaligned data transfers is a word write to a long word with an address offset of 1 byte (A0/A1 = 01). MC68020 & iAPX386 contain internal bus multiplexers to route the data to the correct data lines. Both also generate the extra cycles necessary to complete a misaligned transfer. However, external circuitry is still needed to generate the correct write enables and chip selects to support both functions. PAL devices, however, can be customized to support both these functions easily.

A1	A0	Offset
0	0	0 Bytes
0	1	1 Byte
1	0	2 Bytes
1	1	3 Bytes

By decoding A0 and A1, the two transfer data-size signals, SIZE0 and SIZE1, and the size of the port being addressed, PAL devices can easily support dynamic-bus sizing and generate appropriate chip-select signals for the four banks of memory.

The DRAM Controller

The DRAM controller consists of two AmPAL23S8s: the first is called the Timing and Arbitration PAL device, and the second is called the Data Sizing and Alignment PAL device (Figure 3-194).

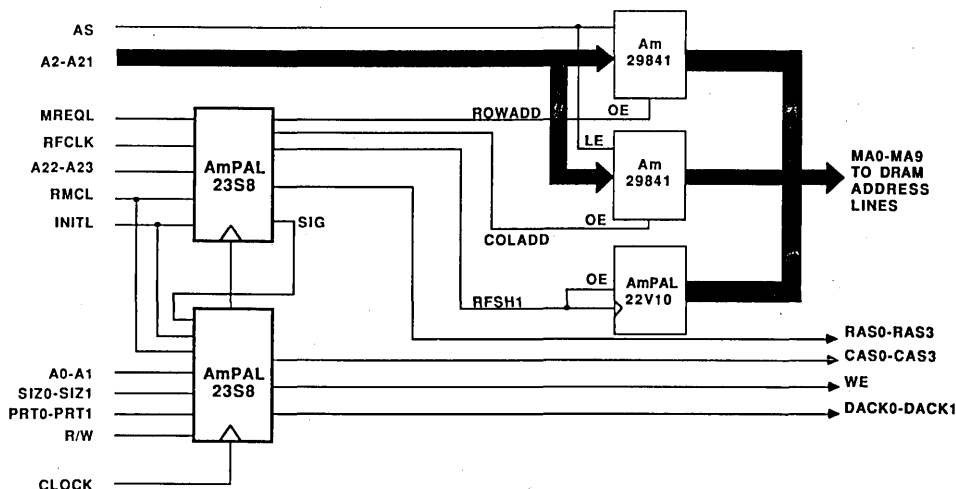


Figure 3-194. A Megabit DRAM Controller

The Timing and Arbitration Controller

The timing and arbitration PAL device performs the following functions:

1. Arbitration between current cycle, refresh cycle & CPU cycle
2. Read-write-cycle execution
3. Read-modify-write-cycle execution
4. Refresh-cycle execution
5. Interface signaling

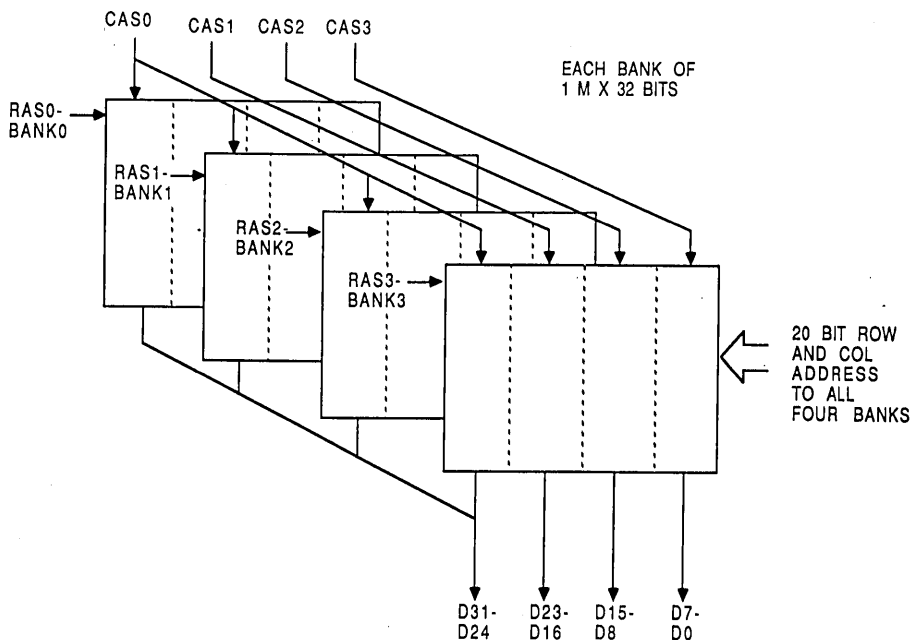
For arbitration, the timing and arbitration PAL device arbitrates between the processor request MREQ signal and the refresh request RFCLK, giving the highest priority to the current cycle, next to the refresh request, and finally to the processor cycle. This sequence supports both intermittent refresh and burst-mode refresh. It also uses a mechanism to store the refresh requests when the processor cycle is in progress for servicing later.

For address multiplexing, it generates the appropriate ROWADD and COLADD signals to multiplex the row and column addresses on to the memory address bus. During refresh

cycles it asserts the RFSH signal by applying the refresh-row address on the memory-address bus.

This PAL device executes the read-and-write cycles as well as read-modify-write cycles. The selection between these two is under the control of the processor signal RMC.

All timing requirements of the different cycles are shown in the timing diagram in Figure 3-196. All the timings of the different cycles are under the control of internal registers. The operation of the state machine to perform all these functions is also shown in Figure 3-197. This state diagram has been derived directly from the timing diagrams and the arbitration requirements of the design. This state machine is implemented by five of the six available buried state registers on the device, and is used primarily for timing and arbitration functions. The sixth buried state register which implements a flag function, is represented as a small, independent state machine, and described below. The state operation was written in AmCUPL, a high level language based PLD software package (Figure 3-198.a), which decodes the state description and generates the Boolean Logic equation file directly (Figure 3-198.b). Figure 3-198.c shows the statistics for this PAL (product term usage etc) and figure 3-198.d shows the corresponding fuse plot for this PAL.



08479A-27

Figure 3-195. Memory Organization

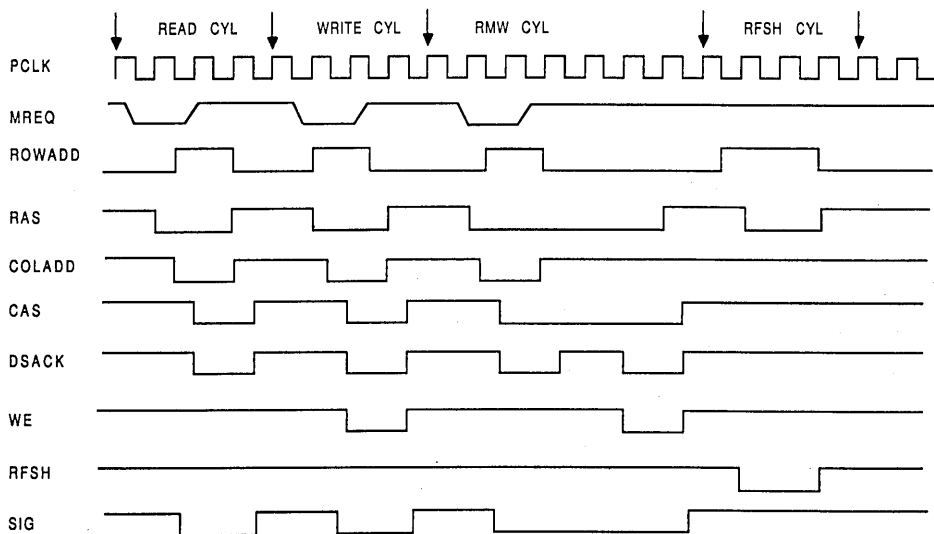


Figure 3-196. Timing Diagram

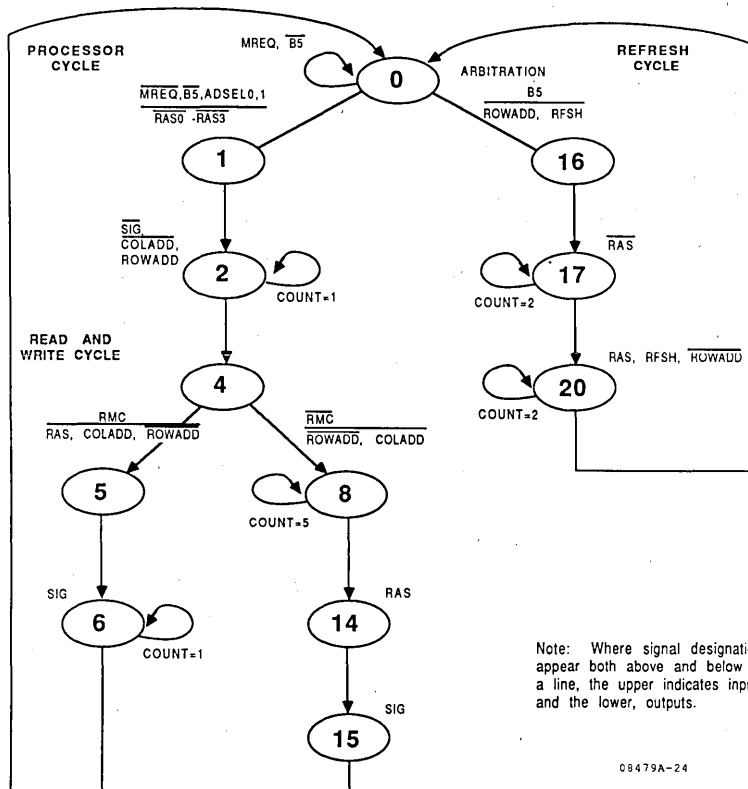


Figure 3-197. State Diagram for the Timing and Arbitration PAL

Arbitration Between Current Cycles, Refresh Cycle, and CPU Cycle

The request for a refresh is received by signal line RFCLK. The CPU requests a cycle by asserting MREQ line. The MREQ signal is assumed to be generated synchronously. This signal can be generated based upon different system requirements—either as a simple decode of address lines, read/write signals or from some virtual to physical address translation scheme, for some complex systems. If a cycle is in progress, as indicated by states other than state 0, no arbitration for a fresh cycle is performed. In such a case, this PAL device continues to execute the current cycle and sequence through different states. This effectively implies that the current cycle is given the most priority and no other cycle is initiated until its completion. When the current cycle is complete (indicated by state 0) the PAL device arbitrates between the MREQ signal and the buried register B5 flag. This buried register flag is used to latch the request for refresh RFCLK, which is usually only

one clock cycle duration. The operation of this flag is controlled by a second small state machine (two states only) in the same device. This second state machine is also shown in Figure 3-197. When RFCLK is HIGH, this state machine toggles to state 1 (B5 = 1) and stays in that state until the refresh cycle is executed (indicated by RFSH signal LOW); it then toggles back to state 0 (B5 = 0), preparing itself for the next refresh request RFCLK.

For the first state machine, the arbitration begins on state 0. If B5 is HIGH, a refresh cycle is given priority and the state machine jumps to state 16 to execute a refresh cycle. If, on the other hand, there is no refresh request pending (indicated by the B5 flag being equal to 0) and MREQ is asserted, a processor cycle is started by jumping to state 1. If none of these condition exist the state machine stays in state 0, polling for any of these events to occur. Such a scheme can support both burst-mode and intermittent-refresh.

```

NAME      MEGABIT DRAM TIMING & ARBITRATION CONTROLLER;
PARTNO    AmPAL 23S8 ;
DATE      03/24/86 ;
REV       01 ;
DESIGNER  KAPIL SHANKAR ;
COMPANY   ADVANCED MICRO DEVICES ;
ASSEMBLY  NONE ;
LOCATION   SUNNYVALE, CA ;
DEVICE    P23S8 ;

```

```

/*****
/*
/*
/*
/*
/*****
/* Allowable Target Device Types: THE ONLY ONE */
/*****

```

/** Inputs **/

```

PIN 2    = MREQ ;
PIN 3    = ADSELO ;
PIN 4    = ADSEL1 ;
PIN 5    = RMC ;
PIN 6    = RFCLK ;
PIN 7    = INIT ;

```

/** Outputs **/

```

PIN 19   = ROWADD ;
PIN 18   = !RAS0 ;
PIN 17   = !COLADD ;
PIN 16   = !RAS1 ;
PIN 15   = !RAS2 ;
PIN 14   = !RFSH ;
PIN 13   = !RAS3 ;
PIN 12   = !SIG ;

```

NODE [B0,B1,B2,B3,B4,B5] ;

/** Declarations and Intermediate Variable Definitions **/

```

FIELD STMA = [B0,B1,B2,B3,B4] ;    /** STATE MACHINE A **/
FIELD STMB = [B5] ;                /** STATE MACHINE B **/
FIELD BANK = [ADSEL1, ADSELO] ;    /** MEMORY BANK ADDRESSED **/

```

```

BANK0 = BANK : 0 ;
BANK1 = BANK : 1 ;
BANK2 = BANK : 2 ;
BANK3 = BANK : 3 ;

```

```

$DEFINE SA0 'b'00000                /** STATE 0 **/
$DEFINE SA1 'b'00001                /** STATE 1 **/
$DEFINE SA2 'b'00010                /** STATE 2 **/
$DEFINE SA3 'b'00011                /** STATE 3 **/
$DEFINE SA4 'b'00100                /** STATE 4 **/
$DEFINE SA5 'b'00101                /** STATE 5 **/
$DEFINE SA6 'b'00110                /** STATE 6 **/
$DEFINE SA7 'b'00111                /** STATE 7 **/
$DEFINE SA8 'b'01000                /** STATE 8 **/
$DEFINE SA9 'b'01001                /** STATE 9 **/
$DEFINE SA10 'b'01010               /** STATE 10 **/
$DEFINE SA11 'b'01011               /** STATE 11 **/
$DEFINE SA12 'b'01100               /** STATE 12 **/
$DEFINE SA13 'b'01101               /** STATE 13 **/
$DEFINE SA14 'b'01110               /** STATE 14 **/
$DEFINE SA15 'b'01111               /** STATE 15 **/
$DEFINE SA16 'b'10000               /** STATE 16 **/
$DEFINE SA17 'b'10001               /** STATE 17 **/
$DEFINE SA18 'b'10010               /** STATE 18 **/
$DEFINE SA19 'b'10011               /** STATE 19 **/
$DEFINE SA20 'b'10100               /** STATE 20 **/
$DEFINE SA21 'b'10101               /** STATE 21 **/
$DEFINE SA22 'b'10110               /** STATE 22 **/

```

Figure 3-198.a Source Code for Timing and Controller PAL


```
$DEFINE SB0 'b'0          /** STATE 0 **/
$DEFINE SB1 'b'1          /** STATE 1 **/
```

```
/** Logic Equations **/
```

```
/** THE FIRST STATE MACHINE **/
```

```
SEQUENCE STMA (
```

```
/** POLLING FOR PROCESSOR OR REFRESH CYCLES **/
```

```
PRESENT SA0 IF B5 & !MREQ & BANK0      NEXT SA1 OUT RAS0 ;
              IF B5 & !MREQ & BANK1      NEXT SA1 OUT RAS1 ;
              IF B5 & !MREQ & BANK2      NEXT SA1 OUT RAS2 ;
              IF B5 & !MREQ & BANK3      NEXT SA1 OUT RAS3 ;
```

```
/** PROCESSOR CYCLE **/
```

```
/** JUMP TO STATE SA1 **/
```

```
/** ASSERT THE RAS FOR THE MEMORY BANK ADDRESSED **/
```

```
      IF B5                      NEXT SA16 OUT ROWADD
                                OUT RFSH ;
```

```
/** REFRESH CYCLE **/
```

```
/** JUMP TO STATE SA16 **/
```

```
/** REMOVE ROW ADDRESS AND SET UP REFRESH ADDRESS **/
```

```
      DEFAULT                    NEXT SA0 ;
/** ELSE POLL AGAIN **/
/** STAY IN STATE SA0 **/
```

```
/** THIS IS THE PROCESSOR CYCLE **/
```

```
PRESENT SA1 IF BANK0      NEXT SA2 OUT RAS0 ;
              IF BANK1      NEXT SA2 OUT RAS1 ;
              IF BANK2      NEXT SA2 OUT RAS2 ;
              IF BANK3      NEXT SA2 OUT RAS3 ;
                                NEXT SA2 OUT ROWADD
                                OUT COLADD
                                OUT SIG ;
```

```
/** REMOVE ROW ADDRESS AND SET UP COLUMN ADDRESS **/
```

```
/** ASSERT SIG TO INFORM DATA SIZING & ALIGNMENT PAL OF PROCESSOR CYCLE **/
```

```
PRESENT SA2 IF BANK0
              IF BANK1
              IF BANK2
              IF BANK3
```

```
NEXT SA3 OUT RAS0 ;
NEXT SA3 OUT RAS1 ;
NEXT SA3 OUT RAS2 ;
NEXT SA3 OUT RAS3 ;
NEXT SA3 OUT ROWADD
      OUT COLADD
      OUT SIG ;
```

```
/** MEMORY ACCESS TIME DELAY **/
```

```
PRESENT SA3 IF BANK0
              IF BANK1
              IF BANK2
              IF BANK3
```

```
NEXT SA4 OUT RAS0 ;
NEXT SA4 OUT RAS1 ;
NEXT SA4 OUT RAS2 ;
NEXT SA4 OUT RAS3 ;
NEXT SA4 OUT ROWADD
      OUT COLADD
      OUT SIG ;
```

```
PRESENT SA4 IF BANK0 & RMC
              IF BANK1 & RMC
              IF BANK2 & RMC
              IF BANK3 & RMC
              IF RMC
```

```
NEXT SA5 OUT !RAS0 ;
NEXT SA5 OUT !RAS1 ;
NEXT SA5 OUT !RAS2 ;
NEXT SA5 OUT !RAS3 ;
NEXT SA5 OUT !ROWADD
      OUT !COLADD
      OUT SIG ;
```

```
/** DISTINGUISH BETWEEN READ WRITE OR READ MODIFY WRITE CYCLES **/
```

```
/** IF RMC HIGH EXECUTE AS READ WRITE CYCLE **/
```

```
/** JUMP TO STATE SA5 **/
```

```
/** ALSO REMOVE COLUMN ADDRESS AND APPLY ROW ADDRESS **/
```

```
/** ALSO REMOVE THE RAS SIGNAL **/
```

```
      IF BANK0 & !RMC
      IF BANK1 & !RMC
      IF BANK2 & !RMC
      IF BANK3 & !RMC
      IF !RMC
```

```
NEXT SA8 OUT RAS0 ;
NEXT SA8 OUT RAS1 ;
NEXT SA8 OUT RAS2 ;
NEXT SA8 OUT RAS3 ;
NEXT SA8 OUT !ROWADD
      OUT !COLADD
      OUT SIG ;
```

```
/** IF RMC LOW EXECUTE READ MODIFY WRITE CYCLE **/
```

```
/** JUMP TO STATE SA8 **/
```

```
/** ALSO REMOVE COLUMN ADDRESS AND ASSERT ROW ADDRESS **/
```

Figure 3-198.a Source Code for Timing and Controller PAL (Continued)

```

PRESENT SA5                                NEXT SA6 OUT !SIG ;                /** THIS IS THE READ WRITE CYCLE **/
/** RAS PRECHARGE TIME DELAY **/

PRESENT SA6                                NEXT SA7 ;

PRESENT SA7                                NEXT SA0 ;
/** PROCESSOR READ WRITE CYCLE COMPLETE **/

/** THIS IS THE READ MODIFY WRITE CYCLE **/

PRESENT SA8 IF BANK0                      NEXT SA9 OUT RAS0 ;
      IF BANK1                          NEXT SA9 OUT RAS1 ;
      IF BANK2                          NEXT SA9 OUT RAS2 ;
      IF BANK3                          NEXT SA9 OUT RAS3 ;
                                          NEXT SA9 OUT SIG ;

/** ACCESS TIME DELAY FOR BOTH READ AND WRITE **/

PRESENT SA9 IF BANK0                      NEXT SA10 OUT RAS0 ;
      IF BANK1                          NEXT SA10 OUT RAS1 ;
      IF BANK2                          NEXT SA10 OUT RAS2 ;
      IF BANK3                          NEXT SA10 OUT RAS3 ;
                                          NEXT SA10 OUT SIG ;

PRESENT SA10 IF BANK0                     NEXT SA11 OUT RAS0 ;
      IF BANK1                          NEXT SA11 OUT RAS1 ;
      IF BANK2                          NEXT SA11 OUT RAS2 ;
      IF BANK3                          NEXT SA11 OUT RAS3 ;
                                          NEXT SA11 OUT SIG ;

PRESENT SA11 IF BANK0                     NEXT SA12 OUT RAS0 ;
      IF BANK1                          NEXT SA12 OUT RAS1 ;
      IF BANK2                          NEXT SA12 OUT RAS2 ;
      IF BANK3                          NEXT SA12 OUT RAS3 ;
                                          NEXT SA12 OUT SIG ;

PRESENT SA12 IF BANK0                     NEXT SA13 OUT RAS0 ;
      IF BANK1                          NEXT SA13 OUT RAS1 ;
      IF BANK2                          NEXT SA13 OUT RAS2 ;
      IF BANK3                          NEXT SA13 OUT RAS3 ;
                                          NEXT SA13 OUT SIG ;

PRESENT SA13 IF BANK0                     NEXT SA14 OUT !RAS0 ;
      IF BANK1                          NEXT SA14 OUT !RAS1 ;
      IF BANK2                          NEXT SA14 OUT !RAS2 ;
      IF BANK3                          NEXT SA14 OUT !RAS3 ;
                                          NEXT SA14 OUT !SIG ;

/** REMOVE THE RAS SIGNAL **/

PRESENT SA14                                NEXT SA15 OUT !SIG ;
/** RAS PRECHARGE TIME DELAY **/

PRESENT SA15                                NEXT SA0 ;

/** PROCESSOR READ MODIFY WRITE CYCLE COMPLETE **/

/** THIS IS THE MEMORY REFRESH CYCLE **/

PRESENT SA16                                NEXT SA17 OUT RAS0
                                          OUT RAS1
                                          OUT RAS2
                                          OUT RAS3
                                          OUT ROWADD
                                          OUT RFSH ;

/** ASSERT ALL FOUR RAS SIGNALS **/

PRESENT SA17                                NEXT SA18 OUT RAS0
                                          OUT RAS1
                                          OUT RAS2
                                          OUT RAS3
                                          OUT ROWADD
                                          OUT RFSH ;

/** REFRESH TIME DELAY **/

PRESENT SA18                                NEXT SA19 OUT RAS0
                                          OUT RAS1
                                          OUT RAS2
                                          OUT RAS3
                                          OUT ROWADD
                                          OUT RFSH ;

```

Figure 3-198.a Source Code for Timing and Controller PAL (Continued)

```

PRESENT SA19                                NEXT SA20 OUT !RAS0
                                           OUT !RAS1
                                           OUT !RAS2
                                           OUT !RAS3
                                           OUT !ROWADD
                                           OUT !RFSH ;

/** REMOVE ALL RAS SIGNALS **/
/** ALSO REMOVE REFRESH ADDRESS AND ASSERT ROW ADDRESS **/

PRESENT SA20                                NEXT SA21 ;
/** RAS PRECHARGE TIME DELAY **/

PRESENT SA21                                NEXT SA22 ;

PRESENT SA22                                NEXT SA0 ;

/** TERMINATE MEMORY REFRESH CYCLE **/
/** GO BACK TO POLLING STATE **/
)
/** THE FIRST STATE MACHINE COMPLETE **/

/** THE SECOND STATE MACHINE **/

SEQUENCE STMB C

PRESENT SB0 IF RFCLK                        NEXT SB1 ;
      DEFAULT                                NEXT SB0 ;
/** JUMP TO STATE SB1 **/
/** REMEMBER REFRESH REQUEST **/

PRESENT SB1 IF RFSH                        NEXT SB0 ;
      DEFAULT                                NEXT SB1 ;
/** JUMP TO STATE SB0 **/
/** REFRESH CYCLE EXECUTED THUS RETURN **/
)

/** THE SECOND STATE MACHINE COMPLETE **/

```

```

/** EQUATIONS FOR SYSTEM INITIALIZATION AFTER RESET **/

[B0..B5].ar = !INIT ;

RAS0.AR = !INIT ;
RAS1.AR = !INIT ;
ROWADD.AR = !INIT ;
COLADD.AR = !INIT ;
RAS2.AR = !INIT ;
RAS3.AR = !INIT ;
RFSH.AR = !INIT ;
SIG.AR = !INIT ;

/* THESE ARE THE OUTPUT ENABLES */

RAS0.OE = 'B'1 ;
RAS1.OE = 'B'1 ;
ROWADD.OE = 'B'1 ;
COLADD.OE = 'B'1 ;
RAS2.OE = 'B'1 ;
RAS3.OE = 'B'1 ;
RFSH.OE = 'B'1 ;
SIG.OE = 'B'1 ;

```

Figure 3-198.a Source Code for Timing and Controller PAL (Continued)

MEGABIT

CUPL 2.11a Serial# 9-99999-999
 Device p23s8 Library DLIB-e-22-4
 Created Wed Jul 30 10:47:14 1986
 Name MEGABIT DRAM TIMING & ARBITRATION CONTROLLER
 Partno AmPAL 23S8
 Revision 01
 Date 03/24/86
 Designer KAPIL SHANKAR
 Company ADVANCED MICRO DEVICES
 Assembly NONE
 Location SUNNYVALE, CA

=====

Expanded Product Terms

=====

RFSH.ar =>
 !INIT

RFSH.oe =>
 1

RFSH.d =>
 !B0 & !B1 & !B2 & !B3 & !B4 & B5
 # !B1 & !B2 & !B3 & B4
 # !B0 & B1 & !B2 & !B3 & B4

B0.ar =>
 !INIT

B0.d =>
 !B0 & !B1 & B2 & !B3 & B4
 # !B0 & !B1 & !MREQ & !B2 & !B3 & !B4 & !B5
 # !B0 & !B2 & !B3 & B4
 # !B0 & B3 & !B4
 # !B0 & B1 & !B3 & !B4
 # !B0 & !B1 & B2 & !B3 & !B4 & RMC

STMA =>

B0 , B1 , B2 , B3 , B4

B1.ar =>
 !INIT

B1.d =>
 !B0 & B1 & !B2 & !B3 & B4
 # B0 & !B1 & B3 & !B4
 # B0 & !B1 & !B3
 # !B0 & B1 & !B4

STMB =>

B5

B2.ar =>
 !INIT

B2.d =>
 !B1 & B2 & !B3 & B4
 # B0 & B1 & !B2 & !B3 & B4
 # !B0 & !B1 & B2 & B3 & !B4
 # B0 & B1 & !B2 & !B4
 # !B0 & !B1 & B2 & !B3 & !B4 & RMC
 # B0 & !B1 & B2 & !B4
 # !B0 & B1 & B2 & !B4

B3.ar =>
 !INIT

B3.d =>
 !B0 & B1 & B2 & B3 & !B4
 # B1 & !B2 & B3 & !B4
 # !B0 & !B1 & B2 & !B3 & !B4 & !RMC
 # !B1 & B3 & !B4

B4.ar =>
 !INIT

B4.d =>

Figure 3-198.b Boolean Logic Equation for the Timing and Arbitration PAL

```

B1 & !B2 & !B3 & B4
# !B0 & !B1 & !B2 & !B3 & !B4 & B5
# !B1 & !B3 & B4

```

```

B5.ar =>
    !INIT

```

```

B5.d =>
    !B5 & RFCLK
    # !RFSH & !B5

```

```

SIG.ar =>
    !INIT

```

```

SIG.oe =>
    1

```

```

SIG.d =>
    B0 & !B2 & !B3 & !B4
    # !B0 & B1 & !B2 & !B4
    # !B0 & !B1 & B2 & !B4
    # B0 & !B1 & B2 & B3 & !B4
    # !B1 & !B2 & B3 & !B4
    # B0 & B1 & !B2 & B3 & !B4

```

```

COLADD.ar =>
    !INIT

```

```

COLADD.oe =>
    1

```

```

COLADD.d =>
    B0 & !B2 & !B3 & !B4
    # !B0 & B1 & !B2 & !B3 & !B4

```

```

BANK0 =>
    !ADSELO & !ADSEL1

```

```

BANK1 =>
    ADSELO & !ADSEL1

```

```

BANK2 =>
    !ADSELO & ADSEL1

```

```

BANK3 =>
    ADSELO & ADSEL1

```

```

ROWADD.ar =>
    !INIT

```

```

ROWADD.oe =>
    1

```

```

ROWADD.d =>
    !B0 & !B1 & !B2 & !B3 & !B4 & B5
    # B0 & !B2 & !B3 & !B4
    # !B0 & B1 & !B2 & !B3
    # !B1 & !B2 & !B3 & B4

```

```

RAS0.ar =>
    !INIT

```

```

RAS0.oe =>
    1

```

```

RAS0.d =>
    !B0 & !B1 & !MREQ & !B2 & !B3 & !B4 & !ADSELO & !B5 & !ADSEL1
    # B0 & !B2 & !B3 & !B4 & !ADSELO & !ADSEL1
    # !B0 & B1 & !B2 & !B4 & !ADSELO & !ADSEL1
    # !B0 & B1 & !B2 & !B3 & B4
    # !B0 & !B1 & B2 & !B3 & !B4 & !ADSELO & !ADSEL1 & !RMC
    # !B1 & !B2 & B3 & !B4 & !ADSELO & !ADSEL1
    # B0 & B1 & !B2 & B3 & !B4 & !ADSELO & !ADSEL1
    # !B0 & !B1 & B2 & B3 & !B4 & !ADSELO & !ADSEL1
    # !B1 & !B2 & !B3 & B4

```

```

RAS1.ar =>
    !INIT

```

```

RAS1.oe =>
    1

```

Figure 3-198.c Statistics for the Timing and Arbitration PAL

```

RAS1.d =>
    !B0 & !B1 & !MREQ & !B2 & !B3 & !B4 & ADSELO & !B5 & !ADSEL1
    # B0 & !B2 & !B3 & !B4 & ADSELO & !ADSEL1
    # !B0 & B1 & !B2 & !B4 & ADSELO & !ADSEL1
    # !B0 & B1 & !B2 & !B3 & B4
    # !B0 & !B1 & B2 & !B3 & !B4 & ADSELO & !ADSEL1 & !RMC
    # !B1 & !B2 & B3 & !B4 & ADSELO & !ADSEL1
    # B0 & B1 & !B2 & B3 & !B4 & ADSELO & !ADSEL1
    # !B0 & !B1 & B2 & B3 & !B4 & ADSELO & !ADSEL1
    # !B1 & !B2 & !B3 & B4

```

```

RAS2.ar =>
    !INIT

```

```

RAS2.oe =>
    1

```

```

RAS2.d =>
    !B0 & !B1 & !MREQ & !B2 & !B3 & !B4 & !ADSELO & !B5 & ADSEL1
    # B0 & !B2 & !B3 & !B4 & !ADSELO & ADSEL1
    # !B0 & B1 & !B2 & !B4 & !ADSELO & ADSEL1
    # !B0 & B1 & !B2 & !B3 & B4
    # !B0 & !B1 & B2 & !B3 & !B4 & !ADSELO & ADSEL1 & !RMC
    # !B1 & !B2 & B3 & !B4 & !ADSELO & ADSEL1
    # B0 & B1 & !B2 & B3 & !B4 & !ADSELO & ADSEL1
    # !B0 & !B1 & B2 & B3 & !B4 & !ADSELO & ADSEL1
    # !B1 & !B2 & !B3 & B4

```

```

RAS3.ar =>
    !INIT

```

```

RAS3.oe =>
    1

```

```

RAS3.d =>
    !B0 & !B1 & !MREQ & !B2 & !B3 & !B4 & ADSELO & !B5 & ADSEL1
    # B0 & !B2 & !B3 & !B4 & ADSELO & ADSEL1
    # !B0 & B1 & !B2 & !B4 & ADSELO & ADSEL1
    # !B0 & B1 & !B2 & !B3 & B4
    # !B0 & !B1 & B2 & !B3 & !B4 & ADSELO & ADSEL1 & !RMC
    # !B1 & !B2 & B3 & !B4 & ADSELO & ADSEL1

```

```

# B0 & B1 & !B2 & B3 & !B4 & ADSELO & ADSEL1
# !B0 & !B1 & B2 & B3 & !B4 & ADSELO & ADSEL1
# !B1 & !B2 & !B3 & B4

```

BANK =>

ADSEL1 , ADSELO

Symbol Table

Pin Variable				Pterms	Max	Min
Pol Name	Ext	Pin	Type	Used	Pterms	Level
! RFSH		14	V	-	-	-
RFSH	ar	14	X	1	1	1
! RFSH	oe	14	X	1	1	1
! RFSH	d	14	X	3	8	1
B0		21	N	-	-	-
B0	ar	21	M	1	1	1
B0	d	21	M	6	10	1
INIT		7	V	-	-	-
STMA		0	F	-	-	-
B1		22	N	-	-	-
B1	ar	22	M	1	1	1
B1	d	22	M	4	10	1
MREQ		2	V	-	-	-
STMB		0	F	-	-	-
B2		23	N	-	-	-
B2	ar	23	M	1	1	1
B2	d	23	M	7	8	1
B3		24	N	-	-	-
B3	ar	24	M	1	1	1
B3	d	24	M	4	8	1
B4		25	N	-	-	-
B4	ar	25	M	1	1	1
B4	d	25	M	3	6	1
ADSELO		3	V	-	-	-
B5		26	N	-	-	-

Figure 3-198.c Statistics for the Timing and Arbitration PAL Device

=====

Fuse Plot

=====

```
Pin #18 6214 Mode -xxx
0414 -----
0460 -x--x-xx-----x--x-x---x----x-
0506 ---x-xx-----x--x-x---x-
0552 ---x-xx-----x--x-x---
0598 -----x--x-x-x--x-
0644 ---x-xx-x--x-x-x--x-
0690 ---x-xx-----x--x---x-
0736 ---x-xx-----x--x-x---x-
0782 ---x-xx-----x-x-x--x-
0828 -----x--x--x---x-
0874 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
Node 25
0920 .....x.....x.....x.....x.....
0966 .....x.....x.....x.....x.....x.....
```

```

1012 -----x-----x-----x-----
1058 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1104 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1150 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17 6218 Mode -x
1196 -----x-----x-----x-----
1242 -----x-----x-----x-----
1288 -----x-----x-----x-----
1334 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1380 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1426 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1518 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1564 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 23
1610 -----x-----x-----x-----
1656 -----x-----x-----x-----
1702 -----x-----x-----x-----
1748 -----x-----x-----x-----
1794 -----x-----x-----x-----
1840 -----x-----x-----x-----
1886 -----x-----x-----x-----
1932 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #16 6220 Mode -x
1978 -----x-----x-----x-----
2024 -x-x--xx-----x--x-x--x-----x-----
2070 ---x--xx-----x--x--x-----x-----
2116 ---x--xx-----x--x-x-----x-----
2162 -----x-----x-----x-----
2208 ---x--xx-x--x--x-x--x-----x-----
2254 ---x--xx-----x-----x-----x-----
2300 ---x--xx-----x--x-x-----x-----
2346 ---x--xx-----x--x-x-----x-----
2392 -----x-----x-----x-----
2438 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2484 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2530 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 21
2576 -----x-----x-x-x-----x-----
2622 -x-----x-----x--x-x-----x-----x-----
2668 -----x-----x-----x-----x-----
2714 -----x-----x-----x-----
2760 -----x-----x-x-x-----x-----
2806 -----x-x-----x-x-x-----x-----
2852 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2898 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2944 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2990 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 22
3036 -----x-----x-x-x-----x-----
3082 -----x-----xx-----x-----
3128 -----x-----xx-----x-----
3174 -----x-----x-x-----x-----
3220 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3266 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3312 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3358 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3404 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3450 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #15 6222 Mode -x
3496 -----x-----x-x-x-----x-----
3542 -x--x-x-x-----x--x-x-----x-----
3588 ---x-x-x-----x--x-----x-----
3634 ---x-x-x-----x--x-x-----x-----
3680 -----x-----x-x-x-----x-----
3726 ---x-x-x-x-----x-x-x-----x-----
3772 ---x-x-x-----x-x-x-----x-----
3818 ---x-x-x-----x--x-x-----x-----
3864 ---x-x-x-----x-x-x-----x-----
3910 -----x-----x-----x-----
3956 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4002 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4048 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 24
4094 -----x-----x-x-x-----x-----
4140 -----x-----x-----x-----
4186 -----x-x-----x-x-x-----x-----
4232 -----x-----x-----x-----
4278 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4324 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4370 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14 6224 Mode -x
4462 -----x-----x-----x-----

```

Figure 3-198.d Fuse Plot and Chip Diagram for the Timing and Arbitration PAL Device (Continued)


```

4508 -----X-----X-X-X-----X-----
4554 -----X-----X-----X-----
4600 -----X-----X-X-X-----X-----
4646 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4692 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4738 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4784 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4830 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 26
4876 -----X-----X-----
4922 -----X-----X-----
4968 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5060 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5106 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13 6226 Mode xxxx
5152 -----
5198 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5244 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5290 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5336 -----X-----X-X-X-X-X-X-X-X-X-X-
5382 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5428 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5474 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5520 -X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-X-
5566 -----X-----X-----X-----
5612 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12 6230 Mode -xxx
5658 -----
5704 -----X-----X-----X-----
5750 -----X-----X-X-X-X-----
5796 -----X-----X-X-X-X-----
5842 -----X-----X-XX-X-X-----
5888 -----X-----X-X-X-X-----
5934 -----X-----X-X-X-X-X-----
5980 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6026 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

LEGEND X : fuse not blown
 - : fuse blown

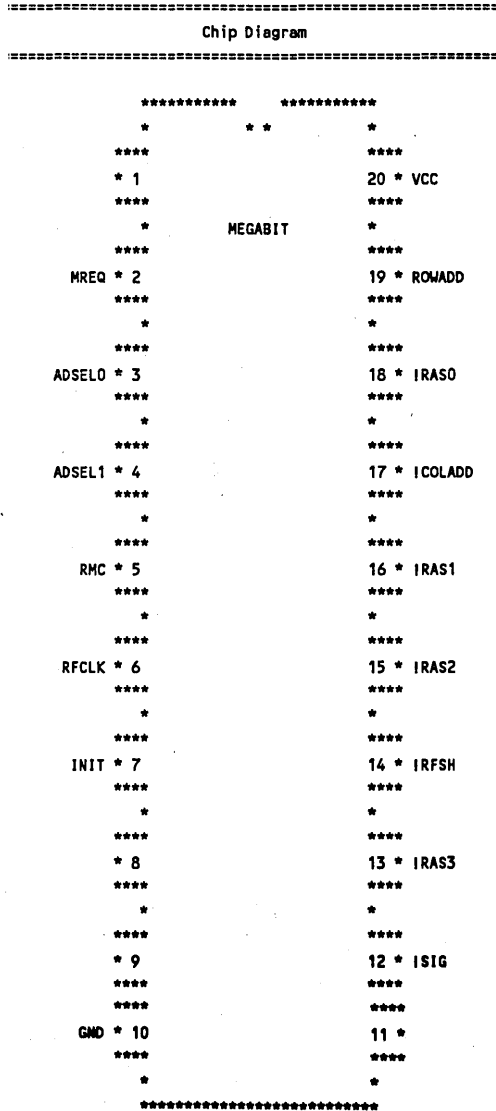


Figure 3-198.d Fuse Plot and Chip Diagram for the Timing and Arbitration PAL Device (Continued)

Processor Read Write Cycle Execution

When the processor cycle wins the arbitration, indicated by state 1, the timing and control signals required for the processor cycle are initiated.

There are four memory banks in the design each with four Mbytes of memory organized as 1M X 32 bits, as shown in Figure 3-195. The Timing and Arbitration PAL device generates the four RAS0-RAS3 signals, one for each memory bank. The selection of one of these four RAS signals depends upon the two input ADSEL0 & ADSEL1 signals. In most systems these would constitute the high bits of system-processor address bus.

After asserting the appropriate RAS signal, the state machine jumps to state 2 to manipulate its address-handling signals. For address multiplexing, this timing and arbitration PAL device generates the appropriate ROWADD and COLADD signals to multiplex the row and column addresses on to the memory-address bus. In state 2 it removes the ROWADD signal and asserts the COLADD signal to allow assertion of the column address to the memory. At the same time it asserts another signal, SIG that is used to inform the Data Sizing and Alignment PAL device of the existence of a processor requested cycle, and to generate the appropriate CAS signals for the memory.

The state machine then sequences through till state 4, allowing for the data-access time. At state 4, it decides between the read-write cycle or a read-modify-write cycle, based on processor signal RMC, and jumps to either state 5 (for read-write cycle) or state 8 (for read-modify-write cycle). For read-write cycle, it removes the RAS signals and waits for another cycle for the RAS precharge time before completing the cycle. Simultaneously, it also removes COLADD signal and asserts ROWADD signal, preparing the memory for the next processor cycle. The SIG signal to the Data Sizing and Alignment PAL device is also removed. The DSACK signal (acknowledgment to the processor) and the WE signals (for a write cycle) are generated if and when required under the control of the Data Sizing and Alignment PAL device.

Processor Read Modify Write Cycle Execution

When the processor requests a read-modify-write cycle by asserting RMC signal, the Timing and Arbitration PAL device automatically increases the length of the cycle. It keeps the

RAS asserted for the appropriate duration of a read-modify-write cycle. At the same time, the Data Sizing and Alignment PAL device generates the appropriate WE and DSACK signals. After this extended RAS assertion, the Timing and Arbitration PAL device completes the cycle, similar to the read-write cycle.

Refresh Cycle Execution

When the arbitration is won by the refresh cycle, the Timing and Arbitration Control PAL device executes a RAS-only refresh cycle. On state 16 it removes the ROWADD signal and asserts the RFSH signal, which applies the refresh row-address on to the memory-address bus. Then in state 17 it generates all four RAS0-RAS3 signals, refreshing all memory banks. It then counts for a few states, depending upon the refresh timing requirements of the memory, before removing the RAS and RFSH signals. On removal of the RFSH signal, the external 10-bit refresh-row address-counter (AmPAL 22V10) is also incremented for the next row.

Interface Signalling

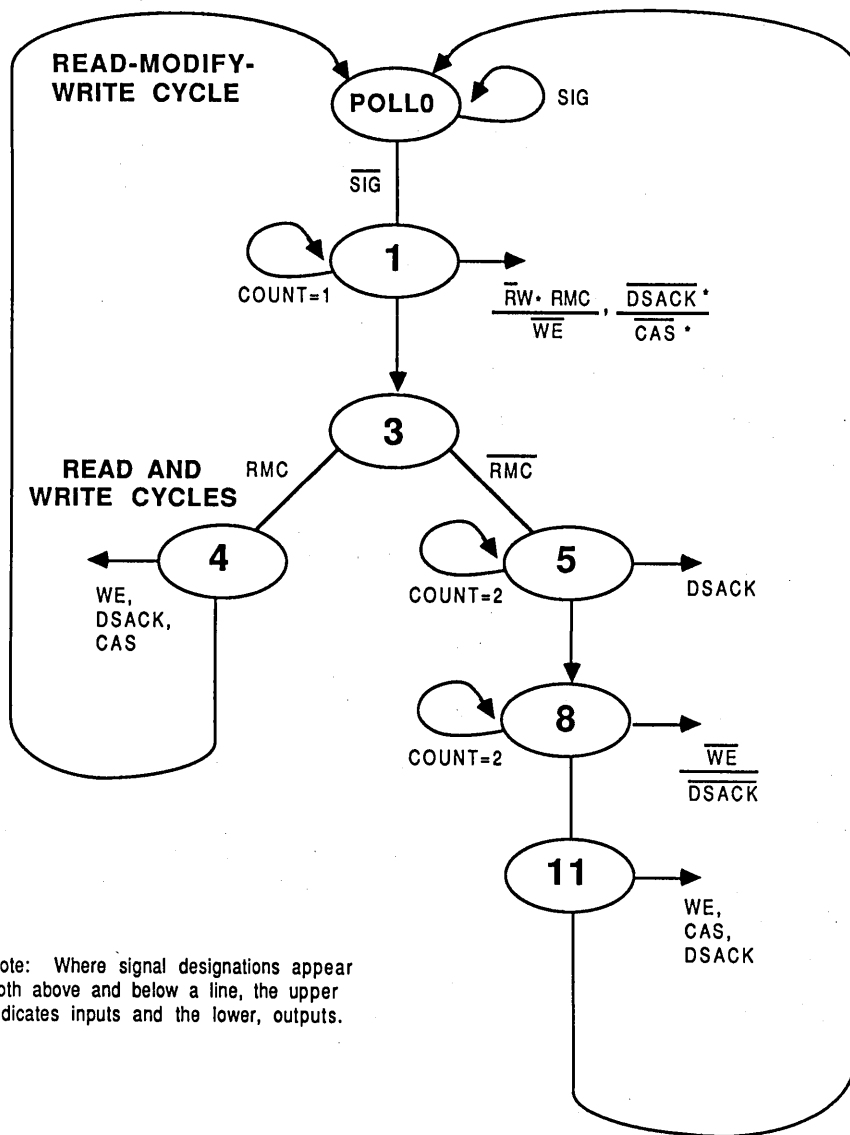
An additional function performed by this PAL device is to inform the Data Sizing and Alignment PAL device of the execution timing reference of a processor cycle. It does this by asserting SIG signal LOW for all processor cycles. This signal is basically used as a synchronizing signal between these two independent PAL devices.

Data Sizing and Alignment PAL

The Data Sizing and Alignment PAL device performs the following functions:

1. CAS generation (for dynamic sizing and alignment)
2. DSACK generation (for dynamic sizing and alignment)
3. WE generation for write/read-modify-write cycles.

The Data Sizing and Alignment PAL device's state machine diagram (shown in Figure 3-199) is synchronized to the Timing and Arbitration PAL device's state machine by using signal SIG. This PAL device has no function for a refresh cycle. For a processor cycle, it asserts appropriate signals CAS0-CAS3, WE and generates the cycle-complete signals DSACK0-DSACK1 to the system processor. The state machine functions are implemented in the four buried state registers on this device.



08479A-25

*Outputs independent
of RW and RMC

Figure 3-199. State Diagram for the Data Sizing and Alignment PAL

CAS Generation

The state machine starts at initial polling state 0, and remains at this state until receiving a SIG signal from the Timing and Arbitration PAL device indicating the start of a processor cycle. The maximum allowable data width for the 68020 processor is 32 bits. The data bus is split into four-byte wide segments. The CAS signals for these four-byte segments are independently controlled, allowing independent byte-wide read-write capability. These four CAS 0-3 signals strobe the column address for the LOW-to-HIGH order bytes of memory. They also allow read and write for low-low, low-middle, high-middle and high-high bytes of a 32-bit data port. For 16-bit data port size, CAS-2-3 provides the low- and high-byte strobes respectively. For an 8-bit port, only CAS3 is used.

In state 1 the CAS signals are asserted. Which CAS signals are asserted depends upon the PORT size available, the DATA size transfer being attempted by the processor, and the alignment of transfer required. As explained before, the attempted transfer data size is encoded on two SIZE0, SIZE1 bits by the processor and the alignment is indicated by the least significant address bits A0, A1 of the processor.

The port size is very application-dependent. Different systems might require either static or dynamic selection of this port size. For example the designer can define different address ranges where the port size may be different. Alternately, jumpers can also define the port size. For illustrative purposes general-purpose means are provided to control data-port size and the consequent generation of CAS signals. Two input pins PRT0 & PRT1 define the size of the port.

Port Size

As described above, based on the input pins PRT1, PRT0 the size of data bus is decided.

PRT0	PRT1	SIZE
L	L	8 bits
L	H	16 bits
H	L	16 bits
H	H	32 bits

For 32-bit ports, all four CAS signals need to be asserted, but for a 16-bit port, only two CAS signals need to be asserted. For

an 8-bit port CAS3 is always asserted. The processor always attempts to transfer the maximum possible amount of data. Its SIZE0,1 signal encodes the amount of data being transferred.

The table below illustrates the relationship of the data size, port size, and alignment that is decided by address bits A1, A0. Assertion of appropriate CAS signals is dependent upon: port size PRT0, PRT1 or; attempted transfer data size SIZE0, SIZE1 and address bits A0, A1 shown in Table 3-25.

The data from Table 3-25 is used in the state description syntax to generate the CAS signals for state 1. The software compiles the boolean equations for this logic directly.

Once asserted, the CAS signals are held asserted for the requisite timing durations by the state machine until the appropriate read-write or read-modify-write cycles are completed.

DSACK Acknowledge to CPU

The amount of data received by any device is dependent upon its port size. Based on the port size PRT0, 1, the encodings for DSACK0, 1 are generated to acknowledge the CPU and inform it of the port size. The acknowledgment timing is synchronized by the state machine. For read-write cycles, the DSACK signals are generated in state 1 and removed on cycle completion in state 4.

For read-modify-write cycle a slight complication exists. The read-modify-write cycle requires two DSACK signals for read and write. The first DSACK (used for both R/W & RMW cycles) is generated in state 1 and removed in either state 4 or state 5. The second, required for read-modify-write cycle only is generated in state 8 and removed in state 11.

WE Generation

For the read-write cycle only, in state 1 based on RMC HIGH (not a read-modify-write cycle) and R/W being LOW (write cycle), the WE is generated for normal write cycles. For read-modify-write cycle, a delayed WE signal is generated always based upon the internal 4-bit state counter in state 8 (Figure 3-196). Figure 3-200.a shows the high-level language specifications for this PAL (using AmCUPPL). Figure 3-200.b shows the corresponding Boolean logic equations; Figure 3-200.c shows the statistics for this PAL device, and Figure 3-200.d shows the fuse plot for this device.

TABLE 3-25. DATA BUS ACTIVITY FOR BYTE, WORD, AND LONG WORD PORTS

Transfer Size	SIZ1	SIZ0	A1	A0	Data Bus Active Sections Byte (B)—Word (W)—Long Word (L) Ports			
					D31–D24	D23–D16	D15–D8	D7–D0
Byte	0	1	0	0	B W L	—	—	—
	0	1	0	1	B	W L	—	—
	0	1	1	0	B W	—	L	—
	0	1	1	1	B	W	—	L
Word	1	0	0	0	B W L	W L	—	—
	1	0	0	1	B	W L	L	—
	1	0	1	0	B W	W	L	L
	1	0	1	1	B	W	—	L
Three-Byte	1	1	0	0	B W L	W L	L	—
	1	1	0	1	B	W L	L	L
	1	1	1	0	B W	W	L	L
	1	1	1	1	B	W	—	L
Long Word	0	0	0	0	B W L	W L	L	L
	0	0	0	1	B	W L	L	L
	0	0	1	0	B W	W	L	L
	0	0	1	1	B	W	—	L

```

NAME      MEGABYT DRAM DATA SIZING & ALIGNMENT CONTROLLER;
PARTNO    AmPAL 23S8 ;
DATE      03/24/86 ;
REV       01 ;
DESIGNER   KAPIL SHANKAR ;
COMPANY    ADVANCED MICRO DEVICES ;
ASSEMBLY   NONE ;
LOCATION    SUNNYVALE, CA ;
DEVICE     P23S8 ;

```

```

/*****
/*
/*
/*
/*
/*****
/* Allowable Target Device Types:  THE ONLY ONE
/*****

```

```

/** Inputs **/

```

```

PIN 2    = SIG ;
PIN 3    = A0 ;
PIN 4    = A1 ;
PIN 5    = RW ;
PIN 6    = RMC ;
PIN 7    = SIZ0 ;
PIN 8    = SIZ1 ;
PIN 9    = PRT0 ;
PIN 11   = PRT1 ;
PIN 12   = INIT ;

```

```

/** Outputs **/

```

```

PIN 19   = !WE ;
PIN 18   = !CAS2 ;
PIN 17   = !CAS0 ;
PIN 16   = !CAS3 ;
PIN 15   = !CAS1 ;
PIN 14   = !DSACK1 ;
PIN 13   = !DSACK0 ;

```

```

NODE [B0,B1,B2,B3] ;

```

```

/** Declarations and Intermediate Variable Definitions **/

```

```

FIELD STM = [B0,B1,B2,B3] ;           /** THE STATE MACHINE **/

```

```

FIELD PORTSIZE = [PRT1, PRT0] ;       /** SIZE OF THE PORT **/
PORT8 = PORTSIZE : 0 ;                 /** INTERFACED TO PROCESSOR **/
PORT16 = PORTSIZE : 2 ;                /** 8 BIT WIDE PORT **/
PORT32 = PORTSIZE : 3 ;                /** 16 BIT WIDE PORT **/

```

```

/** DEFINE THE STATES **/

```

```

$DEFINE SA0 'b'0000                    /** STATE 0 **/
$DEFINE SA1 'b'0001                    /** STATE 1 **/
$DEFINE SA2 'b'0010                    /** STATE 2 **/
$DEFINE SA3 'b'0011                    /** STATE 3 **/
$DEFINE SA4 'b'0100                    /** STATE 4 **/
$DEFINE SA5 'b'0101                    /** STATE 5 **/
$DEFINE SA6 'b'0110                    /** STATE 6 **/
$DEFINE SA7 'b'0111                    /** STATE 7 **/
$DEFINE SA8 'b'1000                    /** STATE 8 **/
$DEFINE SA9 'b'1001                    /** STATE 9 **/
$DEFINE SA10 'b'1010                   /** STATE 10 **/
$DEFINE SA11 'b'1011                   /** STATE 11 **/

```

```

/** Logic Equations **/

```

```

/** THE STATE MACHINE **/

```

```

SEQUENCE STM (

```

```

/** POLLING FOR SIG INDICATING A PROCESSOR CYCLE **/

```

Figure 3-200.a Source Code for Data Sizing and Alignment PAL

```

/**      PORT      DATA      OFFSET STATE      OUTPUT **/
PRESENT SA0
/** FOR PORTSIZE 8 **/
    IF !SIG & PORT8                      NEXT SA1 OUT CAS3 ;

/** GENERATE ACKNOWLEDGE **/
    IF !SIG & PORT8                      NEXT SA1 OUT DSACK0 ;

/** FOR PORTSIZE 16 **/
    IF !SIG & PORT16 & !A0                NEXT SA1 OUT CAS3 ;
    IF !SIG & PORT16 & !SIZ0 & !SIZ1 & A0 NEXT SA1 OUT CAS2 ;

/** GENERATE ACKNOWLEDGE **/
    IF !SIG & PORT16                      NEXT SA1 OUT DSACK1 ;

/** FOR PORTSIZE 32 **/
/** BYTE TRANSFER SIZE **/
    IF !SIG & PORT32 & !A1 & !A0          NEXT SA1 OUT CAS3 ;
    IF !SIG & PORT32 & !A1 & A0           NEXT SA1 OUT CAS2 ;
    IF !SIG & PORT32 & !A1 & !SIZ0        NEXT SA1 OUT CAS2 ;
    IF !SIG & PORT32 & !A1 & !SIZ1        NEXT SA1 OUT CAS2 ;
    IF !SIG & PORT32 & A1 & !A0           NEXT SA1 OUT CAS1 ;
    IF !SIG & PORT32 & !A1 & !SIZ1 & !SIZ0 NEXT SA1 OUT CAS1 ;
    IF !SIG & PORT32 & !A1 & !SIZ1 & !SIZ0 NEXT SA1 OUT CAS1 ;
    IF !SIG & PORT32 & !A1 & A0 & !SIZ0   NEXT SA1 OUT CAS1 ;
    IF !SIG & PORT32 & A1 & A0            NEXT SA1 OUT CAS3 ;
    IF !SIG & PORT32 & !SIZ1 & !SIZ0      NEXT SA1 OUT CAS2 ;
    IF !SIG & PORT32 & A0 & !SIZ1 & !SIZ0 NEXT SA1 OUT CAS1 ;

/** GENERATE ACKNOWLEDGE **/
    IF !SIG & PORT32                      NEXT SA1 OUT DSACK0
                                         OUT DSACK1 ;

/** JUMP TO STATE SA1 **/
/** ASSERT THE CAS FOR THE RIGHT PORTSIZE DATASIZE AND ADDRESS OFFSET **/

    IF !SIG & !RW & !RMC                  NEXT SA1 OUT WE ;
/** GENERATE WE FOR READ WRITE CYCLE ONLY **/

    IF SIG                                NEXT SA0 ;

/** ELSE WAIT FOR THE PROCESSOR CYCLE **/
/** POLL AGAIN **/
/** STAY IN STATE SA0 **/

/** THIS IS THE PROCESSOR CYCLE START **/
PRESENT SA1
    IF CAS0                      NEXT SA2 OUT CAS0 ;
    IF CAS1                      NEXT SA2 OUT CAS1 ;
    IF CAS2                      NEXT SA2 OUT CAS2 ;
    IF CAS3                      NEXT SA2 OUT CAS3 ;
    IF !RW & !RMC                NEXT SA2 OUT WE ;
    IF PORT8                     NEXT SA2 OUT DSACK0 ;
    IF PORT16                    NEXT SA2 OUT DSACK1 ;
    IF PORT32                    NEXT SA2 OUT DSACK0
                                         OUT DSACK1 ;
                                NEXT SA2 ;

/** HOLD CAS GENERATED IN THE LAST STATE **/
/** IF WRITE CYCLE AND NOT READ MODIFY WRITE CYCLE ASSERT WE **/
/** JUMP TO STATE SA2 **/
PRESENT SA2
    IF CAS0                      NEXT SA3 OUT CAS0 ;
    IF CAS1                      NEXT SA3 OUT CAS1 ;
    IF CAS2                      NEXT SA3 OUT CAS2 ;
    IF CAS3                      NEXT SA3 OUT CAS3 ;
    IF !RW & !RMC                NEXT SA3 OUT WE ;
    IF PORT8                     NEXT SA3 OUT DSACK0 ;
    IF PORT16                    NEXT SA3 OUT DSACK1 ;
    IF PORT32                    NEXT SA3 OUT DSACK0
                                         OUT DSACK1 ;
                                NEXT SA3 ;

/** HOLD CAS GENERATED IN THE LAST STATE **/
/** HOLD WE FOR ACCESS DURATION **/
/** JUMP TO STATE SA3 **/
PRESENT SA3
    IF RMC                      NEXT SA4 OUT !CAS0
                                         OUT !CAS1
                                         OUT !CAS2

```

Figure 3-200.a Source Code for Data Sizing and Alignment PAL (Continued)

```

        OUT !CAS3
        OUT !WE
        OUT !DSACK0
        OUT !DSACK1 ;

/** FOR READ WRITE CYCLE REMOVE CAS AND WE FOR CYCLE COMPLETION **/
/** ACKNOWLEDGE CYCLE COMPLETION BY ACKNOT **/
/** JUMP TO STATE SA4 **/

    IF CAS0 & !RMC      NEXT SA5 OUT CAS0 ;
    IF CAS1 & !RMC      NEXT SA5 OUT CAS1 ;
    IF CAS2 & !RMC      NEXT SA5 OUT CAS2 ;
    IF CAS3 & !RMC      NEXT SA5 OUT CAS3 ;
    IF !RMC             NEXT SA5 OUT !DSACK0
                        OUT !DSACK1 ;

/** FOR READ MODIFY WRITE CYCLE CONTINUE TO ASSERT CAS SIGNALS **/
/** ACKNOWLEDGE COMPLETION OF READ PORTION OF THE CYCLE BY ACKNOT **/
/** JUMP TO STATE SA5 **/

/** REMAINING PORTION OF THE READ WRITE CYCLE **/

PRESENT SA4      NEXT SA0 ;
/** GO BACK TO POLLING STATE SA0 **/

/** COMPLETION OF READ WRITE CYCLE **/

/** REMAINING PORTION OF THE READ MODIFY WRITE CYCLE **/

PRESENT SA5
    IF CAS0      NEXT SA6 OUT CAS0 ;
    IF CAS1      NEXT SA6 OUT CAS1 ;
    IF CAS2      NEXT SA6 OUT CAS2 ;
    IF CAS3      NEXT SA6 OUT CAS3 ;
                NEXT SA6 ;

        /** CONTINUE THE READ MODIFY WRITE CYCLE **/

PRESENT SA6
    IF CAS0      NEXT SA7 OUT CAS0 ;
    IF CAS1      NEXT SA7 OUT CAS1 ;
    IF CAS2      NEXT SA7 OUT CAS2 ;
    IF CAS3      NEXT SA7 OUT CAS3 ;
                NEXT SA7 ;

        /** CONTINUE THE READ MODIFY WRITE CYCLE **/

PRESENT SA7
    IF CAS0      NEXT SA8 OUT CAS0 ;
    IF CAS1      NEXT SA8 OUT CAS1 ;
    IF CAS2      NEXT SA8 OUT CAS2 ;
    IF CAS3      NEXT SA8 OUT CAS3 ;
    IF PORT8     NEXT SA8 OUT !DSACK0 ;
    IF PORT16    NEXT SA8 OUT !DSACK1 ;
    IF PORT32    NEXT SA8 OUT !DSACK0
                OUT !DSACK1 ;
                NEXT SA8 OUT WE ;

        /** THIS IS THE WRITE PORTION OF THE READ MODIFY WRITE CYCLE **/
        /** GENERATE THE WE SIGNAL FOR ENABLING WRITE **/
        /** GENERATE THE CORRECT ACKNOWLEDGE FOR THE WRITE CYCLE PORTION **/
        /** CONTINUE TO ASSERT CAS **/

PRESENT SA8
    IF CAS0      NEXT SA9 OUT CAS0 ;
    IF CAS1      NEXT SA9 OUT CAS1 ;
    IF CAS2      NEXT SA9 OUT CAS2 ;
    IF CAS3      NEXT SA9 OUT CAS3 ;
    IF PORT8     NEXT SA9 OUT !DSACK0 ;
    IF PORT16    NEXT SA9 OUT !DSACK1 ;
    IF PORT32    NEXT SA9 OUT !DSACK0
                OUT !DSACK1 ;
                NEXT SA9 OUT WE ;

        /** THIS IS THE WRITE PORTION OF THE READ MODIFY WRITE CYCLE **/

```

Figure 3-200.a Source Code for Data Sizing and Alignment PAL (Continued)


```

PRESENT SA9
  IF CAS0
  IF CAS1
  IF CAS2
  IF CAS3
  IF PORT8
  IF PORT16
  IF PORT32
    NEXT SA10 OUT CAS0 ;
    NEXT SA10 OUT CAS1 ;
    NEXT SA10 OUT CAS2 ;
    NEXT SA10 OUT CAS3 ;
    NEXT SA10 OUT DSACK0 ;
    NEXT SA10 OUT DSACK1 ;
    NEXT SA10 OUT DSACK0
      OUT DSACK1 ;
    NEXT SA10 OUT WE ;

/** THIS IS THE WRITE PORTION OF THE READ MODIFY WRITE CYCLE **/

PRESENT SA10
    NEXT SA11 OUT !CAS0
      OUT !CAS1
      OUT !CAS2
      OUT !CAS3
      OUT !DSACK0
      OUT !DSACK1
      OUT !WE ;

/** COMPLETION OF THE WRITE PORTION OF THE READ MODIFY WRITE CYCLE **/
/** REMOVE ALL CAS WE AND ACKNOWLEDGE COMPLETION **/

/** REMAINING PORTION OF THE READ MODIFY WRITE CYCLE **/

PRESENT SA11
  /** GO BACK TO POLLING STATE SA0 **/
  /** COMPLETION OF THE READ MODIFY WRITE CYCLE **/
}
/** THE STATE MACHINE COMPLETE **/

/** EQUATIONS FOR SYSTEM INITIALIZATION AFTER RESET **/

[B0..B3].ar = !INIT ;

CAS0.AR = !INIT ;
CAS1.AR = !INIT ;
DSACK0.AR = !INIT ;
DSACK1.AR = !INIT ;
CAS2.AR = !INIT ;
CAS3.AR = !INIT ;
WE.AR = !INIT ;

/* THESE ARE THE OUTPUT ENABLES */

CAS0.OE = 'B'1 ;
CAS1.OE = 'B'1 ;
DSACK0.OE = 'B'1 ;
DSACK1.OE = 'B'1 ;
CAS2.OE = 'B'1 ;
CAS3.OE = 'B'1 ;
WE.OE = 'B'1 ;

```

Figure 3-200.a Source Code for Data Sizing and Alignment PAL (Continued)

MEGABYT

CUPL 2.11a Serial# 9-99999-999
 Device p23s8 Library DL1B-e-22-4
 Created Wed Jul 30 10:13:11 1986
 Name MEGABYT DRAM DATA SIZING & ALIGNMENT CONTROLLER
 Partno AmPAL 23S8
 Revision 01
 Date 03/24/86
 Designer KAPIL SHANKAR
 Company ADVANCED MICRO DEVICES
 Assembly NONE
 Location SUNNYVALE, CA

=====

Expanded Product Terms

=====

B0.ar =>

!INIT

B0.d =>

!B0 & !B2 & B3

B0 & B1 & !B2 & !B3 & !RMC

!PRTO & !B0 & !B1 & !B2 & !B3 & !SIG

!B0 & B1 & !B3

!B0 & !B1 & !B2 & !B3 & RMC & !SIG & !RW

PRTO & PRT1 & !B0 & !B1 & !B2 & !B3 & !SIG

B1.ar =>

!INIT

B1.d =>

!B0 & B1 & !B2 & B3

B0 & !B1 & !B2 & B3

!B0 & B1 & !B3

PRTO & PRT1 & B0 & !B1 & !B2

!PRTO & B0 & !B1 & !B2

B0 & !B1 & !B3

DSACK0.ar =>

!INIT

DSACK0.oe =>

1

DSACK0.d =>

!PRTO & !PRT1 & !B0 & !B1 & !B2 & !B3 & !SIG

PRTO & PRT1 & !B0 & !B1 & !B2 & !B3 & !SIG

!PRTO & !PRT1 & B0 & !B1 & !B2

!PRTO & !PRT1 & !B0 & B1 & !B2 & !B3

PRTO & PRT1 & !B0 & B1 & !B2 & !B3

PRTO & PRT1 & !B1 & !B2 & B3

!PRTO & !PRT1 & B0 & B1 & B2 & !B3

PRTO & PRT1 & B0 & B1 & B2 & !B3

!PRTO & !PRT1 & !B0 & !B1 & !B2 & B3

B2.ar =>

!INIT

B2.d =>

!B0 & B1 & B2 & !B3

B0 & !B1 & B2 & !B3

B0 & B1 & !B2 & !B3

DSACK1.ar =>

!INIT

DSACK1.oe =>

1

DSACK1.d =>

PRT1 & !B0 & !B1 & !B2 & !B3 & !SIG

PRT1 & B0 & !B1 & !B2

PRT1 & !B0 & B1 & !B2 & !B3

PRT1 & !B0 & !B1 & !B2 & B3

PRT1 & B0 & B1 & B2 & !B3

B3.ar =>

!INIT

Figure 3-200.b Boolean Logic Equations for the Data Sizing and Alignment PAL

```

B3.d =>
    !B0 & B1 & !B2 & B3
    # !B1 & !B2 & B3
    # B0 & B1 & B2 & !B3

PORT32 =>
    PRT0 & PRT1

PORT16 =>
    !PRT0 & PRT1

DSACKO =>
    PRT0 & PRT1 & B0 & !B1 & !B2 & !B3

STM =>
    B0 , B1 , B2 , B3

PORTSIZE =>
    PRT1 , PRT0

WE.ar =>
    !INIT

WE.oe =>
    1

WE.d =>
    !B0 & !B1 & !B2 & !B3 & RMC & !SIG & !RW
    # B0 & !B1 & !B2 & !B3 & RMC & !RW
    # !B0 & B1 & !B2 & !B3 & RMC & !RW
    # !B1 & !B2 & B3
    # B0 & B1 & B2 & !B3

CAS0.ar =>
    !INIT

CAS0.oe =>
    1

CAS0.d =>
    B0 & !B1 & !B3 & CAS0

# !B0 & B1 & !B3 & CAS0
# !B1 & !B2 & B3 & CAS0
# B0 & B1 & !B2 & !B3 & !RMC & CAS0
# B0 & B1 & B2 & !B3 & CAS0

CAS1.ar =>
    !INIT

CAS1.oe =>
    1

CAS1.d =>
    PRT0 & PRT1 & !A0 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & !SIZ0 & PRT1 & !SIZ1 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & SIZ0 & PRT1 & SIZ1 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & !SIZ0 & PRT1 & A0 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & SIZ0 & PRT1 & SIZ1 & A0 & !B0 & !B1 & !B2 & !B3 & !SIG
    # B0 & !B1 & !B3 & CAS1
    # !B0 & B1 & !B3 & CAS1
    # !B1 & !B2 & B3 & CAS1
    # B0 & B1 & !B2 & !B3 & !RMC & CAS1
    # B0 & B1 & B2 & !B3 & CAS1

CAS2.ar =>
    !INIT

CAS2.oe =>
    1

CAS2.d =>
    !PRT0 & !SIZ0 & PRT1 & SIZ1 & A0 & !B0 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & PRT1 & A0 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & !SIZ0 & PRT1 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & PRT1 & SIZ1 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
    # PRT0 & !SIZ0 & PRT1 & !SIZ1 & !B0 & !B1 & !B2 & !B3 & !SIG
    # B0 & !B1 & !B3 & CAS2
    # !B0 & B1 & !B3 & CAS2
    # !B1 & !B2 & B3 & CAS2
    # B0 & B1 & !B2 & !B3 & !RMC & CAS2
    # B0 & B1 & B2 & !B3 & CAS2

CAS3.ar =>

```

Figure 3-200.b Boolean Logic Equations for the Data Sizing and Alignment PAL (Continued)

```

!INIT
CAS3.oe =>
1
CAS3.d =>
!PRT0 & !PRT1 & !B0 & !B1 & !B2 & !B3 & !SIG
# !PRT0 & PRT1 & !A0 & !B0 & !B1 & !B2 & !B3 & !SIG
# PRT0 & PRT1 & !A0 & !B0 & !A1 & !B1 & !B2 & !B3 & !SIG
# PRT0 & PRT1 & A0 & !B0 & A1 & !B1 & !B2 & !B3 & !SIG
# B0 & !B1 & !B3 & CAS3
# !B0 & B1 & !B3 & CAS3
# !B1 & !B2 & B3 & CAS3
# B0 & B1 & !B2 & !B3 & !RMC & CAS3
# B0 & B1 & B2 & !B3 & CAS3

```

```

PORT8 =>
!PRT0 & !PRT1

```

```

INIT.oe =>
0

```

=====

Symbol Table

=====

Pin Variable		Ext	Pin	Type	Pterms	Max	Min
Pol	Name				Used	Pterms	Level
---	-----	---	---	----	-----	-----	-----
	PRT0		9	V	-	-	-
	SIZ0		7	V	-	-	-
	PRT1		11	V	-	-	-
	SIZ1		8	V	-	-	-
	A0		3	V	-	-	-
	B0		21	N	-	-	-
	B0	ar	21	M	1	1	1
	B0	d	21	M	6	10	1
	INIT		12	V	-	-	-
	A1		4	V	-	-	-
	B1		22	N	-	-	-

B1	ar	22	M	1	1	1
B1	d	22	M	6	10	1
! DSACK0		13	V	-	-	-
DSACK0	ar	13	X	1	1	1
! DSACK0	oe	13	X	1	1	1
! DSACK0	d	13	X	9	10	1
B2		23	N	-	-	-
B2	ar	23	M	1	1	1
B2	d	23	M	3	8	1
! DSACK1		14	V	-	-	-
DSACK1	ar	14	X	1	1	1
! DSACK1	oe	14	X	1	1	1
! DSACK1	d	14	X	5	8	1
B3		24	N	-	-	-
B3	ar	24	M	1	1	1
B3	d	24	M	3	8	1
RMC		6	V	-	-	-
SIG		2	V	-	-	-
PORT32		0	I	1	-	-
PORT16		0	I	1	-	-
DSACK0	d	0	I	1	-	-
STM		0	F	-	-	-
PORTSIZE		0	F	-	-	-
! WE		19	V	-	-	-
WE	ar	19	X	1	1	1
! WE	oe	19	X	1	1	1
! WE	d	19	X	5	8	1
! CAS0		17	V	-	-	-
CAS0	ar	17	X	1	1	1
! CAS0	oe	17	X	1	1	1
! CAS0	d	17	X	5	8	1
! CAS1		15	V	-	-	-
CAS1	ar	15	X	1	1	1
! CAS1	oe	15	X	1	1	1
! CAS1	d	15	X	10	12	1
! CAS2		18	V	-	-	-
CAS2	ar	18	X	1	1	1
! CAS2	oe	18	X	1	1	1
! CAS2	d	18	X	10	10	1
! CAS3		16	V	-	-	-
CAS3	ar	16	X	1	1	1
! CAS3	oe	16	X	1	1	1

Figure 3-200.c Statistics for Data Sizing and Alignment PAL Device

LEGEND F : field D : default variable M : extended node
 N : node I : intermediate variable T : function
 V : variable X : extended variable U : undefined

OBS
6072 xxx

SP

6118 xxx

AR
6164 -----x

Pin #19 6210 Mode -xxx

```

0000 -----
0046 -x-----x-x-x--x-x--x-
0092 -----x-x-x--xx--x-
0138 -----x-x-x--x-x--x-
0184 -----x--x--x-
0230 -----x--x-x--x-
0276 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0322 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0368 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #18 6214 Mode -xxx

```

0414 -----
0460 -x-x-----x-x-x--xx-x--x-x-----
0506 -x-x-x-x-----x-x-x-x--x-x--x-x-----
0552 -x-----x-----x-x-x-x--xx-x-x-----
0598 -x-----x-----x-x-x-x--x-x--x-x-----
0644 -x-----x-x-x-x--xx-x-x-x-x-----
0690 -x-----xx-x-----

```

```

0736 -----x-----x-x-x-----
0782 -----x-----x-x-----
0828 -----x-----xx-x-x-----
0874 -----x-----x-x-x-----

```

Node 25

```
0920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0966 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1012 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1058 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1104 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1150 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Pin #17 6218 Mode -x

```

1196 -----
1242 -----X-X-----XX--X-
1288 -----X--X--X-X--X-
1334 -----X--X--X--X--X-
1380 -----X-XX--X-X--X-
1426 -----X--X--X-X--X-
1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1518 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1564 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Node 23

```

1610 -----X-X-X-X-----
1656 -----X--XX--X-----
1702 -----X--X-X--X-----
1748 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1794 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1840 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1886 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1932 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

Pin #16 6220 Mode -x

1978
2024 -x-.....-x-x-x-.....-x-.....-x-x-.....
2070 -x-x-x-.....-x-x-x-.....-x-.....-x-x-.....
2116 -x-x-x-x-.....-x-x-x-.....-x-.....-x-x-.....
2162 -x-x-x-x-.....-x-x-x-.....-x-.....-x-x-.....
2208-x-xx-.....-x-.....-x-x-.....
2254-xx-x-.....-x-.....
2300-x-x-x-.....-x-.....
2346-xx-x-x-x-.....-x-.....

Figure 3-200.d Fuse Plot and Chip Diagram for Data Sizing and Alignment PAL Device

```

2392 -----x-x-x-x-----x-----
2438 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2484 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2530 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 21
2576 -----x--x-----x-----
2622 -----xx--x-x-----x-----
2668 -x-----x--x-x-----x-----x-----
2714 -----x-x-x-----x-----
2760 -x-----x-x-x-x-----x-----
2806 -x-----x--x-x-----x-----x-x-----
2852 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2898 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2944 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2990 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 22
3036 -----x--x-x-----x-----
3082 -----x--xx-----x-----
3128 -----x-x-x-----x-----
3174 -----x--xx-----x-x-x-----
3220 -----x--xx-----x-----
3266 -----xx-x-----x-----
3312 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3358 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3404 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3450 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #15 6222 Mode -x
3496 -----
3542 -x--x-x-----x--x-x-----x-x-x-----
3588 -x-----x-----x--x-x--xx-x-x-x-----
3634 -x-----x-----x--x-x--x-x-x-----x-x-x-----
3680 -x-x-x-x-----x--x-x--xx-x-x-x-----
3726 -x-x-----x--x-x--x-x-x-----x--x-x-----
3772 -----xx-x-x-x-----
3818 -----x-x-x-x-----
3864 -----x--x-x-x-----
3910 -----xx--x-x-x-x-----
3956 -----x-x-x-x-x-----
4002 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

```

4048 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 24
4094 -----x--x-x-----x-----
4140 -----x--x-----x-----
4186 -----x--x-x-----x-----
4232 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4278 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4324 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4370 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #14 6224 Mode -x
4462 -----
4508 -x-----x--x-x-----x-----x-----
4554 -----x--xx-----x-----
4600 -----x--x-x-----x-----x-----
4646 -----x--x-x-----x-----x-----
4692 -----x--x-x-----x-----x-----
4738 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4784 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4830 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Node 26
4876 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4922 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4968 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5060 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5106 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #13 6226 Mode xxxx
5152 -----
5198 -x-----x--x-x-----x-----x-x-----
5244 -x-----x--x-x-----x-----x-x-----
5290 -----x--xx-----x--x-x-----
5336 -----x--x-x-----x-----x-x-----
5382 -----x-x-x-x-----x-x-x-----
5428 -----x-----x-----x-x-x-----
5474 -----x--x-x-x-----x-x-x-----
5520 -----x--x-x-x-----x-x-x-----
5566 -----x--x-x-x-----x-x-x-----
5612 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12 6230 Mode -xx-

```

Figure 3-200.d Fuse Plot and Chip Diagram for Data Sizing and Alignment PAL Device (Continued)

```

5658 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5750 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5796 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5842 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5934 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5980 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6026 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

```

LEGEND X : fuse not blown
 . : fuse blown

Chip Diagram

```

*****
*      * *      *
****
* 1          20 * VCC
****
*          MEGABYT          *
****
SIG * 2          19 * IWE
****
*          *
****
AO * 3          18 * ICAS2
****
*          *
****
A1 * 4          17 * ICAS0
****
*          *
****
RW * 5          16 * ICAS3
****

```

```

*
****
RMC * 6          15 * ICAS1
****
*
****
SI20 * 7          14 * IDSACK1
****
*
****
SI21 * 8          13 * IDSACK0
****
*
****
PRT0 * 9          12 * INIT
****
*
****
GND * 10          11 * PRT1
****
*
*****

```

Figure 3-200.d Fuse Plot and Chip Diagram for Data Sizing and Alignment PAL Device (Continued)

3.6.6 DYNAMIC MEMORY CONTROL STATE SEQUENCER

An example of a control path application for an AMD PAL is in a memory system. Most large memory systems use MOS dynamic RAMs. Their high density allows packing a large memory size into a small board area. Dynamic RAM prices also make them very cost effective.

Dynamic RAMs require external logic for address multiplexing, timing generation and refresh control. This application note shows the use of an AmPAL16R8A and an Am2964B to provide the necessary external logic for a typical dynamic memory system. The PAL is used as a state sequencer for timing generation and the Am2964B provides specialized control circuitry and reduces timing skew between control signals. This implementation replaces about 20 SSI/MSI packages.

Design Requirements

A system block diagram is shown in Figure 3-201. The control bus provides most of the inputs to the PAL state sequencer. These include: Memory Request (MREQ), READ/WRITE (RW), RESET (RST), Refresh Clock (RFCK), and Read-Modify-Write (RMW). Two upper address lines of the address bus serve as board selects (BS₁, BS₀), and one local signal, SLOW/FAST Memory (FAST), allows use of either slow or fast memory. A READ/WRITE sequence is initialized by MREQ ANDed with the proper board select conditions and a refresh sequence is initialized by RFCK. If both sequences are requested at the same time, a refresh sequence is performed. RW when HIGH selects a READ operation and when LOW selects a WRITE operation. RMW when HIGH selects a Read-Modify-Write cycle.

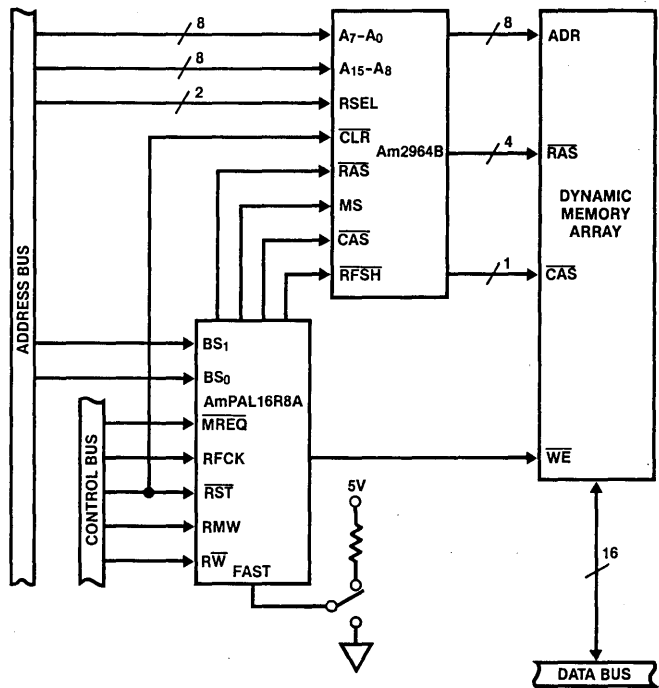
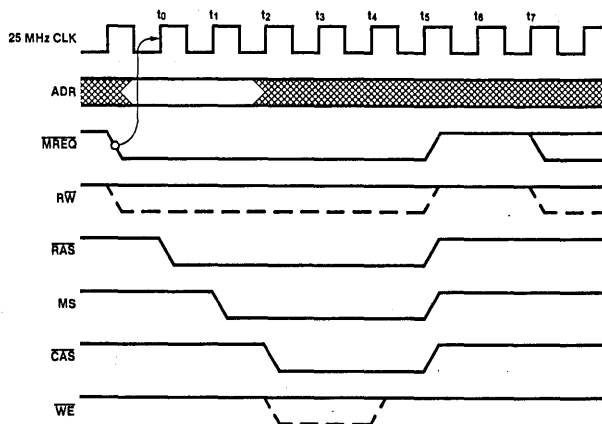


Figure 3-201. Dynamic Memory Controller 03862A-86

The outputs of the PAL provide the timing and control inputs to the Am2964B. These are: Row Address Strobe ($\overline{\text{RAS}}$), Address Multiplexer Select (MS), Column Address Strobe ($\overline{\text{CAS}}$), and Refresh ($\overline{\text{RFSH}}$). In addition, the PAL provides the Write Enable ($\overline{\text{WE}}$) to the Memory Array. Figure 3-202 shows the timing for fast READ/WRITE cycles. The memory cycle is initiated by $\overline{\text{MREQ}}$ going LOW. The PAL responds by bringing $\overline{\text{RAS}}$ LOW at t_0 , followed by MS going LOW at t_1 , and finally bringing $\overline{\text{CAS}}$ LOW at t_2 . If RW is LOW, $\overline{\text{WE}}$ is

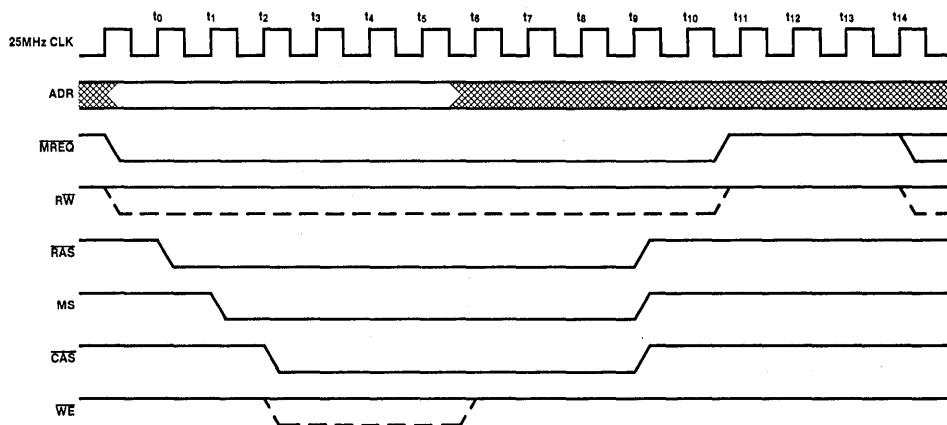
held LOW until t_4 . $\overline{\text{RAS}}$, MS and $\overline{\text{CAS}}$ are brought HIGH at t_5 . The rising edge of any of these 3 signals may be used to latch output data during a Read operation. The state sequencer is then disabled for 3 states to allow for memory precharge.

By holding the FAST input LOW, an extended memory cycle is available to accommodate slower RAMs. The timing appears in Figure 3-203.



03862A-87

Figure 3-202. Fast READ/WRITE Cycle

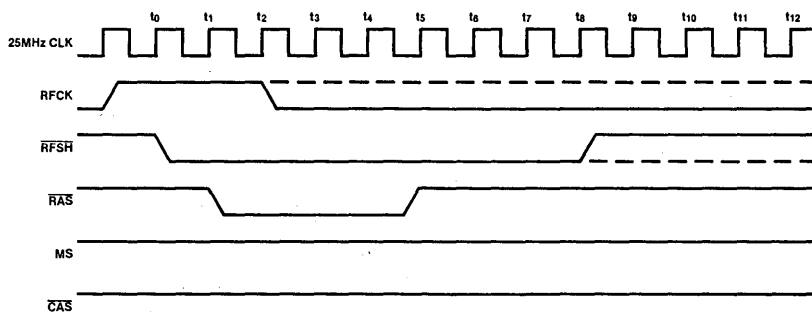


03862A-88

Figure 3-203. Extended Memory Cycle

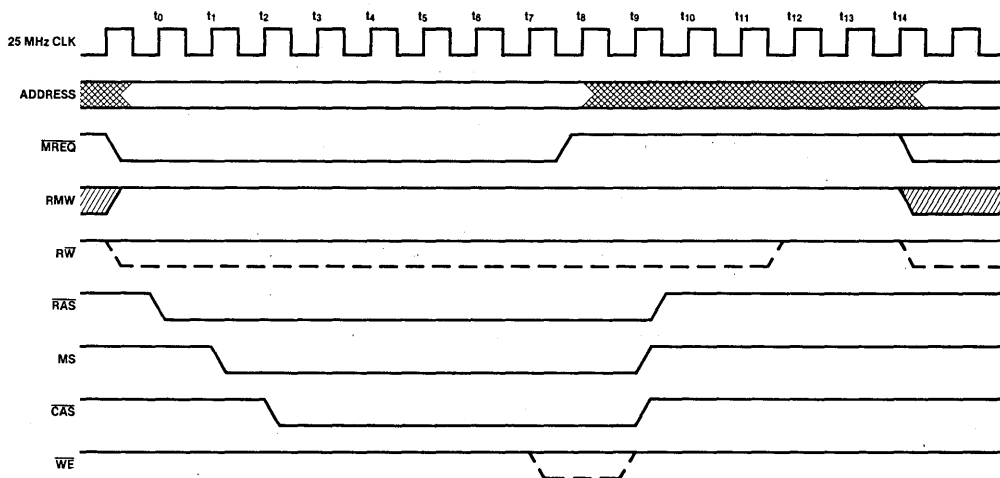
$\overline{\text{RAS}}$ -Only refresh cycle timing is shown in Figure 3-204. The refresh cycle is initiated when RFCK goes HIGH. The RFSH output goes LOW at t_0 , followed by $\overline{\text{RAS}}$ at t_1 . The Am2964B supplied the necessary refresh address. $\overline{\text{RAS}}$ is brought back HIGH at t_5 and precharge is then timed out. An extended refresh cycle for slower memory is available also. Burst refresh can be accomplished by leaving RFCK HIGH for as many refresh cycles as desired.

Read-Modify-Write cycle timing is activated by setting RMW HIGH. This is especially valuable in systems with Error Detection/Correction (EDC) capability. Data can be read, modified by the EDC circuitry (Am2960), and if necessary, written back to memory in a single memory cycle. Read-Modify-Write cycle timing is shown in Figure 3-205. Note that $\overline{\text{WE}}$ goes LOW at the end of the cycle.



03862A-89

Figure 3-204. $\overline{\text{RAS}}$ -Only Refresh Cycle



03862A-90

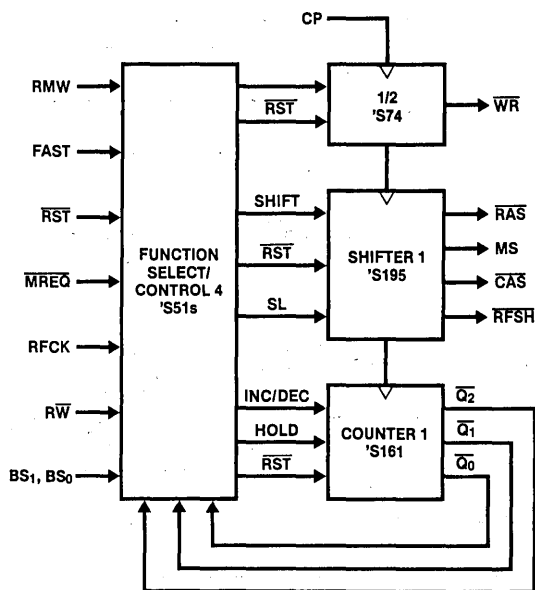
Figure 3-205. Read-Modify-Write Cycle

Design Approach

The first step in the state sequencer design process is to define the timing waveforms for all of the functions desired. Figures 3-202, 3-203, 3-204 and 3-205 are the result. Next, characteristics of the resulting waveforms are examined. Initially, the sequencer is waiting on the $\overline{\text{MREQ}}$ or $\overline{\text{RFCK}}$ input. If $\overline{\text{MREQ}}$ goes LOW, the $\overline{\text{RAS}}$ to $\overline{\text{MS}}$ to $\overline{\text{CAS}}$ sequence is initiated. If $\overline{\text{RFCK}}$ goes HIGH, the $\overline{\text{RFSH}}$ to $\overline{\text{RAS}}$ sequence is initiated. Both sequences are equivalent to a simple "shift" function. Once the shift sequence is completed and the signals are asserted, they must stay asserted for a specific time depending on the selected function. To time the length that signals must stay asserted re-

quires a "counting" function. The precharge sequence at the end of all cycles also requires "counting". This partitions most of the design into two smaller functional blocks; a shifter and a counter. The remaining function select and control logic is partitioned into a "multiplexer-like" functional block. Figure 3-206 shows the PAL partitioned into functional blocks. By dividing the design into blocks, its implementation becomes simple.

Figures 3-207, 3-208, 3-209, and Table 3-26 show PAL equations and a logic diagram for the AmpAL16R8A dynamic memory state sequencer.



03862A-91

Figure 3-206. Partitioned Design/PAL Equivalent

PAL16R8

PAL DESIGN SPECIFICATION

PAT002

BRAD S. KITSON 2/10/82

DYNAMIC MEMORY CONTROL STATE SEQUENCER

ADVANCED MICRO DEVICES

CK RFCK /RST RW /MREQ RMW FAST BS1 BSO GND

/E /Q0 /Q1 /Q2 /RFSH /WE /CAS MS /RAS VCC

```

Q0 := /RST* /MS*/Q0      +
      /RST* RFSH*RAS*/Q0 +
      /RST*/FAST*/Q0*Q2  +
      /RST*/FAST*/Q0*Q1  +
      /RST*/FAST* Q1*Q2   +
      /RST* FAST*/RMW*Q0*/Q1 +
      /RST* FAST*/RMW*Q0*/Q2

Q1 := /RST* RAS*/Q0* Q1   +
      /RST* RAS* Q0*/Q1   +
      /RST*/RAS* Q0* Q1   +
      /RST*/RAS*/Q0* Q2

Q2 := /RST*RAS*Q2        +
      /RST* Q0*Q2        +
      /RST*RAS*Q0*Q1

RFSH := /RST*RFCK*/Q2*/Q1*/Q0*/RAS +
        /RST*RFSH*RAS              +
        /RST*RFSH*/FAST* Q1        +
        /RST*RFSH* Q2

WE := /RST*/RW*/MS*/RFSH*/RMW*/Q0*/Q2 +
      /RST*/RW*/MS*/RFSH*/RMW*/Q1*/Q2 +
      /RST*/RW*/MS*/RFSH* RMW*/Q0* Q1*Q2 +
      /RST*/RW*/MS*/RFSH* RMW* Q0*/Q1*Q2

CAS := /RST*/RFSH*/MS*/Q0      +
      /RST*/RFSH*/MS*/Q1      +
      /RST*/RFSH*/MS*/Q2

/MS := /RST*/RFSH*RAS*/Q0      +
      /RST*/RFSH*RAS*/Q1      +
      /RST*/RFSH*RAS*/Q2

RAS := /RST*/RFCK*/Q0*/Q1*/Q2*MREQ*/BS1*/BSO +
      /RST*/RFSH*/Q0*/Q1*/Q2*MREQ*/BS1*/BSO +
      /RST* RFSH*/Q0*/Q1*/Q2 +
      /RST*RAS*/Q0          +
      /RST*RAS*/Q1          +
      /RST*RAS*/Q2

```

Figure 3-207. Source Listing for Dynamic Memory Control State Sequencer

TABLE 3-26.

FUNCTION TABLE

CK /E /RST /MREQ BS1 BSO RFCK RW RMW FAST RAS /MS CAS WE RFSH Q2 Q1 Q0

; INITIALIZE															
C	L	L	X	X	X	X	X	X	X	L	L	L	L	L	L
; FAST WRITE OPERATION															
C	L	H	L	L	L	L	X	X	X	H	L	L	L	L	L
C	L	H	X	X	X	X	X	X	X	H	H	L	L	L	L
C	L	H	X	X	X	X	L	L	H	H	H	H	L	L	H
C	L	H	X	X	X	X	L	L	H	H	H	H	L	L	H
C	L	H	X	X	X	X	L	L	H	H	H	H	L	L	H
C	L	H	X	X	X	X	L	L	H	L	L	L	L	L	L
C	L	H	X	X	X	X	L	L	H	L	L	L	L	L	L
C	L	H	X	X	X	X	L	L	H	L	L	L	L	L	L
; RAS ONLY REFRESH CYCLE															
C	L	H	H	X	X	H	X	X	X	L	L	L	L	H	L
C	L	H	X	X	X	H	X	X	X	H	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	H	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	H	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	H	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	L	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	L	L	L	L	H	L
C	L	H	X	X	X	X	X	X	H	L	L	L	L	L	L
; READ-MODIFY-WRITE OPERATION															
C	L	H	L	L	L	L	X	X	X	H	L	L	L	L	L
C	L	H	X	X	X	X	X	X	X	H	H	L	L	L	L
C	L	H	X	X	X	X	L	H	L	H	H	H	L	L	H
C	L	H	X	X	X	X	L	H	L	H	H	H	L	L	H
C	L	H	X	X	X	X	L	H	L	H	H	H	L	L	H
C	L	H	X	X	X	X	L	H	L	H	H	H	L	L	H
C	L	H	X	X	X	X	L	H	L	H	H	H	L	L	H
C	L	H	X	X	X	X	L	H	L	L	L	L	L	L	H
C	L	H	X	X	X	X	L	H	L	L	L	L	L	L	H
C	L	H	X	X	X	X	L	H	L	L	L	L	L	L	H

DESCRIPTION

DYNAMIC MEMORY CONTROL STATE SEQUENCER FOR USE WITH THE AM2964B MEMORY CONTROLLER. THE SEQUENCER PROVIDES /RAS,MS,/CAS, & REFRESH TIMING GENERATION TO THE AM2964B AND /WE TO THE DRAMS. IT SUPPORTS BOTH FAST (150NS) AND SLOW (300NS) READ/WRITE CYCLES, /RAS ONLY REFRESH, BURST REFRESH, AND READ-MODIFY-WRITE FOR MEMORY BOARDS OF UP TO 256K.

PAL16R8

PAL DESIGN SPECIFICATION

PAT002

BRAD S. KITSON 2/10/82

DYNAMIC MEMORY CONTROL STATE SEQUENCER

ADVANCED MICRO DEVICES

*D9724

F0

```
L0000 1011 0111 1111 1011 1111 1101 1001 1001 *
L0032 1111 0111 1111 1011 1101 1101 1001 1001 *
L0064 1111 0111 1111 1111 1110 1101 1101 1101 *
L0096 1110 0111 1111 1111 1111 1111 1111 1101 *
L0128 1110 0111 1111 1111 1111 1111 1101 1111 *
L0160 1110 0111 1111 1111 1111 1101 1111 1111 *
L0256 1110 0111 1111 1111 1101 1111 1111 1101 *
L0288 1110 0111 1111 1111 1101 1111 1101 1111 *
L0320 1110 0111 1111 1111 1101 1101 1111 1111 *
L0512 1111 0110 1111 1111 1101 1111 1111 1101 *
L0544 1111 0110 1111 1111 1101 1111 1101 1111 *
L0576 1111 0110 1111 1111 1101 1101 1111 1111 *
L0768 1111 0110 1011 1111 1001 1101 1111 1101 *
L0800 1111 0110 1011 1111 1001 1101 1101 1111 *
L0832 1111 0110 1011 1111 0101 1110 1110 1101 *
L0864 1111 0110 1011 1111 0101 1110 1101 1110 *
L1024 0101 0111 1111 1111 1111 1101 1101 1101 *
L1056 1110 0111 1111 1111 1110 1111 1111 1111 *
L1088 1111 0111 1111 1111 1110 1011 1110 1111 *
L1120 1111 0111 1111 1111 1110 1110 1111 1111 *
L1280 1110 0111 1111 1111 1111 1110 1111 1111 *
L1312 1111 0111 1111 1111 1111 1110 1111 1110 *
L1344 1110 0111 1111 1111 1111 1111 1110 1110 *
L1536 1110 0111 1111 1111 1111 1111 1110 1101 *
L1568 1110 0111 1111 1111 1111 1111 1101 1110 *
L1600 1101 0111 1111 1111 1111 1111 1110 1110 *
L1632 1101 0111 1111 1111 1111 1110 1111 1101 *
L1792 1111 0110 1111 1111 1111 1111 1111 1101 *
L1824 1110 0111 1111 1111 1110 1111 1111 1101 *
L1856 1111 0111 1111 1111 1111 1010 1111 1101 *
L1888 1111 0111 1111 1111 1111 1011 1110 1101 *
L1920 1111 0111 1111 1111 1111 1010 1110 1111 *
L1952 1111 0111 1111 1111 1011 0111 1101 1110 *
L1984 1111 0111 1111 1111 1011 0101 1111 1110 *
```

C713B*

```
V0001 CX0XXXXX00HHHHHHHH1 *
V0002 C01X0XX0000HHHHHHHL1 *
V0003 CX1XXXXX00HHHHHHHL1 *
V0004 CX10X01XX00LHHHLLLL1 *
V0005 CX10X01XX00LLHHLLLL1 *
V0006 CX10X01XX00LHLHLLLL1 *
V0007 CX10X01XX00LLHLHLLL1 *
V0008 CX10X01XX00HHLHHHHH1 *
V0009 CX10X01XX00HLHHHHHH1 *
V0010 CX10X01XX00HHHHHHHH1 *
V0011 C11X1XXXXX00HHLHHHH1 *
V0012 C11XXXXX00HHHLHHHL1 *
V0013 CX1XXX1XX00LHHLHHHL1 *
V0014 CX1XXX1XX00LLHLHHHL1 *
```

Figure 3-208. Fuse Plot and Test Vectors for Dynamic Memory Control State Sequencer

```

V0015 CX1XX1XX00LHLLHHHL1 *
V0016 CX1XX1XX00LLLLHHHL1 *
V0017 CX1XX1XX00HLLHHHH1 *
V0018 CX1XX1XX00HLHLHHHH1 *
V0019 CX1XX1XX00HHHHHHHH1 *
V0020 C01X0X0000HHHHHHHL1 *
V0021 CX1XXXXX00HHHHHHLL1 *
V0022 CX10X10XX00LHHHLLLL1 *
V0023 CX10X10XX00HLHHLLLL1 *
V0024 CX10X10XX00LLHHLLLL1 *
V0025 CX10X10XX00HLLHHLLLL1 *
V0026 CX10X10XX00LHLHHLLLL1 *
V0027 CX10X10XX00HLLHLLLL1 *
V0028 CX10X10XX00LLHLLLL1 *
V0029 CX10X10XX00LHLHHHH1 *
V0030 CX10X10XX00HLLHHHH1 *
V0031 CX10X10XX00LLHHHHH1 *
395F

```

Figure 3-208. Fuse Plot and Test Vectors for Dynamic Memory Control State Sequencer (Continued)

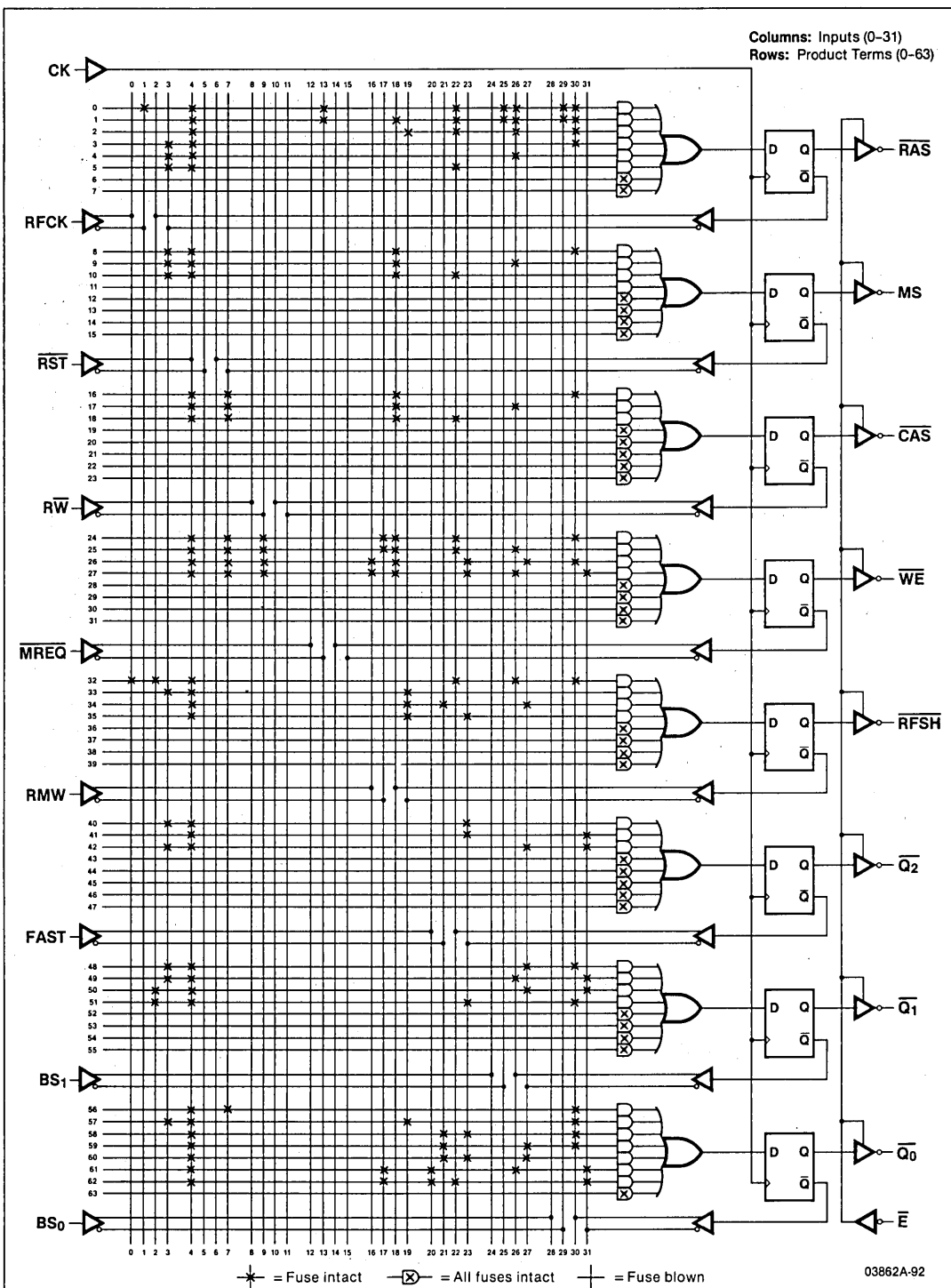


Figure 3-209. Logic Diagram for Dynamic Memory Control State Sequencer Using AmpAL16R8A

3.6.7 82284 and 82288 EMULATION IN AN IBM PC/AT COMPUTER USING TWO AmPAL16R8B DEVICES

Two AmPAL16R8B devices and four TTL packages generate the signals necessary to run an 80286 system at 12.5 MHz. This reduces the cost and improves the speed of the 80286 microsystem. The first PAL device, AmPAL284, is used to emulate the 82284 Clock Driver and Ready Interface functions. The second PAL device, AmPAL288, is used to emulate the 82288 Bus Controller functions. This application generates the signals used in most non-MULTIBUS 80286 systems. The design has been tested in an AT computer. These PAL devices are not pin-for-pin compatible with the parts they emulate nor do they generate all the signals that an 80286 can use including some MULTIBUS signals. However, applications requiring additional signals can be designed by adding additional circuitry.

The simplified Block Diagram below shows the relationship of the PAL devices to the AT computer. Figure 3-210 is the schematic diagram of the emulator and Table I shows the AT interface wiring chart.

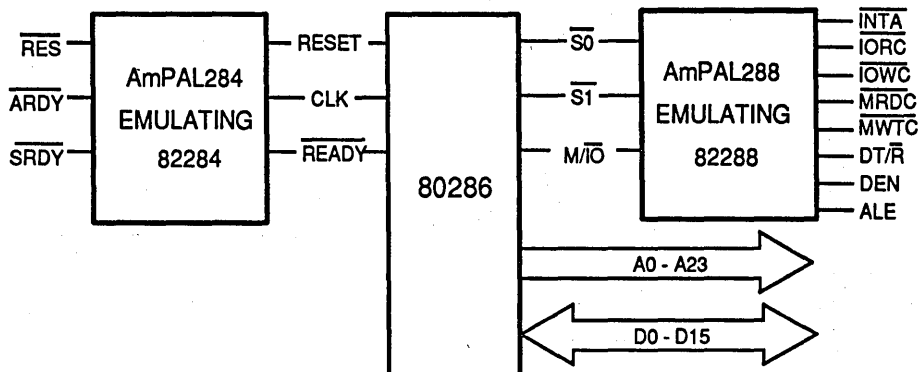
The PAL device equations are written in ABEL and are included in this application note. The source code, the reduced equations, and a fuse map is included for each PAL device.

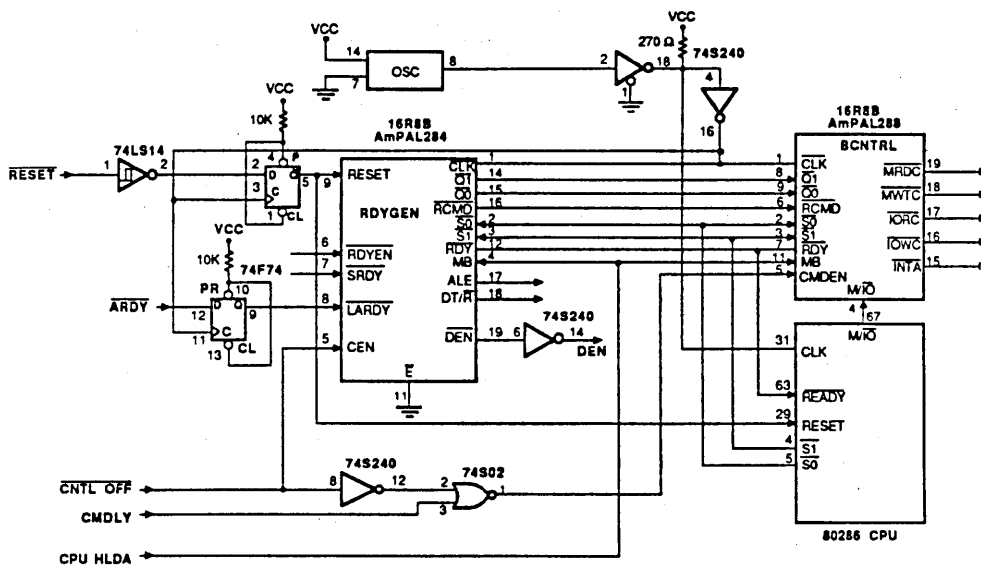
The AmPAL284 Description

A crystal oscillator is used to generate a clock at twice the 80286's internal clock speed. The PAL device is clocked with the inverted CPU clock. The Output Enable pin is grounded to always generate the output signals. The Reset signal is buffered with a Schmitt trigger 74LS14 so that an RC circuit can be used to provide a time delay. After the Schmitt trigger, a 74F74 flip-flop is used to synchronize to the inverted CPU clock. The Ready Enables of the 82284 are connected together on the AT design and are connected to the RDYEN pin of the AmPAL284. The ARDY signal is synchronized to the inverted 80286 clock. The latched ready LARDY and the SRDY are sampled to generate the READY to the 80286.

The state machine shown in Figure 3-211 is used to generate the TS and the TC states. This state machine has four states IDLE, TS2, TC1 and TC2. The state machine requires registered outputs Q0 and Q1 to implement it. The state machine goes to the TS2 state when either $\overline{S0}$ or $\overline{S1}$ goes LOW signalling the start of a bus cycle (refer to Table 3-27 and Figure 3-212). The ALE signal goes active HIGH for the TS2 state and then goes inactive. On the next clock the state machine goes to the TC1 state making $\overline{DEN}$ and $\overline{DT/\overline{R}}$ signals active. The next clock causes the state machine to go to state TC2. The state machine either goes to IDLE, making $\overline{DEN}$ and $\overline{DT/\overline{R}}$ inactive if RDY is active, or to TC1 if RDY is not active. The state machine must return to TC1, if RDY is not active, to keep an even number of clock cycles for each inserted TC state.

Block Diagram





08479A 3-210A

Figure 3-210. Schematic Diagram of the 82284 and 82288 Emulator

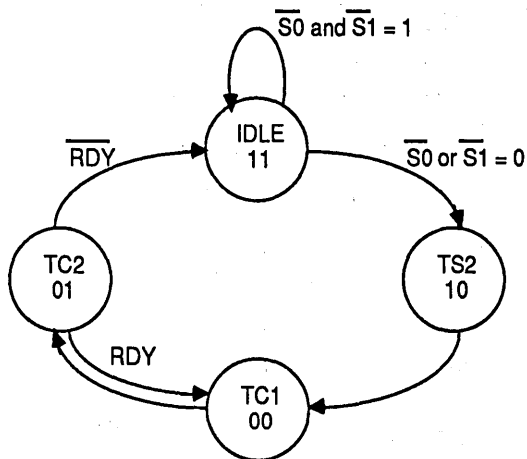
Table 1. Emulator to IBM PC/AT Interface Wiring Chart

Socket Pins on AT Board	Emulator Board Devices (Pin Nos.)				
	AmPAL284	AmPAL288	74F74	74S240	74S02
82284 (Pin No.)					
ARDY (1)	SRDY (7)		D (12)		
SRDY (2)	RDYEN (6)				
ARDYEN (3)	READY (12)	RDY (7)			
READY (4)				Out (18)	
CLK (10)					
RES. (11)			D (2)		
RESET (12)			Q (5)		
82288 (Pin No.)					
S1 (3)	S1 (3)	S1 (3)			
ALE (5)	ALE (17)				
MB (6)	MB (4)	MB (11)			
CMDLY (7)					
MRDC (8)		MRDC (19)			
MWTC (9)		MWTC (18)			
IOWC (11)		IOWC (16)			
IORC (12)		IORC (17)			
INTA (13)		INTA (15)			
CEN (15)	CEN (5)				
DEN (16)					
DT/R (17)	DT/R (18)			In (8)	
M/I/O (18)		M/I/O (4)		Out (14)	
S0 (19)	S0 (2)	S0 (2)			
					In (3)

3.5.7 TBL 1-A

TABLE 3-28. Command and Control Outputs for Each Type of Bus Cycle

Type of Bus Cycle	M/ $\overline{\text{IO}}$	$\overline{\text{S1}}$	$\overline{\text{S0}}$	Command Activated	DT/ $\overline{\text{R}}$ State	ALE, DEN Issued
Interrupt Acknowledge	0	0	0	$\overline{\text{INTA}}$	LOW	YES
I/O Read	0	0	1	$\overline{\text{IORC}}$	LOW	YES
I/O Write	0	1	0	$\overline{\text{IOWC}}$	HIGH	YES
None; idle	0	1	1	None	HIGH	NO
Halt/ Shutdown	1	0	0	None	HIGH	NO
Memory Read	1	0	1	$\overline{\text{MRDC}}$	LOW	YES
Memory Write	1	1	0	$\overline{\text{MWTC}}$	HIGH	YES
None; idle	1	1	1	None	HIGH	NO



	Q1	Q0
IDLE	1	1
TS2	1	0
TC1	0	0
TC2	0	1

08479A 3-211

Figure 3-211. AmpAL State Machine

The $\overline{\text{RCMD}}$ signal goes active when the $\overline{\text{S0}}$ and $\overline{\text{S1}}$ signals decode a Read or Interrupt Acknowledge cycle. This signal is used by the DT/R signal to control the transmit or the receive direction of the data transceivers and by the AmPAL288. The Command and Control outputs for each type of bus cycle are given in Table 3-28. Figure 3-212 shows the cycle timing.

The AmPAL284 source program listing, the reduced equations and the JEDEC fuse map are shown in Figure 3-213.

The AmPAL288 Description

The AmPAL288 uses the Q0, Q1, $\overline{\text{S0}}$, $\overline{\text{S1}}$, and MIO inputs to latch the state of the current cycle. As shown in the PAL equations, these signals are internal signals and are not used by the AT. The $\overline{\text{RCMD}}$ signal is an input and $\overline{\text{WCMD}}$, $\overline{\text{MEM}}$, and $\overline{\text{INT}}$ are internally generated. With these latched signals and the $\overline{\text{CMDEN}}$, the commands are generated. The five commands are $\overline{\text{IOWC}}$, $\overline{\text{IORC}}$, $\overline{\text{INTA}}$, $\overline{\text{MRDC}}$, and $\overline{\text{MWTC}}$. These commands are only enabled when the MB signal is LOW. When MB goes HIGH the commands are high impedance (off).

The AmPAL288 source program listing, reduced equations and the JEDEC fuse map are shown in Figure 3-214.

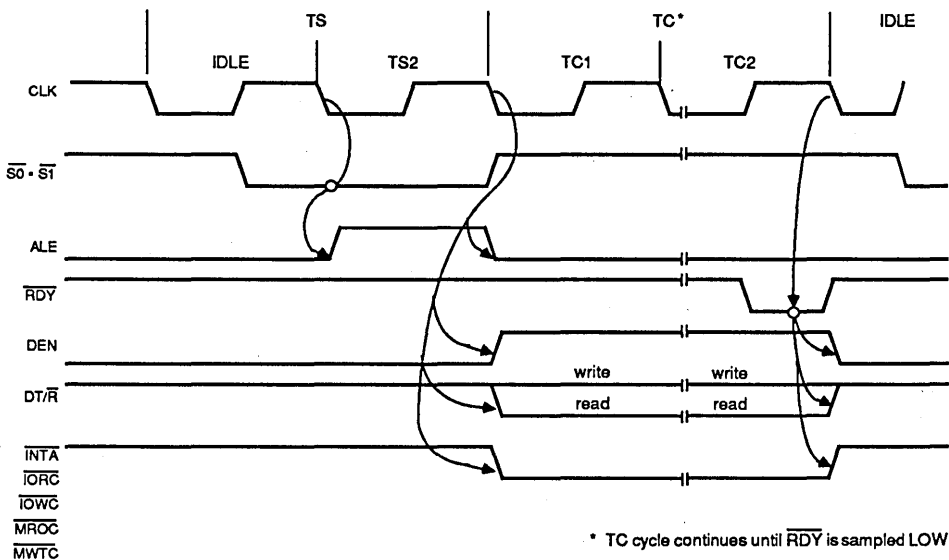
PAL Device Selection

AmPAL16R8B speed devices are required for operation at 12.5 MHz and up. The specifications on any faster 80286 are not established at this time, but this PAL device solution can go faster. For 10 MHz and slower operation, AmPAL16R8A speed devices are adequate.

Limitations of This Design

This PAL device design does not fully emulate the 82284 and the 82288. It provides the signals necessary for an IBM PC/AT design. An oscillator is not included, so an external crystal oscillator must be used. The clock requires a pullup resistor to meet the V_{OH} of the 80286. Multibus operation is not performed. PCLK and MCE signals are not generated. The $\overline{\text{READY}}$ signal is not open collector. The $\overline{\text{ARDYEN}}$ and $\overline{\text{SRDYEN}}$ signals would have to be externally gated if required.

3



08479A 3-212

Figure 3-212. Cycle Timing Diagram

```

flag '-R2';

title
'PAL16R8
82284 PAL device emulation for the AT
COPYRIGHT 1986 ADVANCED MICRO DEVICES, INC.
PAL DEVICE LOGIC EQUATION
Doug Kern 8/13/86

PAL284 device 'P16R8';

"declarations

TRUE,FALSE = 1,0;
HIGH,LOW = 1,0;
X,Z,C = .X.,.Z.,.C.;

GND,VCC
pin 10,20;

CLK,S0,S1,MB,CEN,RDYEN,SRDY,LARDY,RESET,EN
pin 1,2,3,4,5,6,7,8,9,11;

DEN,DT_R,ALE,RCMD,Q0,Q1,AR,RDY
pin 19,18,17,16,15,14,13,12;

QSTATE = [Q1,Q0];"STATE MACHINE REGISTERS
"STATE ASSIGNMENTS

IDLE = ^B11;
TS2 = ^B10;
TC1 = ^B00;
TC2 = ^B01;

STATE_DIAGRAM QSTATE

STATE IDLE: CASE !S0 :TS2;
               !S1 :TS2;
               S1 & S0 :IDLE;
            ENDCASE;

STATE TS2: GOTO TC1;

STATE TC1: GOTO TC2;

STATE TC2: CASE RDY :TC1;
               !RDY :IDLE;
            ENDCASE;

```

(Continued)

Figure 3-213. AmPAL284 Source Program Listing

EQUATIONS

```

!DT_R      := !RCMD & Q1 & !Q0
            # !DT_R & !Q1 & !Q0
            # !DT_R & !Q1 & Q0 & RDY;

!RCMD      := Q1 & Q0 & !S1 & !S0 "INTA
            # Q1 & Q0 & !S1 & S0 "READ
            # !RCMD & Q1 & !Q0
            # !RCMD & !Q1 & !Q0
            # !RCMD & !Q1 & Q0 & RDY;

!RDY       := !SRDY & !RDYEN & S1 & S0
            # !LARDY & !RDYEN & S1 & S0
            # RESET;

!DEN       := Q1 & !Q0 & !MB & CEN
            # !Q1 & !Q0 & !MB & CEN
            # !Q1 & Q0 & !MB & CEN & RDY
            # Q1 & !Q0 & MB & !CEN
            # !Q1 & !Q0 & MB & !CEN
            # !Q1 & Q0 & MB & !CEN & RDY;

ALE        := Q1 & Q0 & !S1
            # Q1 & Q0 & !S0;

```

Figure 3-213. (Continued) AmPAL284 Source Program Listing

ABEL(tm) Version 1.00 - Document Generator
PAL16R8
82284 PAL device emulation for the AT
COPYRIGHT 1986 ADVANCED MICRO DEVICES, INC.
Equations for Module AMD

26-Aug-86
PAL DEVICE LOGIC EQUATION

Doug Kern 8/13/86

Device PAL284

Reduced Equations:

Q1 := !((RDY & !Q1 & Q0 # !Q0));

Q0 := !((!S1 & Q1
(!S0 & Q1
(RDY & !Q1 & Q0
Q1 & !Q0))));

DT_R := !((RDY & !Q1 & !DT_R
(!RCMD & Q1 & !Q0
!Q1 & !Q0 & !DT_R));

RCMD := !((!S1 & Q1 & Q0
(RDY & !RCMD & !Q1
!RCMD & !Q0));

RDY := !((!SRDY & S1 & S0 & !RDYEN
(S1 & S0 & !RDYEN & !ARDY
RESET));

DEN := !((RDY & !Q1 & !MB & CEN
(RDY & !Q1 & MB & !CEN
(!Q0 & !MB & CEN
!Q0 & MB & !CEN))));

ALE := !((S1 & S0 # (!Q1 # !Q0));

Figure 3-213. (Continued) AmPAL284 Reduced Equations

PAL DEVICE LOGIC EQUATION

Doug Kern 8/13/86*

```

1111111111111111111101111011111101
1111111111111111111110110111111111
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
1111111111111111111101111101111101
1111111111111111111110111111111111
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0111011111111111111101110111111111
0111011111111111111101111110111111
1111111111111111111111111111110111
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000*
C4D0B*
C8A0

```

3

3-297

flag '-R2';

title

'PAL16R8

PAL DEVICE LOGIC EQUATION

82288 emulation PAL device for the AT

COPYRIGHT 1986 ADVANCED MICRO DEVICES, INC.

Doug Kern 8/13/86

PAL288 device 'P16R8';

"declarations

TRUE,FALSE = 1,0;
HIGH,LOW = 1,0;
X,Z,C = .X.,.Z.,.C.;

GND,VCC

pin 10,20;

CLK,S0,S1,M_IO,CMDEN,RCMD,RDY,Q1,Q0,MB
pin 1,2,3,4,5,6,7,8,9,11;

MRDC,MWTC,IORC,IOWC,INTRA,MEM,INT,WCMD
pin 19,18,17,16,15,14,13,12;

EQUATIONS

!MRDC := !MEM & !RCMD & Q1 & !Q0 & CMDEN
!MEM & !RCMD & !Q1 & !Q0 & CMDEN
!MEM & !RCMD & !Q1 & Q0 & RDY & CMDEN;

!MWTC := !MEM & !WCMD & Q1 & !Q0 & CMDEN
!MEM & !WCMD & !Q1 & !Q0 & CMDEN
!MEM & !WCMD & !Q1 & Q0 & RDY & CMDEN;

!IORC := MEM & INT & !RCMD & Q1 & !Q0 & CMDEN
MEM & INT & !RCMD & !Q1 & !Q0 & CMDEN
MEM & INT & !RCMD & !Q1 & Q0 & RDY & CMDEN;

!IOWC := MEM & !WCMD & Q1 & !Q0 & CMDEN
MEM & !WCMD & !Q1 & !Q0 & CMDEN
MEM & !WCMD & !Q1 & Q0 & RDY & CMDEN;

!INTRA := !INT & Q1 & !Q0 & CMDEN
!INT & !Q1 & !Q0 & CMDEN
!INT & !Q1 & Q0 & RDY & CMDEN;

!MEM := Q1 & Q0 & M_IO & !S1 & S0
Q1 & Q0 & M_IO & S1 & !S0
!MEM & Q1 & !Q0
!MEM & !Q1 & !Q0
!MEM & !Q1 & Q0 & RDY;

(Continued)

Figure 3-214. AmPAL288 Source Program Listing

```

!INT    := Q1 & Q0 & !M_IO & !S1 & !S0
        # !INT & Q1 & !Q0
        # !INT & !Q1 & !Q0
        # !INT & !Q1 & Q0 & RDY;

!WCMD   := Q1 & Q0 & S1 & !S0
        # !WCMD & Q1 & !Q0
        # !WCMD & !Q1 & !Q0
        # !WCMD & !Q1 & Q0 & RDY;

```

Figure 3-214. (Continued) AmPAL288 Source Program Listings

Reduced Equations:

```

MRDC := !((RDY & !RCMD & !Q1 & !MEM & CMDEN
        # !RCMD & !Q0 & !MEM & CMDEN));

MWTC := !((!WCMD & RDY & !Q1 & !MEM & CMDEN
        # !WCMD & !Q0 & !MEM & CMDEN));

IORC := !((RDY & !RCMD & !Q1 & MEM & INT & CMDEN
        # !RCMD & !Q0 & MEM & INT & CMDEN));

IOWC := !((!WCMD & RDY & !Q1 & MEM & CMDEN
        # !WCMD & !Q0 & MEM & CMDEN));

INTRA := !((RDY & !Q1 & !INT & CMDEN # !Q0 & !INT & CMDEN))

MEM := !((!S1 & S0 & Q1 & Q0 & M_IO
        # (S1 & !S0 & Q1 & Q0 & M_IO
        # (RDY & !Q1 & !MEM
        # !Q0 & !MEM))));

INT := !((!S1 & !S0 & Q1 & Q0 & !M_IO
        # (RDY & !Q1 & !INT
        # !Q0 & !INT)));

WCMD := !((S1 & !S0 & Q1 & Q0
        # (!WCMD & RDY & !Q1
        # !WCMD & !Q0));

```

Figure 3-214. (Continued) AmPAL288 Reduced Equations

3.6.8 INTERFACING THE 80186 MICROPROCESSOR TO THE 8087 WITH THE AMPAL22V10A DEVICE

This application note shows how to interface the 8087 Numeric Data Processor (NDP) to the 80186 High Integration Microprocessor using two AmpAL22V10A devices. It demonstrates some of the details of 80186 and 8087 design, as well as tradeoffs and design decisions made in designing with PAL devices. The hardware features of the 80186 and 8087, and AmpAL22V10A are in their respective data sheets and are not discussed herein.

Introduction

The proliferation of 8088-based IBM PC Personal Computers and PC-Compatibles has led to the development of higher performance, lower cost versions of that "de facto" standard. One important requirement for these "compatibles" is to provide 8087 Numeric Processor support, as does the IBM PC. An 80186/8087 combination would be the ideal choice, given the low cost and high performance of the 80186 and the vast throughput improvements gained from the 8087 in number crunching.

The 80186 is a much higher performance microprocessor than the 8088 used in the IBM PC. The 80186 incorporates several system components on-chip (i.e., DMA, Counter/Timers, Interrupt Controller), as well as improved control circuitry with pipelining to provide instruction execution performance up to six times that of the standard 8086. With the addition of the 8087 NDP co-processor to perform concurrent numeric operations, the 80186/8087 combination is the lowest cost-per-function number cruncher available.

Unfortunately, the interface of the 80186 to the 8087 is not as straight forward as it is with the 8088 to 8087 interface; however the interface can be implemented with two AmpAL22V10A devices and a few discrete components (Figure 3-215). The circuit allows full speed 8 MHz operation (with the 8087-2). With faster microprocessor devices (80186-1, and 10 MHz 8087), and faster PAL devices, the circuit can operate at 10 MHz. In addition, the interface of the 80188 serves to decrease system cost by reducing the data path width (and hence the number of components) while still yielding high performance.

Design Considerations

The 8087 acts as a master/slave co-processor to the 80186. The 8087 tracks all 80186 operations

(instruction fetch, data read/write, idle, etc.), monitoring various external I/O signals of the 80186 to determine its internal state. Upon detecting a numeric instruction in the instruction stream fetched by the 80186, the 8087 requests control of the system bus, and becomes a bus master until its operations are completed.

It is very important that the 8087 remains in complete synchronization with the 80186. Herein lies the problem with an 80186 to 8087 interface. A number of issues impacting synchronization; and their solutions are discussed below.

CPU Clock Synchronization

The 80186 contains an internal crystal oscillator/clock generator circuit which generates the 80186 timing. All external bus operations are synchronized to the 50% duty cycle clock generated. In contrast, the 8087 requires a 33% duty cycle clock normally generated by the 8284 Clock Generator chip.

This problem is solved with a clock shaping circuit using delay lines and a few discrete gates; this circuit shortens the clock HIGH time into the 8087. This is sufficient, as most input signals to the 8087 are synchronous to the rising edge of the 8087 CPU clock, not the displaced falling edge (Figure 3-216).

Queue Status

The 8087 tracks the 80186 status via several external signals. The Queue Status signals (QS0, QS1) provide valuable information to the 8087 regarding the 80186 internal pre-fetch buffer and instruction execution status. The 80186 provides (and removes) these signals 1/2 clock cycle before the 8087 requires them. A simple circuit performs the appropriate delay for the 8087 to use these signals.

This circuit also forces the QS0 and QS1 signals to an inactive state upon System Reset, signalling an "idle" condition to the 8087 until the circuit is enabled by the 80186. This is necessary to force the 8087 into instruction synchronization with the 80186 right after Reset, when the 80186 operation is unusual.

The 80186 does not usually emit the QS0 and QS1 outputs in its default configuration. These signals are generated on ALE/QS0 and WR*/QS1 by strapping RD*/QSM* LOW at Reset. An 8288 Bus Controller is required to decode the S0-S2 status lines from the 80186 or 8087 (whichever is bus master) to generate the proper Read/Write and ALE signals.

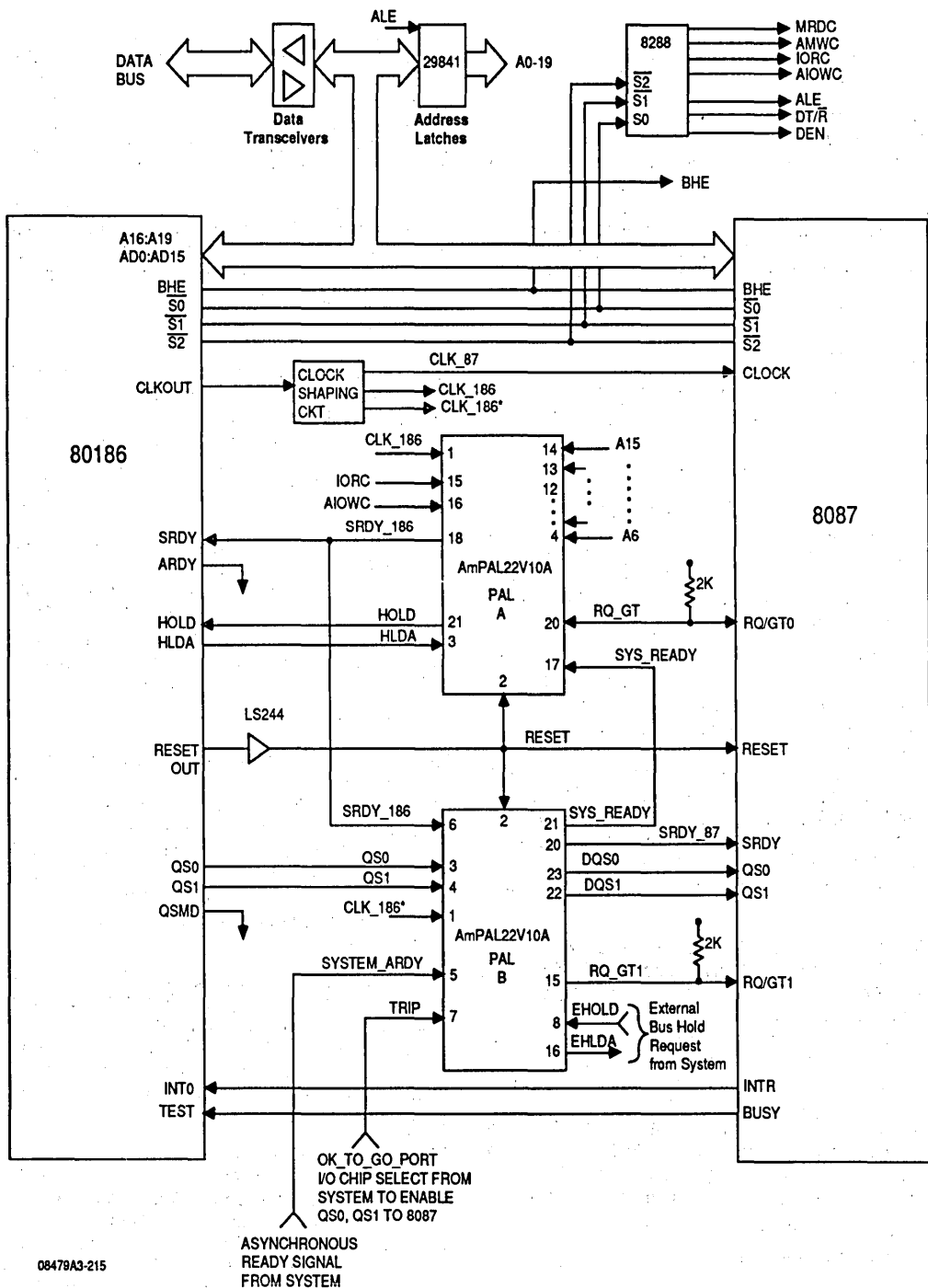


Figure 3-215. 80186 to 8087 Interface

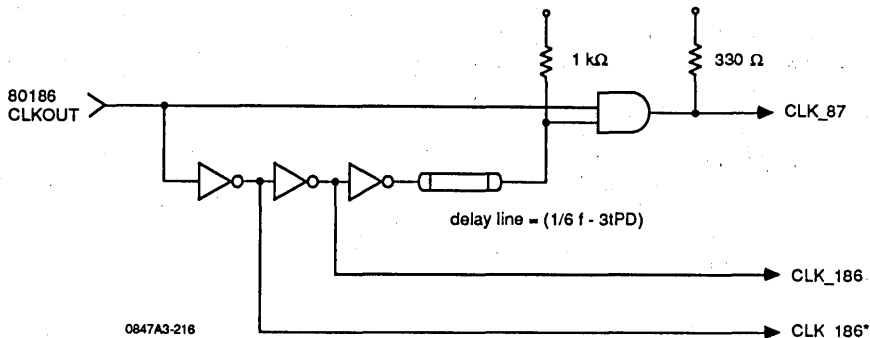


Figure 3-216. Clock Shaping Circuit

Ready Synchronization

The 8087 monitors the current 80186 bus activity via its S0-S2 Status Lines and the Ready Input line. There are no problems with the S0-S2 lines, but Ready may cause some problems.

First, the 80186 has on-board software-programmable chip-select and wait-state generation logic. This is useful in reducing external chip count, but the 8087 is unable to externally track these internally generated wait-states. The solution is to disable all programmable wait states (program them to "0"), recognize the external Ready signal, and insert any necessary wait states to the 80186 and 8087 with the Ready input line. In addition, since the 8087 cannot access the chip-select logic on the 80186, any memory that is used by the 8087 must be selected with external chip-select logic. This is typical only for main system RAM; other EPROM and I/O ports typically are not accessed by the 8087. They may still be selected via the 80186-generated MCS/UCS and PCS lines, as long as "zero wait states" are programmed.

Note that the 80186 has several on-board I/O registers that the 8087 "doesn't know about". These registers are all accessed in zero wait states,

except for the Counter/Timer registers, which cause the insertion of one wait state. Ready signals must be simulated for the 8087 when the 80186 selects these on-board registers, forcing zero or one wait states, depending on the particular register selected. It must be assumed that the on-board registers remain at the default base address FF00H in the I/O space (i.e., the Relocation Register is not altered).

Secondly, consider the situation just after Reset. Following Reset, the 80186 automatically inserts three internal wait states for all memory accesses (i.e., to a Boot PROM) until the chip-select and wait-state registers are programmed. Again, the 8087 will lose synchronization with the 80186, because it does not "see" these wait states. The solution is to force the 8087 into an idle state after Reset by forcing its QS0 and QS1 inputs to "00" (idle), so it won't track the 80186 operations at all. When the 80186 completes programming its internal wait-state registers, it enables the QS0 and QS1 inputs to the 8087 via an I/O signal, and executes at least two consecutive JUMP instructions. The JUMP instructions effectively flush the internal prefetch queues, and the 8087 acquires synchronization with the 80186. The following code is an example of the final synchronization routine, after Reset, and in programming the on-board registers:

```

; The 80186 register initialization code
; completed above. The 8087 is idle.

out      ok_to_go_port, al      ; enable QS signals
jmp      L1                     ; jump to flush queue

L1:      jmp      L2             ; jump to flush queue

L2:

; System code follows here. The 8087
; is now synchronized and active.

```

Finally, the proper Ready inputs must be provided to the 80186 and the 8087. The 8087 requires a synchronous Ready signal (SRDY) to be valid at the rising edge (middle) of T3 or Tw, while the 80186 requires the synchronous Ready input to be valid on the falling (leading) edge of T3 or Tw. Circuitry must be provided to perform the proper synchronization and to provide adequate setup times for the 80186 and 8087 SRDY signals. In addition, extra circuitry is necessary to synchronize an asynchronous Ready signal from the system hardware.

RQ/GT to HOLD/HLDA Conversion

Upon detecting any NDP instructions from the 80186 instruction stream, the 8087 must become a Bus Master in order to obtain and store operands/results in memory. The 8087 requests, and is granted, bus control via its RQ/GT0 signal. This is normally a single-wire handshake supported by the 8086 microprocessor; however, the 80186 uses a two-wire HOLD/HLDA interface. The solution is to build a small state machine that converts the RQ/GT0 pulsed protocol to the HOLD and HLDA signals required by the 80186.

In order to support other Bus Masters in addition to the 8087 (i.e., an external DMA controller), the RQ/GT1 input to the 8087 is used. In this manner, the external Bus Request is "daisy-chained" through the 8087 to the 80186. Again, the RQ/GT1 is a single-wire handshake signal. A second state machine is built to convert external HOLD and HLDA type signals of typical DMA controller chips to the single-wire RQ/GT1 protocol of the 8087.

Interface Circuits

The interface circuit is implemented with two AmpAL22V10A devices, a delay line (or capacitor), and a few Schottky or "AS" gates (see Figures 3-215 & 3-216). The system requires two synchronously clocked circuits based on the 80186 clock (CLK_186), as well as the inverted 80186 clock (CLK_186*). Therefore, the design is partitioned among two AmpAL22V10A's (PALA and PALB), with CLK_186 and CLK_186* as inputs.

Clock Shaping Circuitry

The Clock is shaped with a Schottky delay line and "S" or "ALS" gates, shown in Figure 3-216. The delay line tap determines the 8087 clock HIGH times, and is ~ 10 ns for an 8 MHz clock, to yield the nominal 42 ns clock HIGH time required. For faster clocks (i.e., 10 MHz), this delay must be decreased appropriately (and may be eliminated) de-

pending on the delays through the three inverters.

The three-inverter chain provides: 1) buffering of the CLK_186 and CLK_186* signals for distribution throughout the system, and, 2) increased drive for the delay line. The output of the delay line is pulled up in this circuit, to provide adequate VIH for the AND gate. Also note the strong 330 ohm pull-up on the CLK_87 clock output; this is to provide the nominal 3.9 V VIH required by the clock input of the 8087 (Ref. 4).

PALA Design

PALA is clocked with the positive 80186 clock, CLK_186. This AmpAL22V10A device contains the RQ/GT0 to HOLD/HLDA (80186) synchronous state machine circuitry, the internal chip-select wait-state generator circuitry (used for accessing internal 80186 registers), and the 80186 Ready signal synchronization. The PALA equations are listed in Figure 3-218.

RQ/GT0 (8087) Conversion to HOLD/HLDA (80186)

The State Machine Transition diagram is shown in Figure 3-217. This circuit samples the RQ/GT0 output from the 8087, and waits for a pulse on that line, signalling an 8087 Hold Request. This causes generation of the HOLD to the 80186. The state machine then waits for an HLDA signal from the 80186, and generates a Hold Acknowledge pulse to the 8087 on RQ/GT0. When the 8087 next pulses the RQ/GT0 line, the state machine removes the HOLD signal from the 80186, signalling the end of the HOLD request.

Since RQ/GT0 is a bi-directional signal, the state machine sets it to high impedance "off", except when enabled during the Hold Acknowledge pulse. Generation of this signal demonstrates the versatility of the AmpAL22V10 device for such bi-directional three-state applications.

Note the values supplied for the State Machine variables: A Gray code was used, as in most state machine designs, to minimize glitching on the outputs as the state variables change from state to state (Gray codes have the benefit that only one bit changes per code transition).

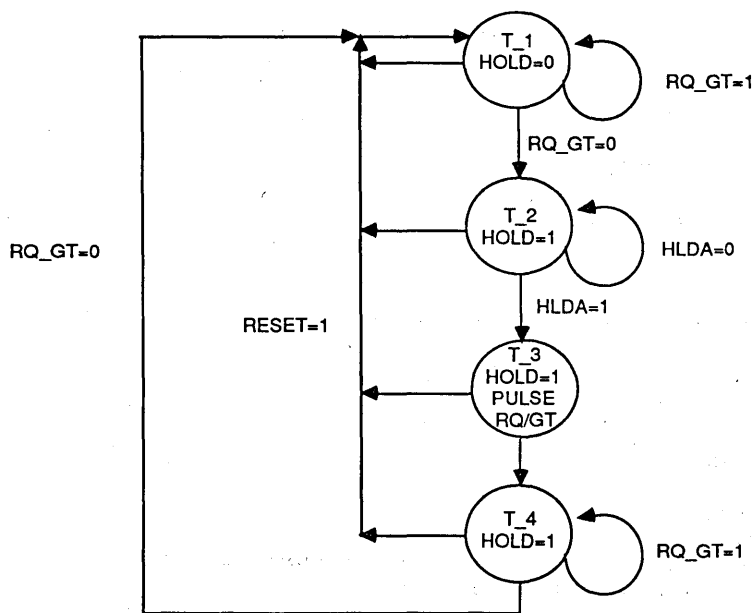
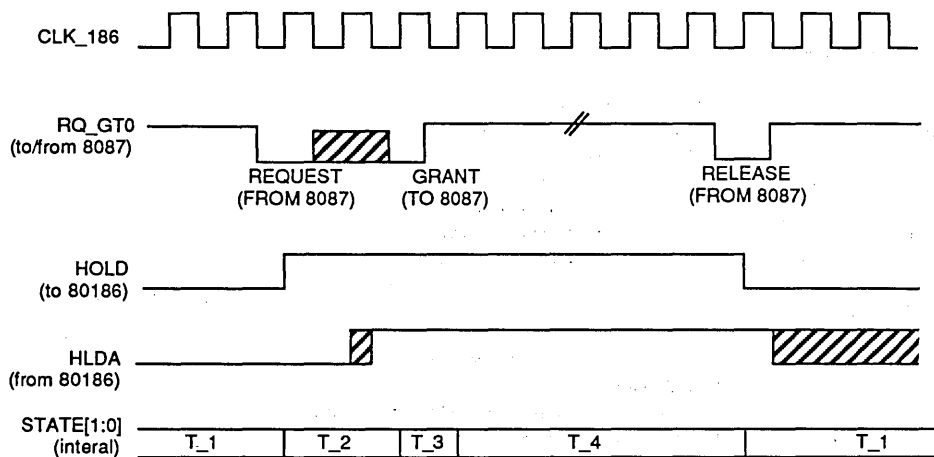
Ready Synchronization and Internal-Register Chip-Select Logic

The signal SRDY_186 is a composite system Ready signal; it is synchronized for the synchronous SRDY input to the 80186 (this signal is in turn synchronized in PALB for the 8087 SRDY). SYS_READY is an input to the PALA

device, and is synchronized (by PALB) version of the external Memory/IO Ready from the system. Note that the 80186 ARDY (Asynchronous Ready) input must be strapped LOW.

In order to generate the proper Ready signals to the 8087 when selecting internal 80186 registers, PALA decodes the 80186 address, and generates the proper Ready signal. The internal

TIMER_RDY signal is generated whenever the internal Counter/Timer registers are selected (I/O address FF50-FF6F). This signal inserts one wait state in the synchronous Ready line, SRDY_186. In addition, the SRDY_186 signal is forced to a "Ready" state whenever any internal 80186 registers, other than the Counter/Timers, are selected.



NOTE: state machine is clocked on positive edge of CLK_186.

08749A3-217

Figure 3-217. RQ/GT0-to-Hold/HLDA Converter


```
DEVICE PALA_8087_80186_Interface (PAL22V10);
```

```

" *****
" ****                                     ****
" **** PAL A                             ****
" ****                                     ****
" **** This PAL converts from 8087 RQ/GT Handshake TO 80186 HOLD/HLDA Signals ****
" **** and generates the ready synchronization logic for the 80186 and 8087 ****
" ****                                     ****
" *****

```

```
PIN
```

```
" **** INPUTS ****"
```

```

CLK      = 1      " 50% duty cycle 80186 clock "
RESET    = 2      " Active HIGH reset from 80186 "
HLDA     = 3      " 80186 HLDA output "
A[15:6]  = 4:11,13,14 " 80186 Address lines 15..6 "
/IORC    = 15     " IO Read Command from 8288 "
/AIOWC   = 16     " 8288 Advanced IO write command"
SYS_READY = 17     " Synchronized READY from system"

```

```
" **** OUTPUTS ****"
```

```

STATE[1:0] = 23:22 " State Machine State Variables "
HOLD       = 21     " 80186 HOLD input "
TIMER_RDY  = 19     " Internal timer chip select (Inserts ONE W.S.) "
SRDY_186   = 18     " Synchronous 80186 READY signal"

```

```
" **** BIDIRECTIONAL INPUT/OUTPUTS **** "
```

```
RG_GTO    = 20;      " RQ/GT I/O. (active low) MUST HAVE PULLUP! "
```

```
DEFINE
```

```
" State variable state value assignments. "
```

```

T_1      = 0 ;      " State 1 "
T_2      = 1 ;      " State 2 "
T_3      = 3 ;      " State 3 "
T_4      = 2 ;      " State 4 "

```

```
BEGIN
```

```

" *****
" ****                                     ****
" **** RQ/GT to HOLD/HLDA Converter State Machine Definition ****
" ****                                     ****
" *****

```

```
IF (RESET) THEN ARESET();
```

```
CASE ( STATE[1:0] ) BEGIN
```

```
  T_1 ) BEGIN
```

```
    IF (RG_GTO)
```

```
        " Wait for 1st request from 8087 "
```

```
        THEN STATE[1:0] := T_1 ;
```

```
        ELSE STATE[1:0] := T_2 ;
```

```
    HOLD := 0;
```

```
        " Deassert HOLD to while here 80186 "
```

```
  END;
```

```

T_2 ) BEGIN
    IF (HLDA)                                " Wait for HOLD ACKNOWLEDGE from 80186 "
    THEN STATE[1:0] := T_3 ;
    ELSE STATE[1:0] := T_2 ;
    HOLD := 1;                                " Assert HOLD to 80186. Also, RQ/GT is "
                                              " pulsed to signal GRANT to 8087 (see below) "
END;

T_3 ) BEGIN
    STATE[1:0] := T_4;    " One idle state to eliminate RQ/GT contention "
    HOLD := 1;
END;

T_4 ) BEGIN
    IF (RG_GTO)            " Wait for RELEASE pulse from 8087 "
    THEN STATE[1:0] := T_4 ;
    ELSE STATE[1:0] := T_1 ;
    HOLD := 1;
END;
END;

RG_GTO = 0;                                " RG_GTO is tri-state controlled I/O, active LOW "
IF ((STATE[1:0] = T_2) * (HLDA))
    THEN ENABLE( RG_GTO );    " Pulse RQ/GT output during state 2 to signal HOLD ACKNOWLEDGE (GRANT) to 8087 "

" *****
" ****
" **** READY Synchronization logic ****
" ****
" *****

" Internal timer registers selected (insert one w.s. with registered output) "

TIMER_RDY := ((IORC + /AIOWC) * (A[15:6] = #B111111101) ;

" 80186 SRDY synchronous ready generation. Synchronize ready inputs for 80186 SRDY input. "
" No wait states if any internal 80186 registers (at I/O address 0ffXXh) selected "
" except for the internal TIMER registers "

SRDY_186 := TIMER_RDY + SYS_READY + ((IORC + /AIOWC) * (A[15:8] = #B11111111) * (A[15:6] /= #B111111101));

END.

```

Listing sum-of-products equations for PALA_8087_80186_Interface

```

SRDY_186 := /IORC*A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[6]
          + A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[6]/A10WC
          + TIMER_RDY
          + SYS_READY
          + A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[7]*A10WC
          + /IORC*A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[7];
SRDY_186.ARESET = RESET;

TIMER_RDY := A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[7]*A[6]
            * /A10WC
            + A[15]*A[14]*A[13]*A[12]*A[11]*A[10]*A[9]*A[8]*A[7]*A[6]
            * /IORC;
TIMER_RDY.ARESET = RESET;

RG_GTO = 0;
RG_GTO.TRISTATE = /STATE[1]*STATE[0]*HLDA;
RG_GTO.ARESET = RESET;

HOLD := STATE[0]
      + STATE[1];
HOLD.ARESET = RESET;

STATE[0] := /RG_GTO*/STATE[1]
          + /STATE[1]*STATE[0];
STATE[0].ARESET = RESET;

STATE[1] := STATE[0]*HLDA
          + RG_GTO*STATE[1]
          + STATE[1]*STATE[0];
STATE[1].ARESET = RESET;

```

PALB Design

PALB is clocked with the inverted 80186 clock, CLK_186*. This PAL device contains the external DMA HOLD/HLDA to RQ/GT1 interface to the 8087, the Queue Status control logic, and the synchronization circuitry necessary for the various Ready signals. The equations for PALB are listed in Figure 3-220.

External HOLD/HLDA Conversion to RQ/GT1 (8087)

This protocol conversion circuit for external bus HOLD requesters performs the converse function of that described above for PALA. The State Machine Transition diagram is shown in Figure 3-219.

The state machine waits for an external HOLD request on the EHOLD input. It then generates a Hold Request pulse on the RQ/GT1 pin to the 8087. It then waits for a Hold Acknowledge pulse from the 8087, and generates EHLDA to the external requestor. When the external Hold Request is removed, the circuit generates a Release pulse to the 8087 on the RQ/GT1 pin, and returns to state 1. Again, the versatility of the AmPAL22V10A is demonstrated by the bi-directional three-state capability of the device.

Gray codes were not used for the state assignments in this case. The codes were chosen to minimize the three-state enable term for the RQ/GT1 output. The RQ/GT1 outputs are enabled during States 2 and 4, to generate output pulses.

State assignments chosen with Gray codes would have required two product terms for the three-state enable ($/\text{STATE}[1]*\text{STATE}[0] + \text{STATE}[1]*\text{STATE}[0]$), and the AmPAL22V10A can only support one product term for the three-state enable signal. The state encoding chosen allows one product term for this function. There is minimal glitching on the outputs of this circuit, even though Gray codes were not used, because the EHLDA term is registered, and the RQ/GT1 term is generated from a minimization that is fully covered on the Karnaugh map (Reference 1).

Queue Status Control Circuit

The Queue Status inputs from the 80186 are delayed by 1/2 clock cycle in this circuit for proper presentation to the 8087 as DSQ0 and DSQ1. In addition, the OK_TO_GO signal forces these signals LOW after Reset, until the TRIP input receives a pulse. Recall that the 8087 must be idle after Reset until all internal 80186 registers are programmed. The TRIP input is generated by a chip select to an I/O port from the 80186.

Ready Synchronization

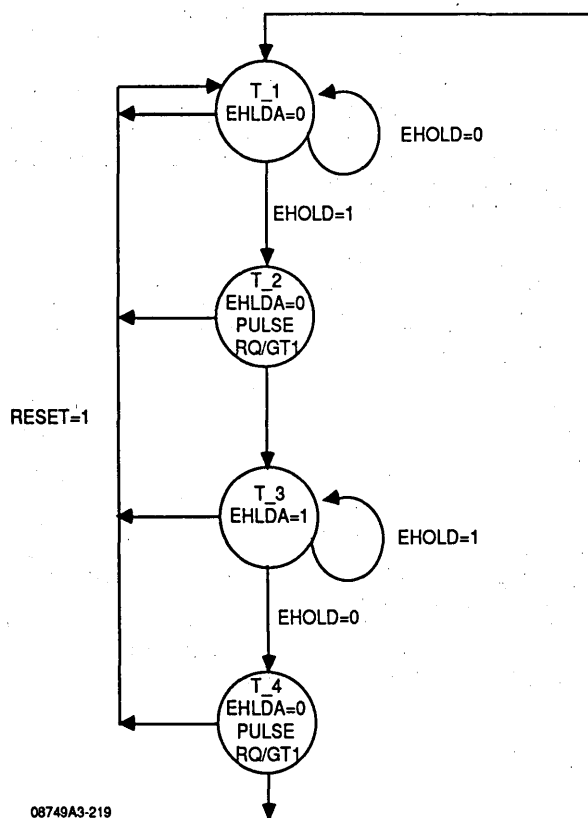
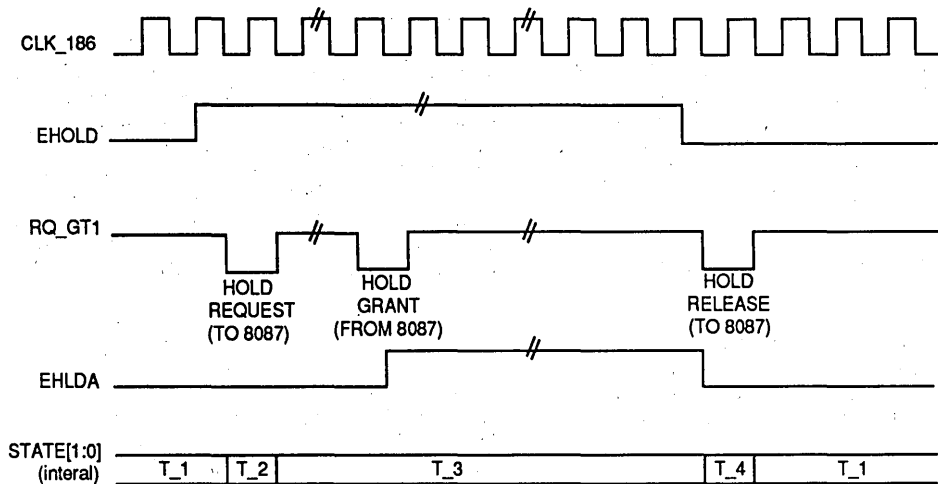
This circuit synchronizes the asynchronous SYSTEM_ARDY Ready signal, and presents the SYS_READY signal to PALA. The SRDY_186 input to the 80186 is also synchronized by this circuit to be presented as the SRDY_87 to the 8087.

Conclusion

Due to the power and flexibility of the AmPAL22V10A device, the interface between the 80186 microprocessor and the apparently incompatible 8087 Numeric Processor can be implemented with a small amount of circuitry.

References

1. Nagle, H. Troy, Carroll, B. D., and Irwin, J. David, *An Introduction to Computer Logic*, Prentice-Hall, Inc, NJ, 1975.
2. Advanced Micro Devices *MOS Microprocessor and Peripherals Data Book*, 1985.
3. Advanced Micro Devices *Programmable Array Logic Data Book*, 1984.
4. Advanced Micro Devices *Bipolar Microprocessor Logic and Interface Data Book*, 1985.
5. *The 8086 Family User's Manual*, Intel Corporation, 1979.



NOTE: state machine is clocked on negative edge of CLK_186.

08749A3-219

Figure 3-219. External Hold/HLDA-to-RQ/GT1 Converter

```
DEVICE PALB_8087_80186_Interface (PAL22V10);
```

```

" *****
" ****                                     ****
" **** PAL B                             ****
" ****                                     ****
" **** This PAL performs part of the 8087 to 80186 interface function. Here, we ****
" **** delay Queue Status from the 80186 properly for the 8087. The logic is ****
" **** clocked off of the INVERTED 80186 clock. We also generate some of the ****
" **** signals used for the synchronous READY inputs to the 8087 and 80186. ****
" **** This logic is clocked off inverted CLK to guarantee adequate set-up times ****
" **** We also perform the HOLD --> RQ/GT1 protocol for EXTERNAL bus masters. ****
" ****                                     ****
" *****

```

```
PIN
```

```
" **** INPUTS ****"
```

```

/CLK      = 1      " 50% duty cycle 80186 clock externally INVERTED      "
RESET     = 2      " Active HIGH reset from 80186 "
QS0       = 3      " Queue Status0 input from 80186 "
QS1       = 4      " Queue Status1 input from 80186 "
SYSTEM_ARDY = 5    " System Asynchronous ready, to be synchronized      "
SRDY_186  = 6      " Synchronous Ready for 80186 "
TRIP      = 7      " Signal from 80186 to enable DQS[1:0] outputs to 8087      "
EHOLD     = 8      " HOLD request from external DMA Master connected to 8087      "

```

```
" **** OUTPUTS ****"
```

```

DQS0      = 23     " Delayed queue status 0 to 8087 "
DQS1      = 22     " Delayed queue status 1 to 8087 "
SYS_READY = 21     " Synchronized system ready      "
SRDY_87   = 20     " Synchronous ready for 8087     "
OK_TO_GO  = 19     " Signal used to enable Queue Status outputs to 8087 (DSQ[1:0] "
STATE[1:0] = 18:17 " EHOLD/EHLDA to 8087 RQ/GT1 handshake state variables      "
EHLDA     = 16     " EHOLD Acknowledge to external DMA MASTER connected to 8087      "

```

```
" **** BIDIRECTIONAL INPUT/OUTPUTS **** "
```

```
RQ_GT1    = 15;    " 8087 RQ/GT1 DMA control for external master      "
```

```
DEFINE
```

```

" State variable state value assignments.      "
" NOTE: These codes were chosen to minimize the RQ/GT signal implementation "

```

```

T_1      = 0 ;      " State 1      "
T_2      = 1 ;      " State 2      "
T_3      = 2 ;      " State 3      "
T_4      = 3 ;      " State 4      "

```

```
BEGIN
```

```

" *****
" ****                                     *****
" **** EHOLD/EHLDA TO RQ/GT1 Converter State Machine Definition *****
" **** for DMA interface to 8087                                     *****
" ****                                     *****
" *****

```

```
IF (RESET) THEN ARESET();
```

```
CASE ( STATE[1:0] ) BEGIN
```

```
    T_1 ) BEGIN
```

```

        IF ( EHOLD )                " Wait for EHOLD REQUEST from external master "
            THEN STATE[1:0] := T_2;
            ELSE STATE[1:0] := T_1;
        EHLDA := 0;                  " De-assert EHOLD acknowledge "
    END;

```

```
    T_2 ) BEGIN
```

```

        STATE[1:0] := T_3; " Kill Time "
        EHLDA := 0;        " and pulse RQ_GT to signal request to 8087 "
    END;

```

```
    T_3 ) BEGIN
```

```

        IF ( EHOLD )                " Wait for master to release EHOLD request "
            THEN STATE[1:0] := T_3;
            ELSE STATE[1:0] := T_4;
        IF (/EHLDA )                " Set EHLDA when RQ/GT GRANT pulse comes from 8087 "
            THEN IF (/RQ_GT1)
                THEN EHLDA := 1;
                ELSE EHLDA := 0;
            ELSE EHLDA := 1;
    END;

```

```
    T_4 ) BEGIN
```

```

        STATE[1:0] := T_1;          " Kill time, and pulse RQ_GT to signal RELEASE "
        EHLDA := 0;
    END;

```

```
END;
```

```

" RQ_GT1 is tri-state controlled, and pulsed active low "
" The following controls the RQ_GT signals "

```

```
RQ_GT1 = 0;
```

```
IF ((STATE[1:0] = T_2) + (STATE[1:0] = T_4)) THEN ENABLE(RQ_GT1);
```

Figure 3-220. (Continued)

```

" *****
" ****
" **** Miscellaneous Synchronization logic/
" ****
" *****

```

```

" OK_TO_GO signal is tripped from an 80186 signal (i/o write) after it has set all
" of its internal registers properly. Until then, OK_TO_GO is false and disables DQS[1:0] "

```

```

IF ( /OK_TO_GO )
    THEN IF (TRIP)
        THEN OK_TO_GO := 1;
        ELSE OK_TO_GO := 0;
    ELSE OK_TO_GO := 1;

```

```

" Delay Queue Status to 8087
" Qualify with OK_TO_GO signal from 80186 after it is reset "

```

```

DQS0 := OK_TO_GO * QS0;
DQS1 := OK_TO_GO * QS1;

```

```

" Synchronize Asynchronous System Ready "

```

```

SYS_READY := SYSTEM_ARDY;

```

```

" Synchronize 80186 SRDY input for 8087 "

```

```

SRDY_87 := SRDY_186;

```

```

END.

```

Figure 3-220. (Continued)

Listing sum-of-products equations for PALB_8087_80186_Interface

```

RQ_GT1 = 0;
RQ_GT1.TRISTATE = STATE[0];
RQ_GT1.ARESET = RESET;

EHLDA := STATE[1]*STATE[0]/RQ_GT1
      + STATE[1]*STATE[0]*EHLDA;
EHLDA.ARESET = RESET;

STATE[0] := EHOLD*STATE[1]*STATE[0]
          + /EHOLD*STATE[1]*STATE[0];
STATE[0].ARESET = RESET;

STATE[1] := STATE[1]*STATE[0]
          + /STATE[1]*STATE[0];
STATE[1].ARESET = RESET;

OK_TO_GO := TRIP
          + OK_TO_GO;
OK_TO_GO.ARESET = RESET;

SRDY_87 := SRDY_186;
SRDY_87.ARESET = RESET;

SYS_READY := SYSTEM_ARDY;
SYS_READY.ARESET = RESET;

DQS1 := OK_TO_GO*QS1;
DQS1.ARESET = RESET;

DQS0 := OK_TO_GO*QS0;
DQS0.ARESET = RESET;

```

Figure 3-220. (Continued)

3.6.9 A MULTIBUS ARBITER DESIGN FOR 10 MHz PROCESSORS

This application note describes the implementation of a bus arbiter using an AmPAL16R4 to interface the 10 MHz 80186 to the MULTIBUS multi-master environment. The PAL equations for bus exchange were developed based on functional and timing requirements specified in the Intel MULTIBUS Specification, June, 1982. The interface shown addresses the bus master case in regards to bus exchange only. No attempt has been made to include the requirements for bus slave operation, interrupt handling or byte operations. This note assumes a knowledge of MULTIBUS operations and signal names.

Overview

The system environment is assumed to be a collection of loosely coupled independent processors which communicate through a common memory on the system bus. Each processor in this system possesses its own local resources.

A block diagram showing the interface of one such processor, an 80186, to the system bus, MULTIBUS, is shown in Figure 3-225. A block of the 80186's address space is allocated to system memory. Access into this portion of the address map generates a request for the system bus via a programmable memory chip-select pin. Once a request is generated by the CPU, the AmPAL arbiter and the 8288 bus controller combine to perform a bus exchange and transfer cycle.

In this design the arbiter maintains control of the system bus and therefore must be forced off either by a higher priority master (BPRN goes HIGH) or a Common Bus Request (CBRQ).

Functional Requirements

On the MULTIBUS, the bus exchange process begins when the Bus Request (BREQ) goes LOW (parallel priority resolution scheme), or when BPRO goes HIGH to succeeding masters (serial priority resolution scheme). If the requesting master is granted priority (BPRN enabled) and the bus is not under control of another master (BUSY disabled), then the master may take control of the bus by forcing BUSY LOW.

An optional request mechanism, Common Bus Request, used in conjunction with either parallel or serial priority has been implemented in this design.

It reduces bus acquisition overhead by allowing bus retention across transfer cycles, unless a request is pending (CBRQ is LOW). For example, in parallel priority the requesting master asserts both BREQ and CBRQ to force the current master off the bus. Once acquisition is complete CBRQ is released for use by another master.

Timing Considerations

Bus exchange on MULTIBUS is a synchronous process. BREQ, BPRO, BUSY and CBRQ are all synchronized with the trailing edge (HIGH-to-LOW transition) of BCLK. BCLK is a MULTIBUS signal having a duty cycle of approximately 50% and a maximum frequency of 10 MHz. There is no requirement for synchronization between BCLK and any other clock in a MULTIBUS system. Synchronizer circuits must be used on signals that cross clock boundaries, because it is possible (as in this design) for the processor clock to be asynchronous to BCLK.

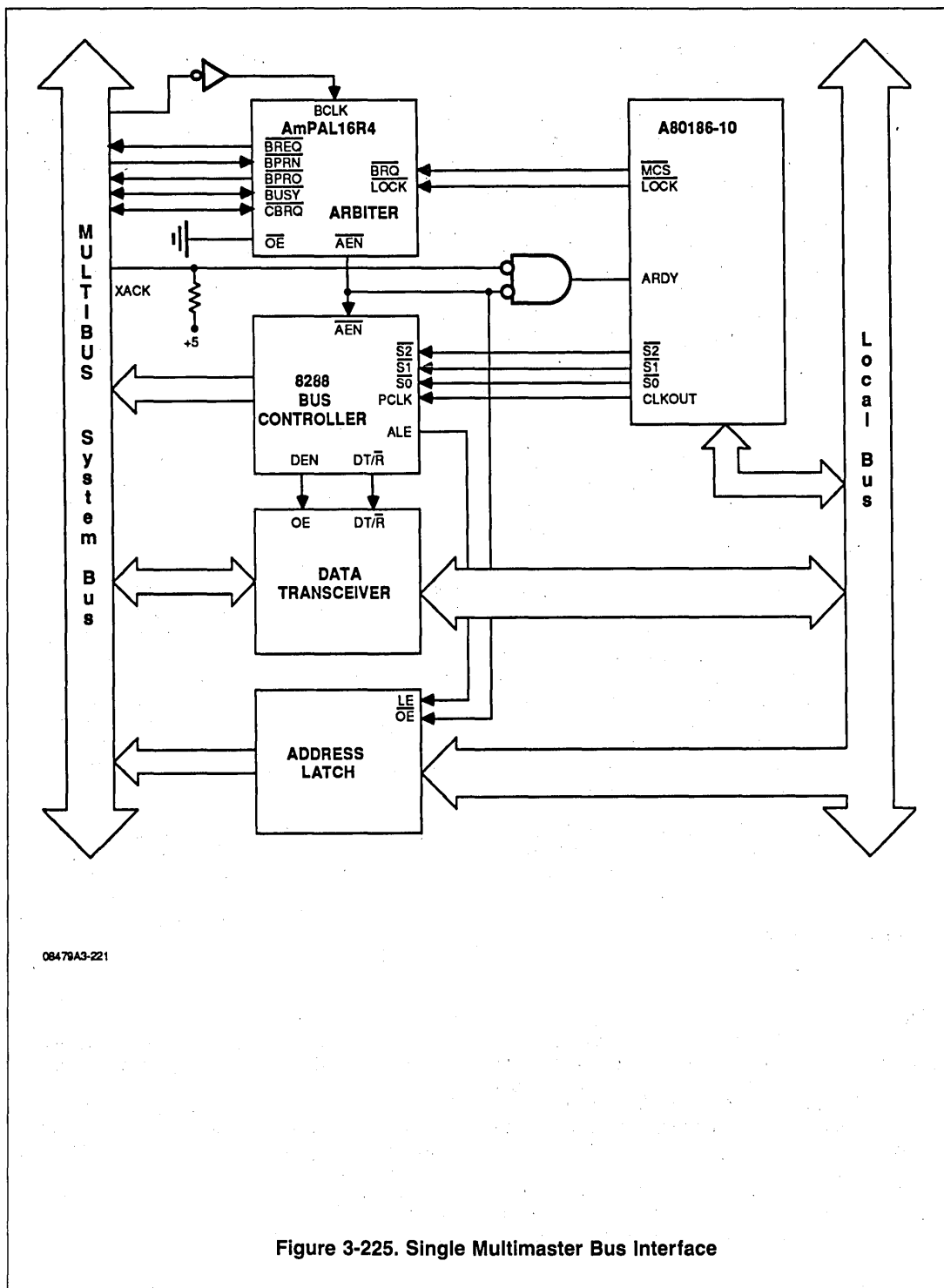
Bus priority on MULTIBUS resolution takes place in one BCLK cycle. If the serial priority scheme is implemented, this requirement will place an upper limit on the number of masters allowed for a given BCLK rate. The parallel priority scheme allows the relationship between BCLK and the number of masters to be independent. Whichever is chosen, the MULTIBUS specified minimum setup time of 22 ns (resolution to clock), plus the desire to keep the BPRN to BPRO propagation delay to a minimum, points to the use of an "A" speed PAL device (Figure 3-226).

The MULTIBUS bus exchange timing requires a BUSY setup time before clock of 25 ns (see Figure 3-226). Release by one master and acquisition by another takes place on two successive HIGH to LOW transitions of BCLK.

PAL Design Description

The PAL design specification with supporting timing diagrams are shown in Figures 3-227 and 3-228. The following remarks are in addition to the comments found on the PAL design specification.

The term "BRQP*AEN" found in the equation for "BREQ" is required to keep the arbiter from recapturing the bus when it is the releasing master. This can occur when a CBRQ forced the surrender, and the processor cycle which follows the just completed cycle accesses the system memory.



08479A3-221

Figure 3-225. Single Multimaster Bus Interface

MULTIBUS address hold after Command (50 ns min.) is satisfied by using the ALE signal from the 8288 instead of the 80186. The 8288 Command lines go inactive on the HIGH to LOW transition of CLKOUT at the beginning of T4 (30 ns max delay). The 8288 ALE signal goes active on the HIGH to LOW transition of CLKOUT at the beginning of T1 (1/2 clock cycle later than ALE from the 80186). The minimum hold time is therefore 100 ns - 35 ns = 65 ns.

Note the inversion of BLCK for use by the PAL device. PAL registers load data on the LOW to HIGH clock transition of pin 1.

- 1) Intel *Multibus Specification* #9800688-04, June 1982
- 2) Intel *iAPX 86,88 User's Manual*, August 1981
- 3) Advanced Micro Devices' *MOS Microprocessors and Peripheral's*, 1985
- 4) Intel *Component Data Catalog*, 1982



3-317

AmPAL16R4

PAT001

Multibus Arbiter

Advanced Micro Devices

BLCK /BPQ /LOCK /BPRN NC NC NC NC GND
/OE /BPRO /BUSY /AEN /BREQ /LOCKP /BRQP /CBRQ NC VCC

BRQP := BRQ ;CPU SYSTEM BUS REQUEST SYNC TO BCLK

LOCKP := LOCK ;CPU LOCK SIGNAL SYNC TO BCLK

BREQ := BRQP*/AEN ;REQUEST SYSTEM BUS IF NOT CURRENT BUS
+ BRQP*BREQ ;MASTER. HOLD TILL CPU RELEASES. WHEN
+ BREQ*BPRN*/CBRQ ;CPU RELEASES HOLD IF NO OTHER
+ BREQ*LOCKP ;REQUESTORS. HOLD ON CPU LOCK.

BPRO = BPRN*/BREQ ;FORCE OFF LOWER PRIORITY REQUESTORS IF
;REQUESTING, IF NOT PASS PRIORITY THRU.

AEN := BREQ*/BUSY*BPRN ;ACQUIRE SYSTEM BUS AND ENABLE ADDRESS
+ BREQ*/AEN ;BUFFERS. WHEN CPU RELEASES HOLD IF NO
+ AEN*BPRN*/CBRQ ;OTHER REQUESTORS.

IF (AEN) BUSY = AEN ;ALLOWS BUSY TO EMULATE O.C. I/O PIN.

IF (BREQ*/AEN) CBRQ = BREQ*/AEN ;REQUEST SYS BUS VIA CBRQ UNTIL ACQUIRED

08479A3-223

Figure 3-227. AmPAL16R4 Arbiter Design Specification

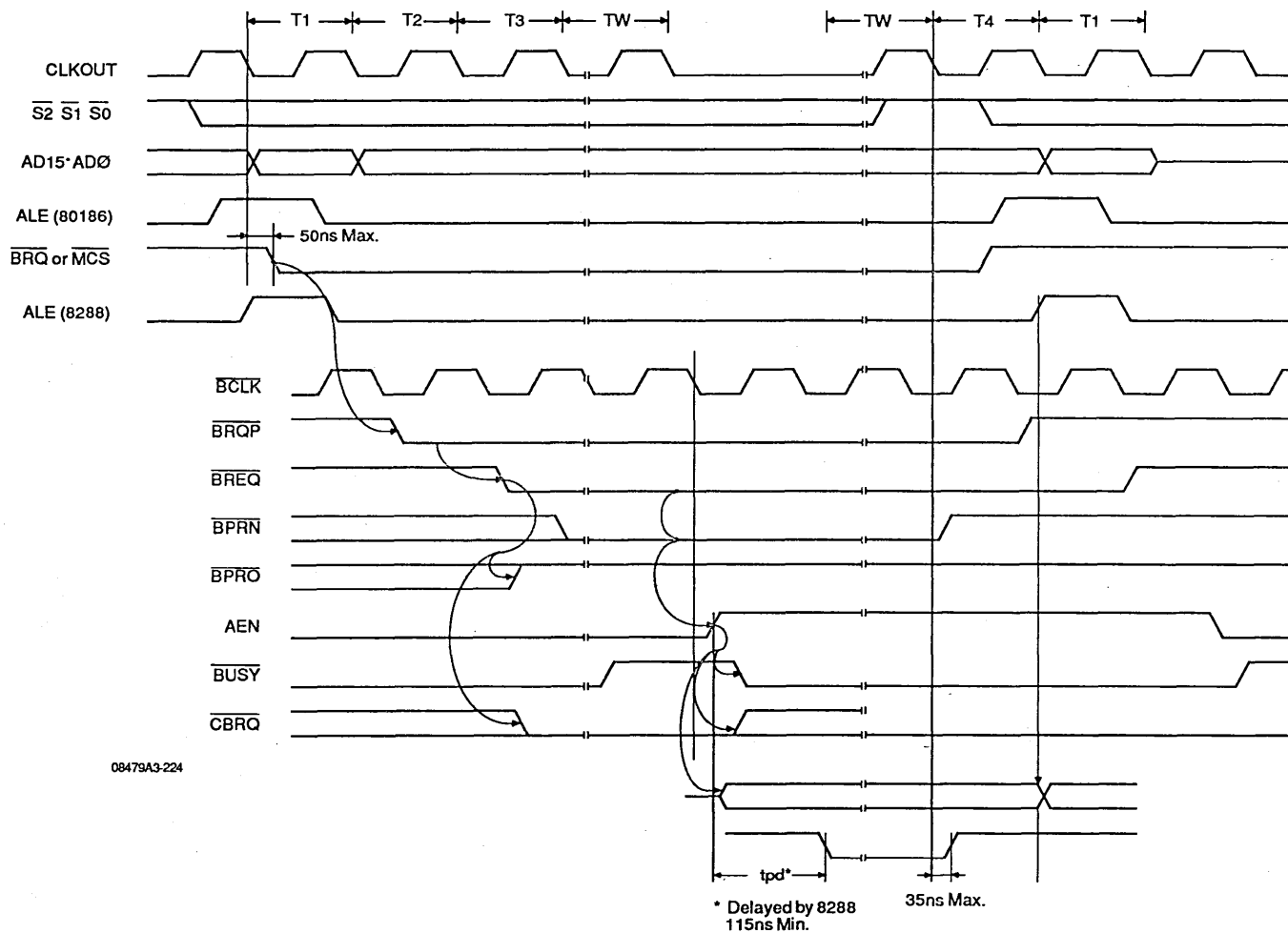
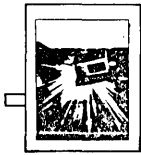
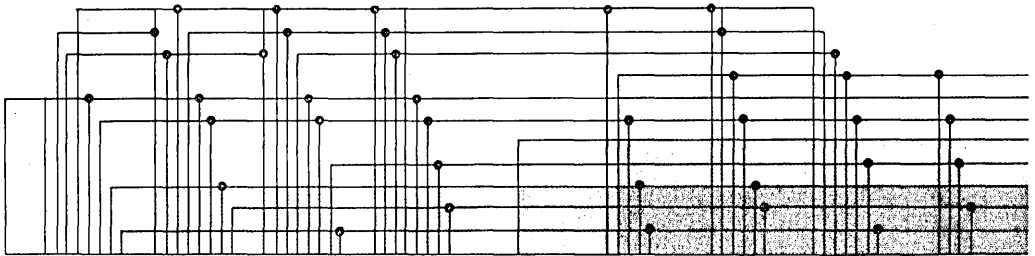


Figure 3-228. Bus Transfer Timing Diagram



LOGICAL ALTERNATIVES IN SUPERMINI DESIGN

Becoming the technology of choice in superminicomputer design, programmable array logic mixes with alternative devices to meet design constraints.



**by Bradford S. Kitson and
B. Joshua Rosen**

Programmable array logic devices have been a driving force behind the latest generation of 32-bit superminicomputer designs. These superminicomputers range from redesigns of existing architectures that reduce cost or overcome packaging limitations, to designs requiring ultrafast turnaround, to high performance architectures specified to take full advantage of such devices.

Superminicomputers designed with programmable array logic (PAL®) include the VAX®11/730

Bradford S. Kitson is section manager in product planning and applications for programmable logic devices at Advanced Micro Devices, 901 Thompson Pl, Sunnyvale, CA 94088. He holds a BS in electrical engineering and computer science from the University of California at Berkeley.

B. Joshua Rosen is manager of processor design at Dataflow Systems Corp, 42 Nagog Park, Acton, MA 01720. He was manager of processor development at Computervision Corp, Bedford, MA, when this article was written. He holds a BA in physics from Lawrence University and an MSEE from Northwestern University.

from Digital Equipment Corp, the MV/8000 and MV/10000 from Data General, and Computervision's APU™ (analytic processing unit). A reimplementa-tion of the original VAX-11/780, the VAX-11/730 sup-plies 25% of the performance in 10% of the board space. Programmable array logic was chosen by the MV/8000 designers to allow the shortest possible design cycle and to catch up with their competitors. The MV/10000 upgrades performance of the MV/8000. The Computervision APU was designed for high performance with PAL devices in mind, and provides excellent examples of how to use such devices to their fullest.

Logic design alternatives

Although by no means the only option, pro-grammable array logic has become the technology of choice in superminicomputer design. Logic design alternatives (Fig 1) include standard prod-ucts (fixed-function devices), semi-custom (programmable logic, gate arrays, and standard cells), and fully-custom logic devices. In supermini-computer design, the primary alternatives are stan-dard products, gate arrays, and PAL devices. The best choice for any given function depends upon the design alternative capabilities, the design con-straints, and the function actually being implemented.

Reprinted with permission of COMPUTER DESIGN

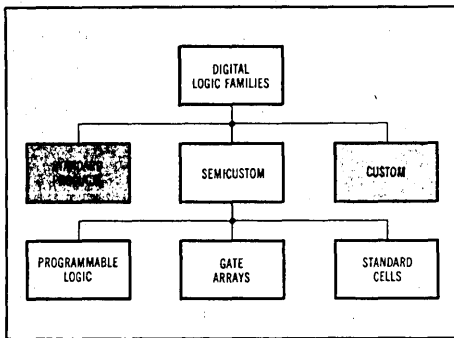


Fig 1 Basic categories of digital logic present designers with trade-off opportunities. Standard products fit where cost is a major concern; custom and semi-custom products can be used where high diversity is desirable.

Via fuse programming, a PAL device allows the designer to construct a custom device or group of devices that precisely implement a desired function. In contrast, fixed-function transistor-transistor logic (TTL) small scale integration/medium scale integration (SSI/MSI) alternatives seldom seem to fit any application in the desired way. Thus, the SSI/MSI designer usually pays penalties via extra logic levels in the critical path and an increased package count. Gate arrays allow custom devices to be created via mask programming. However, designers using a gate array must finalize architecture early in the design cycle. Any errors will require a mask change, which can take months.

In most cases, the best choice is a combination of the alternatives and probably includes some memory as well. Typically, the six basic design constraints are performance (speed), cost, density (packaging), power dissipation, reliability, and design turnaround. Assigning a priority to these constraints will usually define the logic alternative that a machine is based on. Actual implementation trade-offs are made at the function level. The three basic functional portions of a design are data path, control path, and interface.

Standard products have their place

Standard products are defined as devices created for a wide market. Examples of standard products are TTL SSI/MSI, fixed instruction set metal oxide semiconductor (MOS) microprocessors, and microprogrammable large scale integration (LSI) building blocks. These devices are usually multiple sourced and produced in high volume, resulting in lower individual device costs. In a design where cost is the main concern, and performance, power dissipation, density, and design turnaround are of little or no importance, standard products are probably the best choice. In a design such as a superminicom-

puter, where these other considerations have a high priority, inherent disadvantages limit standard product use.

While SSI/MSI devices offer fast individual gates, on a system level their density, power dissipation, and reliability characteristics are not as good as those of the alternatives. In addition, design turnaround characteristics are inadequate because changes usually require printed circuit (PC) boards to be laid out again. In most superminicomputers, therefore, SSI/MSI gates are used only in selected critical path functions that require one or two gate levels (at the expense of density and power dissipation), and in interface applications, such as bus buffers, latches, registers, and transceivers.

Standard LSI products (mostly bipolar) offer exceptional performance, power dissipation, density, and reliability characteristics, but their inflexible architectures limit their application range. Superminicomputer designers rely on proprietary, highly complex architectures to differentiate their designs from those of their competitors, and LSI imposes an architecture. Therefore, applications are limited to general purpose, well-defined functions in the data path, such as parallel multipliers and arithmetic logic units (ALUs) that can benefit from their high performance characteristics.

Gate arrays are "cast in concrete"

Semi-custom gate arrays are defined by integrated circuit (IC) manufacturers as large arrays of unconnected gates. End users specify how gates are interconnected with actual interconnection occurring at the metal-mask layer of the IC process. The main advantages of gate arrays are high density and the ability to customize a design, while primary disadvantages are cost and design turnaround. Each custom device is single sourced and low volume; therefore they are not cost-effective unless the application is density limited or the volume is very high. Gate arrays can adversely affect design turnaround because the system designer must design both the IC and the system in which the IC is used. In addition, any change in the gate array requires new masks and a delay for each mask iteration.

By using gate arrays only where the design can be defined early, designers minimize these turnaround disadvantages. They then hedge their bets by surrounding the gate arrays with logic that can correct any design bug(s) discovered later on. As in LSI, what is "cast in concrete" is usually the data path. However, gate arrays allow more of the data path to be integrated because the device can be optimized for specific design requirements. A proprietary 16-bit ALU slice might be a typical gate array.

Gate arrays are also used to interface multiple onboard buses together as data path "glue." Control-path and interface applications, however, tend to be too likely to change. Therefore, control

path functions are based on implementation techniques such as writable-control-store (WCS). Interface applications frequently require changes because of the need to interface one designer's board to another's. Consider the critical timing between a cache, an instruction fetching unit, and multiple register-files. The exception handling capability required for typical operations, such as a cache miss, can be an indeterminant problem affecting all of the above-mentioned functional units in the system. Should a bug occur in the cache unit, the interface section of all other units will have to be changed to accommodate the correction.

PAL structure paves the data path

Combining simplicity and flexibility, the basic PAL structure is a fuse-programmable AND gate array that drives fixed connection OR gates, allowing logic to be implemented in sum-of-products (AND-OR) Boolean form.

By selectively blowing the appropriate fuses, a PAL device can implement any logic function as long as the number of inputs or AND gates required does not exceed the number provided in the device

chosen. In the AmPAL16R4, for example, the true and complement version of each of the 16 inputs is connected via fuses to each of the 64 AND gates in the device (Fig 2). PAL devices provide additional features, such as programmable input/output (I/O) pins and registered outputs with internal feedback that further enhance their ability to implement logic functions efficiently. Programmable I/Os are especially useful for trading off the number of device outputs for inputs to fit the exact number required by the logic functions being implemented. Internal registered feedback is desirable when implementing complex state machine designs.

Programmable logic devices, like gate array, are semi-custom devices. Created by the IC manufacturer, they are alterable by fuse programming for a specific application. Designed specifically for logic-oriented applications, PAL devices enable designers to create custom devices with fast turnaround time. PAL devices compare favorably to alternative devices in terms of performance, cost, density, power dissipation, and reliability. They fall behind gate arrays and LSI in density and power dissipation, and behind SSI/MSI in individual device cost.

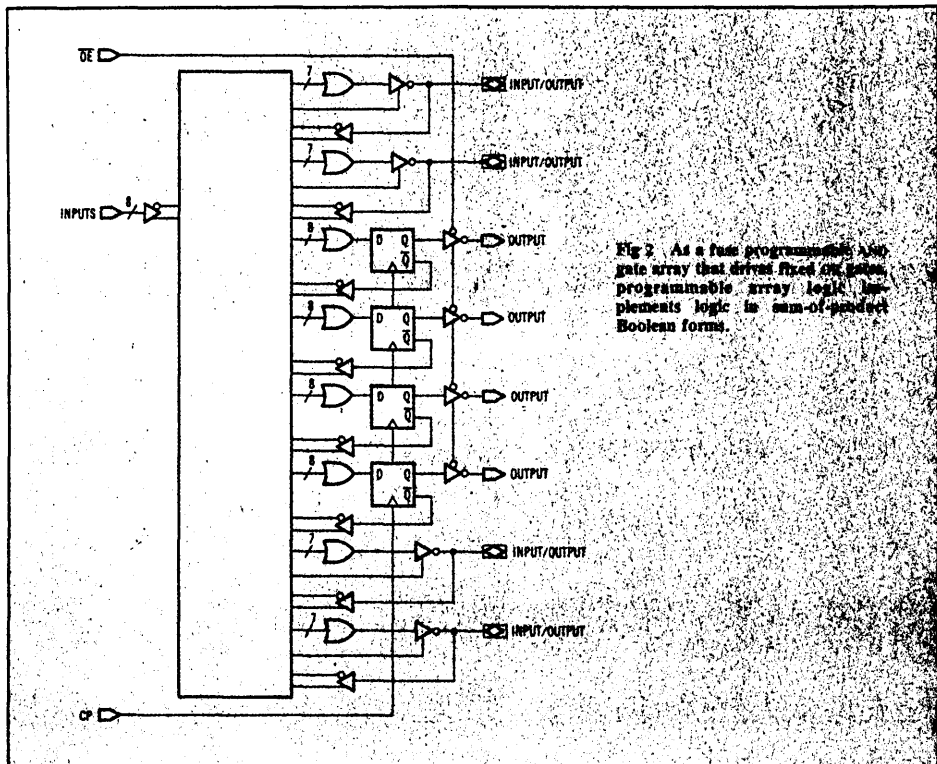


Fig 2 As a fuse-programmable AND gate array that drives fixed OR gates, programmable array logic implements logic in sum-of-products Boolean forms.

(Note however, that PAL devices are cheaper on a system basis.)

Performance and density characteristics of PAL devices have led to significant applications in the data paths of superminis where they are used along with LSI and/or gate arrays. These devices serve to "glue" the LSI and the "cast in concrete" gate arrays into the system. If necessary, the design turn around capability that they display can be used to optimize the data path architecture and, in some cases, fix a bug in a gate array by reprogramming the PAL devices around it. Typical data path functions for the devices include barrel shifters, masking, code conversion, and multiple bus interface.

Design turnaround becomes especially beneficial in the control path and interface portions of a design. Most control path functions are highly random and are prone to change and/or error. While WCS allows design changes to be made by rewriting microcode, PAL devices permit changes that cannot be made in microcode, or would adversely affect system performance. For instance, one supermini-computer manufacturer has established the rule that a gate array can be used in the control path only if eight or more PAL devices are necessary for the same function. In interface design, from a density standpoint, PAL devices allow the interface to merge with the data path "glue" function. Since the interface is just as likely to change as the

control path, but without the benefit of WCS, design turnaround becomes a factor. If one board's designer needs to change the interface, the designs of many other boards are impacted. Updates on all boards can be easily made by reprogramming one or more PAL devices. These devices can also provide the drive capability required by this application area.

Designing with PAL devices in mind

Trade-offs among PAL devices and other digital logic alternatives were key factors in Computervision's APU design. In conjunction with standard products, such as ALUs, LSI multipliers, and memory devices, PAL devices were used to create a patented architecture not feasible in standard TTL SSI/MSI.

Designed as a very high speed 32-bit supermini-computer for engineering applications, the Computervision APU is twice as fast as competitive designs, yet occupies the same board space. Its designers, instead of merely replacing TTL SSI/MSI with PAL devices, used PAL as customizable logic building blocks. This allowed them to implement powerful logic functions in a minimum of space.

The APU processor board set is divided into four modules. The parser/sequencer contains an instruction processor that fetches and decodes instructions in parallel with the execution unit. The control processor performs address and integer computations and the floating point pipe (FPP) performs both scalar and vector floating point operations. The cache/address translation unit contains a 256-slot area page table entry cache and a 16K-byte memory cache.

Approximately 25% of the chips in the APU board set are in PAL. Since the APU is the first implementation of the new CPU architecture, many design aspects are subject to change as the architecture evolves. PAL devices permit designers to rapidly modify hardware to fit the architecture needs, and to implement feature and performance enhancements with minimal impact on the development schedule.

Ability to generate a very large number of custom ICs (over 200 different PAL codes are used in the APU), significantly reduces the processor's size while increasing overall performance. This means that, although the APU and Digital Equipment Corp's VAX-11/750, a gate array-based machine, consume exactly the same amount of board space, the APU is more than twice as fast. In fact, the APU's Fortran performance is substantially faster than that of the VAX-11/780, a machine that consumes 5.2 times as much board space as the APU.

Designed as a high speed arithmetic extension to the APU execution engine, the APU FPP, unlike comparable machines, is an integral part of the

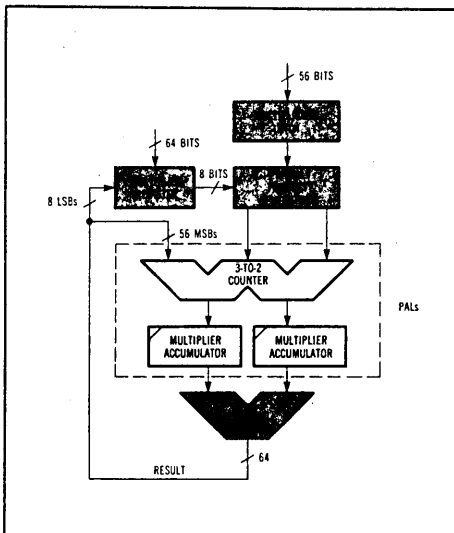


Fig 3 Multiplier calculates 56-bit x 56-bit product. Partial product generation logic uses seven 8 x 8 slices to form 8 x 56-bit multiplication array. Carry-save adders implement 3-to-2 counting techniques to form two operands that are summed in a lookahead ALU.

Without adding pipe latency, the APU is able to accumulate partial products at a rate of 8 x 56 bits every 112 ns.

combination of 3 equally weighted bits can be recoded into a 2-bit field.

This makes it possible to reduce the three operands generated by the multiplication process (high and low partial products and 64-bit intermediate product) into only two operands which may then be summed together in a single lookahead ALU. The 3-to-2 recoding requires no-carry propagate logic and is therefore very fast. Only one level of pipelining is required because of the 3-to-2 counter's speed, resulting in both a reduced parts count and a reduced pipe latency.

In the APU floating point engine, 16 AMPAL16R6s, programmed as triple 3-to-2 counters, are used to reduce the three multiplication operands to two intermediate results [Fig 4(b)]. The registered PAL outputs are connected to the input buses of the Mantissa ALU that is also used for floating point addition and subtraction. The Mantissa ALU then calculates the next intermediate product in parallel with the partial products calculations occurring in the 8 x 8 multipliers. This intermediate product and the new partial products are recoded by the 3-to-2 counter PAL devices to form the next pair of intermediate results. This process continues until the complete 56 x 56 product is generated. Thus, without adding pipe latency, the APU is able to accumulate partial products at a rate of 8 x 56 bits every 112 ns, which coincides with the basic nanocycle machine time.

Barrel shifter, a 3-level implementation

The APU's barrel shifter performs left shift, right shift, and rotate operations of 0 to 63 bits in a single microcycle. Used mainly for floating point prescale and normalize operations, the barrel shifter (Fig 5) is implemented in three stages: the word rotator, nibble shifter, and bit shift and mask logic. It is controlled by associated prescale, leading zero detect, and mask control logic.

Prescale logic converts the signed difference produced by the exponent arithmetic units based on the comparisons of the two operand exponents, into an absolute shift distance. This shift distance is then used to right shift (prescale) the smaller operand Mantissa of a floating point add or subtract operation. The leading zero detect logic determines the left shift distance required to produce a left-justified (normalized) result. Mask control logic converts rotated data to shifted data by masking off the appropriate leading or trailing bits to

implement right or left shifts. These three sections are implemented in PAL.

Implementing the 3-level, 64-bit barrel shifter in MSI requires the use of AM25S10 4-bit shifters. The first level is the word rotator which performs a circular rotate of 0, 16, 32 or 48 bits. Although implementation is simple, the MSI solution requires 16 packages. The second level is the nibble shifter, essentially identical to the word rotator but wired to rotate 0, 4, 8, or 12 bits. The final barrel shifter stage requires not only bit rotate but also leading and trailing bit masking and sticky bit computation (ie, the logical OR of the masked-out bits). An MSI solution requires not only the 16 packages of AM25S10s, used in each preceding level, but also 16 AND gate packages for masking, with another 16 AND packages for the sticky bit computation. Control logic for the mask operation requires as much logic as the entire shift path. A more practical solution consists of building separate left and right

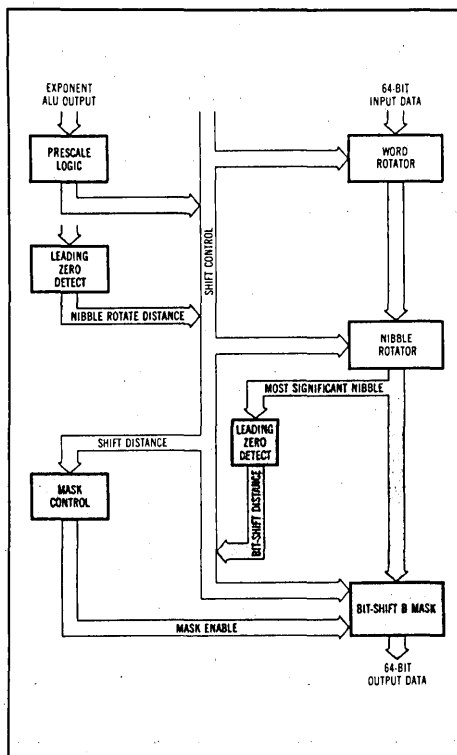


Fig 5 Implementation of 64-bit "Nearest Neighbor Shifter" is in three stages: word rotator, nibble shifter, and bit shift and mask logic. Control derives from associated prescale, leading zero detect, and mask control logic.

internal architecture and not an optional add-on. As a result, the FPP not only accelerates scalar and vector floating point arithmetic, but also performs byte, word, double-word, and quad-word string operations. In addition, it serves to enhance the performance of important nonfloating point instructions such as Procedure Call and Return. A total of 79 PAL devices are used for both control- and data-path applications on the FPP board.

Counter solves multiplication problem

The heart of the FPP is the multiplier section (Fig 3). Double-precision floating point multiplication requires the calculation of a 56-bit x 56-bit product. Unfortunately, 56 x 56 parallel multipliers do not exist on silicon. From a cost/performance standpoint, the best solution is to use a number of small multipliers to build an intermediate sized parallel multiplier and then produce a large product (56 x 56) in multiple cycles.

Partial product generator FPP logic uses seven Am25S558 8 x 8 multiplier slices to implement an 8 x 56-bit multiplication array. Each multiplier chip produces a 16-bit product. In general, the 8 most significant bits (MSBs) of each partial product generator must be added to the 8 least significant bits (LSBs) of the next higher slice to generate the full

64-bit partial product. Exceptions are the 8 MSBs and LSBs.

This technique also requires the ability to accumulate partial products with the partial products from previous cycles. Thus, each cycle must be accompanied by two additions: the partial product summation and the intermediate product accumulation. While this can be done by following the multipliers with two levels of lookahead adders, usually 74S181s, the resulting nanocycle time is approximately three times longer than the partial product generation time of the 8 x 8 multipliers. Modifying this scheme, however, by adding registers between each level of logic, the pipeline multiplier reduces the nanocycle time to near the propagation delay time of the multiplier chips plus the clock to output time of the multiplier register plus the setup time of the intermediate result register. This scheme has two disadvantages: increased pipe latency, caused by the two extra levels of pipelining, and a high parts count.

Still another technique involves replacing one level of the pipe and one level of lookahead adders with carry-save adders between the partial product generators and the pipeline registers. Carry-save adders are used to implement a technique that is called 3-to-2 counting. As seen in Fig 4(a), any

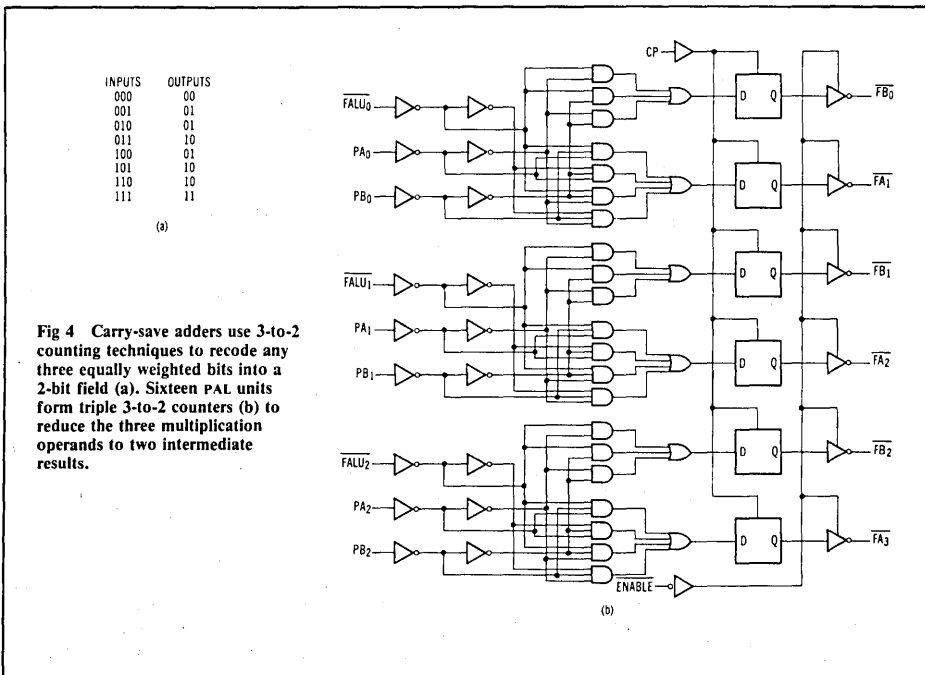


Fig 4 Carry-save adders use 3-to-2 counting techniques to recode any three equally weighted bits into a 2-bit field (a). Sixteen PAL units form triple 3-to-2 counters (b) to reduce the three multiplication operands to two intermediate results.

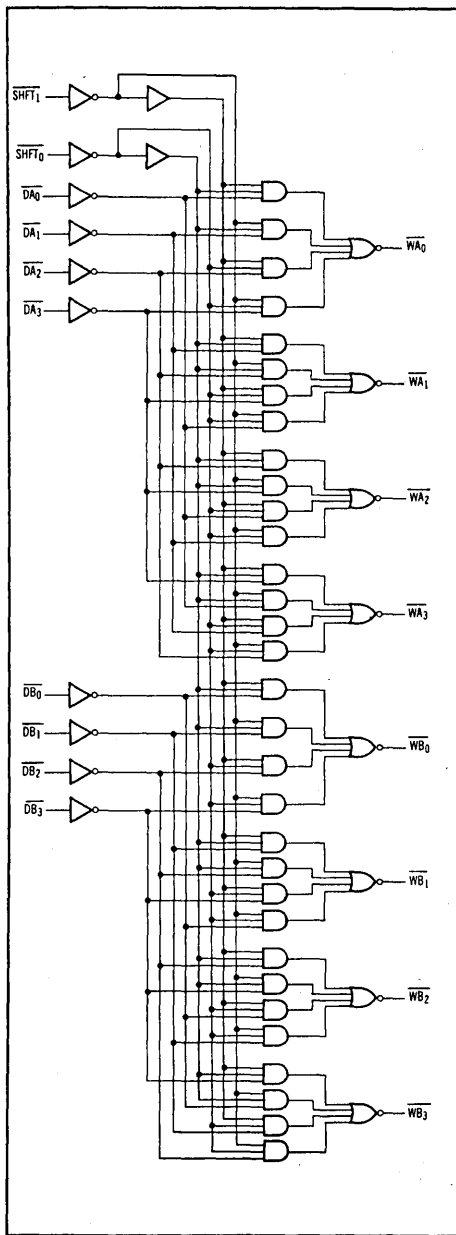


Fig 6 Word rotation, first level of 3-level shifter, consists of eight PAL units, each programmed as two 4-bit rotators. It performs a circular rotate of 0, 16, 32, or 48 bits.

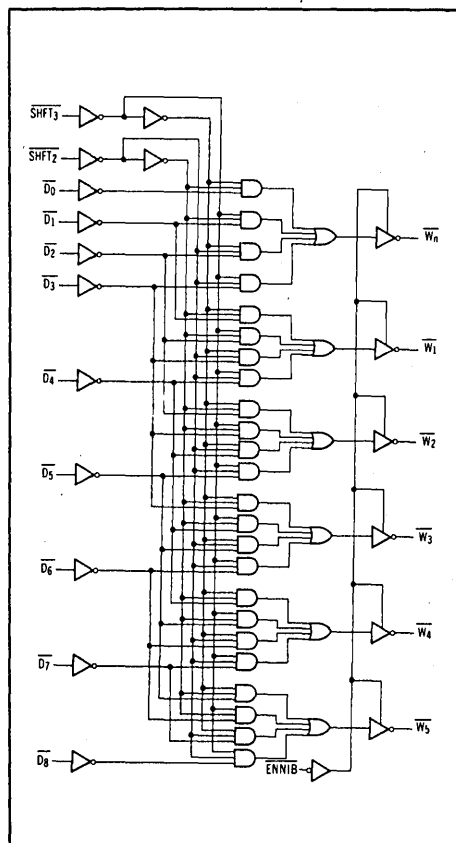


Fig 7 Nibble shifter makes up second level of barrel shifter. It is wired to rotate 0, 4, 8, or 12 bits.

shifters, 48 packages apiece, and not implementing a sticky bit at all.

Implemented in PAL, a 64-bit rotator and shifter with sticky bit computation requires considerably fewer packages than a unidirectional MSI shifter. The word rotator consists of eight identical PAL devices programmed as two 4-bit rotators/package (Fig 6). The nibble shifter requires four Am25S10s and eight PAL devices programmed as 6-bit wide, 4-place shifters (Fig 7). The bit shift and mask logic requires 16 PAL devices in the data path and two PAL devices in the control path. (Fig 8).

To implement the masking function required for shifting, a technique called "Nearest Neighbor Shifting" (U.S. patent pending, Computervision Corp) is used. Each shift and mask PAL device has an enable input from one mask control PAL unit. In addition, each shift and mask PAL device is also

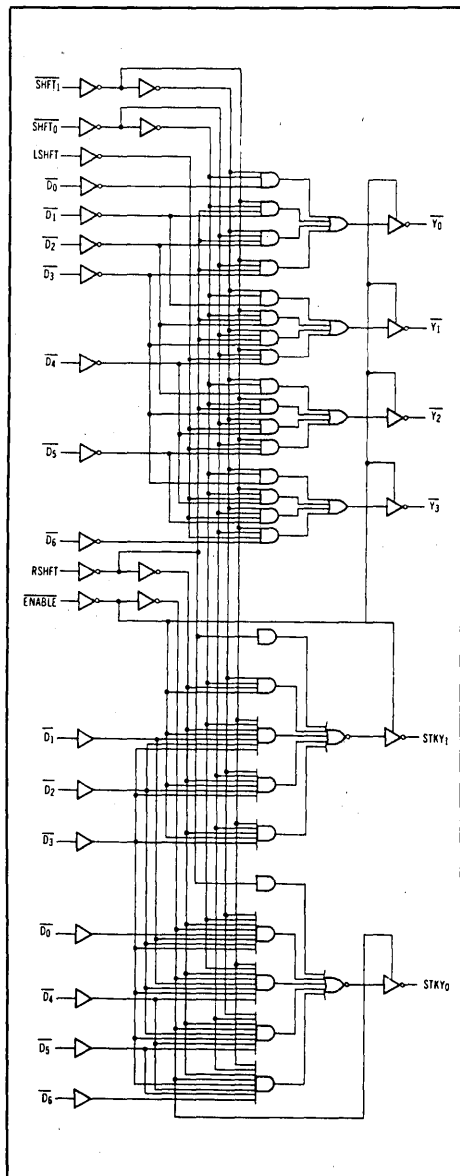


Fig 8 Bit shift and mask logic for barrel shifter requires 16 PAL devices in the data path and two PAL devices in the control path. Each shift and mask PAL device has an enable input from one mask control PAL device and is connected to enable inputs of its left and right neighbors. The 64-bit masking operation requires only 16 control lines.

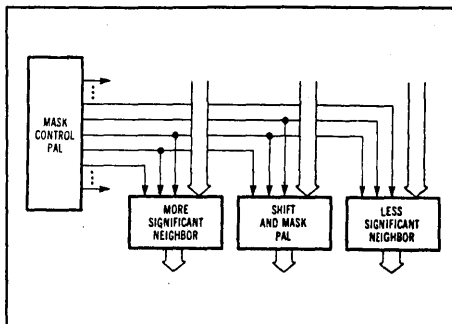


Fig 9 "Nearest Neighbor" interconnection is used to implement masking function for shifting. A mask control PAL device determines if all 4 bits are to be masked; enables from the adjacent slice determine if the PAL device is at shift boundary.

connected to the enable inputs from its left and right-hand neighbors (Fig 9). The mask control PAL input determines if all 4 bits from the slice should be masked off. Enables from the adjacent slice determine if a PAL unit is at a shift boundary. If only one of the neighboring slices is disabled, the shift and mask PAL unit masks off from 0 to 3 of the bits adjacent to the disabled slice, depending on the bit-rotation distance. In this way, the 64-bit masking operation can be implemented with only 16 control lines as opposed to at least 64 for an SSI/MSI solution.

In addition to performing the final shift and mask operation, the bit-shifter PAL unit also computes the logical OR of the masked-out bits at each slice position. These outputs are then logically ORED to generate a sticky bit. The extra hardware required is less than two SSI packages. The entire PAL barrel shifter requires 38 devices to implement 64-bit rotation, left shifting, right shifting, and sticky bit accumulation. An MSI-based left/right shifter, without sticky bit computation, requires a minimum of 96 parts. In addition, the logic required for implementing the prescale and normalize operations is significantly reduced through the use of PAL devices.

PLDs As Semicustom Substitutes

Om Agrawal, Advanced Micro Devices Inc., Sunnyvale, CA

Jim Beck, Chronon Computer Corp., Mountain View, CA

Today's system designers can choose from at least five ways to implement their applications: Standard products (SSI, MSI, LSI—fixed-function devices), programmable logic, gate arrays, standard cells, and full-custom logic devices. With each alternative come tradeoffs related to architectural flexibility, time, and development cost (Figure 1). The best choice for a given design usually depends on the designer's priorities. Table 1 illustrates the selection criteria for various alternatives.

Generic Advantages of PLDs

Programmable logic devices (PLDs) are off-the-shelf customizable devices that combine the flexibility of custom logic with the easy availability of standard products. The major advantages of PLDs over other semicustom solutions such as gate arrays and standard cells include reduced development time, greater design interactivity, and lower overall system development cost.

PLDs offer the fastest design cycle of any semicustom device. Since the architecture of a PLD is defined by its programmable fuse pattern, PLDs can be programmed within hours, instead of the weeks to months required for gate arrays and standard cells. Besides allowing an interactive approach to system design, this fast turnaround time encourages experimentation in the sense that different architectures can be tried out with a minimum of overhead. If the implementation does not work, new devices can be programmed quickly and inexpensively, in comparison to other semicustom alternatives. The system development costs and capital equipment costs associated with PLDs are also substantially less than those of gate arrays and standard cells.

In addition, the total engineering effort required to design, test, debug, and bring into production a system using PLDs is substantially less than that required for other semicustom alternatives. Currently, high-level tools are being developed to further simplify the process of designing with PLDs. Compiler-based software languages for specifying original logic equations can generate a minimum number of intermediate equations with built-in optimizers. There are support tools, such as simulators and test vector generators, for debugging and testing in both the prototyping and production environments.

Architectural Variations in PLDs

Substantial progress has been made with PLDs in the past few years. They are currently accepted as a viable option for system design, and they are becoming a preferred approach to logic design.

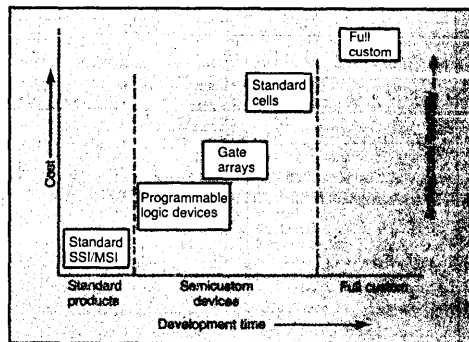


FIGURE 1. Development time, cost, and architectural flexibility tradeoffs.

PROMs, PAL devices, and PLAs constitute the three most prevalent programmable logic devices. All these PLDs have the same basic two-level AND-OR architecture, but they vary in their allocation of logic features and their methods of programmability.

The basic AND-OR structure makes PLDs natural for implementing logic equations in Boolean sum-of-products form. However, unlike PROMs, which have a fixed AND array structure, the programmable AND array structure of PALs and PLAs makes them suitable for handling a larger number of inputs. This programmable AND structure also means that the inputs are not fully decoded, allowing more than one address to select the same word. Also, more than one word in the array can be selected simultaneously. PAL devices and PLAs therefore overcome one of the key inefficiencies of PROMs: a fixed number of inputs. Because the PAL devices do not need the fuses and programming and testing circuitry for the OR array, they are typically 15% faster than PLAs for a given logic capability.

Besides supporting a greater number of inputs, the other architectural benefits of PAL devices include programmable bidirectional I/O pins, programmable combinatorial or registered outputs, polarity control, and flexible product term distribution.

The basic PAL architecture incorporates a fuse-programmable AND array that drives fixed-connection OR gates. By blowing the appropriate fuses of the AND array, a designer

*PAL is a trademark of, and used under license from, Monolithic Memories, Inc.

	Programmable PAL	Programmable PAL	Gate Arrays	Standard Cell	Full Custom
Time to Market	Short-Medium	Short	Medium	Long	Very Long
Development Lead Time	Immediate	Hours	Weeks/Months	Wks/Months	Years
Development Cost	None	Low	Medium	Medium/High	Very High
Availability	High	High	Medium	Low	Low
Alternate Sources	Many	Many	Poor	Poor	Poor
Volume Independence (Sensitivity)	Low	Low	High	High	High
Application Support	Much	Some	Some	Some	None
IC Expertise	None	None	Some	Some	Some
Architectural Flexibility	Low	Medium/High	High	Higher	Highest
Design Ease/Changes	Medium	High	Low	Low	Low
Performance	Low/Medium	Medium	Medium/High	Medium/High	High
Logic Density	Low	Medium	Medium/High	Medium/High	High
Cost of Design Changes	Low/Medium	Low	High	High	Very High
Solution Efficiency	Low	Medium	High	High	Very High
Reliability	Medium/High	High	Medium High	High	High

TABLE 1. Selection criteria for different alternatives.

can create a PAL device to implement any logic function expressed in a sum-of-products form, so long as the number of inputs per AND gate, and the number of OR outputs, do not exceed the capacity of the given device.

For a PAL device, the degree of its programmability and architectural flexibility is determined somewhat by the number of fuses that form its programmable AND/OR gate. Each programmable AND gate is called a product term; these can be configured to provide either the logic functions (for forming the outputs) or control functions (for generating control signals). The first generation, 20-pin PAL devices such as 16L8 or the 16R8 have 64 logical product terms and 8 control product terms, with a total of 2K fuses. The second-generation PAL device, illustrated by the AmPAL22V10, has an array of 5808 fuses with a total of 132 product terms. Of these 132, 120 are logical product terms and 12 are control product terms (Figure 2).

Each product term consists of a fixed number of inputs. These inputs may be dedicated or fed back, or they may consist of bidirectional I/O pins. Whereas the first-generation devices had 32 inputs per product term, the 22V10 has 44 inputs to drive each product term (up to 22 inputs with both its true version and its complements).

Programmable I/Os allow the programming of the output buffers through a control product term; they also allow the output to be configured as a dedicated output pin, a dedicated input pin, or a dynamically controllable input/output. This makes the device useful in fitting the exact number of inputs

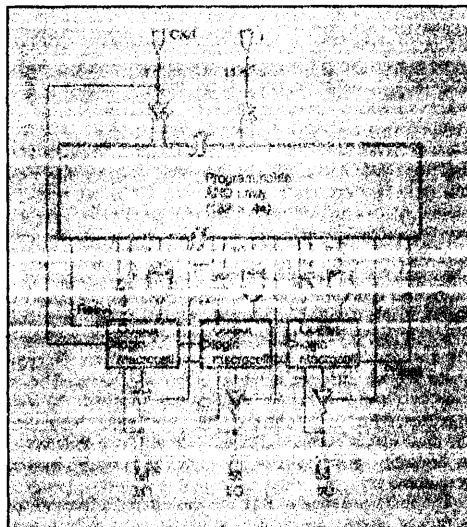


FIGURE 2. Block diagram of the AmPAL22V10 second-generation PAL device.

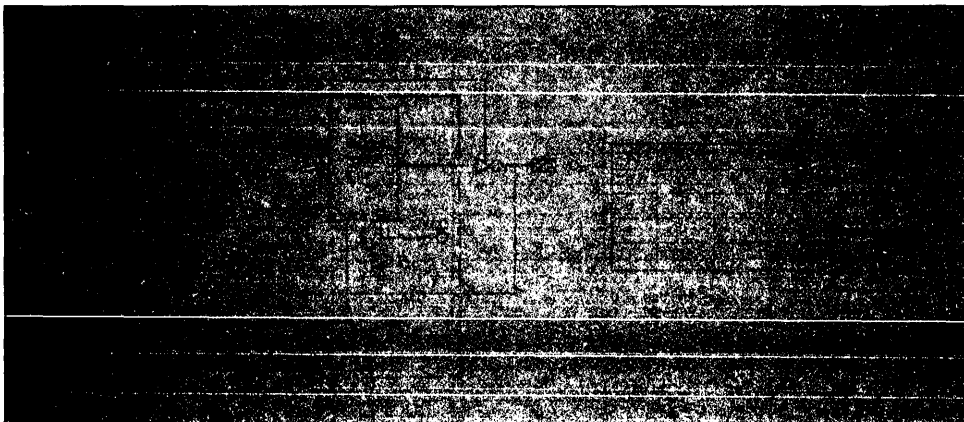


FIGURE 3. The 22V10 output logic macrocell diagram.

or outputs required for desired functions. To be used as an output, the output buffer is always enabled (i.e., all the fuses associated with that product term are blown), and the pin behaves as an output pin. For an input pin, the product term associated with that output buffer is unused. Dynamically controllable input/output buffers result from the output-enable product term being enabled or disabled by a logical combination of one or more inputs. Such buffers are useful in microprocessor bus-oriented applications, such as data steering, storage, or manipulation.

Output macrocells increase the regularity and design freedom available with the PAL devices. These macrocells allow any or all of the I/O pins to have either registered or combinatorial outputs, of either polarity (active high or active low). The programmable output-polarity control feature of the macrocell allows the designer to program either the true or complementary version of a logical term, whichever makes the most efficient use of the chip's AND array. Also, associated with each macrocell is an individual output-enable control term that configures the I/O pins for either input or output. Macrocells with individual I/O-enables allow designers to customize each chip's architecture to fit its own application, using different combinations of registered and combinatorial inputs and outputs (Figure 3).

Another concept that is quite useful to system designers is the "variable distribution of product terms." This approach aids in implementing functions such as counters, exclusive OR functions, or complex state machines, where different states require different numbers of product terms. This is especially useful in counter applications, in which the least significant bit(s) of a counter require fewer product terms than the most significant bit(s). Besides offering more architectural flexibility, this results in optimum use of chips' internal resources.

The first generation of PAL devices offered a fixed distribution of product terms, with a maximum of eight logical product terms per output. With variable distribution of product terms, and a larger number of product terms per output (up to 16), the 22V10 allows more complex functions to be

implemented than was previously possible in a single PAL device.

Application of PALs in System Design

A system designer typically partitions his task into three general areas: data-path, control-path, and interface (glue) applications.

Data-path applications typically include data manipulation (ALUs), data storage (register files, pipeline registers), and data steering/selection (multiplexing/demultiplexing). In these applications, performance and density are critical. Typically, the data-path portion of a system is the most structured portion, so this is likely to be defined relatively early in the design cycle and is unlikely to require changes during prototyping. Hence turnaround time normally is not critical for the data-path portion of a typical system.

Control-path applications usually include the timing, sequencing, and decision-making portions of a digital system. These sections are normally implemented with state machines—either random logic in either PROMs or RAMs. The control section is the most complex portion of a digital design, and it is likely to contain subtle errors and to require many changes during prototyping. Performance and turnaround time tend to be critical for the control path, while density considerations are less important.

Interface or glue applications fulfill the miscellaneous functions. Typically, interface circuitry connects LSI modules, such as microprocessors, peripherals, and gate arrays. For interface applications, turnaround time is quite critical.

PAL devices, because of their general-purpose nature, are used in all of these applications—data path, control, and interface. In the data path, they can be used for data steering and data manipulation. Using PAL devices for data steering simplifies the implementation of a multiple-bus architecture. Data manipulation may include functions such as 16/32 bidirectional shifters, barrel shifters, constant-generation logic, and sign-extension logic. PALs are also used extensively for optimizing control functions, such as instruction pre-decoding, double pipelining control, register file control, and

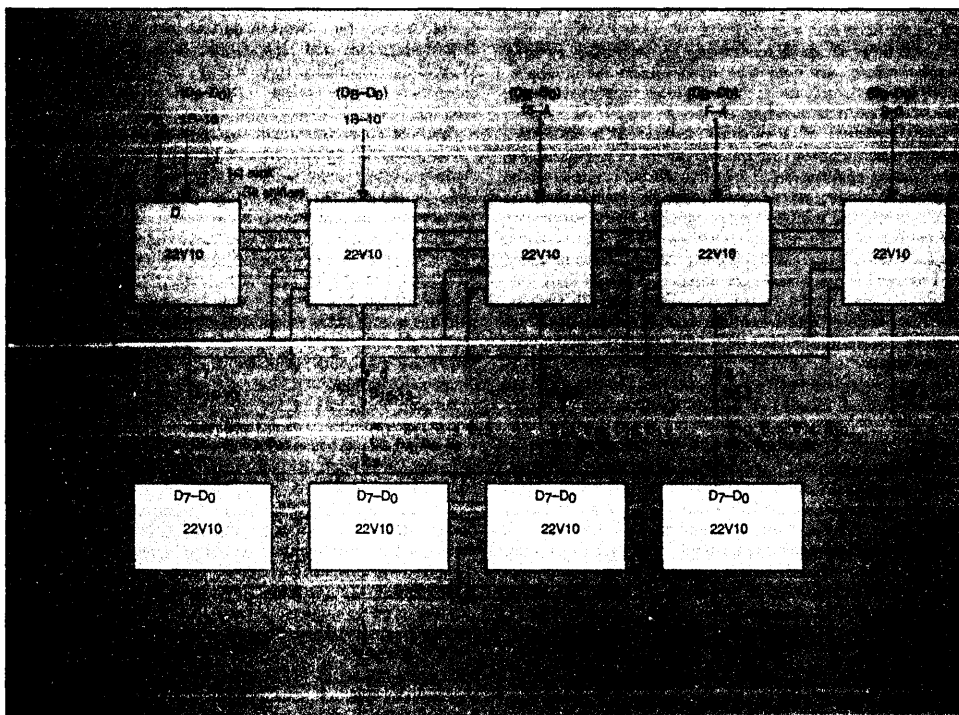


FIGURE 4. A 32-bit bidirectional shifter using 22V10s.

special-instruction control.

Most of the design support for PAL devices has been concentrated on either sequential controllers or expanded versions of available MSI parts. However, in these two applications, alternative choices often diminish the appeal of PALs. For example, in sequential-controller designs, PROMs compete with PAL devices, because they do not have the product term limitation of the PAL devices. In the case of expanded MSI functions, the idea of implementing a 28-bit counter with a single 40-pin PAL device, for example, requires careful cost analysis before one can seriously consider replacing multiple-sourced SSI and MSI chips.

Here we will look at some alternative uses for PAL devices that are not normally presented in the literature. The applications described make extensive use of the Am22V10. Although the 22V10 is not the most complex device offered in the PAL universe, it does have the speed and features necessary for high-performance TTL designs.

Two applications using the 22V10 are presented here: a 32-bit bidirectional shifter, and an interrupt mask and control register. The shifter is a relatively straightforward design that can easily be implemented with SSI/MSI multiplexers; however, using PAL devices offers some significant component savings. The interrupt mask and control register demonstrates how PALs can combine several functions to achieve a clean

design of a traditionally messy circuit. The two applications make use of some features of PAL devices that are especially valuable for digital system design, as follows.

Control Decoding/Pre-Decoding. This function is normally done outside the data path, early in the CPU clock cycle, but can often be performed in parallel inside a data-path PAL. The savings in both clock time and complexity of control can be substantial.

The use of PAL devices as macro instruction registers illustrates how chips can be used to optimize a routine function and help pre-decode the functions. An instruction register typically is used to receive instructions from the data bus during memory fetch operations. The instruction is then decoded by the control sequencer section and executed. Separating the information used during the decoding cycle from the execution cycle helps improve performance. This may allow the instruction register to pre-decode instructions "on the fly" as they come off the data bus, without losing any data in the instruction word. This results in a significant increase in control-sequence speed.

Concatenation of Functions. This reduces parts counts and eliminates the associated interchip delays. A simple example is an MSI part followed by a latch or register, which can easily be replaced with a PAL.

Reduction of Signal Loading by PALs. When several MSI

parts are replaced with a single PAL, the number of loads and the signal delay are both reduced.

Availability of Multiple Outputs of Selectable Polarity Simultaneously. As with the signal loading point above, multiple outputs may mean faster signals, simplified wiring, and simplified data steering.

Implementation of a program counter (PC) with PAL devices is an excellent example of how PALs can be used in data steering applications. Typically, the PC is implemented as an incrementing register that is parallel-loaded from the ALU output, and which sources multiple buses: a memory address bus to initiate memory fetch operations, and an internal bus for relative branch address computation. Since a typical MSI device has few three-state outputs, it cannot drive two separate buses. Implementing this with SSI/MSI devices, therefore, requires additional three-state controls adding unnecessary delay in the memory path, compared to the PAL-device solution.

Inclusion of Additional Functionality Not Envisioned in the Original Design. Whether this takes the form of "fixing an oversight" or "upgrading the features," unused PAL pins and extra product terms are available for the system designer to consider "what else" a computer might do.

Support for Minimum Parts Count. Often, the objective is to build high-performance CPUs that require a minimum of board space. This requires a design with as few parts as possible while implementing a complex machine in parallel. For such applications, PAL devices can aid system designers, because of their ability to implement complex custom functions at relatively high speeds.

A 32-Bit Bidirectional Shifter Using PAL Devices

For data-path applications, designers often need greater bit-manipulation and bit-shifting capability than is offered by off-the-shelf SSI/MSI devices. A conventional shift register usually shifts its entire contents only one bit position in a clock cycle. Multiple bit shifts require multiple clocks, thus penalizing performance.

Bidirectional shifters can shift data both to the left and to the right. For a left shift, data is shifted (toward the most significant bit), usually with zero fill at the least significant bit (LSB). For a right shift, data moves toward the LSB, filled by a "bit" that is controlled externally. Usually, this fill bit is either zero (for zero fill) or a sign bit.

Designing a shifter with MSI to allow a simultaneous 32-bit bidirectional shift can be approached in several ways. Conceptually, 32 32-to-1 multiplexers would be the simplest design. Here, only one rank of parts would do the job in the time defined by the propagation delay of this multiplexer. However, the loading of each data and control bit (32 loads), the wiring mess, and the lack of real 32-to-1 SSI/MSI multiplexers in particular force slower, multiple-rank designs.

In a multiple-rank shifter, each level of logic (rank) performs a shift operation of something less than the full amount. The shift amounts of all the ranks, when multiplied together, must equal or exceed the total shift amount desired. For the 32-bit case, five ranks of 2-to-1 multiplexers would do the job ($2^5 = 32$), using 40 16-pin parts.

In order to minimize the chip count, off-the-shelf "shifters" or parts designed to permute input and output lines can

be used. Some of the available devices include the Am25S10 (TTL 4-bit), the F100158 (100K ECL, 8-bit) and the MC10808 (10K ECL, 16-bit). For the target 32-bit TTL bidirectional shifter design, 3 ranks of the Am25S10, with each rank using 8 devices, would perform the shift in 39 ns (typical) or 60 ns (maximum).

The design presented here uses only two ranks of shifters, made from PAL devices. The first rank, using five 22V10s, shifts the input bits to the left or right by 0, 1, 2, or 3 places. The second rank uses four 22V10s and rotates each input bit to one of 8 outputs Figure 4.

The first-rank shifter acts as a 7-to-1 multiplexer for each of the 32 output bits. The number of product terms available in the 22V10 varies between 8 and 16, depending on which output is used. The 32 data inputs are numbered from bit 0 to 1F, and the fill signal is controlled externally. This capability provides logical shift, with either zeroes pulled in, or any other bit controlled externally, for a shift-down operation.

Bits S0 through S4 provide the total (32-bit) shift amount. The actual number of bit positions to be shifted can be programmed by the user with these bits. The direction signal specifies either a left or right shift. Table 2 shows the output equations for B_{1F}, B_{1E}, B₁₂, and B₀ bits (intermediate signals).

The first-rank PAL devices provide 0-, 1-, or 3-bit bidirectional shift capability, specified by S0 and S1. The three-state control OE signal facilitates rapid data-bus access in bus-organized systems and can be used to increase the number of places shifted. As seen from Table 2, the equations use seven product terms to implement the 7-to-1 multiplexer function for the 7 left or right bits.

The second rank of PALs uses four 22V10s and rotates each input bit to one of 8 outputs. Just for providing the 8-place rotator function, exactly eight product terms are needed for each output, as shown below:

$$\begin{aligned} 1F &= /S2 * /S3 * /S4 * B_{1F} + \\ &S2 * /S3 * /S4 * B_{1B} + \\ &/S2 * S3 * /S4 * B_{17} + \\ &S2 * S3 * /S4 * B_{13} + \\ &/S2 * /S3 * S4 * B_F + \\ &S2 * /S3 * S4 * B_B + \\ &/S2 * S3 * S4 * B_7 + \\ &S2 * S3 * S4 * B_3 \end{aligned}$$

Because the AmPAL22V10 can provide up to 22 inputs and 10 outputs, it has more than enough capacity for this simple 8-bit rotating function. With its excess input/output and product-term capability, the AmPAL22V10 device is able to incorporate more functions in the second rank, as shown below.

The second rank of 22V10 shifters can operate in any four user-programmable modes, determined by the logic levels applied to pins F0, F1 (function pins) as shown in Table 3.

MODE 0

Mode 0 specifies rotation to the right by N nibbles as determined by the three bits S2, S3, and S4; it also inserts the "FILL" bit, which specifies the logic level to fill the positions left empty by the shift. The FILL bit is controlled externally. Mode 0 also specifies "LOAD WORD" operations from memory. For this, the shift amount has to be "0 nibble," because there is no filling or shifting in this mode.

S4 =	DIRECTION	*S0	*S1	*D6	+	
	DIRECTION	*S0	*S1	*D7	+	LEFT SHIFT
	DIRECTION	*S0	*S1	*D6	+	
	DIRECTION	*S0	*S1	*D6	+	PASS
	DIRECTION	*S0	*S1	*D7	+	FILL
	DIRECTION	*S0	*S1	*D6	+	RIGHT SHIFT
	DIRECTION	*S0	*S1	*D7	+	FILL
S3 =	DIRECTION	*S0	*S1	*D5	+	
	DIRECTION	*S0	*S1	*D6	+	LEFT SHIFT
	DIRECTION	*S0	*S1	*D7	+	
	DIRECTION	*S0	*S1	*D6	+	PASS
	DIRECTION	*S0	*S1	*D7	+	FILL
	DIRECTION	*S0	*S1	*D6	+	RIGHT SHIFT
	DIRECTION	*S0	*S1	*D7	+	FILL
S2 =	DIRECTION	*S0	*S1	*D5	+	
	DIRECTION	*S0	*S1	*D6	+	LEFT SHIFT
	DIRECTION	*S0	*S1	*D7	+	
	DIRECTION	*S0	*S1	*D6	+	PASS
	DIRECTION	*S0	*S1	*D7	+	FILL
	DIRECTION	*S0	*S1	*D6	+	RIGHT SHIFT
	DIRECTION	*S0	*S1	*D7	+	FILL
S1 =	DIRECTION	*S0	*S1	*D5	+	
	DIRECTION	*S0	*S1	*D6	+	LEFT SHIFT
	DIRECTION	*S0	*S1	*D7	+	
	DIRECTION	*S0	*S1	*D6	+	PASS
	DIRECTION	*S0	*S1	*D7	+	FILL
	DIRECTION	*S0	*S1	*D6	+	RIGHT SHIFT
	DIRECTION	*S0	*S1	*D7	+	FILL
S0 =	DIRECTION	*S0	*S1	*D5	+	
	DIRECTION	*S0	*S1	*D6	+	LEFT SHIFT
	DIRECTION	*S0	*S1	*D7	+	
	DIRECTION	*S0	*S1	*D6	+	PASS
	DIRECTION	*S0	*S1	*D7	+	FILL
	DIRECTION	*S0	*S1	*D6	+	RIGHT SHIFT
	DIRECTION	*S0	*S1	*D7	+	FILL

TABLE 2. 32-bit bidirectional shifter first-rank output equations.

S4	S3	S2	Function
0	0	0	Rotate right by 0 bytes
0	0	1	Rotate right by 1 nibble or 2 bytes
0	1	0	Rotate right by 1 nibble or 2 bytes
0	1	1	Rotate right by 1 nibble or 2 bytes

TABLE 4. 32-bit bidirectional shifter second-rank equations.

MODE 1

This mode specifies the "LOAD BYTE" operation, enabling a "rotate right" by N nibbles (or 2N bytes), after which a byte is loaded and empty bits are filled with the FILL bit. For this, the least significant bit (S2) must be 0, as shown in Table 4. The upper empty bits in the left-hand side are filled with the "FILL" signal.

MODE 2

This mode specifies "ROTATE LEFT" by N nibbles. The amount of nibbles to be shifted (N) is determined by S2, S3, and S4. For a left shift, the right end is filled with zeros. This mode also specifies store operation by byte, half-word (two bytes), and full word. The proper amount of shift, specified in S2 through S4, has to be provided by the appropriate external control signals.

MODE 3

This mode is similar to Mode 1, except that it loads a half-word (2 bytes). Data is always right-aligned.

As shown in Table 5, the most significant bit needs 22 product terms. After optimization, these product terms are reduced to 14 product terms, a number that the 22V10 can accommodate.

Mode	F1	F0	Function
0	0	0	Rotate right by N nibbles (specified by S2-S4) and LOAD WORD.
1	0	1	LOAD BYTE.
2	1	0	Rotate left by N nibbles (specified by S2-S4) and store Half, WORDS and Bytes.
3	1	1	LOAD HALF.

TABLE 3. Functional modes of the second rank of the bidirectional shifter.

DEFINE	RIGHT	=	/F1	+	/F0	
	LOAD-BYTE	=	/F1	+	F0	
	LEFT	=	F1	+	/F0	
	LOAD-HALF	=	F1	+	F0	
SEL0	=	/S4	+	/S3	+	S2
SEL1	=	/S4	+	/S3	+	S2
SEL2	=	/S4	+	S3	+	S2
SEL3	=	/S4	+	S3	+	S2
SEL4	=	S4	+	S3	+	S2
SEL5	=	S4	+	S3	+	S2
SEL6	=	S4	+	S3	+	S2
SEL7	=	S4	+	S3	+	S2

1F = SEL0	{	RIGHT * D7	+	
		LEFT * D7	+	
		LOAD-BYTE * FILL	+	
		LOAD-HALF * FILL	+	
SEL1	{	RIGHT * FILL	+	
		LEFT * D6	+	
SEL2	{	RIGHT * FILL	+	
		LEFT * D5	+	
		LOAD-BYTE * FILL	+	
SEL3	{	RIGHT * FILL	+	
		LEFT * D4	+	
SEL4	{	RIGHT * FILL	+	
		LEFT * D3	+	
		LOAD-BYTE * FILL	+	
		LOAD-HALF * FILL	+	
SEL5	{	RIGHT * FILL + LEFT * D2	+	
SEL6	{	RIGHT-FILL + LEFT * D1 + LOAD-BYTE * FILL	+	
SEL7	{	RIGHT-FILL + LEFT * D0	+	

TABLE 5. The Mode 1 LOAD BYTE operation requires that S2 be zero.

The eight outputs of the second-rank PALs need 14, 13, 13, 12, 13, 12, 10, and 9 product terms, in that order. The number of product terms available in the 22V10 varies between 8 and 16, depending on which output is used. Using eight outputs of the 22V10, which have 14, 14, 16, 12, 16, 12, 10, and 10 product terms for these outputs, only eight product terms (for these outputs) are left unused.

Implementing a 32-bit bidirectional shifter will take 16 first-generation PAL devices (AmPAL 16H8/16L8). Implementing these with SSI/MSI, such as Am25S10, will increase the device count to 24.

Gate Array Implementation of the 32-Bit Bidirectional Shifter

Implementing the straight 32-bit bidirectional shifter would take about 1,412 gates in CMOS, as shown in Table 6. However, the minimum number of I/O pads will be more than 96 [32 data inputs, 32 outputs, and 32 intermediate signals (B_{1F}=0), plus at least 7 control signals]. Also, for the gate

8-to-1 Mux	13 gates	$13 \times 64 = 832$
Input buffer	4	$4 \times 32 = 128$
Gates/FF	9	$9 \times 32 = 288$
Gate/output buffer	7	$7 \times 32 = 224$
		1412

TABLE 6. Gates required for implementation of a 32-bit bidirectional shifter in CMOS.

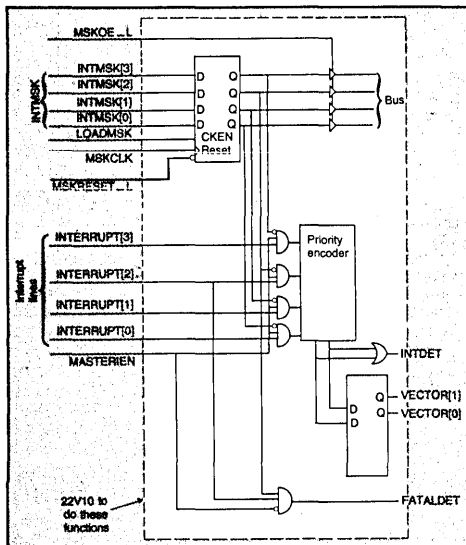


FIGURE 5. A 4-bit vectored priority interrupt controller implemented with the 22V10.

array implementation, 32 on-chip buffers and 32 2-to-1 multiplexers will be needed for the 32 intermediate bus signals (B_{1F} to B_0). This will require a total of approximately 1,600 to 1,700 gates for a gate array implementation. With the requirement for at least 96 I/O pads, the choice for the CMOS gate array (using LSI Logic's gate arrays) will be either LL5220 (2224 gates) or LL7420 (4242 gates).

Interrupt Mask and Control Register

This example illustrates the use of 22V10s to simplify the design of an interrupt system; to reduce its cost, size, and package count; and to increase its speed and reliability.

A good interrupt system should handle multiple interrupt requests, prioritize interrupt requests, nest the interrupts' service routines (for serving higher-order interrupts), and enable/disable interrupts on the fly (dynamically). It should also provide selective masking of individual interrupt requests, and flexibility in the method of clearing interrupt requests, as well as a fast interrupt-response system. Ideally, it should also support hardware modularity.

Handling the qualification, latching, and priority ordering of interrupts often seems to require a disproportionate number of chips. With the 22V10, however, a registered 4-bit interrupt mask with tristate outputs, a master interrupt-mask bit,

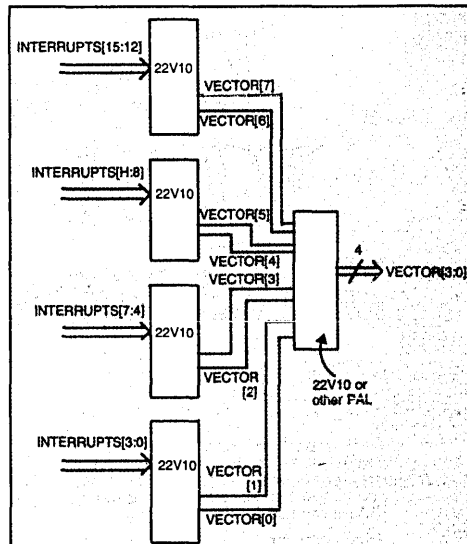


FIGURE 6. A 16-bit vectored priority interrupt controller using 22V10s.

four individual interrupt inputs, a registered priority encoder, and a non-latched interrupt detected output can all be included in one chip. The interrupt-mask register includes a master reset and a load-enable pin to ease interfacing with other tristate buses. The priority encoder can also detect any combination of input signals considered to be "illegal," and generate a fatal error signal. Figure 5 shows the use of the 22V10 in this application, which uses 13 inputs and 8 outputs of the 22V10.

A four-bit mask register is used to mask individual interrupts. Considerable flexibility is provided for controlling the mask register. Interrupt requests on lines INTO through INT3 are ANDed with the corresponding bits of the MASK register, and results are sent to a 4-input priority encoder; this produces a 2-bit encoded vector representing the highest priority interrupt, which is not masked. The LOADMSK signal loads the MASK register from the input lines. The entire mask register is cleared by the MSKRESET_L signal. The MSKOE_L signal enables the MASK register outputs onto the three-state output bus. VECTOR[1:0] provides the encoded interrupt-vector outputs. The interrupt-detect circuitry flags any unmasked interrupt input and generates the INTDET signal. FATALDET signal provides the illegal state. Table 7 shows the PLPL (programming language for programmable logic) description of this 4-bit vectored priority-interrupt controller.

Structuring of the interrupt logic with a PAL device is beneficial in two ways. This approach provides hardware modularity, which easily supports expansion. Additional modules may be added as the need to service additional requests arises. This hardware modularity also provides a structural regularity to the hardware design, which simplifies

3

Hanging a user-customized macrocell at each output of an LSI-level fuse-programmable array logic gives it the flexibility of a gate array. The result: Less hardware is needed for high-level functions.

Reprint with permission from ELECTRONIC DESIGN

Programmable logic chip rivals gate arrays in flexibility

Designers of semicustom ICs have long appreciated the in-house programming capabilities of programmable array logic devices. But gate arrays, with their LSI densities and wider opportunities for customization, are an inviting alternative, even if they must be sent out for metal-mask processing. Which approach should the designer use?

An agonizing choice between convenience and versatility is no longer necessary. The first fuse-programmable chip capable of implementing LSI circuits blends the advantages of both types. The result is more cost-effective designs with higher-level functions than previous programmable logic devices offered and a shorter turn-around time than is possible with gate arrays.

The new chip, the AmPAL22V10, can be thought of as a fuse-programmable gate array, since at twice the density of previous programmable array logic devices, it replaces the equivalent of logic circuits having 500 to 1000 gates (see "Blending Programmable Logic and Gate Arrays," p. 97). In

fact, because of its flexible, programmable architecture, the chip spans the breadth of standard SSI, MSI, and present-generation PAL devices, often accomplishing what could not be practically done with several such chips.

Designers program the 22V10 by opening fusible links in any or all of its 10 output macrocells, as well as in its AND-gate array, developing the needed logic functions. Though the AND-gate structure makes the unit similar to other PAL devices, the chip's 10 output logic macrocells afford a substantial new degree of design freedom (Fig. 1).

What's more, at the chip's output pins are 10 three-state output buffers, each fed by a macrocell and controlled through a separate output-enable AND gate.

The macrocells provide the 22V10's key features: They can be configured to

make any or all of the I/O pins act either sequentially or combinatorially and have either active highs or active lows. Additionally the Output Enable can individually control the I/O terminals to act as output pins, input pins, or bidirectional ports. Thus the designer can readily change the chip's architecture.

Within the array, each AND gate is fed by the true and complementary levels of the chip's 12 input lines, one of which doubles as a clock line. Also, each AND gate is fed by the true and complementary levels of the 10 feedback outputs from the output



Brad Kitson, Section Manager
David Laws, Managing Director
Warren Miller,* Section Manager
Advanced Micro Devices Inc.
901 Thompson Place, PO Box 3453
Sunnyvale, Calif. 94088

*Warren Miller is now at Step Engineering Co.

Semicustom ICs: Programmable logic

logic macrocells.

Each of the AND gates therefore has 44 inputs, and there are a total of 132 AND gates in the array. The result is over 5800 fuses that can be programmed to perform the desired function.

A total of 120 of the input AND gates are distributed to drive 10 OR gates, forming the logical part of the AND array. In addition, one AND gate is assigned to control each of the 10 three-state outputs, and the remaining two gates activate synchronous register presets and asynchronous register resets.

The array itself consists of 10 AND-gate macrocells (five similar pairs), each logically ORing the outputs of anywhere from 8 to 16 AND gates. The AND gates are distributed so that the two outermost macrocells in the array logically OR together eight AND gates. The next two outermost macrocells logically OR together 10 AND gates; the

next, 12 AND gates; and so on, traveling inward along the array. In other words, moving toward the array's center, two AND gates are added to successive pairs of macrocells, ending with the two innermost macrocells, each ORing 16 AND gates apiece.

The variable distribution of AND gates is particularly useful when implementing counting, exclusive-OR functions, and complex state machines. For example, a 10-bit binary counter has only one AND gate on the LSB but requires an additional gate for each succeeding bit, for a total of 10 AND gates to implement the MSB. Should a hold, parallel load, or specialized function also be desired, the MSB requires even more AND gates.

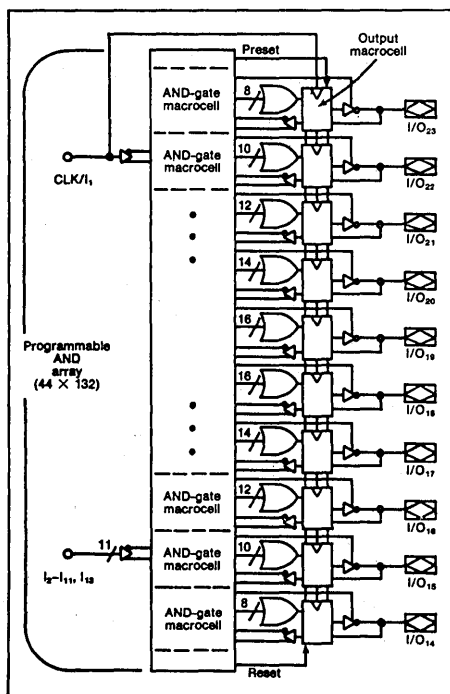
To program the array into its familiar sum-of-products Boolean function, a designer blows the corresponding fuse links. Those links initially connected the chip's pins and their complements to all 132 of the 44-input AND gates. As mentioned, the designer can also program any OR-gate output to be a dedicated input or a dynamically controlled I/O pin. This is possible because the connection from an output macrocell to its output pin is through a three-state output buffer and is controlled by an output-enable AND gate. Turning off the connection turns the output pin into an input. Otherwise the pin remains connected to the output macrocell, fed by the OR gate.

An output with a difference

The chip's output section is where it appears more like a gate array than a PAL device. The output macrocells allow the designer, by individually setting the outputs to be either sequentially or combinatorially and either active high or active low, to program either the true or complementary version of a logical term—which ever makes the most efficient use of the chip's AND array and fits the chip to the application. Furthermore, independent control over the active state helps the designer conform to common logic conventions, thus avoiding confusion and error.

For example, certain functions, such as chip selects and resets, tend to be active low. The output macrocells avoid contradictions with convention by inverting outputs, individually changing them when necessary to conform. Finally, by individually selecting either sequential or combinatorial outputs, the macrocells impart to the chip the capacity for implementing more functions than is possible with existing PALs.

The macrocell (Fig. 2a) contains three main logic blocks: a rising-edge-triggered D-type flip-flop with asynchronous reset and synchronous preset inputs, a four-to-one output-path selection multiplexer, and a two-to-one feedback-path selection multi-



1. A fuse-programmable LSI device combines the efficiency of programmable logic arrays with the flexibility of gate arrays. Ten user-definable output macrocells allow the designer to set the logic outputs individually for one of four configurations. They can even be used as inputs or dynamically controlled I/O.

plexer. Two fuses are imbedded in each macrocell to control the four possible output configurations: R_n selects sequential or combinatorial operation; P_n selects active low or active high.

With both fuses intact, the output is sequential and active low (Fig. 2b). The Q output of the flip-flop passes to an inverting output buffer, and the Q output feeds back to the AND array internally. This configuration is particularly suitable for state-machine designs. Alternatively, by programming (blowing) P_n , the designer can make the output active high.

On the other hand, by programming only R_n , the designer makes the output combinatorial and active low. For this scheme, the OR-gate output bypasses the flip-flop and feeds the inverting output buffer directly. At the same time, the feedback multiplexer passes the output back into the array (Fig. 2c).

By programming both fuses, the designer makes the output combinatorial and active high, and, again, the feedback multiplexer passes the output back to the array. This configuration serves best for "glue" logic and for generating complex control functions.

Finally, when the device is programmed in the combinatorial configuration, the programmable Output Enable can be used to transform an output pin into a dedicated input or a dynamically controlled I/O terminal. Once enabled as a dedicated

input, however, a pin's output function is sacrificed. This capability is extremely valuable when partitioning functions into the device. The limitation is no longer the numbers of input devices, but the total number of device pins. Lastly, as a dynamically controlled I/O, the pin serves both functions, meeting most bus requirements.

Intelligent use

One of the most challenging jobs in designing microprocessor systems is installing intelligent peripheral controllers—that is, controllers that can transfer data themselves and so unburden the processor from inefficient polling operations. One solution is to build a custom intelligent controller from scratch using the 22V10 (Fig. 3). This controller governs data flowing between a microprocessor and up to eight peripheral devices, all conforming to the Small Computer Systems Interface standard (SCSI, the ANSI X3T9.2/82-2).

The interface consists of an 8-bit bidirectional data port (DB_7 – DB_0); three control outputs—Acknowledge (ACK), Select (SEL), and Reset (RST); and five control inputs—Request (REQ), Control/Data ($\bar{C}/D$), Message (MSG), Busy (BSY), and Input/Output ($\bar{I}/O$). There are additional signals, unrelated to the SCSI standard. They include several microprocessor controls, such as address lines A_1 and A_2 for setting the data's desti-

Blending programmable logic and gate arrays

Centered on a fuse-programmable AND array, the architecture of a PAL (programmable array logic) device offers designers a powerful, instantly modifiable logic building block. As a result, PAL devices are used in a wide variety of applications, ranging from microprocessor-based systems to supermini-computers.

Since gate arrays are not programmed in house, they have a much slower turnaround time and higher cost. Yet gate arrays' simple logic structure makes them easier to customize than PAL devices, and their larger size makes for denser logic structures.

A gate array's simpler structure can be a drawback, however. For example, typical gate arrays are based on just two- or three-input gates; therefore, to implement complex functions, they often require many gate levels, which results in slow logic structures. In contrast, PAL devices, like memories, make more efficient use of their silicon area and can be configured into fast logic structures. Consequently, even with a slight overhead for programming circuitry, PAL densities are beginning to rival those of gate arrays.

However, innovative circuit techniques and an oxide-isolated bipolar process team up in the AmPAL22V10 and combine the advantages of PAL

devices and gate arrays. For example, the programmable output macrocells are designed to add only one Schottky diode in the delay path. As a result, the chips selected for high speed have a worst-case input-to-output propagation delay and setup time of only 25 ns, as well as a worst-case clock-to-output delay of a speedy 15 ns, over the commercial temperature range. Also, the oxide-isolated process fits the chip's ability to implement complex functions into a 24-pin, 0.3-in.-wide package.

For convenience, a feature called Preload loads the chip's internal registers with an arbitrary value, so that the designer can perform a complete post-programming functional test. In fact, Preload is a necessity for testing state machines, where the successive states depend on past and present values, as well as on the inputs. Without Preload, complex and sometimes impossible sequences would have to be entered before the first test could be performed.

What's more, platinum silicide fuse technology makes programming faster and increases reliability, and a well-controlled melting rate yields large, stable, nonconductive gaps. Address circuitry, much like that of a PROM, blows one fuse at a time, and programming is done with an algorithm that is easily adapted to a wide variety of available machines.

Semicustom ICs: Programmable logic

nation, read and write controls ($\overline{\text{IORD}}$ and $\overline{\text{IOWR}}$), Interrupt Request ($\overline{\text{INTREQ}}$), Chip Select ($\overline{\text{CSSCSI}}$), System Clock (SYSCLK), and buffer and latch enables ($\overline{\text{BE}}$ and $\overline{\text{LE}}$) for passing bus data between the processor and its peripherals.

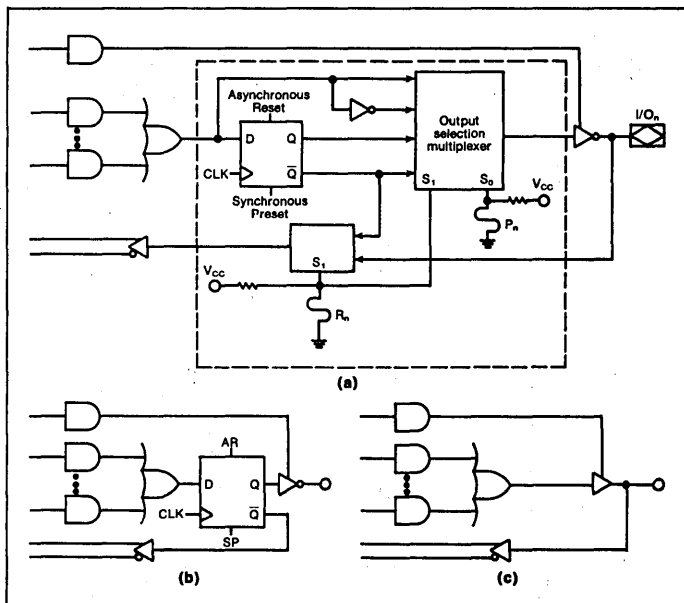
In operation, the controller assigns one of eight peripheral devices to be a master by activating $\overline{\text{SEL}}$ and at the same time sends the master's address to the SCSI data port. From then on, when a device notifies the controller that it needs to transfer either a command or data, the controller interrupts the host processor. In turn, the processor reads the controller's status register to determine which device is requesting service and the type of request it is making. After its status register is read, the controller removes the interrupt request and carries out the required transfer.

Meanwhile the selected device issues a $\overline{\text{BSY}}$ signal, letting the other devices know that the bus is in use. The data transfer is managed by the handshake signals $\overline{\text{ACK}}$ and $\overline{\text{REQ}}$, and the processor activates $\overline{\text{RST}}$ as needed by pulling both $\overline{\text{WR}}$ and $\overline{\text{RD}}$ low at the same time.

Matching the chip to its application starts with an assessment of the available resources against those needed to do the job. In general, that means establishing that the chip has enough input and output pins and product terms to accommodate the logic functions required by the application.

As to the pinouts, the 10 outputs required by the application do, in fact, match the 10 available on the chip. Four of the pins are used for the bidirectional data bus and the rest are dedicated outputs. As for inputs, the SCSI controller requires 11, which the chip more than satisfies with its total of 12.

Next the designer develops the product terms needed to execute the controller's job and compares the number and size of the required terms with the number available from the chip. The resulting controller equations (see Table 1) number 10, and these nestle neatly into the chip's 10 available AND-gate macrocells. Also, the maximum number of terms in any one equation—in this case, those making up the acknowledgment function—is only 6, compared with the chip's capacity of 8 to 16. Thus the controller function fits easily into a single programmable



2. Each of 10 output macrocells (a) are independently fuse-programmed for one of four configurations. Fuse R_n controls whether the outputs are sequential or combinatorial, and fuse P_n determines whether they are active high or active low. With no fuses blown, the output is sequential and active low (b). Blowing only R_n gives a combinatorial active-low output, and blowing only P_n gives a sequential active-high output. Finally, blowing both fuses changes the output to combinatorial and active high (c).

Semicustom ICs: Programmable logic

logic array, as can more complicated micro-processor peripheral interfaces.

The 22V10's real power, however, becomes apparent when it is needed to fill an application that appears to demand more than the chip's resources can deliver. Insufficient input or output pins or product terms are frequent shortcomings of many devices when the job is very formidable. With its highly versatile I/O architecture, however, the 22V10 can accommodate conditions that would make chips even larger than itself inadequate.

Table 1. Peripheral controller equations

Combinatorial equations

$$BE = CS \cdot RD \cdot A_2 \cdot \bar{A}_1$$

$$LE = \bar{CS} \cdot WR \cdot A_2$$

$$D_3 = \bar{A}_1 \cdot C + A_1 \cdot \bar{ACK}$$

$$D_2 = \bar{A}_1 \cdot M + A_1 \cdot SEL$$

$$D_1 = \bar{A}_1 \cdot B + A_1 \cdot RST$$

$$D_0 = A_1 \cdot I + A_1 \cdot R$$

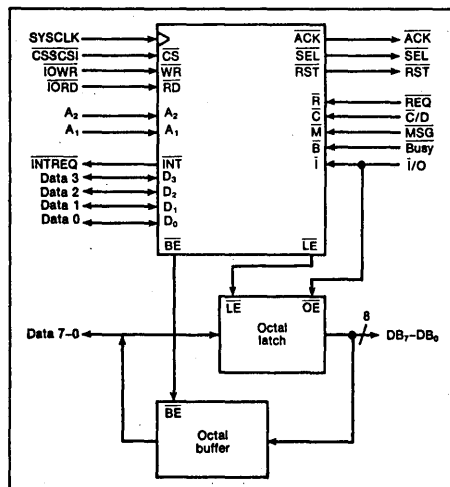
Sequential equations

$$INTREQ = \bar{R} \cdot (I \cdot C \cdot M + \bar{J} \cdot \bar{C} \cdot M + I \cdot C \cdot M + ICM)$$

$$ACK = \bar{CS} \cdot WR \cdot A_2 \cdot A_1 \cdot D_3 + R \cdot \bar{ACK} \cdot (\bar{CS} \cdot WR \cdot A_2 \cdot A_1) + R \cdot CS \cdot RD \cdot A_2 \cdot A_1$$

$$SEL = \bar{CS} \cdot WR \cdot A_2 \cdot A_1 \cdot D_2 + B \cdot SEL \cdot (\bar{CS} \cdot WR \cdot A_2 \cdot A_1)$$

$$RST = RD \cdot WR \cdot CS \cdot WR \cdot A_2 \cdot A_1 \cdot D_1 + RST \cdot (\bar{CS} \cdot WR \cdot A_2 \cdot A_1)$$



3. The problem of designing an intelligent peripheral controller is simplified by the 22V10's flexible I/O structure. With the freedom to assign different architectures to different pins, the designer can make better use of the chip's input and output pins and product terms.

For example, one of the most complicated functions to perform with a digital computer is floating-point arithmetic, or more specifically the normalizing of numbers in the process. As a result, no standard MSI or LSI logic for normalization exists. Yet the programmable array is well-suited to integrating at least part of this complicated logic function.

Normalization is necessary because a floating-point number is represented in two parts: a mantissa and an exponent. The mantissa is a fractional number that is adjusted, or normalized, so that a binary 1 always occupies the most significant bit. To normalize a floating-point number without changing its value, a compensating adjustment must be made to the exponent. In effect, the exponent, a power of two, is incremented for each bit position that the mantissa is shifted to the left.

Because the process of normalization can be very costly if done wholly in parallel, it is more efficient to process a number one byte at a time. Furthermore, because of the function's complexity, it is reasonable to partition the normalization unit into two units: an execution unit, which acts on the incoming data, and a control unit, which directs the execution unit. Thus the two units together form a byte-wide floating-point normalization (BFN) unit, with the programmable logic array making up the execution unit (Fig. 4).

The design of the execution unit breaks up logically into the four main instruction categories; Detect, Normalize, Merge, and Shift. Detect tests the incoming data for a binary one in any bit position. When such a nonzero byte appears, a flag, F, informs the control unit and tells it to begin the normalization process. Next, the Normalize command determines the number of bit positions to shift the incoming byte, in order to put its first bit in the most significant position. Then the Shift command adjusts the input data to the left by the specified number of bit positions, filling the lower bits with 0s. Finally, the Merge command adds in the full numbers' remaining bits, adjusting them so that the entire number is intact and shifted.

From the beginning, even before the equations of the execution unit have been fully developed, it appears that the resources of the programmable array will be insufficient for the job (see Table 2). Fourteen input pins are needed, against the chip's 12, and 12 output pins are needed, whereas the chip has only 10. Nevertheless, a closer look at the execution unit equations, with an eye toward consolidating I/O pins and functions, turns up paths around the seeming shortages.

One "problem" that proves not to be one is the number of outputs. Although they number 13, not

Semicustom ICs: Programmable logic

all are active at the same time. For example, the outputs V_0 - V_2 , which specify the number of shift positions as a three-bit binary number, are active only when no other outputs are. Thus they can share their output circuitry with other signals, relieving the output shortage.

However, combining the shift outputs with other terms creates a product-term deficiency. Yet, again, careful observation reveals a solution. One way to reduce the product terms of a function is to invert it, using the programmable array's output macrocell. The most likely candidate for such a reduction is the function with the most product terms. Although there are several, the best, because it is relatively removed functionally from the others, is the detection flag, F. Its function is given by:

$$F = D_7 + D_6 + D_5 + D_4 + D_3 + D_2 + D_1 + D_0$$

where D_7 through D_0 are input data bits, which, if any are high, indicate a nonzero byte.

When De Morgan's theorem is applied, the long OR string of active-high data reduces to a single AND term, saving a total of seven product terms. Still, two input pins must be found for the programmable array to fully incorporate the logic of the execution unit. Since De Morgan's theorem cannot minimize the logic any further, the only answer is to locate any product terms that underuse the chip's resources, combine them, and lend their output pins to the cause by making them I/O pins.

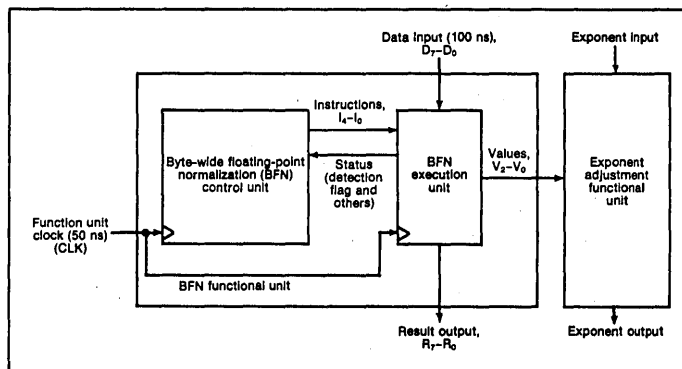
Examination of the instruction input specification (see Table 3) reveals that during Detect only three of the five instruction inputs are necessary, freeing the two others to be used as outputs. Therefore instruction bits I_0 and I_1 can share the outputs that were previously dedicated to shift bits V_0 and V_1 , thus meeting the execution unit's final requirements in a single chip. □

Table 2. Allocation of chip resources

	Input pins		Output pins		Product terms
	Number	Names	Number	Names	
Initial requirements	14	CLK, D_7 - D_0 , I_4 - I_0	12	F (detection flag), V_2 - V_0 , R_7 - R_0	114 required
Resources	12	None	10	None	120 available
Final requirements	12	CLK, D_7 - D_0 , I_4 - I_2 , I_1 / V_1 , I_0 / V_0	12	I_1 / V_1 , I_0 / V_0 , R_7 / V_2 , R_0 / F , R_6 - R_1	108 required

Table 3. Instruction definitions

	Instruction code (I_4 - I_0)	Detection flag
Detect	00XXX	0
Normalize	00XXX	1
Merge	01N ₂ N ₂ N ₀	X
Shift	10N ₂ N ₁ N ₀	X



4. Floating-point arithmetic is a hardware-intensive process that, for the most part, has defied integration. Yet by partitioning the tasks into smaller pieces, a designer can apply the power of the programmable array to the job. The chip's flexibility contributes to its successful implementation as a floating-point execution unit, even though its resources may seem insufficient.

DESIGN ENTRY

ELECTRONIC DESIGN EXCLUSIVE

PAL device buries registers, brings state machines to life

Om Agrawal and Kapil Shankar

Advanced Micro Devices Inc., 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088; (408) 732-2400.

Designers who build state machines from programmable logic devices (PLDs) must develop good juggling skills. The job usually takes a careful balancing of the I/O pins, the product terms, and the registers that store states and output bits. But because these resources all are limited in a PLD, performance and compactness often have to be sacrificed in favor of the desired function.

Ideally, state-machine designers need a PLD with buried, or internal, state registers, compact packaging and high speed. Buried registers free valuable I/O pins; a small package saves precious board space; and high operating speed speaks for itself.

Approaching the ideal is the AmpAL-23S8, the first 20-pin programmable array logic (PAL) device de-

Housed in a 20-pin DIP, the smallest PAL device to date for building state machines sports registers that save I/O pins, yet it is easy to test.

signed specifically for building state machines and sequencers. Its architecture is more flexible than any 24- or 28-pin PAL device or programmable logic array (PLA). For example, it has 14 edge-triggered, D-type flip-flops (Fig. 1). That alone compares favorably with 24-pin bipolar PAL devices, which have up to 10 registers, and with 24- and 28-pin PLAs, which have 12 to 14 registers. Only 40-pin PAL devices with 16 registers surpass the new chip.

As significant is the fact that of the chip's registers, six hold buried state bits. By definition, the buried registers do not merely free I/O pins: Indeed, they actually enhance the chip's value as a state machine. Driven by the chip's AND-OR array the buried state registers, along with the eight registers dedicated to I/O pins, make possible a wide range of states and outputs.

Moreover, the buried state bits can be easily accessed, controlled, and observed. Although the

first two attributes are common among PAL devices, the third has always presented problems. The 23S8 is the first chip whose product-term array makes the buried state registers observable.

Of the other eight registers, four serve in output logic macrocells and four store output states. Each macrocell contains three main blocks (Fig. 2). One block is the register, a rising-edge-triggered flip-flop sharing asynchronous reset and synchronous preset inputs with all the other flip-flops. The other two blocks are multiplexers, one of them selecting from one of four output paths, the other from one of two feedback paths.

The logic chip is small, fast, and easy to program. Packaged in a 0.3-in. DIP, it is half the width of 28- and 40-pin PAL and PLA type devices. Moreover, it comes in 33- and 28.5-MHz versions, with propagation delays of 20 and 25 ns, respectively. The faster version is an ideal companion for new 8-MHz, 16-bit microprocessors, as well as for 16-MHz, 32-bit units. Even at 28.5 MHz, the new chip is faster than existing bipolar PLA-based sequencers, which operate from 20 to 25 MHz. It also beats equivalent-density PAL devices operating at or below 25 MHz.

Programming takes place by blowing reliable and fast platinum silicide fuses, and using available software packages. Like all PAL devices, the 23S8's fusible AND-OR array is a direct measure of its capability. With 6200 fuses, it is three to six times

Distribution of logic product terms

Flip-flops		AND gates ORed
Location	Number	
Macrocells	2	8
	2	10
Output pins	2	8
	2	12
Buried	2	6
(Internal)	2	8
registers	2	10

DESIGN ENTRY ■ State-machine PLD

the size of standard 20-pin bipolar PAL devices. In fact, it is larger than all other bipolar PLDs except for the 40-pin chips, which are three to four times as large and have some 8000 fuses. The new chip's fuse array handles all 14 registers, with 6 to 12 logic products for each output, and up to 23 inputs—9 of them dedicated, 8 for feedback, and 4

6 for the buried states.

The number and distribution of the chip's product terms are two other measures of its capability. It has altogether 135 product terms for logic and control functions. Moreover, the AND array that drives the 14 flip-flops is variably distributed (see the table, p. 101).

The distribution of the 124 logic product terms helps adapt the chip to system requirements, while optimizing its internal resources. (Of the 11 other terms, 8 control output, the remaining 3 control preset, reset, and observability, respectively.) For additional flexibility in designing powerful state machines, the chip can trade off the product terms of its buried registers.

OBSERVABILITY CLOSE UP

The chip's observable product term is generated by the AND-OR array, along with an ability to preload the buried registers. Observability helps check a buried register's state, so that a designer can monitor any register during debugging.

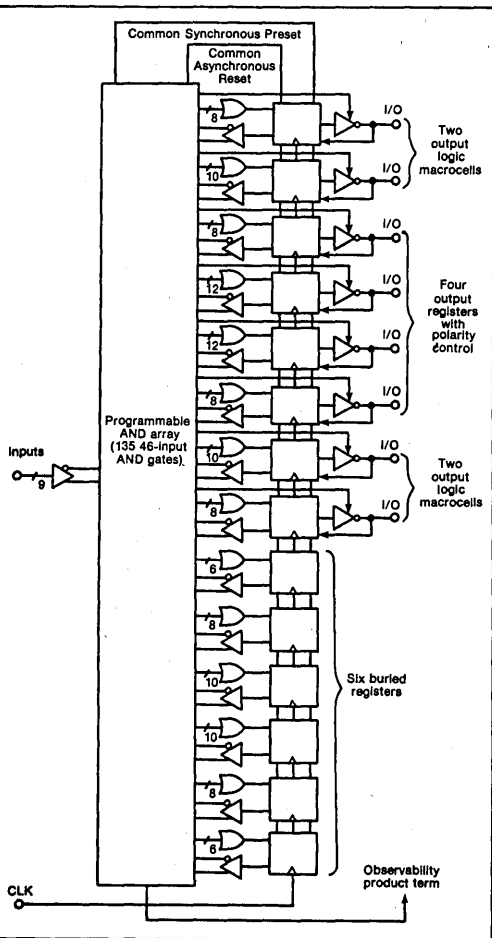
Under control either of a single pin or of a combination of six input signals, the observable product term drives a set of inverting buffers. In normal operation, the buffers also serve the four dedicated output registers and two of the four macrocells. When activated, therefore, the observable product term disables signal flow from those registers and macrocells and simultaneously connects the six buried registers to their respective I/O pins.

Flexibility in configuring a state machine derives from the independent control of output and feedback multiplexers. Each output macrocell has three fuses, two to determine the output path and one the feedback path. The fuses can configure the macrocells in eight ways: a combinatorial or a registered output with either an I/O pin or a registered feedback. Each of those four variations offers either active high or active low outputs.

In the output path, fuse S_1 determines whether the output is active high or active low. The output-nature fuse, S_2 , selects sequential or combinatorial operation. When both of those fuses are intact, a flip-flop's Q pin passes through an inverting buffer, making the output active low and registered.

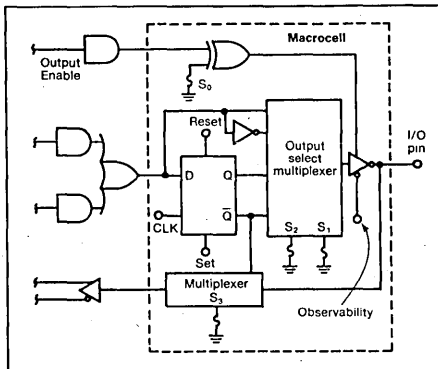
On the other hand, programming or blowing S_1 renders the output term active high; blowing S_2 causes the output to bypass the register, making it combinatorial. A combinatorial output with feedback can, moreover, provide more than one level of ORing logic. Finally, fuse S_3 controls the feedback path, sending either the flip-flop's Q output or the I/O pin back into the array.

Controlling each output's enable line through a product term from the AND array creates further options by transforming an I/O into a dedicated input, a dedicated output, or a dynamically controlled I/O pin. A designer can thus adjust the number of input and output pins, and have the dynamic I/O capability required by most buses.

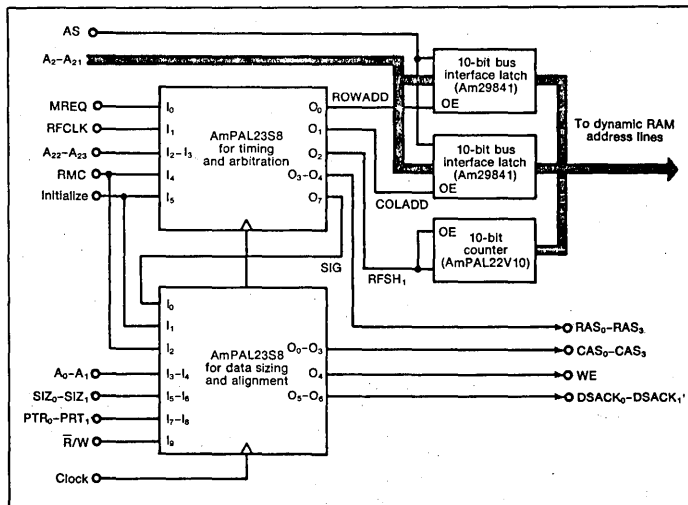


1. The AmPAL2358 is the first 20-pin PAL device designed specifically for building state machines and sequencers. It contains 14 registers, 6 of them buried and requiring no I/O pins, 4 dedicated to output macrocells, and 4 storing output bits. The states of the buried registers are easily observed, which simplifies debugging.

Each output enable product term is further associated with an individual polarity fuse. This polarity fuse allows designers to modify a control signal to enable outputs using De Morgan's theorem, which is especially useful for bus control applications.



2. Each programmable macrocell has three main blocks: a rising-edge-triggered D-type flip-flop, an output multiplexer, and a feedback multiplexer. The output multiplexer selects from one of four output sources; the feedback multiplexer feeds either an I/O pin or the flip-flop's Q output into the programmable AND-OR array.



3. Two 2358s lie at the heart of an advanced dynamic RAM controller that compares well in its data-transfer features—such as dynamic bus sizing and handling misaligned data—with recent 32-bit microprocessors. One of the two main PAL devices controls the timing and arbitration of the memory cycles; the other sizes and aligns data transfers.

The chip's high speed and buried state registers make it an ideal foundation for a flexible and fast dynamic RAM controller that meets the complex requirements of the latest microprocessors. Even at a 16-MHz processor clock rate it can become a controller with few or no wait states that might otherwise have to be built from scratch.

There is, for example, no ready-made controller that can take advantage of dynamic bus sizing—the automatic adjustment to the amount of data movable in one bus cycle. Nor can any off-the-shelf unit handle misaligned data, that is, when an operand falls outside its proper memory boundaries. Bus sizing and data alignment are features of the recently arrived Motorola 68020 and the Intel 80386, both 32-bit microprocessors. But a controller able to perform the two functions can be built around two 2358s, one for timing and arbitration logic and the other for data sizing and alignment (Fig. 3).

The timing and arbitration chip arbitrates among present, refresh, and CPU cycles; executes read or write, read-modify-write, and refresh cycles; and asserts interface signals. The chip first arbitrates between the processor's Memory Request (MREQ) and Refresh Request (RFCLK) signals. It gives priority to the cycle currently being executed, follows this with the refresh cycle, and concludes with the processor cycle. Both intermittent and burst-mode refresh schemes are therefore possible.

The design's timing relationships and arbitration requirements define a state-machine diagram (Fig. 4). The

actual state machine employs five of the six available buried registers. The sixth register is a flag and so acts as a small, independent state machine. Either of two software packages—Cupl from Personal CAD Systems Inc.'s Assisted Technology Division (San Jose, Calif.) or Abel from Data I/O Corp. (Redmond, Wash.)—easily describes the state operations. (Cupl is the language used in this particular example.)

No arbitration takes place during the execution of a cycle, which is indicated by a state other than 0. The timing and arbitration chip stores any refresh request and responds to it

later, continuing to follow the current cycle through its various states. Thus the current state automatically takes priority, and no other cycle is started until the current one is complete.

At the end of the cycle, at state 0, the timing and arbitration chip arbitrates between MREQ (a CPU cycle) and RFCLK. In the small independent state machine, the buried register flag, B_5 , latches RFCLK, which may last no longer than one clock cycle. When RFCLK is high, B_5 toggles to state 1 and stays there until the refresh cycle is executed. Then it toggles back to state 0 to be ready for the next RFCLK.

During refresh cycles, the timing and arbitration chip asserts the Refresh signal (RFSH), which supplies the refreshed row address to the memory address bus. The chip also generates the Row and Column Address Enable Lines, ROWADD and COLADD, which multiplex addresses onto the memory bus. At state 16, the chip removes ROWADD and asserts RFSH; state 17 asserts the Row Address Strobe signal (RAS) and refreshes all memory banks before removing RAS and RFSH. Re-

moving RFSH increments an external 10-bit refresh-row-address counter (AmPAL22V10) to the next row.

If no refresh request is pending during state 0 and MREQ is asserted, a processor cycle begins and the machine jumps to state 1. Otherwise the machine stays in state 0 and polls for either a refresh or processor cycle.

The PAL chip responds to the processor's signal RMC to execute a read, a write, or a read-modify-write cycle, with the internal registers controlling timing for the different cycles. If, for instance, the memory consists of 4 Mwords, organized as four equal banks of 32-bit words, the chip asserts a RAS signal for each bank—RAS₀ through RAS₃.

The strobe signal being asserted depends on address-select signals ADSEL₀ and ADSEL₁, which in most systems are the high bits of the processor's address bus. After asserting the proper strobe, the machine jumps to state 2 to manipulate the address-handling signals. For instance, to multiplex addresses, ROWADD and COLADD are again asserted.

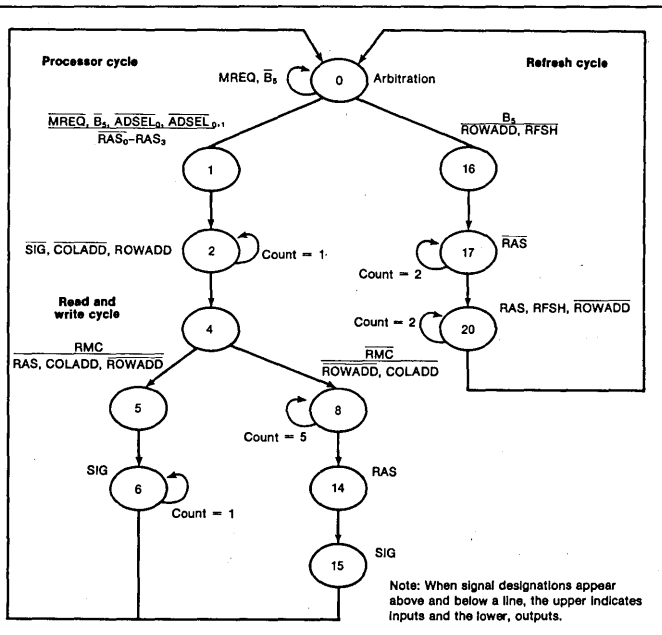
In state 2, the timing and arbitration chip removes

ROWADD and asserts COLADD, along with SIG, the reference that tells the data-sizing and alignment chip that a processor cycle is in progress. SIG synchronizes the two 23S8s and instructs the data-sizing and alignment PAL to send a column-address strobe (CAS) to memory.

The state machine remains in state 2 for an extra-cycle, and then moves to state 4 to allow time to access the data. There it decides, based on processor signal RMC, whether to execute a read, a write, or a read-modify-write cycle. Before completing a read or write access, the timing and arbitration chip removes RAS and waits for one more cycle to allow for the RAS precharge time. Simultaneously, it switches back to ROWADD from COLADD in preparation for the next processor cycle, and disables SIG.

When the RMC signal requests a read-modify-write cycle, the timing and arbitration chip automatically lengthens the cycle. It holds RAS until DSACK and WE signals have been asserted, and only then completes the cycle.

The data-sizing and alignment chip works only during a CPU cy-



4. A state diagram for the timing and arbitration chip reflects its operation for selecting and orchestrating the memory cycles. The chip waits in state 0 to handle either a processor cycle or a refresh cycle; the latter takes priority over any but a cycle currently in progress. The chip performs refresh, read, write, and read-modify-write cycles.

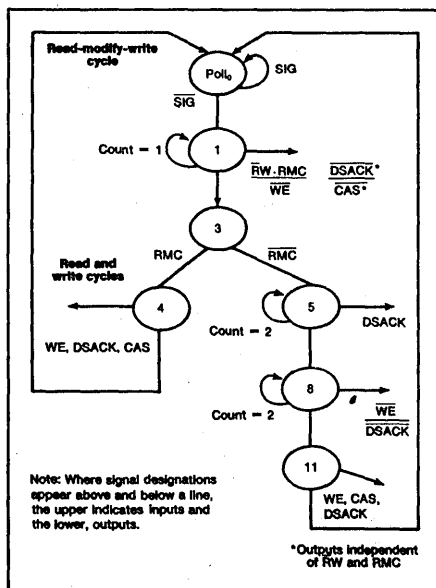
DESIGN ENTRY ■ State-machine PLD

cle. It creates WE for the write and read-modify-write cycles, and it accepts SIG, which synchronizes the chip's different states (Fig. 5) with those of the timing and arbitration chip.

Four of the data-sizing and alignment chip's buried state registers generate its state-machine function. The machine starts polling in state 0 and stays there until it receives SIG, the start of a processor cycle.

The 68020 has a 32-bit maximum data width and a data bus divided into four byte-wide segments, each independently controlled by a CAS signal from the data-sizing and alignment chip. The processor can therefore read and write byte-wide data over any segment.

The four CAS signals, CAS₀ to CAS₃, strobe the column address from the high- to low-order memory bytes. The signals also allow the microprocessor to read and write across the low-low, low-middle, high-middle, and high-high bytes of a 32-bit data port. The data-sizing and alignment chip must assert all four CAS signals for a 32-bit port; only CAS₂ and CAS₃ are needed to supply a 16-bit port's low- and high-byte strobes; while for an 8-bit port, CAS₃ is enough.



5. The data-sizing and alignment chip works only during a processor cycle and waits in state 0 until one is requested. If the processor calls for a read or a write cycle, the chip asserts DSACK, CAS, and, for a write cycle, WE. For a read-modify-write cycle, it asserts two DSACK signals, one each for the read and write segments.

The data-sizing and alignment chip asserts the CAS signals in state 1, and selects them according to the size of the port in use, the width of the data being moved, and how the data aligns with the port. Coding on input pins PRT₀ and PRT₁ shows the port size; signals SIZ₀ and SIZ₁ from the microprocessor show data width; and the microprocessor's least-significant address bits, A₀ and A₁, give the alignment.

THE PROCESSOR'S PREFERENCE

The size of the port dictates the amount of data that can be transferred in any one cycle. Based on the PRT signals the state machine asserts DSACK signals, which acknowledge the CPU and tell it how much data it can move, since it will always try to move as much as possible. The state machine synchronizes the acknowledgment timing, asserts the required CAS signals, and holds them until the read, write, or read-modify-write cycle is complete.

A read cycle's DSACK signal is asserted in state 1 and remains until the end of the cycle at state 4. A write cycle is identical, except for the additional WE signal. But a read-modify-write cycle requires two DSACK signals, one for reading, the other for writing. The first DSACK is the same as that for a read or a write cycle; the second DSACK occurs in state 8 and remains until state 11, with a delayed WE asserted in state 8 by an internal 4-bit counter. □

Om Agrawal is product planning manager for programmable logic devices at Advanced Micro Devices. He has designed 16-bit and 32-bit minicomputers and is co-author of a book on high-speed memory systems. He holds a PhD in electrical engineering and computer science from Iowa State University and an MBA from the University of Santa Clara.

Kapil Shankar is a senior product planning engineer for programmable logic and memory devices, and has designed advanced graphics systems. He has an MS in computer and systems engineering and an ME in electrical power engineering, both from Rensselaer Polytechnic Institute.

MIXING DATA PATHS EXPANDS OPTIONS IN SYSTEM DESIGN

Chip designers are creating powerful CPUs and peripherals with 16- and 32-bit parts. Mixing these with 8-bit parts overcomes limitations imposed by established designs, incomplete families, and software incompatibility.

by **Mark S. Young and
James R. Williamson**

Integrating 16- and 32-bit peripherals and CPUs into 8-bit designs, at the simplest level, means separating the control and data paths from new peripherals and the systems. Mixing different data path widths and control protocols, however, makes possible major improvements in function, performance, and cost.

The price/performance curve of VLSI chips, for example, allows designers to obtain more and better functions for the same amount of money every year. Alternately, the functionality of a device can remain constant while the price falls.

Moreover, these new devices with wider data paths can extend the life of older designs. For example, many of the most popular personal computers today use the 8088 microprocessor and, therefore, are constrained to an 8-bit data path. Designers of add-on accessories for these personal computers prefer the

newer 16-bit peripherals. These peripherals will let users preserve their software investments, improve performance, and stave off obsolescence.

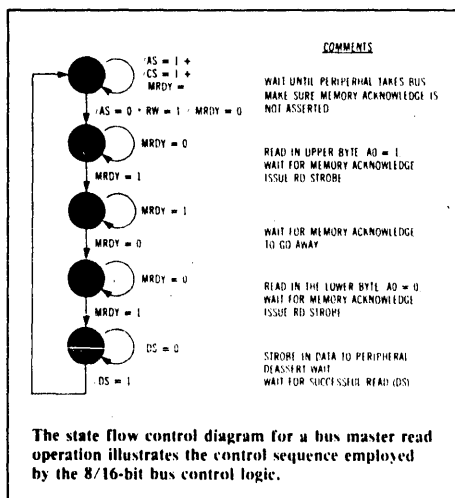
Mixing different data path widths can also enhance new designs. For example, it is less expensive to use an 8-bit bus in a new design because the memory requirements are generally cheaper. Only half as many dynamic RAMs are necessary for the same number of kilobytes of memory. In addition, an 8-bit bus needs much less control and support logic. Designers can mix smaller data path peripherals with wider data path CPUs. This allows them to introduce systems based on the newer, more powerful 32-bit CPUs even before 32-bit peripherals are available.

Designers can use this mixing method to obtain wider data paths from existing designs until a new system design is warranted. They can also use parts in unexpected applications. For example, cost-conscious terminal manufacturers might want to use the Am8052/8152A chip set (the 8052 is an advanced CRT controller and the 8152A is a video system controller) in new terminals based on the relatively inexpensive 8051 microprocessor. Mixing the 8-bit, single-chip microprocessor with the 16-bit CRT controller allows designers to maximize the cost/performance ratio of the terminal.

Mixed data path widths can improve bus utilization as well. A 16-bit peripheral in a 32-bit system only occupies half the data bus for data transfers. If the designer mixes the data paths correctly, however, the 16-bit peripheral could transfer data as

Mark S. Young is a product planning engineer at Advanced Micro Devices, Inc (Sunnyvale, Calif). He holds a BA in computer science from the University of California at Berkeley.

James R. Williamson is an applications engineer at AMD. He holds a BS in electrical engineering from the California State Polytechnic University, Pomona.



32-bit chunks and improve bus efficiency by 100 percent for that peripheral.

Two central concerns stem from mixing devices that communicate over different-sized buses. The first problem results when two devices communicate on a "common" data bus. Consider, for example, a 32-bit system utilizing 8- and 16-bit peripherals. Overcoming the mismatched data paths requires some form of controlled multiplexing/demultiplexing of the different data paths. In addition, extra control signals for partitioning the 32-bit word into 8-, 16-, and 32-bit chunks may be required.

Many 16-bit CPU-based systems that use 8-bit peripherals normally use just the lower 8 bits of the data bus to transfer data to and from the peripheral. This method does not work in systems using 16-bit peripherals and 8-bit CPUs, however, and it tends to break down in systems with 8-bit peripherals having bus master capability.

A bus multiplexing method involves multiple transfers when taking data from or adding data to a mismatched data bus. For example, before a 16-bit peripheral can transfer data over an 8-bit bus, the 16-bit data must be divided into two 8-bit chunks. It is then transferred sequentially. First, the lower 8 bits are transferred out on the bus. Then, in the next transfer cycle, the upper 8 bits of the 16-bit word are sent out. The major difference in the opposite case—a bus read operation from an 8-bit bus to a 16-bit device—is that the first byte read from the system must be latched. Once the second byte has been fetched, the 16-bit peripheral reads in the assembled 16-bit (2-byte) word. Additional provisions may be needed when the 16-bit peripheral only wants to access a single byte.

The other major problem in mixed data path transfers is the actual data read/write operation. The nature of the multiple transfer forces designers to guarantee that the stretched transfer will occur and that it will not be interrupted. Two aspects of stretching the transfer cycle from or to the peripheral illustrate the complexity of this problem.

The first case, when the peripheral is the bus master, is the simplest. A 16-bit peripheral holds its data available for what normally would be two complete bus transfer cycles. This function can be performed when the transfer acknowledge signal to the peripheral is delayed. If the data was latched instead of holding the peripheral in a multiple word transfer, however, the device could try to send the next 16-bit data word and its "new" address. The procedure of latching the data and releasing the peripheral should not be used, therefore, because it may interfere with the addressing of the remaining (pending) 8-bit transfer.

Whenever a device acts as a bus slave to a CPU that cannot access the device's natural word width in a single operation, a different constraint appears. The sequence must be set up so the peripheral cannot obtain the bus while the CPU is in the middle of a slave read/write operation. In a typical system, the CPU is the last device in the interrupt queue. It is possible for the peripheral to become bus master between the first and second read operations and invalidate the results of the first read operation in a realtime system. This is because an 8-bit CPU would have to perform two consecutive read operations to examine a 16-bit peripheral control register.

This function can be handled two different ways. If the CPU has a bus lock instruction, as in the iAPX family of CPUs, the programmer must use one of these instructions before the CPU accesses the peripheral. Alternately, the CPU needs to disable the arbitration logic while it is performing the uninterruptible access with the 16-bit peripheral.

Crucial cycle

The uninterruptible word transfer cycle is crucial for maintaining the integrity of the data transferred. When either the CPU or a peripheral on the bus makes an access using the 8/16-bit control logic, it must complete the larger device's word access before relinquishing the bus. If this requirement is not met, a transfer's integrity can be violated easily by some other device. This interrupts the transfer, and corrupts or aborts the multiplexing sequence.

To illustrate this point, consider a system consisting of an 8-bit CPU and several 8- and 16-bit peripherals. Assume one of the peripherals is executing a block transfer of 16-bit data onto the 8-bit bus. If the CPU interrupted the transfer in order to poll the peripheral during a half-word transfer, two undesirable events would occur. Either the multiplexing

sequence would be damaged irreparably when the CPU polled the peripheral, or the CPU would read garbage from the peripheral.

Designing the control interface to allow mixing of 8- and 16-bit peripherals requires attention to the data and control flow. During a write operation, the data is written out sequentially: the lower byte comes before the upper byte (or vice versa). The read operation differs only because the data bus is 8 bits and because it forgets the last byte transferred; it knows the current byte only. Hence, the interface requires that one of the bytes be latched until the full 16-bit word has been assembled.

The slave mode of operation works almost the same as the peripheral bus master mode. The single exception is the slave write operation. When the interface is defined, the designer must make a conscious choice about which byte (upper or lower) to latch during peripheral read operations (or conversely, slave peripheral write operations). Once this decision has been made, the CPU must always access the latched data byte first (during a slave write) and then access the non-latched byte to complete the transfer. This restriction is minor, requiring no extra software overhead. It could affect the ease of the programmer's coding if not handled properly, however. For example, if the programmer used a compiler to generate the software for the system, extra care may be necessary to ensure the compiler generates the correct addressing sequence.

An alternative solution would be to latch both the upper and lower data bytes. In this case, however, the cost of the interface would increase, as would the complexity, with no appreciable gain. The control flow in these designs derives from two differ-

ent sources: the state control flow itself and the 16-bit peripheral interfacing with the 8-bit bus. A state diagram can be used to specify how uninterrupted word transfers will occur and how the upper and lower byte address is generated.

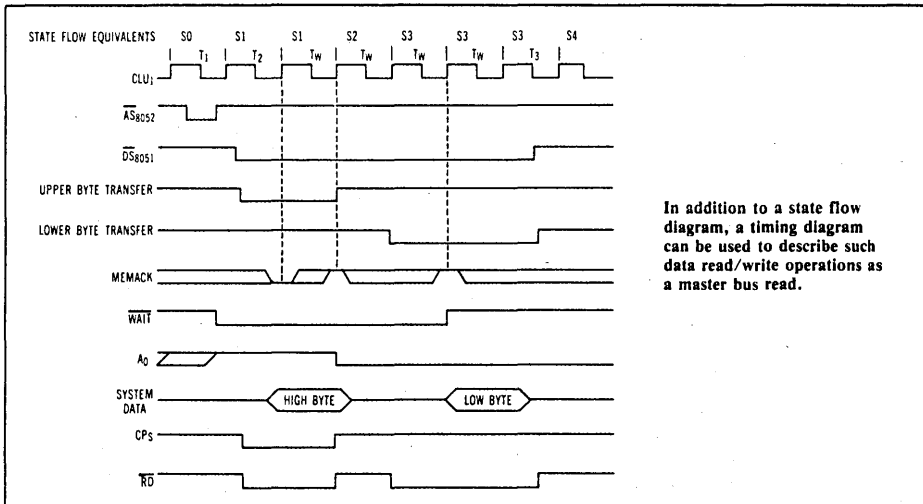
In addition, the specific bus timing of the peripheral and the data bus must be examined to quantify the state control flow. These timing specifics also provide information on data latching, read/write control strobes, and addressing to and from the peripheral. The state control flow is divided into four operations: bus master read, bus master write, slave read, and slave write.

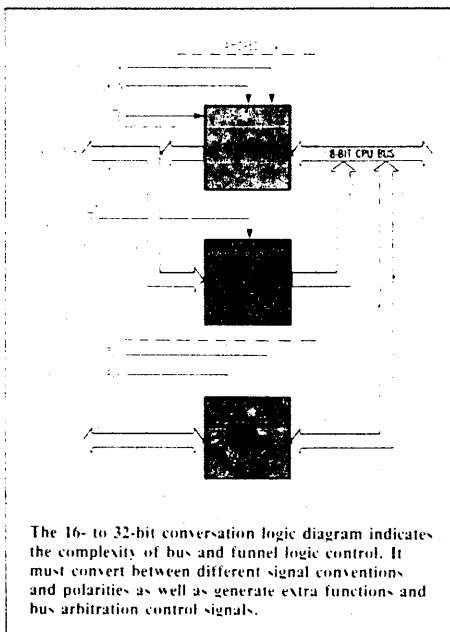
For a bus master read/write operation from a 16-bit peripheral device operating on an 8-bit bus, four control signals must be generated by the 8/16-bit control unit: address bit 0 (A0), peripheral hold (WAIT), bus read (RD), and bus write (WR). The A0 line is generated by the 8/16-bit control logic to indicate which byte is to be transferred in bus master modes only. Otherwise, the A0 generated by the system is used to indicate which byte is being accessed. The WAIT line holds up the peripheral during transfers. The RD and WR lines are required to indicate successive transfer cycles on the bus.

Hidden transfers

The peripheral's signals will only strobe active once because it does not know that two transfers are being executed. The slave transfer flows are almost identical, except the CPU is generating the bus signals and the transfer directions are reversed (ie, a bus write goes into the peripheral).

For this 16- to 8-bit data flow example, the data on the upper byte only needs to be latched when data





is being read (as bus master) or written (as a bus slave). An interface to handle this operation needs to latch data coming from the 8-bit data bus into the peripheral, it also needs to act as transceiver when the peripheral is sending data out to the system. A device with a clocked, tri-state output that has an 8-bit wide latch in one direction and a tri-state transceiver in the other direction would be ideal for accomplishing such an interface.

The Am2952 8-bit bidirectional I/O port provides a good enough match to the logic and allows the upper data bus latch and upper data transceiver chips to be combined on one IC. It provides two 8-bit clocked I/O ports, each with tri-state output controls and individual clocks and latch enables. An Am2949 bidirectional bus transceiver completes the logic required for the data path function.

The state flow control requires logic that can move sequentially from state to state, hold in a particular state, and be reset or initialized back to a predefined state. Depending on the number of states required (generally less than 16 distinct states for a design of this complexity), a 3- or 4-bit counter should be able to solve the problem nicely.

Considerable bus control logic is required to generate the data path flow logic and the bus control signals. This is especially true if the peripherals and CPUs use different signal conventions (eg, when $\overline{AS}$, $\overline{DS}$, and $\overline{R/W}$ use address latch enable, $\overline{RD}$, and $\overline{WR}$). Conversion from one signal convention to

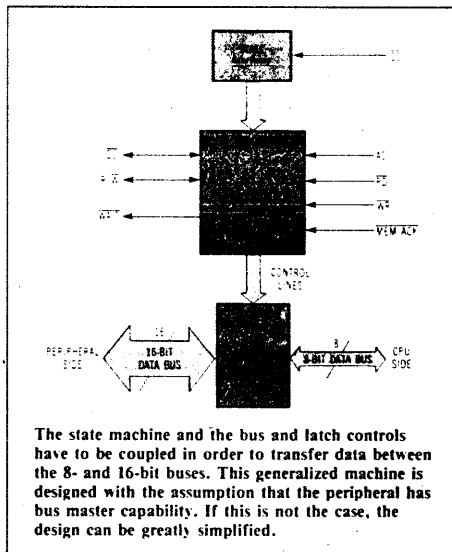
another, changes in signal polarity, and provision for extra functions (such as generating $A0$) require a lot of logic synthesis ability. If the peripheral has bus master capability, such additional information as bus arbitration controls must be fed into the next state determination logic in order to decide what control sequence to follow.

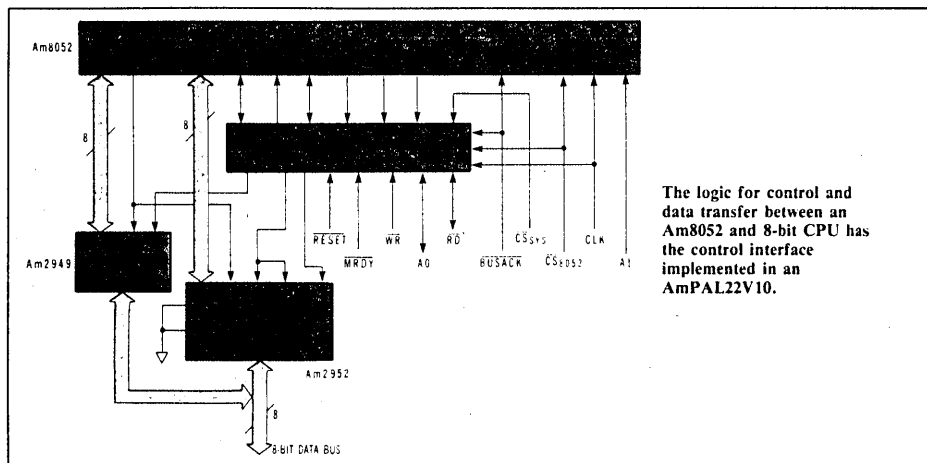
Customized interface minimizes cost

An 8/16-bit control interface between the Am8052 CRT controller and an 8-bit CPU provides a good example of how customizing a general interface can reduce costs. (The CRT controller is designed with a 16-bit data interface.) The onboard DMA unit fetches data from system memory and the CPU polls the CRT controller's internal status and control registers. Because the CRT controller does not modify system memory, however, a bus master write operation is unnecessary. Thus, there is no reason to generate a system write control signal ($\overline{WR}$).

In addition, the control and display information must be aligned on word boundaries. This requirement relieves the 8/16-bit control logic from funneling the bytes and performing odd/even byte transfers. It also saves control inputs from the CRT controller because all transfers are words; that is, no need exists for upper and lower data strobes or byte high enable inputs.

The bus master read operations are standard 16-bit data transfers divided into two 8-bit transfers. The CPU's slave accesses are either pointer writes (to select the desired control/status register) or 16-bit data read/write operations. (Pointer write operations





are actually 8-bit operations because only the lower 8 bits of the data form the register address.) The bus master read operation can be represented by a state flow diagram or a timing diagram. Conceptually, state flow diagrams are easier to understand, but timing diagrams usually convey more information. Other state flow diagrams can be derived directly from the timing diagrams of the CRT controller to 8-bit interface.

Simplifications allow synthesis on one device

Two special conditions must be met in the state machine implemented in the 8/16 interface. First, before a new transfer cycle is attempted (when the state machine is waiting in the initial state, S0), memory acknowledge (MRDY) must be inactive. This prevents interference from the last transfer.

The second special condition occurs when the CRT controller asserts the R/W line to indicate a write operation. Although the CRT controller does not write data into system memory, when it updates the upper 8 bits of the 24-bit address latch the R/W line indicates a write operation (in conjunction with AS). The CRT controller is not actually performing a system data write, only an address latch update. The state machine, therefore, must not start a bus sequence if the R/W line is held active low by the CRT controller during a bus master operation.

These simplifications in design allow the CRT controller to 8-bit CPU control interface to be synthesized in a single AmPAL22V10 programmable logic array device. In addition, the bus control signals are converted from AS, DS, and R/W to RD and WR. The minimum CRT controller and bus control signals that must be generated are RD, A0, DS, and R/W. Although the CRT controller uses DS and R/W as inputs during a bus master operation, the

PAL device must convert the CPU RD and WR signals to DS and t/W for slave I/O operations.

The signals A0 and RD are generated by the control logic when the CRT controller is performing a read access to system. The WAIT (or not READY) signal to the CRT controller must also be generated by the control logic. The data flow controls require six additional controls to load and strobe the latch, and to enable transceivers to pass data to and from the 8-bit bus. Theoretically, 4 more bits (outputs) are required to represent all the control states needed to manipulate the 8/16-bit control logic. This means the design appears to need 14 output logic units in a PAL device to perform the required task.

Reducing the 14 output cells to the 10 cells available in the PAL device requires a closer look at the timing and output switching functions. The A0 and RD control lines are in effect part of the system bus control and, therefore, cannot be multiplexed easily. The DS and R/W lines to the CRT controller are also fixed because they must be valid throughout the entire transfer cycle as well.

This leaves 6 of the 10 output logic cells of the PAL device to represent the remaining 10 identified control lines. This method of minimization involves careful state synthesis, analysis of the signal switching functions during the transfers, and utilization of several control pins on the CRT controller. By using the BREQ, BACKI, BACKO, CS, and C/D inputs to the PAL device, we can reduce the number of unique states required to 8 instead of 15. This reduces the number of logic cells required for the state machine from 4 to 3 bits.

At this stage, the design requires seven control signals to manipulate the data transfer registers and WAIT line. The two latch enables (CES and CDR) on the Am2952 bidirectional I/O port can be

permanently enabled. By controlling the clock signal to the latches, the controls required for three pins can be reduced to one. The interface control state machine will only use the correct side of the dual latches on the bidirectional I/O port.

The Am8052 CRT controller helps considerably with its own control bus interface. Two signals provided by the CRT controller, $\overline{\text{TBEN}}$ and $\overline{\text{RBEN}}$, switch the data transceivers in the correct direction regardless of the type of data transfer (as a bus master or bus slave). When the controller is a bus master performing a read operation, or when it is a bus slave undergoing a write operation, therefore, the $\overline{\text{RBEN}}$ signal is strobed to obtain the correct polarity. By using this line, two of the remaining six control lines can be eliminated ($\overline{\text{REN}}$ on the Am2949 and $\overline{\text{OEAS}}$ on the Am2952). Although the $\overline{\text{TBEN}}$ line performs a similar function, it does not function correctly in a 16- to 8-bit multiplexed bus environment.

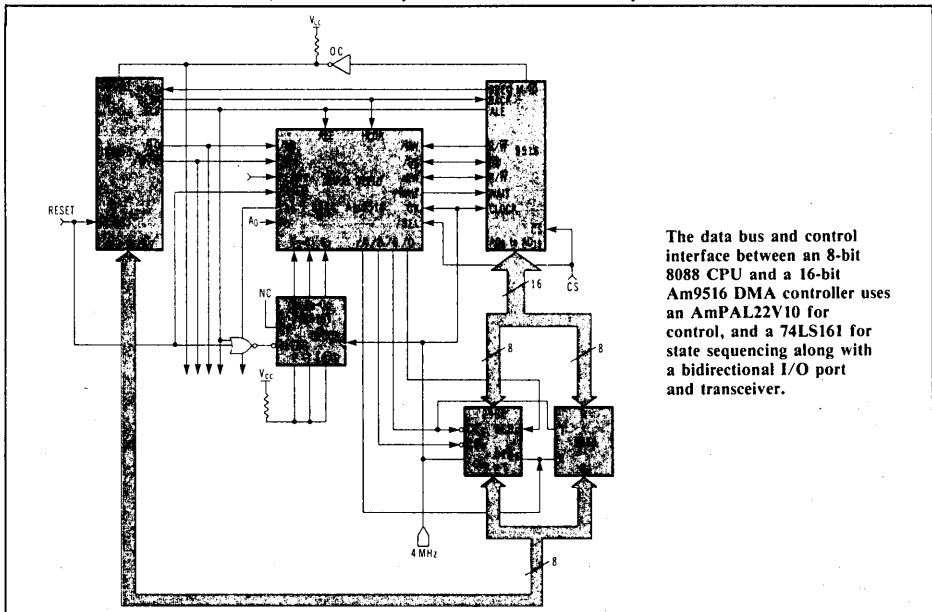
Two of the remaining control lines ($\overline{\text{OEAS}}$ on the Am2952 and 10 on the bidirectional bus transceiver) must be generated by individual cells in the PAL device. The two clock enables on the Am2952 are permanently enabled. The two Am2952 clocks are tied together to minimize the amount of logic required in the PAL device used to generate clock strobes to the latches.

This leaves the design with three logic cells and four output functions (the $\overline{\text{WAIT}}$ line to the CRT controller and the 3 state bits). Careful analysis of

the state flows and timing diagrams indicates that the $\overline{\text{WAIT}}$ line is only asserted in 4 of the 8 states. A clever assignment of state numbers to the state flow sequence allows the $\overline{\text{WAIT}}$ line to be absorbed into the 3 state encoding bits. The logic equations for the AmPAL22V10 device can be derived directly from the timing diagrams.

An unusual problem might occur when a peripheral device operates as a bus slave on a smaller data bus, such as a 16-bit peripheral to 8-bit CPU. During the first slave write operation, the chip select $\overline{\text{CS}}$ is enabled by the bus master making the access. No actual data—just the data latch—is strobed into the peripheral, however. After the first byte of data has been written, the second access causes the full 16-bit data to be strobed into the peripheral.

If the designer is using a common $\overline{\text{CS}}$ function to both the peripheral and the 8/16-bit control logic, the controller logic must be designed not to glitch or strobe any of the control lines to the peripheral (it must prevent $\overline{\text{DS}}$, R/W from being enabled, for example). For some peripheral devices, glitches on the control lines might cause the register to be written accidentally onto a register that will be overwritten in the next write cycle anyway. With other peripherals this might be a catastrophic event. Many devices acting as bus slaves have write recovery time requirements (ie, a certain minimum interval between consecutive write operations). Glitches on the control lines might force the next (and final) write operation to be delayed—or cause a violation of the



device specifications. Glitches might evade any special addressing/register accessing scheme used in the peripheral. This might occur, for example, if the slave device requires the user to write the address of the register that was accessed immediately before the register was written. In this case, glitches or useless control strobes could wreck the sequence.

The problem can also be solved by using two lines. In this solution, one of the lines would go to the peripheral device and the other would connect to the 8/16-bit controller. The chip select to the peripheral is activated each time a slave read occurs (for both upper and lower byte accesses), or when a slave write operation occurs and the unlatched 8-bit data is being written. The chip select function to the 8/16-bit controller is chosen each time the peripheral is selected normally (for slave read/writes on both upper and lower 8-bit data transfers). This problem is bypassed completely when two separate chip select functions are used: one for loading up the Am2952 latch during a slave write/read and one to strobe the Am8052 controller into action when it is needed by the 8-bit CPU.

Bus conversion maximizes flexibility

A data bus and control interface to an 8088 8-bit microprocessor and Am9516 16-bit DMA controller can be created using four devices: an AmPAL22V10 for the control block, a 74LS161 counter for the state sequencer, an Am2952 bidirectional I/O port, and an Am2949 bidirectional transceiver.

This design incorporates certain simplifications. The DMA controller requires word accesses only during command chaining and for slave register accesses. The 8/16-bit data transfer interface for bus master operations (ie, DMA data transfer functions) is handled automatically as a programmable option. During slave write operations, the first byte output to the DMA controller must have an odd address and the following second byte an even address. Conversely, during a slave read cycle, the first byte read from the DMA controller must be at an even address and the second at the next higher odd address.

Furthermore, for bus master operations, the system must use the latched address line A0 (LA0) from the AmPAL22V10 as its sole A0. Because the logic is already available, the system does not have to provide this function. LA0 now becomes the system address bit 0 with full 24-mA drive capability.

Deciding on a means for controlling the funneling of the data stream—that is, transforming 16-bit data into 8-bit data and vice versa—was the first step in deriving this example. As mentioned earlier, simply dividing each 16-bit access into two 8-bit data transfer cycles presents one way of doing this. On outgoing accesses (16-bit path from the DMA controller) during the first cycle, the upper half of the 16-bit path is latched while the lower half passes through

```

PIN
CK      = 1  /RD  = 23
S[0:2]  = 2:4  /WR  = 22
AO      = 5    LA0  = 21
/SEL    = 6    /DS  = 20
ALE     = 7    /RW  = 19
HLDA    = 8    /WAIT = 18
/BW     = 9    /A   = 17
READY   = 10  /B   = 16
RESET   = 11  /C   = 15
        /D   = 14;

BEGIN
  IF (RESET) THEN ARESET[ ];
  This section defines the wiggles when the Am9516 is bus master

  IF (HLDA) THEN ENABLE[ ];
  IF (/S[2] * HLDA) THEN BEGIN
    IF (S[1] * /S[0]) THEN
      LA0 = /CK * BW * /BW * AO *
            ALE + /BW * LA0 * /ALE ;
    ELSE
      LA0 = BW * /BW * AO *
            ALE + /BW * LA0 * /ALE ;
  END;
  IF (HLDA) THEN
    (CASE) (S[2:0])
    BEGIN
      1) BEGIN
        RD = /RW * DS ;
        A  = /BW * /RW * /CK ;
        WR = /BW * RW * DS ;
        C  = /BW * RW ;
        WAIT = 1 ;
      END;
      2) BEGIN
        RD = /RW * DS ;
        B  = BW ;
        A  = /BW * /RW ;
        WR = /BW * RW * DS ;
        C  = /BW * RW ;
        WAIT = /BW ;
      END;
      3) BEGIN
        RD = /RW * DS * B ;
        B  = BW * CK ;
        A  = /BW * RD ;
        WR = /BW * RW * DS ;
        C  = /BW * RW ;
        WAIT = BW ;
      END;
      5) BEGIN
        RD = /RW * DS ;
        A  = /BW * /CK ;
        WAIT = BW ;
      END;
      6) BEGIN
        RD = /RW * DS ;
        A  = /BW ;
      END;
      7) BEGIN
        RD = /RW * DS ;
        A  = /RD ;
      END;
    END;
  END;
  This section defines the wiggles when the 8088 is bus master
  BEGIN
    LA0 = AO * ALE * SEL + LA0 * /ALE * SEL
    B   = LA0 * WR * SEL
    A   = /LA0 * WR * SEL
    DS  = A + /LA0 * RD * SEL
    C   = /LA0 * RD * SEL
    D   = LA0 * RD * SEL
  END;
END.

```

This PLPL file implements an interface between the 8-bit 8088 and the 16-bit Am9516.

Programming the PAL and the counter

In writing the Programming Language for Programmable Logic (PLPL) file to control the operation of the AmPAL22V10 and the 74LS161 counter, the inputs to the PAL device from the counter are assigned S0, S1, and S2, respectively. Then, it is possible to apply a "sculptured design" technique to the entire timing diagram (see figure in Panel, "A matter of timing") by using the Case statement from PLPL. By assigning combinatorial equations to only one binary partition or column at a time (Case), the designer can ignore all other aspects of the design for the time being and generate simple equations directly from the timing waveforms.

During clock time T1 of the Am9516's word read cycle the state of the 74LS161 (S0, S1, S2) is cleared to 000 by the assertion of address latch enable (ALE). LA0 is the only output control signal from the CRT controller asserted during this period. This signal is handled as a special case, however. During time T2 of the DMA controller's word read cycle, the RD and WAIT outputs from the CRT controller must be asserted. This time partition corresponds to the state inputs S2, S1, S0 = 001. Therefore, the first Case equations are

```
CASE (S[2:0])
BEGIN
1) BEGIN
    RD = /RW*DS ; Transform Control
                ; Signals /RW and DS
                ; into Intel /RD

    WAIT = 1    ; Assert Wait
                ; unconditionally

END;
```

During time T2 of the DMA controller's byte read cycle, A is the only additional output not already

accounted for in the Case statement. This signal allows a byte of data to flow through the bidirectional bus transceiver into the DMA controller during byte read operations. Some additional constraints are placed on this signal, however: it must only be asserted in time T2 on byte read operations (the B/W input) and it must be delayed by a half clock period from the rising edge of T2 (CK signal). Thus the Case statement becomes

```
CASE (S[2:0])
BEGIN
1) BEGIN
    RD = /RW*DS ;
    A = /B/W*/RW*/CK ; enable the
                        ; receiver

    WAIT = 1

END;
```

Finally, by examining the last time T2 elements (WR and C) during the DMA controller's byte write cycle, the remaining terms in Case 1 are derived. With the exception of LA0, the remaining equations were developed in the same fashion. Clearly, this "sculptured" technique is a very simple and methodical means for arriving at the Boolean requirements for a logic block.

As the PLPL listing shows, the signal LA0 was handled slightly differently from the previously discussed method. The number of product terms generated via the Case statement made this approach necessary. The number exceeded the upper limit (16 terms) for a programmable logic array. As a practical matter, therefore, it was necessary to optimize this signal manually. However, it should be noted that this step will not be necessary once the fully optimized version of PLPL becomes available.

a tri-state buffer onto the 8-bit bus. During the second cycle, the tri-state buffer is turned off and the previously latched half of the data is driven onto the bus. On incoming accesses (8-bit path to 16-bit path), the process is reversed.

The control mechanisms that perform this cycling depend on the WAIT and R/W signals passing to and from the DMA controller, and on the ability to enable or disable the latches and transceivers selectively. The Am2952 bidirectional I/O port was chosen because of its dual registers and its flexible control. The AmPAL22V10 device was chosen to match the required number of control pins and functions. Since the complexity of this design requires the use of all of the PAL's I/O pins for control functions, however, it was necessary to use a 74LS161 counter to provide the state sequencer function.

Programming with PLPL

It has long been the logic designer's "art" to merge the often very different concepts and notations of timing information with Boolean logic. Yet, the evolu-

tion of a syntax to fully express this art has taken a long time. AMD recently developed such a language for programming the AmPAL22V10, however.

"Programming Language for Programmable Logic," or PLPL, allows the designer to specify a design using multiple input formats. This specification flexibility supports the variety of design approaches necessary to express different design problems efficiently. These formats range from simple sum-of-products Boolean equations to high level constructs. PLPL also supports the input specifications for many types of AND/OR based devices, including all of the current AMD programmable logic array and PROM devices.

PLPL is block structured, and includes the high level language constructs If-Then-Else, Case, and For; all familiar to many programmers of the C and Pascal languages. Macros, functions, constants, and variables may also be used in PLPL. The language also facilitates use, clarity, and self-documentation.

Such current programmable logic technology and associated programming language as PLPL allow

A matter of timing

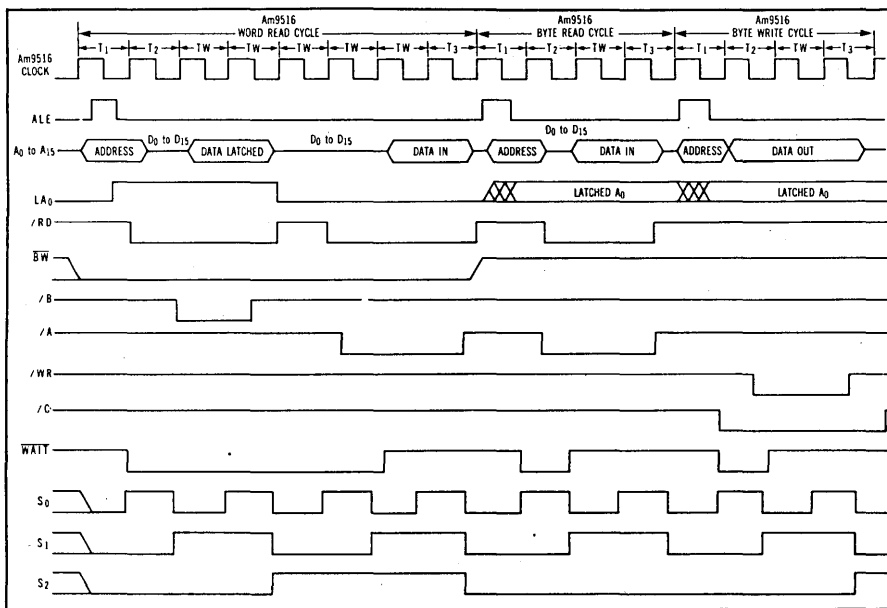
The complex AmPAL22V10 design used the accompanying timing diagram to correspond to the desired waveforms. They are partitioned by the respective binary state (or count) from the counter.

The desired timing requirements during the period when the DMA controller is bus master appears below. During time T1, address latch enable (ALE) is asserted by the DMA controller to denote the beginning of the cycle; a short time later, an address is driven onto the bus. This address is valid at the falling edge of ALE. The control signal LA0 (latched A0), therefore, must be valid at this time, as well. In this phase of the cycle, it must also be high to enable the odd byte from memory to be loaded into the bidirectional I/O port. In addition, the assertion of ALE performs the function of resetting the 74LS161 counter to 0000 in order to synchronize the cycle.

During time T2, the DMA controller will assert its DS signal. The timing for this signal, in conjunction with the R/W signal (asserted in T1) must be trans-

formed into an 8088-equivalent RD signal. During a word read cycle, this RD signal also must be artificially negated and then reasserted to accomplish a double byte read. At the same time, the DMA controller must be "parked" in order to multiplex or assemble a word. Thus, the WAIT signal is also asserted at time T2. During time TW (S2, S1, S0 = 010), the receiver clock enable control signal B must be asserted in order to allow the next system clock's rising edge to strobe the upper byte into the bidirectional I/O port. This is accomplished during the next TW period (S2, S1, S0 = 011).

During the remainder of the word read cycle, RD is negated and then reasserted after LA0 has been forced low to address the even byte. A is then asserted to allow both the previously latched upper byte and the current lower byte to be driven onto the DMA controller's pins. And finally, the WAIT signal is negated, allowing the DMA controller to finish its read cycle by strobing in the 16 bits of command data on its data pins.



highly organized application-oriented control blocks to be formed easily. These tools can conceptually raise the designer above the details of the design at the logic level and directly translate the necessary response characteristics from a timing diagram.

This approach can be referred to as a "sculptured design" technique because it is analogous to the way solid stone is formed according to an artist's image. Raw logic can be transformed directly into useful control functions from the desired timing information.

The AmPAL22V10 is, in essence, a fuse-programmable gate containing up to 22 inputs and 10 outputs. It can define and program that architecture of each output on a pin by pin basis. Thus, the designer is free to optimize the design mix between registered and combinatorial functions as needed.

The AmPAL22V10 is programmed by opening fusible links in any or all of its 10 output macrocells, as well as in its AND gate array. The AND gate structure is very similar to other PAL devices; therefore

it allows the same powerful, yet familiar features. However, it is the AmPAL22V10's 10 output logic macrocells that give the designer substantial new design freedom. Moreover, at each macrocell output is a tri-state output buffer controlled by a separate output-enable AND gate.

These macrocells provide the AmPAL22V10's key features. They can be configured to make any or all of the I/O pins act either in sequence or in combination and have either active-high or active-low characteristics. Furthermore, the output enables can individually control the direction of the pins so they act as outputs, inputs, or bidirectional ports.

A number of trade-offs and limitations are apparent in a design that so dramatically affects the input and output of the system. The most obvious limitation stems from under utilization of 16-bit peripherals on an 8-bit bus—the speed of all I/O operations are cut in half. As a result, bus utilization will increase if the 16-bit peripheral represents a significant factor of the bus use. A CRT controller such as the Am8052 might use 5 to 10 percent of the bus bandwidth for display information when using 16-bit I/O. Converting to 8-bit I/O would double bus use to 10 to 20 percent. Another factor that might affect the bus usage is the efficiency of the 8- to 16-bit con-

version control logic. If the state machine designed to perform the 8/16-bit (or 16/32-bit) conversion is improperly designed, extra transfer overhead might be introduced. This might mean a sequential transfer of two 8-bit values would take twice as long a single 16-bit transfer.

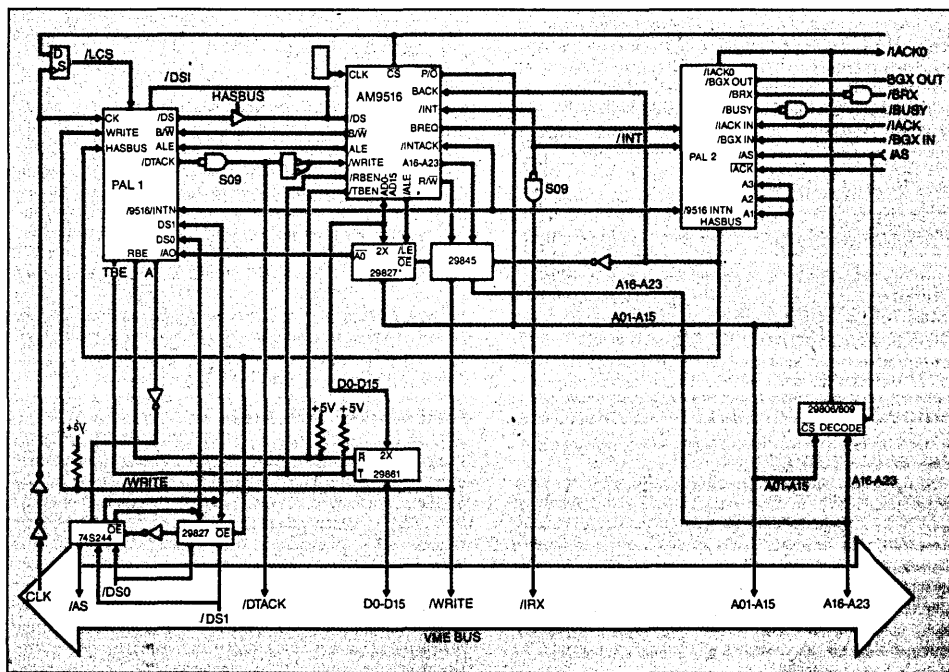
The design constraints might limit the use of the peripheral to byte-only operations during data transfers (as in the design using the DMA Am9516 controller), and slow it down by a factor of two during command operations. For such a DMA device as the Am9516, the extra time required for command fetching is not usually a significant portion of bus time.

System designers will have to weigh the cost of the extra overhead on a case-by-case basis. The benefits may well justify these limitations—particularly when the bus is self-limiting, but the device characteristics allow for value-added designs. In addition to bus degradation for certain configurations, extra logic and design effort are involved. Most interfaces outside a system's immediate family require some kind of extra interface logic, however. By manipulating the signals and incorporating them into programmable logic devices such as the AmPAL22V10 device, therefore, most of this logic is free.

High Performance DMA For The VMEbus

3

To remain compatible with existing peripherals, the Am9516 can perform a byte packing/unpacking function. This allows efficient use of word-organized memory when interfacing to byte-organized memory or peripherals. The part has the ability



DIGITAL DESIGN ■ NOVEMBER 1985

to load its own control parameters into its on-board registers in order to facilitate loading of initialization parameters. The CPU writes these parameters to local memory, informs the Am9516 of the data block address and issues a start chain command to key the initialization process.

Upon completion of a DMA operation, the Am9516 can do one of several things. For systems requiring DMA between two areas of memory, on-board base registers are provided for each channel. This allows fast reloading of the current address and operation count registers. Changing direction of the transfer simply requires changing the state of the "flip bit" in the channel mode register. Upon completion, the Am9516 can perform another chain operation to prepare for a different type of DMA transfer. Using its on-board interrupt logic, the device can inform the host of its completion status by sending an interrupt and responding to the host with a user-specified interrupt vector.

To accommodate slow memory or peripherals, the Am9516 provides a means to add wait states. The wait line can be driven with normal types of transfer acknowledge signals, such as XACK in a Multibus environment, READY in an 8086 system or DTACK in a 68000 system. With the Am9516, wait states can also be added under software control. Users simply program the transfers' required number of wait states (0, 1, 2 or 4 may be selected) into the address register tag field. This feature makes it possible to stretch bus cycles without hardware overhead.

The VMEbus makes provisions for address modifier lines to allow implementation of paging schemes and memory management functions. These lines also allow access to several sequential locations of storage while providing only one address. VME also caters to 32-bit data fields and 32-bit address fields. The bus design makes provisions for system or module failures. Signals such as bus time-out, bus error and power fail can serve as fault-finding mechanisms to ensure overall system integrity.

Interfacing The Bus

A basic interface to the VMEbus may be based on the Am9516. Although features such as bus exceptions or address modifier codes are not considered here, the basic approach can expand to allow for additional features or special user requirements. The system design uses an Option One priority arbitration scheme, but could easily be expanded to implement other formats. PAL devices minimize the hardware required for user-customization of this system. The design uses two PALs: an AmPAL16R4 device, which performs basic signal translations to and from the Am9516 when it is a master or a slave on the VMEbus; and an AmPAL16L8 for handling bus arbitration and interrupt responses. A breakdown of the signal descriptions used are shown in Table 1.

PAL1 provides the logic that interfaces to the VMEbus when the peripheral chip is either a master or a slave. When the Am9516 is configured as a bus slave to another master residing on the VMEbus, the address is decoded to generate an Am9516 chip select signal. This signal is also latched (LCS) to begin an internal state sequence in PAL1. State variables A and B determine both the start and the end of the required Am9516 data strobe (DS) and the VME acknowledge signal (DTACK). Timing of these signals changes slightly depending on whether the cycle is read or write.

When the Am9516 is a bus master, PAL1 generates all VME-

CLOCK(CK)	VME Bus Clock used to gauge state sequence timing
WRITE	Am9516 Read/Write control signal—PAL Input
LCS	Latched Chip Select, starts PAL state timing sequence—PAL Input
HASBUS	Signal to indicate the Am9516 has control of VMEbus—Output of PAL2, Input to PAL1
BW	Am9516 Byte/Word control signal—PAL Input
A₀	Am9516 Low Order Address bit—PAL Input
DS₀, DS₁	VME Bus Data Strobes—Input and Output to PAL1
DS	Data Strobe generated by PAL1 to drive Am9516 DS line
AS	Address Strobe—Input to PAL2
DTACK	Data XFER Acknowledge signal generated by PAL1 to drive the VME DTACK control line
A, B	Internally-generated state variables used to create other required control signals
TBE, REE	Transmit and Receive Buffer Enables—Outputs of PAL1
DSI	Data Strobe Input to PAL1—Output of Am9516 DS
INTNTA	Indicates an Interrupt Response to the Am9516
BREQ	Am9516 Bus Request signal—Input to PAL2
BGXIN	VME Bus Grant In signal—Input to PAL2
A₁, A₂, A₃	Decoded to determine if present Interrupt Acknowledge cycle is at user-specified request level—Input to PAL2
MYLEVEL	Indicates the interrupt is at the user-specified request level
IACKOUT	Interrupt Acknowledge Out from VME—Input to PAL2
IACK	Interrupt Acknowledge—Input to PAL2
IACKIN	Interrupt Acknowledge In from VME daisy-chain—Input to PAL2
INT	Interrupt Request from Am9516—Input to PAL2
BGXOUT	VME Bus Grant Out signal—Output from PAL2
BBSY	VME Bus Busy signal—Output from PAL2
BRX	VME Bus Request signal—Output from PAL2

Table 1: Breakdown of the signal descriptions used in the interface of the Am9516 to the VMEbus.

required control signals not directly generated by the Am9516. The address strobe generated by 8-MHz and 10-MHz versions of the Am9516 does not meet the VME address valid to address strobe falling edge set-up time. Thus, state variables A and B must be used to create this address strobe signal. An external buffer on the output of A provides sufficient drive for the VME address strobe control line. PAL1 also generates the correct VME Data Strobe (DS₀, DS₁) combination from the Am9516's Data Strobe, Address Line A₀ and the Byte/Word signals.

Table 2 shows the PAL equation, the pinout of the device, logical equations for all signals and a function table (truth table). Figure 1 details the interconnection of the Am9516 to the two interface PAL devices and the VMEbus.

Requesting The Bus

PAL2 controls the acquisition of the VMEbus by the Am9516 and generates the signals for both the Am9516 and the VMEbus during interrupt acknowledge cycles. To obtain use of the VMEbus, the Am9516 requests the bus by driving its BREQ line high. This signal is received as an input by PAL2. A VME bus request (BRX) is generated only if there is no active bus-grant-in (BGXIN). Such gating ensures that the Am9516 does not steal the bus from a lower priority master that was just granted use of the bus. Upon receipt of an active bus-grant-in (BGXIN) from the VME arbiter, PAL2 generates an active HASBUS signal indicating to the Am9516 that it now has control of the system bus. Simultaneously, PAL2 generates the VME "bus busy" signal (BBSY), indicating to the VME arbiter

and other VME masters that the bus is in use.

PAL2 also generates the proper levels on bus-grant-out (BGXOUT) to continue the bus acknowledge daisy chain. This design assumes an Option One arbiter which responds only to requests on bus request line 3 (BR₃) and depends upon the daisy chain for prioritization. VME also specifies a parallel method for bus arbitration that could, with minor modifications, be implemented using this PAL design. Because an Option One arbiter is assumed, PAL2 does not generate a bus clear signal. The signal is only used by Option One priority arbiters to inform the present master on the bus that a higher priority master would like use of the bus. This design also implements an option called "release when done," which releases bus busy (BBSY) when the Am9516 indicates that it no longer wants use of the VME.

The other portion of this PAL creates the proper signal sequence during an interrupt acknowledge cycle. When the chip encounters a condition that should generate an interrupt, it drives its interrupt (INT) line low. This interrupt signal drives one of the VME interrupt request lines (IRQ₁-IRQ₇) as well as an input to PAL2. Upon receipt of an interrupt acknowledge (IACK) signal, the PAL device decodes the three low-order address lines. If the present interrupt acknowledge cycle is at the Am9516's predetermined level when interrupt acknowledge in (IACKIN) is received low, PAL2 generates the 9516INTA signal that drives the Am9516 interrupt acknowledge input (INTACK). The Am9516 then responds by driving the data bus with its user-programmable vector.

PAL1 also plays a role in interrupt acknowledge cycles. When it detects a valid 9516INTA signal, an internal state sequence begins which generates the proper transfer acknowledge (DTACK) to the interrupt handler. When PAL1 sees the data strobe ($\overline{DS}_0$) go high, the cycle ends and DTACK is driven back high. PAL2 also generates the proper signal level on interrupt-acknowledge-out (IACKOUT) to complete the interrupt acknowledge daisy chain. Table 3 is the PAL equation for PAL2 and details the pinout of this part as well as the equations that define its operation.

Meeting VME Power Requirements

The VME specification puts some rather stringent dc requirements on bus signals. AMD manufactures a family of bus interface parts that satisfy the requirements for most of the VMEbus signals. The bus grant and the interrupt acknowledge daisy-chain signals need not be buffered since their drive requirement is moderate. Other signals, such as AS and $\overline{DS}$, are buffered by a 74S244.

As a bus slave, the DMA card must receive an address from the VMEbus and decode it to generate a chip select to the Am9516. AMD's 29809 and 29806 equal-to comparators suit this need. In this slave mode, the Am29827 10-bit buffer receives all necessary control signals from the VMEbus to initiate the required bus cycle at the Am9516. To complete the cycle, PAL1 generates a DTACK signal which must go through a 74S09 gate to meet the drive requirements of DTACK on VME. The Am29861 is a data transceiver needed to drive the VMEbus or the Am9516.

As a bus master, the same Am29861 ICs provide the transceiver functions necessary on the VME data bus. The direction in this case is controlled by the Am9516 rather than the PAL1 outputs. The lower 16 bits of address out of the Am9516 are lat-

CK	WRITE	LCS	HASBUS	BW	AO	ALE	DSI	/9516INTA	GND
/OE	/RBE	/TBE	/B	/A	/DTACK	/DS	/DS1	/DS0	VCC

:INTERNAL STATE VARIABLES

A	:=/LCSxDSx/HASBUS	+ :EON FOR /A-SLAVE
	/LCSx/HASBUSxAxD0xWRITE	+ :EON FOR /A-SLAVE RD ONLY
	/LCSx/HASBUSxAxD01xWRITE	+ :EON FOR /A-SLAVE RD ONLY
	ALEx/BxHASBUS	+ :EON FOR /A-9516 MASTER
	/ALEx/BxHASBUSxA	+ :EON FOR /A-9516 MASTER
B	9516INTAx/Ax/BxDS0	+ :EON FOR /A-INTA CYCLE
	9516INTAx/AxBxDS0	:EON FOR /A-INTA CYCLE
	:=/LCSxAx/HASBUS	+ :EON FOR /B-SLAVE RD/WR
	/LCSx/HASBUSxBxD0xWRITE	+ :EON FOR /B-SLAVE RD ONLY
	/LCSx/HASBUSxBxD01xWRITE	+ :EON FOR /B-SLAVE RD ONLY
	Ax/BxHASBUS	+ :EON FOR /B-9516 MASTER
	9516INTAx/Ax/BxDS0	+ :EON FOR /B-INTA CYCLE
	9516INTAx/AxBxDS0	+ :EON FOR /B-INTA CYCLE
	9516INTAx/AxBxDS0	:EON FOR /B-INTA CYCLE
	:9516 OUTPUTS TO VME	
DS	:=/LCSx/Bx/HASBUS	+ :EON FOR /DS-SLAVE RD/WR
	/LCSxBxD0xWRITE	+ :EON FOR /DS-SLAVE RD ONLY
	/LCSxBxD01xWRITE	+ :EON FOR /DS-SLAVE RD ONLY
DTACK	:=/DSx/HASBUSxAxD0xWRITE	+ :EON FOR /DTACK-SLAVE WR
	/DSx/HASBUSxAxD01xWRITE	+ :EON FOR /DTACK-SLAVE WR
	DSx/HASBUSxAxBxD0xWRITE	+ :EON FOR /DTACK-SLAVE RD
	DSx/HASBUSxAxBxD01xWRITE	+ :EON FOR /DTACK-SLAVE RD
	9516INTAx/AxBxDS0	:EON FOR /DTACK-INTA CYCLE

: BUFFER ENABLE

IF (/HASBUS) TBE=Ax/WRITE

Table 2a: PAL1 handles signal translations to and from the Am9516 for both master and slave situations.

ched in an Am29845. Since it meets all requirements for driving the VME address lines, the Am29845 drives the bus directly. An additional Am29827 buffers the high-order address lines (A₁₆-A₂₃), as well as the read/write line, onto the system bus. Because of the high drive requirements specified for the VME data strobes ($\overline{DS}_0$, $\overline{DS}_1$) and address strobe (AS), the PAL cannot drive the bus directly. A 74S244 is used in this instance since it meets the 64-mA sink current specified by the VME. In addition to DTACK, three other signals must go through a 74S09 open collector AND gate to provide the drive capability and proper output type. These are an interrupt signal out of the Am9516 (INT) and bus request (BRX) and bus busy (BBSY) out of PAL2.

Handling Bus Exceptions

Both the Am9516 and the VMEbus provide users with mechanisms for handling bus errors and subsystem failures. The VME specification provides for a bus time-out module that can flag system errors or problems. This module monitors the data strobe lines, DTACK and bus error (BERR) signals. When a falling edge is detected on either of the data strobes, this

SEMICONDUCTOR TECHNOLOGY

module begins a time-out sequence. If the module times-outs before a valid DTACK signal or BERR signal, or a rising edge of either of the data strobes occurs, then this module will drive BERR low, which completes the data transfer and automatically flags a bus error.

The Am9516 can also use a portion of this feature to abort a DMA in progress if a bus exception is detected. In master mode, the Am9516 could monitor the bus error signal ($\overline{\text{BERR}}$) and, when active, it drives the Am9516 $\overline{\text{OP}}$ input low. Subsequently,

the Am9516 stops current DMA operation and relinquishes the system bus. The Am9516 could then generate an interrupt to the system indicating that a system error or problem has occurred. The VME board could also monitor the bus ac fail line which indicates that power is going down. In such a fatal situation, should the Am9516 have control of the bus, it should give it up. This scheme could be implemented using the Am9516 end-of-process (EOP) control line.

Although this design represents the implementation of an interrupt driven system, a simpler system could be designed that does not take advantage of the Am9516 interrupt logic. In fact, such a simple system might use one AMD 24-pin PAL device (AmPAL22V10) rather than two. However, the built-in interrupt controller adds flexibility within the system and provides for fast

	W R I T E	L C S	H A S B U S	B W	A 0	/ D S 0	/ D S 1	/ D S	/ D T A C K	/ A	/ B	/ T B E	/ R B E	A L E	D S I	/ N T A	COMMENTS
X	X	H	H	X	X	X	X	H	H	H	H	Z	Z	X	X	H	:CHECK HI-Z VARIABLES
C	H	L	L	X	X	X	L	L	L	H	H	H	H	X	X	H	:START OF WRITE CYCLE
C	L	L	L	L	X	X	L	L	L	L	L	L	L	X	X	H	:SECOND CLOCK CYCLE
C	L	L	L	L	X	X	L	L	L	L	L	L	L	X	X	H	:THIRD CLOCK CYCLE
C	L	L	L	L	X	X	L	L	L	L	L	L	L	X	X	H	:FOURTH CLOCK CYCLE
C	L	L	L	L	X	X	L	L	L	L	L	L	L	X	X	H	:FIFTH CLOCK CYCLE
C	L	L	L	L	X	X	L	L	L	L	L	L	L	X	X	H	:SIXTH CLOCK CYCLE
:																	:FINAL CYCLE
H	H	H	L	X	X	X	X	H	H	H	H	H	H	X	X	H	:START OF READ CYCLE
C	H	L	L	X	X	L	L	L	L	H	H	H	H	X	X	H	:FIRST CYCLE
C	H	L	L	X	X	L	L	L	L	H	L	L	L	X	X	H	:SECOND CLOCK CYCLE
C	H	L	L	X	X	L	L	L	L	L	L	L	L	X	X	H	:THIRD CLOCK CYCLE
C	H	L	L	X	X	L	L	L	L	L	L	L	L	X	X	H	:FOURTH CLOCK CYCLE
C	H	L	L	X	X	L	L	L	L	L	L	L	L	X	X	H	:9516 DATA VALID
C	H	L	L	X	X	L	L	L	L	L	L	L	L	X	X	H	:DATA STAYS VALID-DS LOW
C	H	L	L	X	X	H	H	H	H	L	L	L	L	X	X	H	:FINISH-DS/0 GO HIGH
C	H	L	L	X	X	H	H	H	H	H	H	H	H	X	X	H	:END CYCLE/START NEXT
:																	
C	H	H	L	X	X	Z	Z	H	H	H	H	H	L	L	H	H	:CHECK DATA STROBE 3-S
C	H	H	L	X	X	H	H	H	H	L	H	Z	Z	L	H	H	:CHECK BUF ENA 3-S
C	H	H	L	L	L	H	H	H	H	L	L	Z	Z	L	L	H	:9516 MASTR-WORD READ
C	H	H	L	L	L	L	L	H	H	L	L	Z	Z	L	L	H	:SECOND CYCLE
C	H	H	L	L	L	L	L	H	H	L	L	Z	Z	L	L	H	:THIRD CYCLE
H	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:WORD READ CK DATA STRBS
H	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:BYTE READ EVEN-CK DS1/DS0
H	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:BYTE READ ODD-CK DS1/DS0
C	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:WORD READ
C	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:FOURTH CYCLE
C	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:FIFTH CYCLE
C	H	H	L	L	L	L	L	H	H	H	H	Z	Z	L	L	H	:9516 DS GONE
:																	
H	H	H	L	X	X	Z	Z	H	H	H	H	H	L	L	H	H	:DS-3S, INTA CYCLE
C	H	H	L	X	X	L	Z	H	H	H	H	H	L	L	H	H	:START-INTA CYCLE
C	H	H	L	X	X	L	Z	H	H	L	L	L	L	L	H	H	:FIRST CLOCK
C	H	H	L	X	X	L	Z	H	H	L	L	L	L	L	L	L	:SECOND CLOCK
C	H	H	L	X	X	L	Z	H	H	L	L	L	L	L	L	L	:THIRD CLOCK
C	H	H	L	X	X	L	Z	H	H	L	L	L	L	L	L	L	:FOURTH CLOCK
C	H	H	L	X	X	L	Z	H	L	L	L	L	L	L	L	L	:DATA BUS VALID
C	H	H	L	X	X	L	Z	H	L	L	L	L	L	L	L	L	:DB STILL DRIVEN
C	H	H	L	X	X	H	Z	H	H	H	H	H	L	L	L	L	:END CYCLE
C	H	H	L	X	X	Z	Z	H	H	H	H	H	H	H	H	H	:START NEXT CYCLE

Table 2b: Truth table for PAL1. PAL1 allows the interface between the Am9516 and the VMEbus.

PAL16LB PAL 2	
NC /BREQ /BGXIN /IACK /IACKIN /AS /INT A1 A2 GND A3 NC /MYLEVL BRX /IACKOUT HASBUS /BGXOUT BBSY /9516INTA VCC	
BUS EXCHANGE VARIABLES	
/BRX =/BREQ+HASBUS+BGXIN+BBSY	
/HASBUS =/BREQ+/BGXINx/BRX+/HASBUS+/BBSY	
/BBSY =/HASBUS	
BGXOUT =BGXINxBRXx/BBSY+ BGXINxBGXOUTx/BBSY	
INTA CYCLE VARIABLES	
9516INTA =MYLEVLxINTxIACKINx/IACKOUT	
IACKOUT =/MYLEVLxIACKINxAS + /INTxIACKINxAS + IACKINxASxIACKOUT	
MYLEVL =A1xA2xA3xIACKxINT	
THIS SETS THE INTERRUPT LEVEL OF THE 9516. THIS DESIGN ASSUMES AN INT LEVEL OF 7. THE USER CAN CHANGE THIS BY REWRITING THIS EQUATION TO SET THE DESIRED INT LEVEL.	

Table 3a: PAL2 is required to interface the AM9516 to the VMEbus.

interrupt processing. Whenever a DMA operation is terminated, the Am9516 can generate an interrupt. There is also a two-level interrupt queue so that a new DMA cycle can start before an interrupt acknowledge cycle occurs. Reading the interrupt save register will give the interrupt handler the data needed to determine the cause of the interrupt.

VME Address Modifier Codes

As specified by VME, address modifier lines allow the master presently on the bus to pass additional information to the slave it is addressing. Some of the address modifier codes are defined by VME to perform certain functions, while others are user-defined. These address modifier codes allow such features as system partitioning (i.e., privileged or nonprivileged accesses), memory mapping or sequential access cycles that allow access of several locations without providing a separate address each time. The design does not account for address modifier codes even though VME specifies that all masters should drive these lines. This is left to the potential user since the system will need to be configured to meet those specific requirements.

□ Editor's note: A complete set of timing diagrams for this design is available upon request from AMD. For further information, please circle 619.

DD

How useful did you find this article? Please circle the appropriate number on the Reader Inquiry Card.

Very Useful 613
Useful 614
Somewhat Useful 615

3

FUNCTION TABLE														
BREQ	/BGXIN	/IACK	/IACKIN	/AS	/INT	/9516INTA								
/IACKOUT	HASBUS	/BGXOUT	/BBSY	/BRX	A1	A2	A3	/MYLEVL						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						
:	:	:	:	:	:	:	:	:						

Table 3b: Truth table for PAL2. PAL2 provides VMEbus arbitration as well as generating the signals for an interrupt acknowledge cycle.

APPLICATIONS NOTEBOOK

A Demultiplexed Analog Subsystem

The new monolithic analog to digital converters generally do not provide a 16-bit formatted data word for immediate processing by a CPU. These A/D converters usually multiplex data through an eight-bit port in byte and nibble form. There is no easy way for a processor, regardless of the ALU bit size, to handle this unformatted data directly since it requires demultiplexing and reformatting the data coming from the converter. This results in inefficient and slow data processing. For an existing system to use one of these new converters it could be a major undertaking to modify the system software to achieve the proper data format for processing.

Advanced Micro Devices' Am6112, which multiplexes 12 bits of data through an eight-bit port in two's complement, requires the following algorithm to reformat the data for a 16-bit processor. In the first read of the converter's port, the four most significant bits of a 12-bit data word are read into one of the processor's registers (bit locations B0 through B3). In the second read, the eight least significant bits are read into a different register (bit locations B0 through B7). To merge the two registers, the processor must do a logical shift of the four most significant bits to locations B8 through B11 in the first register. B11 is tested for sign, and, if negative, bits B12 through B15 are set. The shifted and corrected register must then be merged with the second register to get a formatted data word of any value for processing. An analogous operation must be done if the conversion code is one's offset. The added processing time required is now equivalent to five or six instructions per data fetch.

The following design, using a PAL device, is a hardware/software solution for the interface of the Am6112 to a 16-bit processor — shown in Figure 1. This hardware design doesn't require the five or six instructions for formatting to be used each time there is a data fetch, thus avoiding an untenable software modification or an increase in processing time.

The first step in the design is to construct a timing chart (Figure 2). The chart shows the timing interactions between the PAL device and the Am6112. After working out the timing, schematics are drawn and the allocation of inputs and outputs can be assigned to the PAL device (Figure

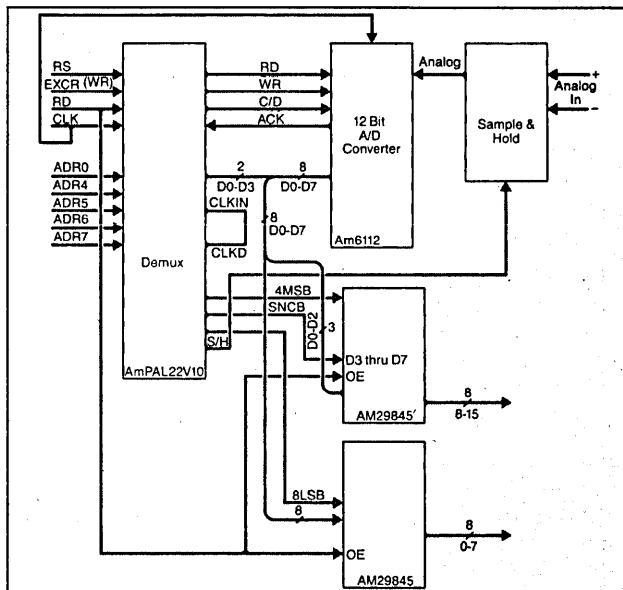


Figure 1: Circuit diagram showing AMD's Am6112, ADC, controlled by AMD's new AmPAL22V10 PAL with a parallel 16-bit bus interface. The AmPAL22V10 functions as a state machine to initialize and control the sample and hold device, the Am6112, and provide the logic necessary to demultiplex 12 bits of data from the Am6112 and place it into a 16-bit data format with corrected sign bit.

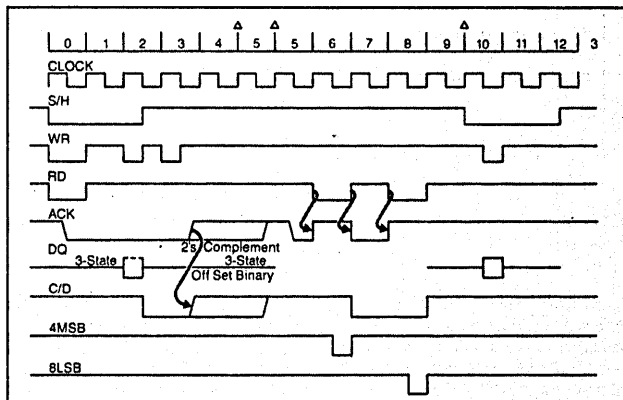


Figure 2: Analog to digital conversion cycle showing power up initialization — states 0 to 2, start of conversion — state 3, conversion cycle — states 5 and 6, and latching of data from conversion — states 6 and 8. The first entry point is the power up sequence in state 0 and the second is in state 9. In states 5 and 9 the clock does not run until conversion is complete or an external conversion is requested, respectively.

APPLICATIONS NOTEBOOK

I). Finally, the sequential states and the Boolean equations are defined and assembled. In this instance the sequential states are the successive binary states of the timing.

The parts used in the interface design are Advanced Micro Devices' AmPAL22V10, the Am6112 12-bit analog to digital converter, and two Am29845 bus registers. The AmPAL22V10 is mnemonically called "DEMUX," meaning de-multiplexer.

The design requires minimal clocking during the actual conversion of the analog signal in the Am6112. This is done by gating the acknowledge signal from the Am6112 and one of the binary states from the state counter with the clock signal through the PAL device. There is also minimal clocking when the Am6112 is idle. Again, by gating the clock with a binary state from the state counter and the signal that requests a conversion, clocking in the circuit is minimized. In the PALASM source listing, the clock control equation has ADR4 through ADR7 factors. These factors are the I/O address lines from the CPU. The I/O address de-

coding for chip select is done in the AmPAL22V10 for the conversion circuitry.

After an external conversion request, the clock is gated on to start the Am6112's conversion. The external conversion request signal is the control bus' write signal. Thus the first equation for the PAL device is state 5 and acknowledge low, or state 9 and external conversion request low which allows the clock to run the state counter.

Not clocking during conversion or while idle prevents unwanted noise, especially in the analog signal, and minimizes the overall system noise generation due to output switching. Using these design rules does not alter the actual conversion time in the Am6112. The Am6112 requires approximately $14\frac{1}{2}$ clock periods for a conversion. The state machine requires 13 clocks but three overlap with the clocking of the Am6112. Thus, the total number of clocks in a conversion is $24\frac{1}{2}$ from the leading edge of the control bus' write signal.

Creating the Boolean equations for the interface design is a heuristic process using the timing chart. For example, in

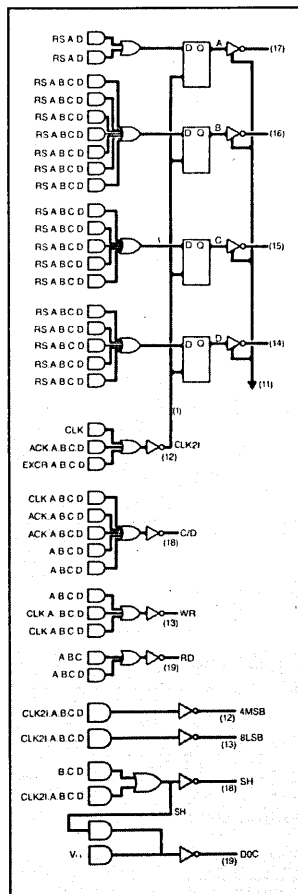


Figure 3: Graphic representation of the functions expressed in the PALASM source listing DEMUX for the AmPAL22V10 PAL.

defining the read equation from the timing figure, read must be low during state zero, state six and state eight. The actual logic for the read signal is implemented as shown in Figure 3, and its Boolean expression is in the accompanying PALASM listing (Figure 4). In states two and three, the write signal will go active low during only the first half of each state's clock cycle rather than for the whole period as in the read signal. For the command/data signal the timing figure shows it going low in the

APPLICATIONS NOTEBOOK

CLKIN RS EXCR SYSRD ACK CLK D3 ADR0 ADR4 ADR5 ADR6 GND
ADR7 A B C D D0 CLK0 /SH /CD /WR /RD VCC

: CONTROL SIGNALS

RD = $/A/B/C + /A/B/C/D$

WR = $/A/B/C/D + /A/B/C/D/CLKO +$
 $A/B/C/D/CLKO + /A/B/C/D/CLKO$

CD = $/A/B/C/D/CLK + A/B/C/D/JACK +$
 $/A/B/C/D/JACK + /A/B/C/D + A/B/C/D$

SH = $/A/B/C/D + A/B/C/D/CLKO +$
 $/A/B/C/D + A/B/C/D + /A/B/C/D/CLKO$

: CLOCK OUTPUT FOR STATE COUNTER

CLKO = $CLK + A/B/ACK/C/D +$
 $A/B/C/D/EXCR/ADR4/ADR5/ADR6/ADR7$

; EXCR & 10 @ I/O ADDR CX

: CONVERSION MODE CONTROL

IF ($/A/B/C/D/CLKO/ADR0 +$; I/O ADDR C0H AND C1H
 $/A/B/C/D/CLKO/ADR0$) D0 = SH ; ADR0 = LOW IS 1's & IF HIGH IS 2's.

: STATE COUNTER AND ADDITIONAL CONTROL SIGNALS

A = $/A/D/RS + /A/D/RS$

IF ($/A/B/C/D/D3$) B = $A/B/C/D/RS + /A/B/C/D/RS + ; [SNCB]$
 $A/B/C/D/RS + /A/B/C/D/RS + ; SIGN CONTROL$
 $A/B/C/D/RS + /A/B/C/D/RS + ; BIT$
 $/A/B/C/D/RS$

IF ($/A/B/C/D/CLKO$) C = $A/B/C/D/RS + /A/B/C/D/RS + ; [8LSB]$
 $A/B/C/D/RS + /A/B/C/D/RS + ; STROBE$
 $A/B/C/D/RS$

IF ($/A/B/C/D/CLKO$) D = $A/B/C/D/RS + /A/B/C/D/RS + ; [4MSB]$
 $A/B/C/D/RS + /A/B/C/D/RS + ; STROBE$
 $A/B/C/D/RS$

FUNCTION TABLE

CLK CLK0 CLKIN EXCR ACK RD WR CD SH D0 D3 D C B A RS

H	H	C	X	X	X	X	X	X	Z	X	X	X	X	X	L	: 0
H	H	H	H	X	L	L	H	L	Z	X	Z	Z	Z	L	H	: 0
L	L	L	X	L	L	L	H	L	Z	X	Z	Z	Z	L	H	: 0
H	H	C	X	L	H	H	H	L	Z	X	Z	Z	Z	H	H	: 1
H	H	H	X	L	H	H	H	L	Z	X	Z	Z	Z	H	H	:

Figure 4: PALASM listing and example of function table for application.

latter half of state two, remaining low through states three and four and rising to the high state, inactive, in state five. It also has a synchronous relationship to the acknowledge signal from the Am6112: command/data is "anded" such that it will not go inactive until the acknowledge signal becomes inactive.

The other control signals are similar to the read, write, and command/data signals. The data signal, D0 of the PAL, go-

ing to D0 of the Am6112, serves as an output during initialization in the first half of state two.

To generate all of the control signals, a four-bit counter was created in the PAL device to count from zero to 12. At reset, or power-up, the counter starts at zero and goes through one cycle, or counts to nine, then stops until the CPU exerts an active low on the write control line. States 10, 11 and 12 occur after the active leading edge

of the write signal, and provide a sampling window for the sample and hold circuit. In state 12, the state counter is loaded with a four-bit binary value of three. On the next clock edge, that which would sequence a counter to state 13, the counter is synchronously loaded with the value three and sequencing of the converter occurs for a conversion.

The B, C and D outputs of the counter are normally tri-stated. B, C and D out-

APPLICATIONS NOTEBOOK

puts are pulled to ground via resistors. When a strobe for data enable is needed for one of the bus latches, either C or D's output is taken out of tri-state and driven active high. B's output state is determined by the sign bit from the Am6112 converter.

It drives input bits B12 through B15 of the high half of the data bus latch in order to move the sign bit to the most significant bit position; this eliminates the need for software modifications. If the sign bit is positive, or active low, the output is not taken

out of tri-state. If the sign is negative, or active high, the output is taken out of tri-state and driven active high.

— Garret Spears, Field Applications Engineer, Advanced Micro Devices, Sunnyvale, CA

3.7.8 Article Reprint

Advanced Programming Language for Programmable Logic Design

by

Om Agrawal
Dept. MGR., Customizable Logic Product Planning
Advanced Micro Devices
981 Thompson Place
Sunnyvale, CA 94088

INTRODUCTION

With persistent advancements of LSI/VLSI technology, new generation of programmable logic devices (PLDs) - especially PAL (s) rivalling density and flexibility of gate arrays are appearing on the horizon.

PLDs with faster speed, denser arrays (with increasing number of logical product terms), better architectural features, built in support for diagnosability and with sophisticated software design tools, are finding wider and wider applications in various systems.

Two major factors which are fueling the growth of such advanced programmable logic devices are: Better software tools and better technology. Software tools are providing the "ease of design"; while better technology is providing the increased architectural flexibility, faster speed, reduced power and increased functional density. The major focus in this paper will be on the software side; on an advanced Programmable Logic Programming Language (PLPL).

ADVANCES IN SOFTWARE

With the increasing complexity and flexibilities of programmable logic devices, the need for better software tools becomes increasingly important. While the first generation software tools such as PALASM was adequate for relatively low complexity, first generation programmable logic devices, they are inadequate in several respects for future generation devices.

DESIRED FEATURES FOR ADVANCED SOFTWARE TOOLS FOR PROGRAMMABLE LOGIC

- o Compiler based rather than Assembler based

While the first generation software package was, relatively low level

* PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

machine language and Assembler based, the next generation software has to be Compiler based for simplifying the tasks of the designers and for utilizing the complexity of the future generation devices.

- o High Level Block Structured Language

For utilizing the complexity of future devices, and minimizing the task of relatively complex logic designs, the software should specify the design equations in a high level, block structured language.

- o Ease of Portability to different Operating Systems

The software should be developed in a high level language to allow it to be ported to different operating systems relatively easily.

- o User Friendly

The software package should be an easy and convenient tool, with all the features tailored towards USER friendliness.

- o Technology Independent, Device Independent and Hardware Programming Equipment independent

The software package should be able to support multiple programmable logic devices (PALs, PLAs and PROMs), with multiple technologies (Bipolar and CMOS); should run on a large number of operating systems (running on multiple machines - from low end PCs to very high end Main frame computers) and should support a large number of hardware programming equipments built by various manufacturers.

- o Should support multiple design formats, and should allow clear problem definition with relatively High Level Constructs

The software package should support problem definitions in multiple design formats and High Level Constructs - multiple design formats such as PALASM like sum-of-products Boolean equations, extended Boolean logic descriptions, DeMorgan's Laws, Macro substitution capability, pin vector specifications; and high level register transfer and state machine descriptions with high level constructs such as IF-THEN-ELSE, FOR, CASE, and FUNCTION CALLS. It should allow the devices to be expressed in terms of their functions rather than the implementation of their functions.

- o Should simplify the tasks of the designers

It should allow the designers to take full advantage of designing with programmable logic devices and should help them with all the aspects of the design - starting from defining the problem to be solved, creating a solution, verifying the solution by simulation, generating the test vectors, optimizing the list of Boolean equations.

- o Interactive or Batch Mode of Operation

The software package should have the capability of providing both interactive and batch mode of operation for designing with programmable logic devices.

- o Ease of adding new devices

The software package should be developed to support relatively easily the addition of new programmable logic devices. The environment should be preferably a Data Base Table driven, to achieve this functionality.

The software package should support problem definitions in multiple design formats and High Level Constructs - multiple design formats such as PALASM like sum-of-products Boolean equations, extended Boolean logic descriptions, DeMorgan's Laws, Macro substitution capability, pin vector specifications; and high level register transfer and state machine descriptions with high level constructs such as IF-THEN-ELSE, FOR, CASE, and FUNCTION CALLS. It should allow the devices to be expressed in terms of their functions rather than the implementation of their functions.

- o Should simplify the tasks of the designers

It should allow the designers to take full advantage of designing with programmable logic devices and should help them with all the aspects of the design - starting from defining the problem to be solved, creating a solution, verifying the solution by simulation, generating the test vectors, optimizing the list of Boolean equations.

- o Interactive or Batch Mode of Operation

The software package should have the capability of providing both interactive and batch mode of operation for designing with programmable logic devices.

- o Ease of adding new devices

The software package should be developed to support relatively easily the addition of new programmable logic devices. The environment should be preferably a Data Base Table driven, to achieve this functionality.

STRUCTURE OF PLPL ENVIRONMENT

Programmable Logic Programming Language (PLPL) is such an advanced software package - with all the above features. It provides an integrated top-down, hierarchical and complete package for designing with programmable logic devices. With its advanced features, it is a new design approach for further enhancing the short design turnaround time associated with all programmable logic devices.

Written in language C for portability and flexibility, it is a high level, block structured hardware description language for programmable logic devices. With its Data Base, table driven format, it has the capability to support multiple AND-OR logic based programmable logic devices such as PALs, PLAs and PROMs. It is designed to be a convenient tool for the USERS; and supports multiple design techniques such as PALASM like sum-of-products Boolean equations, extended Boolean logic descriptions (Parenthesized equations, DeMorgan's laws, Macro substitution capability, and Pin vectors). With its interactive or batch mode of operation; optional tri-level menu driven capability; extensive error checking and help facility; and with direct hardware programming equipment interface, it is quite user friendly and easy to use.

PLPL is the heart of a new programmable logic computer aided design environment. This environment with all its flexibility allows the designer to take full advantage of designing with programmable logic devices and allows all the tasks associated with the designing of programmable logic devices to be performed with ease and confidence.

Structured in a top-down, hierarchical fashion, in PLPL CAD environment, each program is governed by a separate program module. Nine independent modules make up the PLPL environment (fig. 1): The Operations Processor, The PLPL Compiler, PLPL Data Base Map, JEDEC Fuse Map Generator, List Equations, PLPL Simulator, PLPL Optimizer, PLPL Semi-automatic Test Vector Generator and PALASM to PLPL Translator.

THE OPERATIONS PROCESSOR

The Operations Processor is an interactive, tri-level, menu driven interface module that provides the interface for all other modules of the PLPL CAD environment such as the Compiler, JEDEC Fuse Map Generator, List Equations, Simulator, Test Vector Generator, Optimizer etc. With its extensive "help" facility, it is designed to be quite user friendly. Also, it provides an "escape" capability to temporarily exit the PLPL environment to execute certain system operations (of the particular operating systems) - such as editing or looking at the contents of other files.

THE PLPL COMPILER

The PLPL Compiler is the heart of the PLPL environment. It converts appropriate PLPL design specifications into an intermediate file form - to be accepted by other PLPL modules - such as JEDEC Fuse Map Generator, List Equations, Simulator etc. The PLPL Compiler parses the PLPL input specifications, checks for possible syntax errors and flags any possible device limitation errors. The PLPL Compiler is designed to be device independent. All the device specific information - such as its architectural features and other features are provided to it from the PLPL Data Base.

THE PLPL DATA BASE

The PLPL Data Base stores all device dependent architectural information (associated with all the programmable logic devices - that are supported by PLPL) and supplies this information to other modules who need it - such as the Compiler, JEDEC Fuse Map Generator, and the Simulator. This centralized nature of the data base, provides the flexibility to add new devices quickly to the PLPL environment. Future addition of new devices to PLPL impact only this module.

JEDEC FUSE MAP GENERATOR

This module has the responsibility for generating the appropriate fuse map pattern for down loading directly to a hardware programmer. It arranges the fuse pattern into appropriate JEDEC transfer format, supported by all suppliers of programmable logic programmers.

LIST EQUATIONS

This module takes the intermediate file (generated by the Compiler) and generates all appropriate Boolean equations.

PLPL SIMULATOR

This module performs the simulation using user created test vector tables, consisting of specified inputs and expected outputs. This simulator uses the specified test vector inputs and the outputs of the Compiler's intermediate file to model the device's output behaviour. The simulation output is compared to the expected test vector outputs, and appropriate error messages are generated, when results do not match.

PLPL OPTIMIZER

This module optimizes the list of Boolean equations generated by the PLPL Compiler. With the elimination of all redundant product terms, this results in an optimum list of equations for optimum utilization of the device.

PLPL SEMI-AUTOMATIC TEST VECTOR GENERATOR

This module generates test vectors (semi) automatically for a PLPL design specification. The test vectors generated by this module can be used by the Simulator to test the device.

PALASM TO PLPL TRANSLATOR

This module reformats PALASM specifications into appropriate transformations that is acceptable to the PLPL Compiler. This provides downward compatibility with existing equations for converting to PLPL.

PLPL PROGRAM STRUCTURE

This section describes the structure of PLPL input specification and some of the capabilities of PLPL language.

All PLPL input specifications require as a minimum the following three basic structures (fig. 2):

- o a HEADING
- o a DECLARATION BODY and
- o a PROGRAM BODY

The HEADING specifies the title, if any, of the design and the device type that is to be programmed.

The DECLARATION BODY is where pin names, macros, constants, and variables are specified. It is separated from the HEADING with a white space. In PLPL, white space is defined to blanks (spaces), tabs, newlines, formfeeds (^L), and comments.

The PROGRAM BODY contains the logic specification that is to be programmed.

Figure 3. shows the detail structure of a PLPL program and table 1. shows the formal requirements of PLPL organization. As seen from this table, every PLPL program must have a PIN declaration. Since all the elements need to be declared before they are used in the program, usually the pin declaration is the very first declaration within a program specification. The declarations are followed by the main program block, which is bracketed by the keywords BEGIN and END. (with the period).

The PROGRAM BODY section begins with the keyword BEGIN (also a statement operator) and is terminated with the statement operator END. (with the period). Any combination of Boolean equations and high level descriptions using the IF-THEN-ELSE, CASE, and FOR constructs may appear between the opening and closing statement operators. The section may be null (containing no statements), but the statement operators (BEGIN-END.) may still appear. This section may only appear once in a specification.

THE HEADING

Excluding comments and white spaces, the HEADING must be the first line of a PLPL input specification. The syntax for the heading is as follows:

DEVICE title_specifier (part_name)

The heading consists of three parts: a) the keyword DEVICE, b) an optional title_specifier and c) a part_specifier or device name. The program HEADING should always begin with the keyword DEVICE. The title_name is optional, but when specified, the title should have no spaces in it. The part_name should be a valid partname. Note that HEADING definition can span multiple lines, with comments inserted in between, if so desired.

THE DECLARATION BODY

Three types of declarations are supported in PLPL:

- o PIN - for pin assignments
- o DEFINE - for MACRO and CONSTANT definitions
- o VAR - for variable list definitions

Only the PIN declaration is mandatory in PLPL.

PIN DECLARATION

PIN declarations section allows to equate user defined identifiers to physical device pins. It begins with the keyword PIN and is terminated with a semicolon. PLPL supports two types of pin assignments: Simple and Vector. Simple pin list declarations assign user defined symbolic names to the pins within the real device, for describing the pins' actual functions. Vector pin list assignments assign a group of multiple pins to a pin vector. Whereas, the Simple pin declaration, equates a pin number (a number that represents the actual pin on the device) to a simple pin_name; the pin vector declaration concatenates the symbolic pin_vector name with the numbers inclosed in brackets, for each symbolic pin. There is a one to one mapping between the order of the numbers with in the brackets on the left hand side and the numbers specified on the right hand side. The syntax for a pin_vector assignment is as follows:

pin_name [subscript_set] = pin_set

In pin_vector assignments, colon and comma can be used together for both symbolic and actual pin numbers. For example:

DATA[7:0] = 3,4,5,6,7,8,9,10 can also be expressed as

DATA[7:5,4,3:0] = 3,4,5:10 or also as

DATA[7:0] = 3:10.

This adds great deal of flexibility, clarity, and conciseness to input specification. Instead of dealing with each pin individually, all of the pins together can be considered as a single unit. A set of pins that interfaces to a bus is a good example of where this can be used quite advantageously. This also provides a clear and powerful way to describe state machines.

DEFINE DECLARATION

This section allows two types of declarations: MACROS and CONSTANTS.

MACROS provide the capability of representing a complicated expression with a single symbolic name. Since, only the macro name and not the whole expression that it represents is used in the equation section of the program body, macros make specifications more concise. Macros also make design changes to be made easily. A single change to a macro definition is much easier to make than a change that must be repeatedly made throughout the equation section.

CONSTANT declaration, while conceptually similar to a MACRO, is restricted to the assignment of a number to a constant_name.

VAR DECLARATION

Like Macro definition section, this is an optional section, and is used to declare all the variables used in PLPL program. It begins with the keyword VAR, followed by a variable_list and is terminated with a semicolon.

THE PROGRAM BODY

PLPL is a block structured language and encourages modular programming. As seen in table 1. PLPL program has at least a declaration body followed by a main program body. The main program body is bracketed by the keywords BEGIN and END.

Blocks of statements delimited by BEGIN and END; may appear in various locations within the program. However, the last program body must be followed by a period. In PLPL, a block of statements bracketed by the statement operators BEGIN and END; is treated as a single statement.

STATEMENT STRUCTURE

Four types of statements are supported in PLPL:

- o The Assignment Statements
- o The Conditional Statements
- o The Loop Statements
- o Function Calls

ASSIGNMENT STATEMENT

An Assignment statement consists of a variable identifier followed by the assignment symbol = or := followed by a valid expression. Expressions that appear on the right hand side of an assignment operator come in two flavors: Boolean and Arithmetic.

Boolean expressions consist of identifiers that represent pins and macros and logical operators combined in the infix format (i.e., operators in between two operands). For example, assuming that the identifiers are defined appropriately

$Q_OUT[1] * /RESET * HOLD + DATA[1]$
is a valid Boolean expression.

Also, multiple assignments to the same variable are allowed in PLPL. If an identifier appears on the left hand side of more than one assignment statement in the program body, the actual "value" of this identifier will be the logical "OR" of each expression.

An Arithmetic expression consists of numbers, constants, and variables combined with arithmetic operators in the infix format. PLPL supports only integer expressions. In integer expressions, all variables, constants and results of functions must be integers.

CONDITIONAL STATEMENTS

PLPL provides two types of conditional statements - to support either binary decision capability or multi-way decision capability. These two statements are:

- o IF-THEN-ELSE statements
- o CASE statements

IF-THEN-ELSE statement is used to make binary decisions. Figure 4 and 5 show the examples of IF-THEN-ELSE statements. Since, the ELSE part is optional, the ELSE is concatenated with the closest previous ELSE-less IF.

The CASE statement provides multi-way decision making capability. It causes a branch to one of several statements depending on the value of CASE argument. This statement is quite powerful in specifying state machines. Used in state machine design, the CASE statement can merge with the IF condition so that a state diagram can be directly translated into a PLPL specification. This statement is also quite powerful for handling memory address range for address decoding functions. Figures 6, and 7 illustrates the use of CASE statement for handling memory address ranges for address decoding.

LOOP STATEMENT

For controlled looping, PLPL supports a high level FOR construct. This construct allows execution of a loop with the automatic incrementation of a counter variable. FOR loops provide a simple way to evaluate many expressions that incorporate pin vectors. This is quite powerful for compacting equation specifications. Compact yet completely understandable specifications can be created with minimal effort. Figure 8. illustrates the use of FOR statement.

FUNCTION CALLS

PLPL supports three types of FUNCTION CALLS (predefined functions):

- o ENABLE ()
- o ARESET ()
- o SET ()

The ENABLE function is used to selectively enable output pins in a device; and the arguments to the function must be output or input/output pins. Typically this function is used in conjunction with the IF statement. If a call to this function never appears in a specification, pins defined as outputs are assumed always enabled.

ARESET is a function provided specifically for the AmPAL22V10, and is used to cause an asynchronous RESET of the output registers. SSET is also a function provided specifically for the AmPAL22V10. A call to this function causes a synchronous set to occur, i.e., the output registers are set to a high state. Since, both the ARESET and SSET mechanism apply to all the outputs, all the arguments to the functions are ignored.

FUTURE TRENDS IN SOFTWARE SUPPORT

For software support of existing and future, complex programmable logic devices, most of the efforts are being spent on the development of high level language based advanced software package. Technology independent, device independent, and programmer independent software tools are becoming mandatory for wider acceptance of these devices. Complete software packages - helping the designers with with all the aspects of the design processes are beginning to emerge. Also, with the development of advanced software packages, the architecture of these devices will be made more transparent to the designers so that they will spend their time in solving the problems rather than incremental optimal utilization of the internal resources of the device.

With the availability of these advanced software package such as PLPL, the tasks of the designers will be simplified. Instead of specifying the complicated designs by Boolean equations only, the designers will be able to specify them directly on state diagram level using High Level Language (HLL) type specifications, directly on engineering work stations. Software tools will help designer in specifying the problem, simulation of design verification, test vector generation, and post programming verification. Advances in programmable logic software will be integrated with capturing the schematic entry, direct state diagram level representation, state minimization, netlist generation, specification of design at high level language, automatic simulation, verification and downloading into a programmer. The improved design methodology will allow the system designers to exploit their system systems knowledge without having to learn the internal details of particular ICs. It will result in better use of engineering talents and bringing down the cost of designing a product.

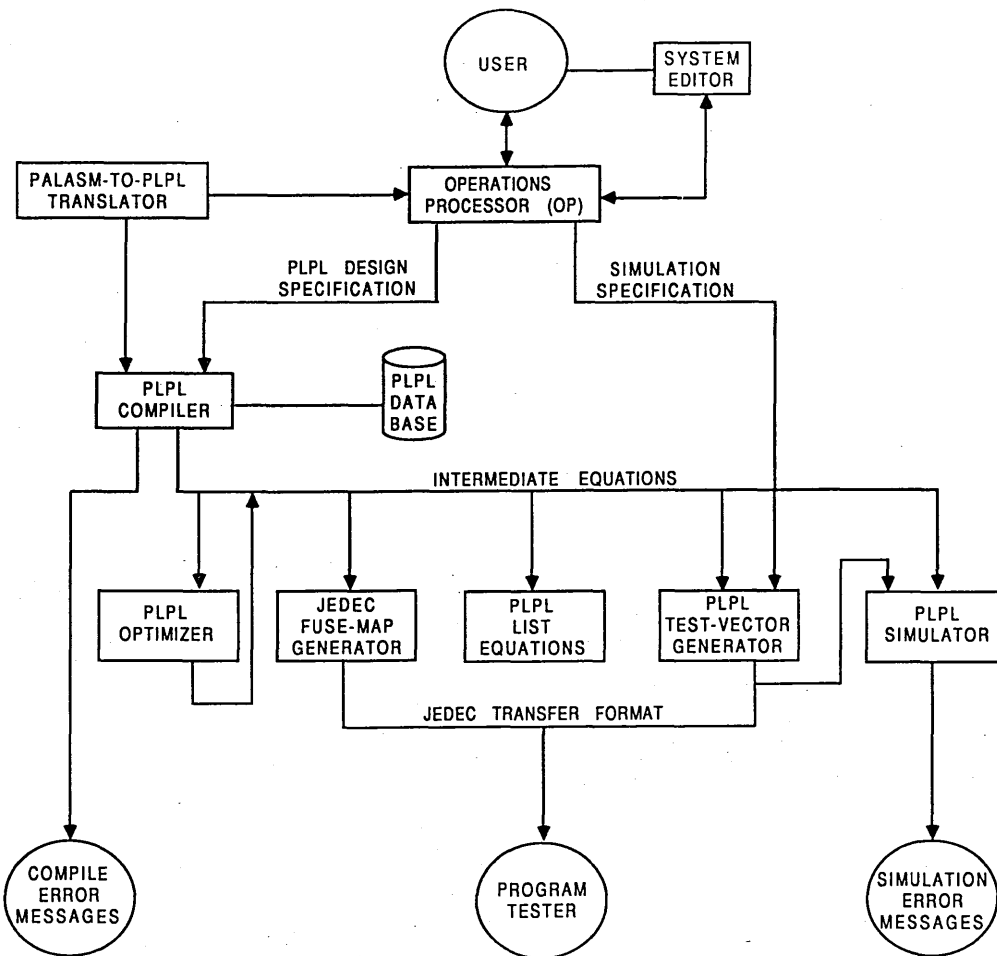


FIGURE 2 PLPL PROGRAM STRUCTURE

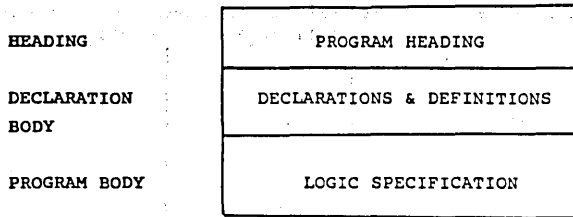


FIGURE 3 PLPL PROGRAM STRUCTURE

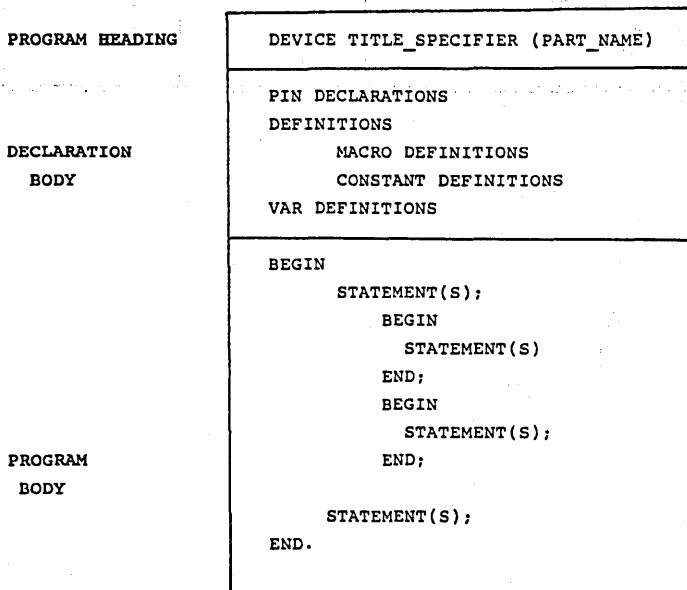


TABLE 1 FORMAT OF PLPL PROGRAM ORGANIZATION

	<u>MANDATORY</u>	<u>OPTIONAL</u>
<u>Program Heading</u>	x	
DEVICE	x	
TITLE_SPECIFIER		x
(PART_NAME)	x	
<u>Declaration Body</u>	x	
PIN LIST declarations	x	
PIN VECTOR declarations		x
MACRO definitions		x
CONSTANT definitions		x
VARIABLE definitions	x	
<u>PROGRAM Body</u>	x	
BEGIN	x	
Statement		x
Statements		x
END.	x	

Figure 4

Examples of valid IF statements

Example 1

```
BEGIN
  IF
    (
      D
      *
    /E
    )
  THEN
    C = A * B
END.
```

"Example of IF statement spanning multiple lines"

Example 2

```
BEGIN
  IF (a) THEN
    IF (b) THEN e = c;
    ELSE e = d;
END.
```

Example 3

```
BEGIN
  IF (a) THEN
    BEGIN
      IF (b) THEN e = c ;
      ELSE e = d ;
    END;
END.
```

Figure 5

```
BEGIN
  IF (a) THEN
    BEGIN
      IF (b) THEN
        BEGIN
          f = c ;
        END;
      ELSE
        BEGIN
          f = d ;
        END;
      END;
    ELSE
      BEGIN
        f = e;
      END;
    END.
```


Figure 6

Example for CASE statement

"The following two examples illustrate the use of CASE statement for handling memory address range for address decoding function. The first example illustrates decoding in a normal fashion. "

DEVICE test (Ampall6L8)

PIN

```

A15 = 1
A14 = 2
A13 = 3
A12 = 4
A11 = 5
/RESET = 6
/MREQ = 7

/MEM1 = 19
/MEM2 = 18
/MEM3 = 17
/MEM4 = 16
/MEM5 = 15
/MEM6 = 14
/MEM7 = 13
/MEM8 = 12;

```

DEFINE

```
MEMOP = /RESET * MREQ;
```

BEGIN

```

"MEM1 is active from 0000H to 0FFFH"
MEM1 = MEMOP * /A15 * /A14 * /A13 * /A12;

"MEM2 is active from 1000H to 1FFFH"
MEM2 = MEMOP * /A15 * /A14 * /A13 * A12;

"MEM3 is active from 2000H to 27FFH"
MEM3 = MEMOP * /A15 * /A14 * A13 * /A12 * /A11;

"MEM4 is active from 2800H to 2FFFH"
MEM4 = MEMOP * /A15 * /A14 * A13 * /A12 * A11;

"MEM5 is active from 3000H to 37FFH"
MEM5 = MEMOP * /A15 * /A14 * A13 * A12 * /A11;

"MEM6 is active from 3800H to 3FFFH"
MEM6 = MEMOP * /A15 * /A14 * A13 * A12 * A11;

"MEM7 is active from 4000H to 5FFFH"
MEM1 = MEMOP * /A15 * A14 * /A13 ;

"MEM8 is active from 6000H to 7FFFH"
MEM8 = MEMOP * /A15 * A14 * A13 ;

```

END.

Figure 7

"the same example with the use of CASE statement"

```

DEVICE test (Ampall6L8)

PIN
    A[15:11] = 1:5    "Doing the decoding just looking at bits
                      A15 - All only"

    /RESET = 6
    /MREQ  = 7
    /MEM[1:8] = 19:12;

DEFINE
    MEMOP = /RESET * MREQ;

BEGIN
    CASE (A[15:11])
    BEGIN
        "MEM1 is active from 0000H to 0FFFH"
        #B00000, #B00001) MEM[1] = MEMOP ;

        "MEM2 is active from 1000H to 1FFFH"
        #B00010, #B00011) MEM[2] = MEMOP ;

        "MEM3 is active from 2000H to 27FFH"
        #B00100) MEM[3] = MEMOP ;

        "MEM4 is active from 2800H to 2FFFH"
        #B00101) MEM[4] = MEMOP ;

        "MEM5 is active from 3000H to 37FFH"
        #B00110) MEM[5] = MEMOP ;

        "MEM6 is active from 3800H to 3FFFH"
        #B00111) MEM[6] = MEMOP ;

        "MEM7 is active from 4000H to 5FFFH"
        #B01000, #B01001, #B01010, #B01011) MEM[7] = MEMOP ;

        "MEM8 is active from 6000H to 7FFFH"
        #B01100, #B01101, #B01110, #B01111) MEM[8] = MEMOP ;

    END;
END.

```

Figure 8

Example for CASE statement

"the following two examples illustrate the use of CASE statements for handling address ranges for address decoding "

DEVICE decoder (Ampall16L8)

```
PIN      ENAB = 4
        D[0:7] = 12:19;

BEGIN
    CASE (ENAB)
    BEGIN
        0) /D[0:7] = #B11111111;
        1) /D[0:7] = D[0:7];
    END;
END.
```

"The same example treated different way"

DEVICE decoder (Ampall16L8)

```
PIN      ENAB = 4
        SEL [1:3] = 1:3
        D[0:7] = 12:19;

BEGIN
    CASE (ENAB)
    BEGIN
        0) /D[0:7] = #B11111111;
        1) CASE (SEL[3:1])
            BEGIN
                0) /D[0] = 1;
                1) /D[1] = 1;
                2) /D[2] = 1;
                3) /D[3] = 1;
                4) /D[4] = 1;
                5) /D[5] = 1;
                6) /D[6] = 1;
                7) /D[7] = 1;
            END;
    END;
END.
```

Figure 9

Some Examples of FOR statements

Example 1

```
BEGIN

    FOR (X = 0 TO 7)
        BEGIN
            Q[X] = D[X]
        END;
    END.
```

Example 2

```
BEGIN

    FOR (X = 1 TO 8)
        BEGIN
            Y = X PLUS 1 ;
            Z = X MINUS 1;
            Q[X] := LOAD*D[X] + HOLD*Q[X] + SHFTR*Q[Y] + SHFTL*Q[Z];
        END;
    END.
```

Example 3

```
BEGIN

    PIN_NUM_PLUS_1 = PIN_NUM PLUS 1;
    PIN_NUM_MINUS_1 = PIN_NUM MINUS 1;

    FOR (PIN_NUM = 1 TO 8)
        Q_OUT [PIN_NUM] := LOAD * DATA [PIN_NUM]      +
                           SHFTR * Q_OUT [PIN_NUM_PLUS_1] +
                           SHFTL * Q_OUT [PIN_NUM_MINUS_1] +
                           HOLD * Q_OUT [PIN_NUM];
    END.
```

3.7.9 Article Reprint

Test Methods for Programmable Logic

by Jenny Yee

Product Planning Applications Engineer
Advanced Micro Devices, Inc
Sunnyvale, CA 94088

Introduction

The purpose of this paper is to investigate the methods used to test programmable logic devices. Because these devices are user-programmable, they offer the user/designer the flexibility of defining the function of a device for a particular application. After the function has been defined the designer needs to verify that the function will work correctly in a programmed device. He needs a simple method that allows him to easily generate test patterns which reflect the functions he has defined. After function definition and verification, the design is implemented by programming a device. Since large quantities of devices may be programmed using the same design, testing is again called for, in order to insure that the devices are fully functional before they are integrated into systems. Because the objectives and methods of testing during design and programming are different, it is reasonable to examine them separately.

The Design Process: Simulation

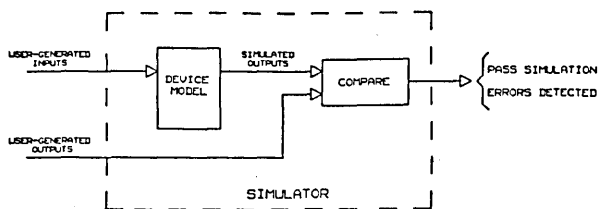
The typical design process tends to be trial-and-error. The first attempt at a design usually gets modified. Because programmable logic devices can be quickly changed --- the old device can be pulled out, a new one programmed in seconds and plugged back into the system - they allow the designer a fast design cycle time.

The necessity to verify that a function will work correctly before it is

actually implemented in a programmable logic device is analogous to the need to verify that a logic design using standard TTL devices is correct before the design is built. The method used in logic design to perform such verifications is logic simulation. This allows the designer to define a model of the logic structure on a system, subject the model to test patterns and use the outputs to determine whether the logic is correct.

Simulation can also provide the necessary verifications for designers using programmable logic devices. Although the process is slightly different than TTL logic simulation, its purpose is the same.

Figure 1 shows the basic flow of programmable logic device simulation. The simulator accepts a test pattern consisting of inputs (both device inputs and internal current state information) and outputs generated by the designer. The designer should create the patterns by specifying the input and output states expected during actual operation of the device. This is the best way to verify that the design is correct and the programmed devices will work properly the first time. Together with the device model, which is itself implicit in the simulator, the simulator generates outputs. It then makes a comparison between the outputs specified by the designer and those it generates and flags any discrepancies. The designer determines if the error occurred in the function definition or in the test pattern and makes the necessary modifications.



BASIC PROCESS OF SIMULATION

FIGURE 1.

Simulation can be recommended for programmable logic device verification not only because it can provide the necessary evaluations for an exact specification of the device, but because the required specification of test patterns (inputs/outputs), or test vectors, can be easily generated and modified by the designer. The manner in which the designer presents the patterns is explicit. Figure 2 shows a test pattern for a decoder in the PALASM function table format. (Note the forced inputs on the left and expected outputs on the right.) For this simple example the table is easily generated, but for more complex designs it may become cumbersome.

Both combinatorial and sequential designs need to be verified using simulation. Sequential devices have registers and internal feedback and combinatorial devices do not. Verification of combinatorial devices is a function of input conditions only, whereas sequential devices require a method by which registers can be loaded to some current state so next state transitions can be sequenced. Figure 3 shows a test pattern for a shift register using the same PALASM* format. It lacks current state definition

* PALASM is a trademark of and is used under license from Monolithic Memories, Inc."

capability. The table must have register initialization in the first vector. All succeeding vectors then use the previous vector output as a current state. Advanced Micro Devices offers a hardware capability in its sequential programmable logic devices called Preload, which allows the user to initialize registers to obtain a current state. This additional capability makes test pattern generation easier, but requires test support. The example in the following section will show how test patterns can be modified to utilize the hardware capability.

Preload Vectors

Advanced Micro Devices offers an upgraded version of PALASM called AmPALASM20 that allows a user to specify a current state of the device, via special Preload Vectors. A preload vector contains the letter P in the normal clock position of the function table and the desired current state in the output position. A vector in the same form as Figure 3 that preloads a current state of all LOWs, or ZEROs, is shown below:

P X X X X X X X X L L L L

FUNCTION TABLE

```

***Forced Inputs***** Expected Outputs***
S2 S1 S0 D7 D6 D5 D4 D3 D2 D1 D0
-----
L L L L L L L L L L H
L L H L L L L L L L L
L H L L L L L L H L L
L H H L L L L H L L L
H L L L L L H L L L L
H L H L L H L L L L L
H H L L H L L L L L L
H H H H L L L L L L L
-----
DESCRIPTION
THIS DEVICE IMPEMENTS A 3 TO 8 DECODER. THE
SAMPLE SHOWS THE DESIGN OF THE DECODER USING
PAL.

```

FIGURE 2

FUNCTION TABLE

; ****Forced Inputs*****								*Input/Output*		*Expected Outputs*				
CK	S1	S0	D3	D2	D1	D0	OE	SRISLO	SLISRO		Q3	Q2	Q1	Q0

;														
;LOAD AND SHIFT RIGHT														
C	L	L	L	L	L	L	L	Z	Z		L	L	L	L
C	H	H	X	X	X	X	L	Z	Z		L	L	L	L
C	L	H	X	X	X	X	L	H	L		H	L	L	L
C	L	H	X	X	X	X	L	L	L		L	H	L	L
C	L	H	X	X	X	X	L	L	H		L	L	H	L
C	L	H	X	X	X	X	L	L	L		L	L	L	L
;														
;LOAD AND SHIFT LEFT														
C	L	L	H	H	H	H	L	Z	Z		H	H	H	H
C	H	H	X	X	X	X	L	Z	Z		H	H	H	H
C	H	L	X	X	X	X	L	H	L		H	H	H	L
C	H	L	X	X	X	X	L	H	H		H	H	L	H
C	H	L	X	X	X	X	L	L	H		H	L	H	H
C	H	L	X	X	X	X	L	H	H		H	H	H	H
;														
;HOLD														
C	H	H	X	X	X	X	L	Z	Z		H	H	H	H

DESCRIPTION														
THIS DEVICE IMPLEMENTS A SHIFT REGISTER. THE LAYOUT PROVIDED														
IS A DEMONSTRATION OF HOW THE SHIFT REGISTER MAY BE DESIGNED														
USING A PAL.														

FIGURE 3

An Alternative Approach to Simulation

Although function tables are widely used for input/output specification, some may find them to be too rigid, tedious and cumbersome. There are methods to improve the format of presentation and manner of description that make the specifications much easier to generate and modify. The following shift register example will suggest a more powerful alternative to the PALASM function table format presented in Figures 2 and 3.

Beginning with the listing of the names of signals in the device, a suggested declaration list might appear as follows:

```
IN      CLK SEL[1:0] DATA[3:0] /OE;
I_O     SRILO SLIRO;
CS      Q[3:0];
NS      ;
OUT     Q[3:0];

VAR      a b c d i;
CONST    LOAD = 0
          SHIFT_RT = 1
          SHIFT_LT = 2
          HOLD = 3;
```

This list suggests that all signals of the device be identified in fields of their respective functions. The signal names are specified in fields as well. This procedure helps clarify the function being simulated and also defines the structure of the function table itself.

In this example, there are five fields to declare device information - INPUTS(IN), INPUTS/OUTPUTS(I_O), CURRENT STATE(CS), NEXT STATE(NS), and OUTPUTS(OUT) -- but only four are actually used. The number of fields used in a function table depends on the design being simulated. For example, a combinatorial design may only require the IN and OUT fields.

The first declaration identifies the inputs(IN) followed by the list of signals: clock (CLK), select0 and select1 (SEL[1:0]), data0, data1, data2, data3 (DATA[3:0]), output enable (/OE). It reserves a one-field bit for the clock, two bits for select, four for data, and one for output enable. Note that for the case of select and data, signals are concatenated into a field to simplify the identification of the vector list.

The second and third declarations list I/O and current state(CS) entries, respectively. The I/O signals, SRILO (shift right input/shift left output) and SLIRO (shift left input/shift right output), are used to specify the state of the input or output bit being shifted. Current state is used to initialize the state of the registers. The fourth declaration identifies the next state (NS) entry. It is usually used in conjunction with current state(CS) to specify sequential state machine transitions. (It is unused in this example and actually does not need to be shown.) The fifth declaration identifies the output(OUT) pins (Q[3:0]).

variables and constants are available and must be declared prior to use in the test pattern. They make the test pattern easier to understand and more convenient for the designer to generate. Variables allow the designer to generate a single vector that may be used iteratively by simply changing the value of the variables. Constants allow meaningful names to be substituted for less descriptive numeric values for bits and fields.

Figure 5 shows a complete function table for the shift register example. A declaration list with variables and constants is included. The test pattern is represented between the BEGIN and END keywords. A comment field (enclosed in double quotes) above the test vectors

Shift Register Example

Simulate

```

IN      CLK SEL[1:0] DATA[3:0] /OE ;
I_O     SRILO SLIRO;
CS      Q[3:0];
OUT     Q[3:0];

VAR      a b c d i;
CONST    LOAD = 0
          SHIFT_RT = 1
          SHIFT_LT = 2
          HOLD = 3;

BEGIN

*****IN***** :***** I_O*****:**CS** :**OUT**
" CLK SEL[1:0] DATA[3:0] /OE : SRILO SLIRO : Q[3:0] : Q[3:0]"
"
X      X      X      H : X      X : X      : Z ;
C      LOAD    abcd    L : X      X : X      : abcd ;
C      SHIFT_RT X      L : i      d : abcd   : iabc ;
C      SHIFT_LT X      L : a      i : abcd   : bcdi ;
C      HOLD    X      L : X      X : abcd   : abcd ;

END.

"Reserved Characters
X = DON'T CARE      C = CLOCK      P = PRELOAD      H = HIGH
Z = TRI-STATE       "COMMENTS"     - = NOT APPLICABLE L = LOW"

```

FIGURE 4

specifies that the fields and signal inputs are in the same order as previously specified. The IN field is first, followed by I_O, CS, and OUT, as well as CLK preceding SEL[1:0], etc.

The first vector of the function table tests the device's ability to generate tristate outputs. The output enable (/OE) is active LOW; forcing it HIGH (H) causes the outputs (Q[3:0]) to be made tristate (Z). The rest of the signals in the vector are not required and are set to don't cares (X). The second vector makes use of variables and constants to test a load function. The

constant LOAD in the select field is used to clarify what function is chosen. The data input field contains the variable string (abcd), which is loaded into the outputs following the clock pulse. The clock (CLK) entry in the table has the letter C to signify a positive-edge-triggered clock pulse that causes the next state of the device to be entered. The third vector uses constants and vectors to test the shift right function. The data field is not required and is set to don't cares. Following the clock transition the outputs will have the result of the shift. The variable string (abcd) will

have the variable (i) shifted in from the right, and the variable (d) will be shifted out. The succeeding vectors in the pattern may be understood in the same way as those above. The result is that only five vectors specify a test of the shift register, as contrasted to twelve for only a partial definition in the PALASM format.

The Production Process: Post-Programming Function Verification

After a design is successfully verified it can be implemented on a device by programming the device's internal fuse array. A few or a great many devices may be programmed depending on the application of the design. Independent of the volume of devices, post-programming verification is necessary to insure that the devices are fully functional devices before they are integrated into systems.

Post-programming verification tests the functional capability of the programmed device using test vectors. These vectors are similar to the test patterns used in simulation, but more of them are needed. Cost and test time increase linearly with the number of vectors being tested. If the vectors are not readily available or complete, they must be generated by the user or a computer. For a user the task of generating the number of vectors needed for an extensive post-programming test can be tedious and error-prone. Post-programming vectors generated by a computer are more complete and accurate.

Fuse Programming and Verification

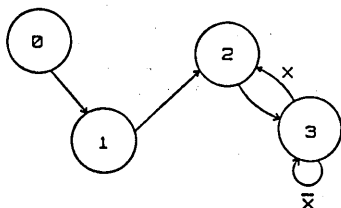
The fuse array is the internal element that defines the logic structure of the device upon programming. When the user obtains the device from the manufacturer all the fuses in the array are intact. After programming, the array is a

combination of intact and blown fuses. Depending on the selection of fuses that are required to implement the desired function. The programming of fuses is done by blowing one fuse at a time. As they are blown, the fuses are individually checked by the programmer for correctness. As soon as programming is completed, the user may also perform a global fuse array verification using a programmer. Fuse array verify checks the fuse pattern on the device against the test pattern stored in the programmer's memory. The success of this verification partially indicates that the device is functional.

Preload

Preload is a hardware feature that has been added to the register circuitry of all AMD's registered programmable logic devices. This feature simplifies testing since current state of the registers can be set to a known value just like any other input. Thus the preload capability reduces the task of sequential testing to the much easier combinatorial test problem.

Figure 5 shows a simple example of state testing. Without preload it would be difficult to access state 3 without sequencing through states 0, 1, and 2. Furthermore, this process would be iterative for each test of state 3. With preload, state 3 can be entered directly and tested. Thus test time is saved because less vectors are required. Power-up initialization is difficult to test. Typically, a state machine has additional don't care states that only affect power-up. Should any of these don't cares be entered on power-up the machine must be able to exit them to start normal operation. Preload allows these states to be entered and tested to see if they can be exited. Preload is invaluable in the test methods examined in the next section.



TESTING OF STATES
FIGURE 5.

Function Testing

The different methods used for post-programming functional testing differ in how the input and output test vectors are generated and applied, and in how many vectors are needed. Functional testing methods for programmable logic devices fall into the categories of User-generated, Exhaustive, Pseudorandom and Path Sensitization.

Of the four methods, user-generated requires the most amount of knowledge about the actual design. Someone, usually the designer, must generate the input and output test vectors. This method is very attractive if the vectors were already created for simulation or some other previous test. The level of testing can be tailored, but if a complete functional test is needed or desired, vectors required may be prohibitive. The simulation vectors of Figure 4 can be used for a complete test if the designer writes software that expands the number of vectors by using all the possible combinations of the variables and don't care inputs, and if the Preload facility is available.

This method of testing is inadequate if the vectors are for some reason not readily available - for example, if pre-programming testing is done out of house.

With exhaustive technique the designer attempts to test all possible input and current state combinations against all possible expected outputs. Input and current state may be created simply by sequencing through the possibilities. The Preload facility is needed to generate the current states. The expected outputs either need to be calculated from the design specification with software, or some signature analysis technique on the output results must be employed.

With this scheme 100% functional testing can be accomplished. An additional advantage is that if signature analysis is employed no vectors need to be calculated. The disadvantage of this methods is that the worst-case number of vectors must be applied: 20-pin PAL* requires 2^{17} or 128,000 vectors. While 128,000 vectors might be applied economically, as devices get more complex, the number of vectors may become unmanageable.

The Pseudorandom testing technique performs a device test by generating random test inputs and utilizing a signature analysis on the output results. The advantages of this technique is that no vectors need to be calculated and the level of testing can be controlled by the number of vectors applied. This type of testing is useful for combinatorial and simple sequential designs, but is of little use on complex sequential designs since with random inputs it is not possible to apply the proper sequence of inputs to cause the proper state transitions. Because of its random nature this technique can't

* PAL is a registered trademark of and is used under license from Monolithic Memories, Inc."

	User-Generated	Exhaustive	Pseudo-random	Path Sensitization
Method of Input Vector Generation	User	Hardware	Hardware	Hardware (Software)
Output Vector Generation	User	Automatic (Software)/Sig. Analysis	Signature Analysis	Automatic (Software)
# of Vectors Required	Application dependent (expandable with software)	Worst-case	Selectable	Minimal
Test Vector Software Req'd	Maybe	Maybe	No	Yes
Preload Capability Req'd	Yes	Yes	No	Yes

Table 1

make use of the important Preload capability which makes sequential testing easy when used with other techniques.

Path Sensitization testing, also known as stuck-at-one/stuck-at-zero testing, has previously been used only in large test programs such as TEGAS. This technique attempts to isolate certain paths of a device and apply tests that assure that each path is functional. In the case of PALs, the paths tested are in the AND array matrix. Each gate must be isolated and tested for stuck-at-one and stuck-at-zero.

The advantages of this technique are that 100% functional testing can be accomplished with few vectors, and Preload can be employed to make sequential testing simple. The disadvantage is that complex software is necessary to generate the test vectors. As yet, no readily available path

sensitization software has been shown to generate a list of vectors that guarantees 100% functional testing.

A summary of functional test methods appears in Table 1.

Conclusion

This paper has attempted to present test methods for the designer to use during the different stages of development of programmable logic devices. Simulation was recommended for verification during the design process, and its requirements and benefits were discussed. Several alternate methods were offered for verification of device functionality following programming and verification of programming. The ability to preload was shown to simplify sequential device verification; its requirements were also discussed.

DESIGN ENTRY

Fuse-programmable chip takes command of distributed systems

The first fuse-programmable controller eliminates bulky and expensive designs, freeing distributed intelligence to carve out a greater niche for itself.

In much the same way that cars and highways spawned the suburbs, standard microprocessor buses and add-on boards have distributed processing intelligence, revolutionizing the design of digital systems. Breaking systems into independent modules shortens the design cycle, eases upgrades, and accelerates fault diagnosis. But despite these advantages, an essential element has been missing: a one-chip controller geared specifically to the needs of distributed intelligence.

Without that critical ingredient, engineers have been forced to turn to less than optimal solutions. One approach relies on boards packed with as many as 35 SSI and MSI devices. Another tack is to go with a powerful—yet costly—VLSI chip. Alternatively, a programmable logic device can be pressed into service, but such circuitry lacks the computing power to control peripherals.

The missing element, a fuse-programmable controller chip, is now here. By mixing intelligence and control, the Am29PL141 stakes out new territory for distributed systems. The 20-MHz IC combines for the first time all of the elements of an intelligent micro-code controller. Its powerful sequencing logic steps through the controller's 64-by-32-bit pipelined PROM. That fuse-programmable memory stores a user-defined microprogram drawn from a set of 29

microinstructions, including a repertoire of jumps, multiple branches (or case statements), and subroutine calls. All can be executed conditionally, depending upon the outcome of one of eight tests. In addition, a serial shadow register on the 28-pin chip helps designers diagnose system troubles right down to a particular IC. (In the past, expediency often dictated that complex trouble-shooting be avoided for as long as possible.)

Four basic blocks

The controller comprises four main functional blocks. Three of them—the microaddress control logic, condition code selector, and microinstruction decoder—form the cornerstone of the controller, the address sequencer. The fourth is a microprogram memory (64 by 32 bits) with a pipelined register and serial shadow register (Fig. 1).

For the most part, the elements of the address sequencer are fairly typical. Nevertheless, the way in which they are organized and connected, as well as the instruction set, make the chip unique. For example, the microaddress control portion of the sequencer contains one register for counting loops and another for stacking subroutine return addresses. Yet either register can be employed to double the capacity of the other. Consequently, the chip can nest two levels of loop counting or two levels of subroutine branching (the instruction set reflects those abilities). And the high degree of interaction between elements, particularly within the microaddress control logic, makes necessary a highly sophisticated micro-

Om Agrawal and Deepak Mithani
Advanced Micro Devices Inc., 901 Thompson Pl.,
P.O. Box 3453, MS 47, Sunnyvale, CA 94088;
(408) 749-2903.

Reprinted with permission from *Electronic Design*, Vol.33, No.24,
Copyright Hayden Publishing Co., Inc., 1985.

Electronic Design • October 17, 1985

Fuse-programmable controller

instruction set.

The microaddress control logic is the brain of the address sequencer, since it generates the addresses that access the microinstructions. At any time, the address that is called depends on the preceding instruction and the outcome of any conditional tests.

Within the control logic, a program-counter multiplexer supplies the PROM's 6-bit address. The multiplexer takes the address from a microprogram counter, incremented program counter, branch control logic, or subroutine register. Because the program counter contains the address of the currently executing instruction, that instruction is executed again when the program counter is selected as the address source. As a result, the counter plays a fundamental role in tallying loops and executing "wait until true" instructions.

The incrementer holds the next address in the sequence, and is the expected source when no jumps are executed and no branch or subroutine conditions exist. When conditional statements like "if... then... else" and multiple branches pass the required tests, or when unconditional jumps are executed, the branch control logic supplies the address. Finally, when the program calls a subroutine, the subroutine register supplies the necessary address.

A multiplexer selects one of three address sources. If only one stack level is needed, the value stored in the subroutine register is chosen. When the count register feeds the subroutine register, however, it furnishes an additional stack level. The third source, the incrementer, supplies the subroutine's return addresses.

Doing double duty

If not needed for a second subroutine level, the count register can, among other functions, execute iterative loops and time external events. To accomplish the former, the controller loads the register with the number of iterations to be run. Each iteration decrements the register until it reaches zero. The zero-detection logic associated with the counter informs the chip's microinstruction decoding logic when the register "bottoms out."

Using the same logic, an instruction can be repeated a set number of times. Repeated executions of the same instruction is a simple way to insert wait states and, therefore, build an interface to different microprocessors and peripherals.

The count register is loaded from any of four sources: a decrements, for normal loops; an instruc-

tion field; the subroutine register; and the branch-control logic. The last derives a 6-bit value from a data field in a microinstruction.

The branch-control logic, a powerful block within the sequencer, calculates the 6-bit value either by applying the microinstruction data field directly or by using it to mask the chip's six test inputs, T_0 through T_5 . In the second case, the masked input actually becomes the branch address. Moreover, either the data field or masked test value serves as both a branch address and a count value.

The same control logic also compares the masked test inputs to a constant in a microinstruction field. The outcome of this check affects a flip-flop. The latter's condition itself becomes a factor in deciding conditional branch and subroutine instructions. If a match occurs, the flip-flop is set. Alternatively, the flip-flop remains unchanged if there is no match. Because the flip-flop does not change when there is no match, it is particularly useful for comparing ASCII characters and other 6-bit fields, as well as for successively checking the chip's test inputs.

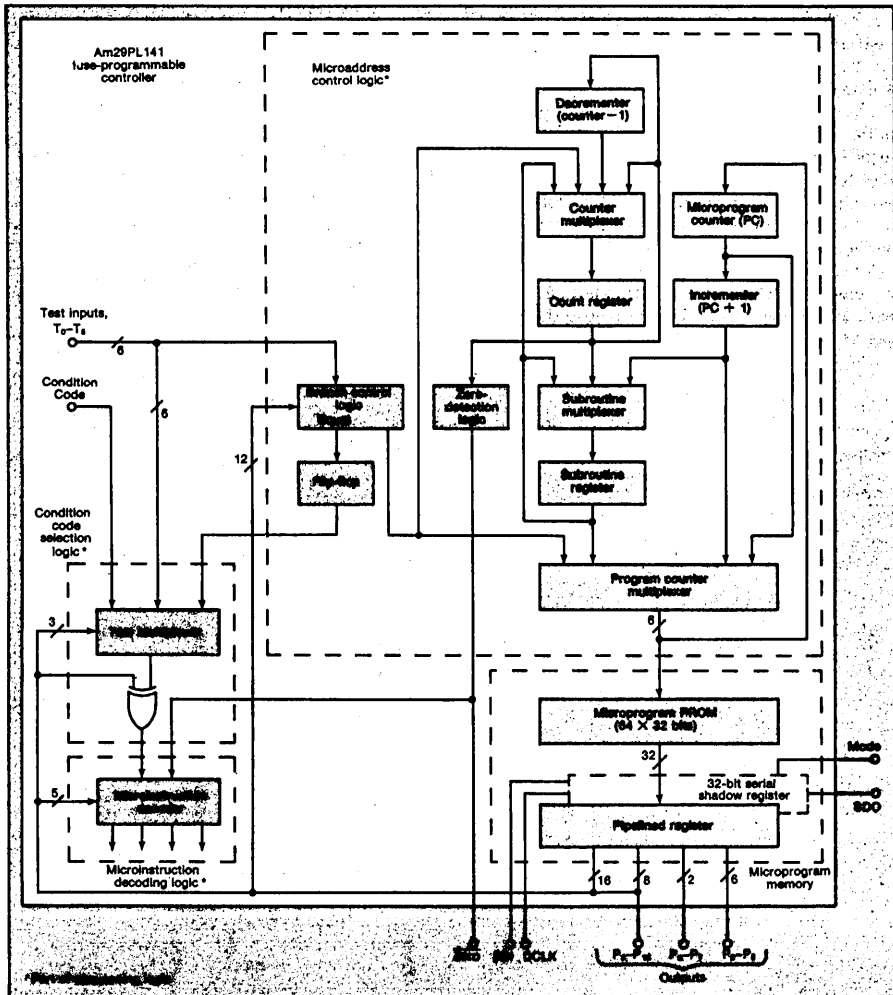
Controlling conditions

A set flip-flop is one of the eight aforementioned tests that fulfills a conditional branch or subroutine. Through its condition-code selection logic, the controller is able to check each of its six test input lines, as well as the Condition Code input. Further, an exclusive-OR gate within the selection logic switches the meaning, or interpretation, of a test result. In other words, with no external hardware, a test condition can be asserted either when a match occurs or when one does not.

The final component of the chip's sequencer section is the microinstruction decoder. That programmable logic array generates the IC's internal control signals based on the microinstruction being executed and the test results reported by the condition-code logic.

The IC's fourth functional block comprises the fuse-programmable microprogram memory, pipelined register, and serial shadow register. The pipeline register is 32 bits wide, and stores the microinstruction being run. The next address is calculated by the sequencer and its contents is fetched from the microprogram memory. The upper 16 bits of the pipeline's output remain within the chip to sequence addresses and control internal functions.

Only the 16 low-order bits link to the outside, as user-defined control lines. Of these, the upper byte is



1. The 20-MHz Am29PL141 is the first complete microprogrammable controller chip, making it an important building block for distributed processing systems. Its powerful sequencing logic steps the controller through its pipelined PROM. The fuse-programmable memory is 64 by 32 bits.

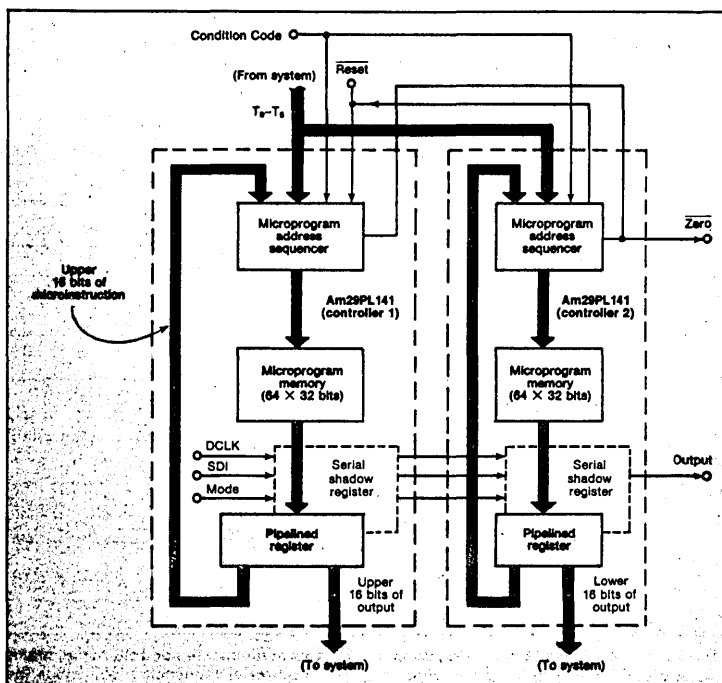
put in the high-impedance state by setting a microinstruction's Output Enable bit to 0. Moreover, chips can be cascaded readily if more than 16 control bits are needed (Fig. 2).

To check out the chip, an instruction is shifted serially into the shadow register and then loaded in parallel into the pipeline. Doing so forces the instruction to be executed, and its results transferred back

The shadow fuse

A separate fuse must be blown to set up the serial shadow register. When that is done, four pins are redefined to handle diagnostics. Specifically, the Condition Code and Zero lines and Output Data Bits 6 and 7 become, respectively, the Serial Data In (SDI), Serial Data Out (SDO), Diagnostic Clock (DCLK), and Mode control lines.

The strength of any controller—and the advantage



2. When an application calls for more than 16 control bits, two or more chips can be cascaded horizontally. The lower 16 bits of each of the chip's 32-bit microinstructions (of which there are 29) serve to control a system's components. Eight of these 16 control bits can be put into a high-impedance state under microinstruction control; the other 8 bits are always enabled.

DESIGN ENTRY

Fuse-programmable controller

of a microprogrammed system that employs it—lies in an engineer's ability to specify the sequence in which microinstructions are executed. To ensure that ability, the controller executes all the basic high-level constructs required for structured microprogramming. Its 29 op codes include sequential instructions, conditional instructions, dual branching forks, and multibranching case statements. Iterative executions, like For, While, When, and Until, round out the set. In addition, Jump, Jump to Subroutine, Loop, and Compare instructions allow designers to store very complex algorithms in the chip's 64-word memory.

Instruction formats fall into two categories. The first is for general microinstructions; the second is for the chip's Compare instructions. The latter compare a 6-bit test input to a masked constant. The Compare instructions are well-suited for character searches, as well as key searches in a look-up table.

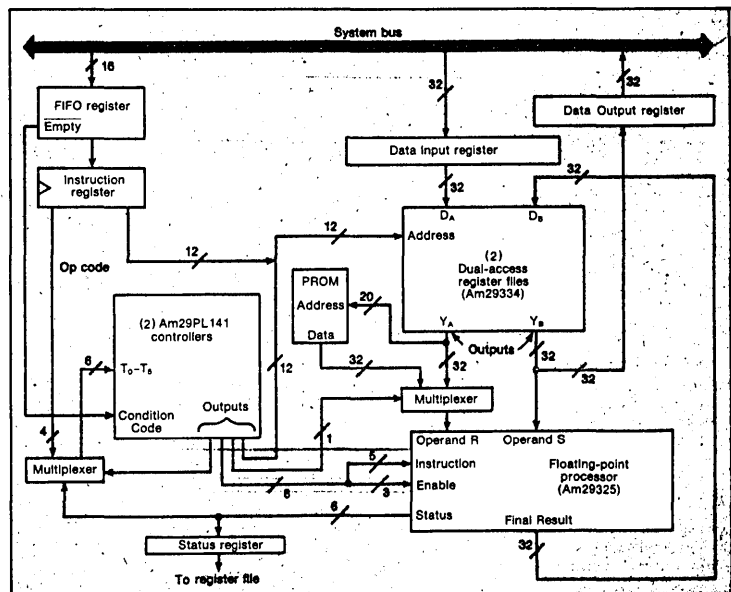
A single-precision, floating-point peripheral board

(Fig. 3) presents a good example of the part that the controller plays in a distributed system. As a microprogrammed design, the peripheral serves as an add-on math accelerator card that plays with different hosts and buses. The controller orchestrates the actions of the floating-point processor, and various registers, register files, and memory chips.

Simple arithmetic

The processor is simple to use, partly because it incurs no pipeline delays. It conforms to IEEE and other industry standards, and takes only a single clock cycle to add, subtract, or multiply. It needs five cycles to divide, using the Newton-Raphson method that inverts one of the factors and multiplies. In operation, to divide X by Y the chip fetches the approximate inverse of Y from a PROM-based table and multiplies it by X. One or two iterations of this method increase the initial accuracy.

The floating-point board works with a microword



3. In a typical application, the controller oversees the workings of a floating-point processor board. Host instructions sent to the FIFO and instruction registers initiate subroutines in the controller that generate the signals that run the board. Two controllers are employed to supply the necessary number of control signals.

DESIGN ENTRY

Fuse-programmable controller

of at least 25 bits, 9 more than available with one controller. Thus the design employs two controller chips. The floating-point processor requires five command bits. Three are instructions and two select the input source. It also needs three control bits to enable its trio of data registers. Two other chips (each a dual-access four-port register of 64 words by 18 bits) temporarily store commands. They accept 12 register address bits (6 for source and 6 for destination) from the host or the controller's microprogram memory.

Data passes to and from the host through input and output registers on the board, which call for their own enable signals. Another bit is needed to advance the FIFO instruction register. One is necessary to enable and another to select a status word. (The floating-point chip supplies status information, which is available to the controller through its test inputs as well as to the host through the register file.) Seven control bits are left for miscellaneous tasks.

Operation begins when at least one 16-bit instruction is loaded from the host into the peripheral's FIFO register. The instruction consists of a 4-bit op code, a 6-bit source-register address, and a 6-bit address for the second source register, which also stores

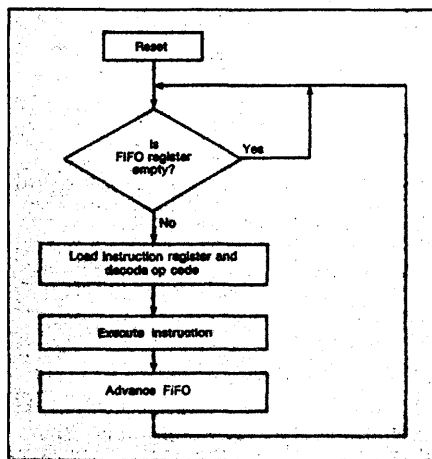
the results. Until an instruction is received, the controller is in the wait state (Fig. 4). When an instruction arrives, however, the FIFO's Empty signal activates the controller, which then reads the command from the FIFO into a separate instruction register. The controller also loads the instruction's op code into its test inputs. It then masks the two unused test bits and jumps to a subroutine that performs the operation specified by the op code. After completing it, the controller advances the FIFO register to load the next instruction.

The peripheral executes up to 16 op codes. The first eight are single-cycle operations and identical to those of the floating-point processor. They consist of addition, subtraction, multiplication, and format conversion instructions. The remaining op codes are used to load, store, and divide data, and a multiple cycle instruction multiplies and accumulates values. The four remaining op codes can be defined by the user to implement application-related operations.

Software and hardware tools are a necessary part of such projects as the foregoing peripheral. The software assembles high-level microprograms and a JEDEC output file that specifies the fuse pattern to be burned into the PROM array. Currently, a program called Fuse Formatter, which runs on the IBM PC personal computer, lets designers enter hexadecimal code that corresponds to PROM data. From that code, the program creates a file that is downloaded directly to one of several PROM programmers. The latter blow the corresponding fuses in the microprogram memory and are the only required hardware tools. □

Om Agrawal is the product planning manager for programmable logic devices at AMD. He has designed 16- and 32-bit minicomputers, and is the coauthor of a book on high-speed memory systems. Agrawal holds a PhD in electrical engineering and computer science from Iowa State University. He also received an MBA from the University of Santa Clara.

As a senior product marketing engineer for the company's microprocessor division, Deepak Mithani, designs and markets bipolar microprocessors. He earned a BSEE from India's Maharaja Sayajirao University and an MSEE from the University of Wisconsin.



4. Loading an instruction into the FIFO starts the peripheral and activates the controller, which loads an external instruction register and decodes the op code field. The op code initiates a subroutine in the controller, issuing the proper control signals, advancing the FIFO register, and loading the next instruction.



Programmable event generator conquers timing restraints



Bruce Threewitt, Manager, Product Planning
Advanced Micro Devices, Sunnyvale, CA

When it comes to generating complex, high-resolution digital waveforms, Advanced Micro Devices' Am2971 picks up where older parts leave off. Precise time delays once required the use of either a hybrid structure consisting of an analog delay line combined with digital logic or costly counters driven by high-frequency clocks. On the one hand, analog delay lines have

a more elegant solution for the problem of generating complex high-speed timing waveforms (see Fig. 1). Applications for the PEG (see box, "A new PEG in the designer's tool box") range from simply correcting the clock skew that results from distributing a clock signal on a backplane to generating complex state-machine timing. The PEG's 12 output lines can be programmed by

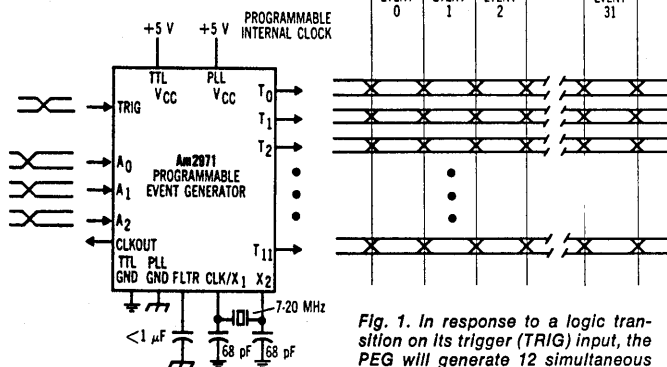


Fig. 1. In response to a logic transition on its trigger (TRIG) input, the PEG will generate 12 simultaneous user-programmed timing waveforms on its T_0 to T_{11} output lines.

short enough tap-to-tap time delays, or resolutions, to be usable for handling the tighter timing relationships of dynamic and static RAMs. However, those lines cannot also accommodate the longer total delays needed by these devices.

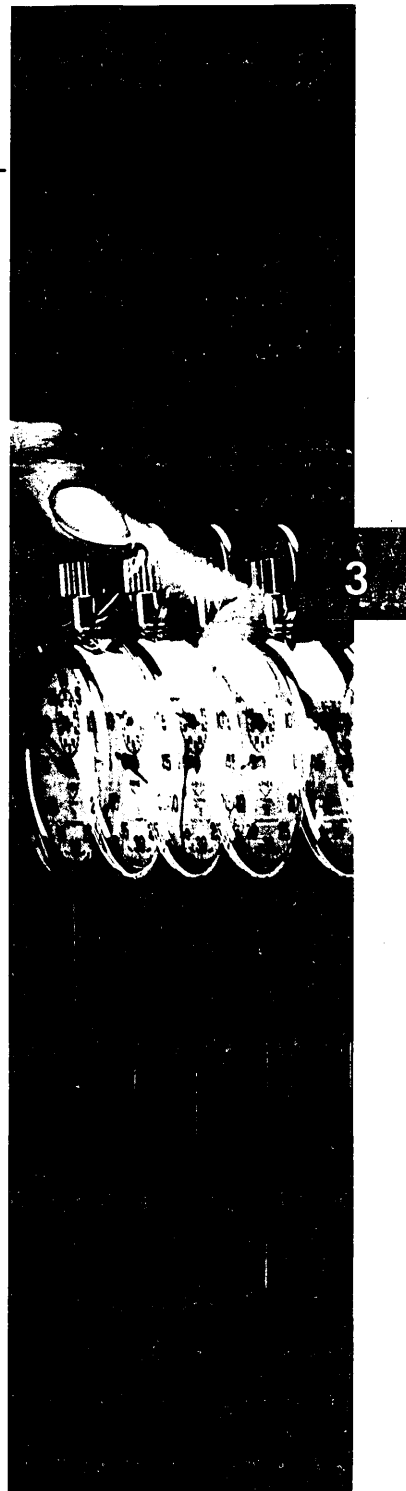
On the other hand, the desired waveforms can be obtained digitally from the outputs of one or more binary counters. But to achieve resolutions of 10 ns, counters must be driven with 100-MHz clock frequencies, which are difficult to distribute around a board.

The programmable event generator (PEG) a monolithic IC, offers

the user to assume either a logic-high or a logic-low level within each of 32 time slots or events. The frequency of these events—that is, the frequency of the output waveforms—is also programmable. But it is the precision and resolution of these waveforms that is the PEG's claim to fame (see Fig. 2).

Internal operation

The PEG is an edge-triggered logic device that generates a pre-programmed digital waveform in response to a triggering signal. It consists of four basic blocks: a next-address and event store, a start-ad-



address store, an oscillator and clock-control block, and control logic (see Fig. 3). A total of 623 user-programmable platinum-silicide fuses, similar to those used in AMD's bipolar PROMs and programmable array logic devices, are located throughout these blocks.

When the PEG's fuses are being programmed, the 12 timing-tap outputs (T_0 to T_{11}) are used as fuse-address inputs. A thorough description of the procedure for programming the PEG can be found on its data sheet, so for the purpose of understanding what goes on inside the chip, let's assume that it has already been programmed.

An output-timing sequence is initiated by a logic transition at the TRIG input. Two user-programmable fuses are located in the trigger-polarity block. One of them is a polarity-select fuse, which when un-

programmed will cause the timing sequence to start during a negative transition of the TRIG input. If it is programmed, the sequence starts when a positive transition occurs.

The second fuse in this block is used to define the end of a timing sequence. If this fuse is left unprogrammed, the timing sequence is stopped on the trailing edge of TRIG pulse. Otherwise if the second fuse is programmed, the end of the timing sequence is defined by the stop bits as programmed by the user into the next-address/event generator functional block.

Customized waveforms

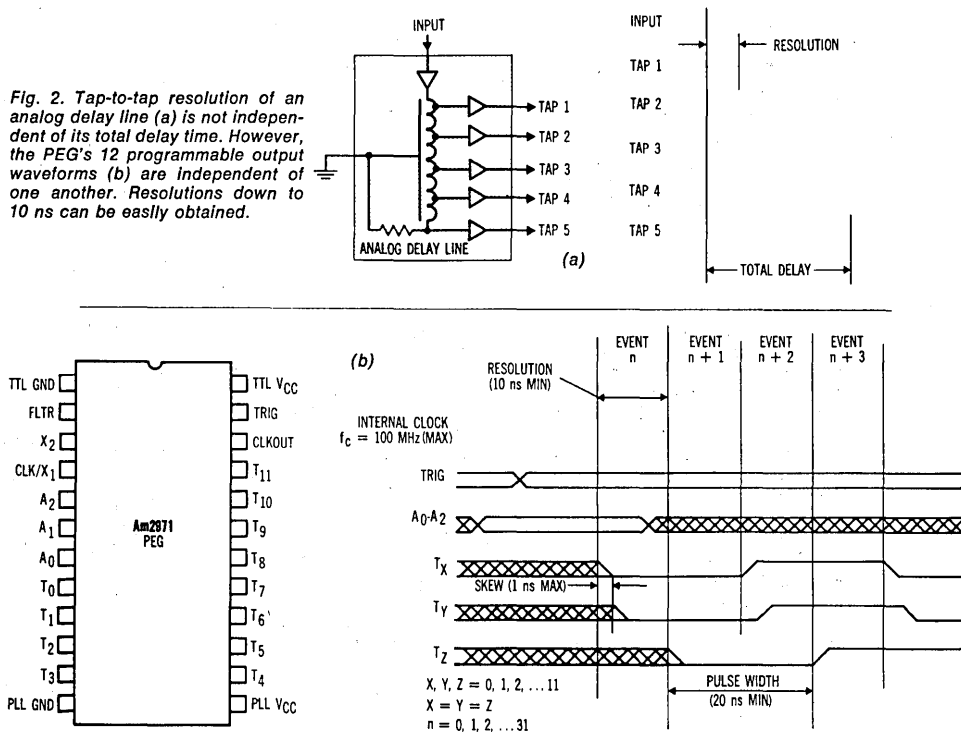
A transition at the TRIG input latches three address bits, A_0 to A_2 , which are decoded by the start-address store, which serves as an 8 x 5-bit mapping PROM. Each of its eight addresses represents a differ-

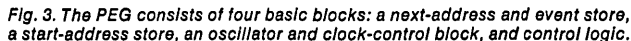
ent starting point for the 12 output waveforms. For example, three different addresses in this store could be used to initialize output waveforms that implement the read-, write-, and page-mode timing sequences of a dynamic RAM.

The five bits of each address are used to select one of thirty-two 18-bit locations from the next-address and event store (also a PROM). Each 18-bit string contains three groups of information. The first 5 bits define the next address of the desired timing sequence. The next 12 bits define the logic levels on each of the timing-output lines (T_0 to T_{11}). The 18th bit is a sequence-stop bit. As the timing sequence progresses from the first to the thirty-second event-store address, the output lines will produce 12 independent waveforms.

The assumption so far is that the

Fig. 2. Tap-to-tap resolution of an analog delay line (a) is not independent of its total delay time. However, the PEG's 12 programmable output waveforms (b) are independent of one another. Resolutions down to 10 ns can be easily obtained.





Typically, a signal provided by the system switches one or more address-multiplexer ICs. This signal must be accurately timed to change state after time interval 1 in *Fig. 5*. The performance of the individual DRAM—and ultimately of the entire memory system in which it resides—depends on reducing interval 2 as much as possible. Interval 1 is 15 to 20 ns in most standard DRAM spe-

Integrated Circuits

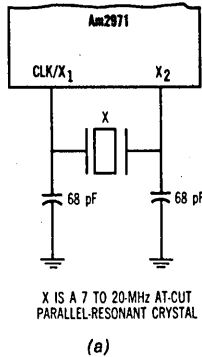
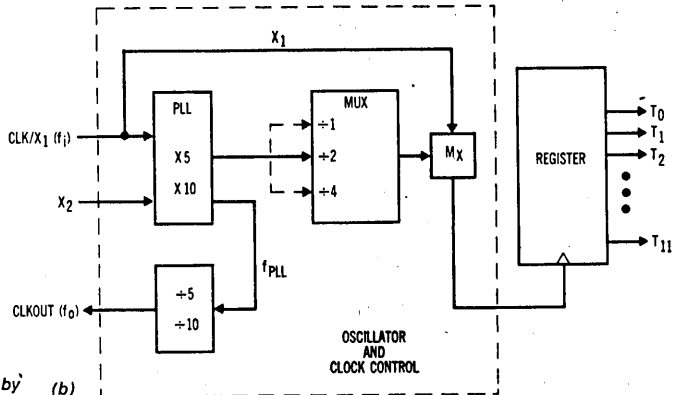


Fig. 4. The PEG can be clocked by an external crystal (a) or by the system clock. On-chip programmable clock-control logic (b) enables the user to set the internal-clock frequency, f_c .

cifications. Thus, a resolution of 10 to 15 ns is desirable for the DRAM-timing waveforms—an easy job for the PEG.

What's more, the PEG can be used to generate the necessary timing waveforms for multi-array memories too. In video-DRAM systems, for example, the rising edge of the transfer-enable signal, TRG, must be accurately placed relative to the rising edge of the serial-port clock signal, SC. When the PEG is used to generate both waveforms, the designer has full control over the reso-



lution between the rising edges of the TRG and SC signals.

More events and/or channels

In some applications, 12 output channels are sufficient but more than 32 event states are necessary. These additional states can be obtained by connecting two PEGs as shown in Fig. 6. After PEG 1 has cycled through its 32 states, its T_0 line triggers PEG 2. The second PEG will then begin its timing sequence at the start address specified by PEG 1. If necessary, the clock frequency of PEG 2 may be changed by using the clock output of PEG 1 as the clock input to PEG 2. Otherwise, the internal clocks and TRIG polarities

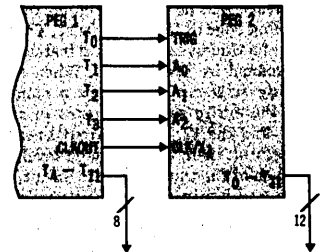


Fig. 6. By connecting two PEGs, designers can obtain timing waveforms that consist of more than 32 events.

of the two PEG chips are independently programmable.

If the waveforms must be nested, one of the output taps of PEG 2 can be used to remove the TRIG signal from PEG 1. PEGs can also be paralleled for applications in which 32 event states are sufficient but more than 12 output channels are needed. For example, when 12 to 24 channels are needed, the designer can simply drive two PEG chips with the same input signals and use as many of the output lines as necessary. □

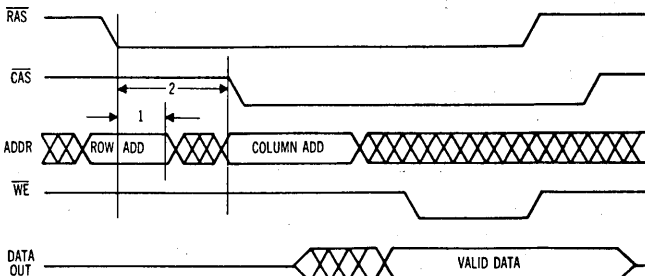


Fig. 5. In this DRAM delayed-write cycle, interval 1 must be kept to within 10 or 15 ns, and interval 2 must be as short as possible. When mounted on the same DRAM memory board, the PEG could be used to generate the RAS, CAS, and WE signals.

PLDs implement encoder/decoder for disk drives

3

By using software to define programmable logic devices as run-length-limited encode/decode systems, you can design disk-drive systems that have 50% more data-storage capacity than drives that implement the MFM code.

Arthur Khu and Rudy Sterner,
Advanced Micro Devices Inc

When you're designing a disk-drive system, you can implement run-length-limited (RLL) 2,7 encoding/decoding circuitry in your design by using only three programmable-logic devices (PLDs) and two shift registers. You design the encoder and decoder as state machines and use the timing diagrams to determine what the timing and control signals must be.

To create a disk controller with encoding/decoding features, you can use three AmPAL22V10 PLDs and a disk controller such as the Am9580/Am9582 chip set (Fig 1). The Am9580 hard-disk controller and the Am9582 disk-data separator perform all the general disk-control functions. The PLDs have appropriate architectures for implementing the encoding and decoding state machines. Further, you can reprogram the PLDs to implement higher density ratios for data encoding, and you can increase your system's speed simply by using faster PLDs.

An RLL code is a code in which the number of zeros between ones—the run length—is definite. The "2,7"

designation means that the code for the binary data string has at least two and at most seven zeros separating the ones. RLL codes increase the density of data stored on a disk by reducing the number of recorded pulses (ones) necessary to represent a given amount of data. This reduction allows the disk-drive circuitry to pack the ones closer together, increasing the amount of data on the disk.

In comparison with the (de facto) industry-standard approach, MFM, the RLL 2,7 code increases by 50% the amount of data you can store on a disk drive (see box, "RLL 2,7 code vs MFM code"). In addition, RLL

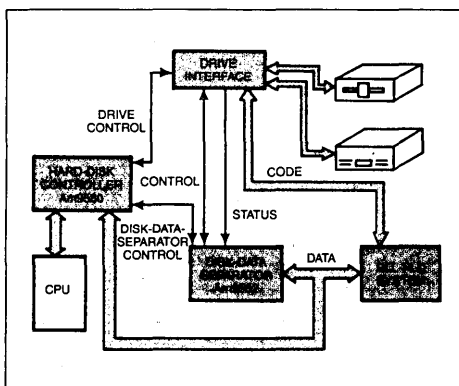


Fig 1—A complete disk controller requires only two VLSI ICs, a PLD-based encoding/decoding system, and a drive interface. The hard-disk controller and the disk-data separator provide generic disk-drive control, and the PLDs provide RLL 2,7 encoding and decoding, which increases disk storage capacity.

TABLE 1 — RLL 2,7 CODING RULES

DATA	2,7 CODE
10	1000
11	0100
000	100100
010	001000
011	000100
0010	00001000
0011	00100100

2,7 decoding circuitry recovers quickly from code-deletion errors.

As Table 1 shows, RLL 2,7 encoding circuitry translates seven data strings into 2,7 code. You can break any binary non-return-to-zero (NRZ) data string into combinations of the seven data strings. To obtain the

decoded data string, the circuitry matches the 2,7 code patterns with the seven 2,7 code strings in Table 1.

Because 2,7 code strings have variable lengths (they can be 2, 3, or 4 bits long), your design will need control logic that controls the output from the encoder/decoder as translation takes place. Encoding and decoding state machines (Figs 2 and 3) implement this control logic from the code in Table 1 (see box, "Convert RLL 2,7 code to a state machine"). The encoding state machine

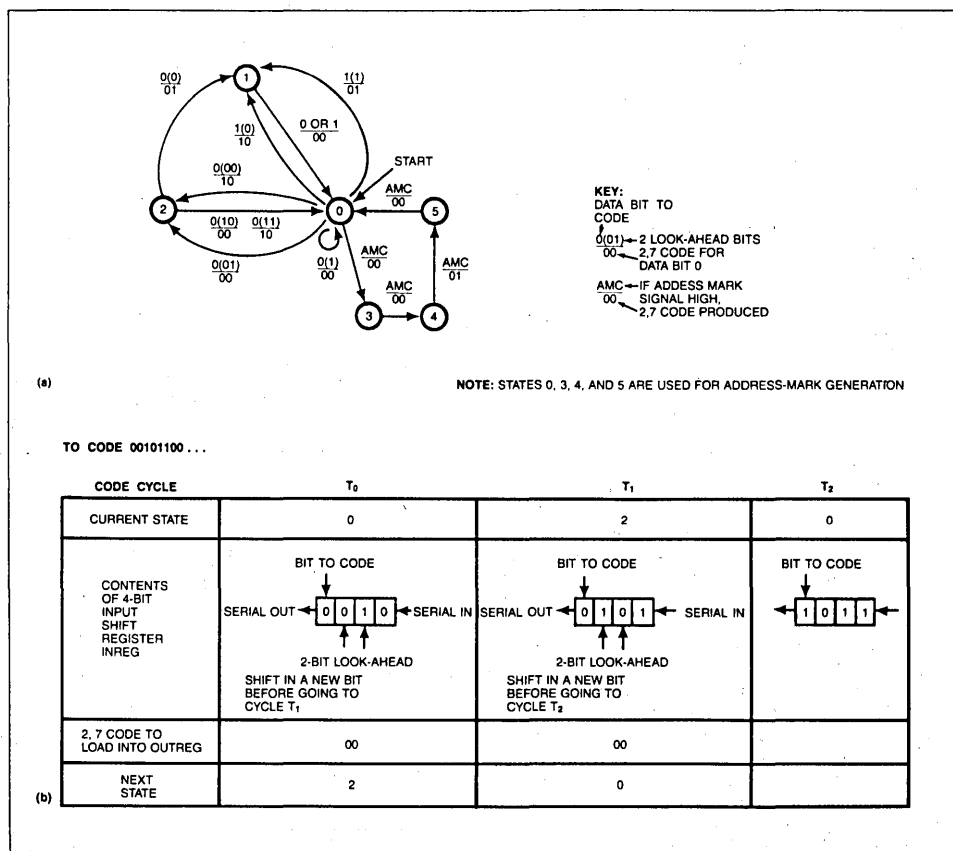


Fig 2—The encoder state machine (a) describes the translation of input data into RLL 2,7 code. The first two cycles in the encoding of the data string 00101100 (b) demonstrate how the encoder determines the correct code by examining both the bit to be encoded and the next two (look-ahead) bits.

During decoding, circuitry matches the RLL 2,7 code patterns with the seven RLL 2,7 code strings to obtain the corresponding decoded data string.

produces two 2,7 code bits for each data bit received. The decoding state machine, on the other hand, produces one data bit for every two bits of 2,7 code. The clocking scheme in these encoding/decoding state machines is simpler than the familiar table-look-up meth-

od, which requires suspended operation during encoding and decoding.

You can implement both of these encoding/decoding 2,7 state machines with one PLD device (IC₁) and two shift registers (INREG and OUTREG) (Fig 4). To

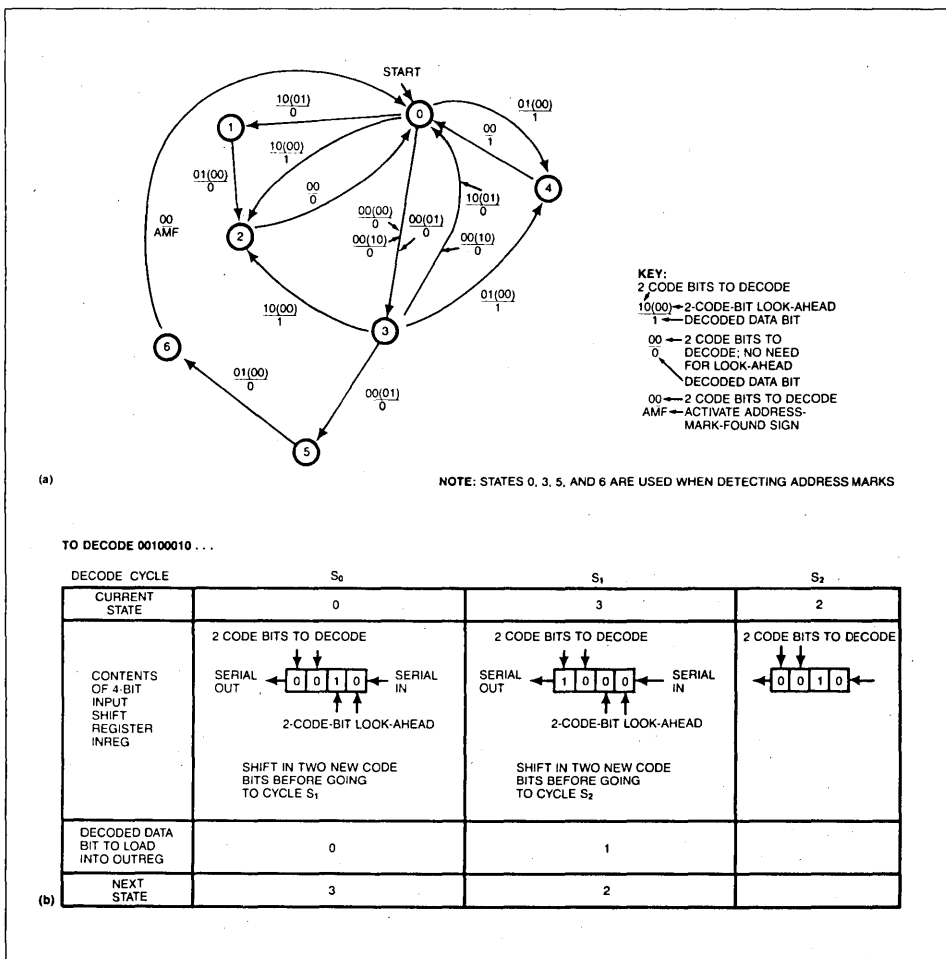


Fig 3—This decoder state machine (a) represents the translation of RLL 2,7 code into an output data stream. To determine the correct output data, the decoder examines four code bits—two bits to be decoded and two look-ahead bits, as shown in the example in b.

Because the RLL 2,7 code contains two bits for every one bit of data, the timing circuitry divides the clock signal for the coded data, producing a data clock.

synchronize the encoder/decoder with the hard-disk controller, you use two other PLDs (IC₂ and IC₃) to provide clock-generation and address-mark-control logic. IC₁ is a AmPAL22V10, which has sufficient capacity to implement the two state machines. IC₂, also an AmPAL22V10, utilizes the PLD's large capacity and programmable output cells to implement the address-mark-control circuitry. An AmPAL16HD8 (IC₃), which is sufficient for implementing the clock circuitry and the random logic, completes the design.

To understand the state-machine implementation of the RLL 2,7 encoder, consider the state machine in Fig

2. The encoding state machine begins in state zero; the first four bits of the data to be encoded are in the shift register INREG. For the data stream in the figure, the encoder, beginning in the first cycle (T₀) reads the first bit in the stream as 0 and sees that the next two bits (the look-ahead bits) in INREG are 0 and 1, respectively. According to the state diagram, the encoder produces a 00 output because, as Table 1 shows, input data starting with 001 translates to a character string that starts with 00. The encoder then enters state 2, shifts the encoded 0 out of INREG, and shifts in the next (fifth) bit of the data to be encoded.

RLL 2,7 code vs MFM code

Although the RLL 2,7 and MFM coding methods can both increase a disk drive's capacity, RLL 2,7 code is more compact. A disk drive that implements the RLL 2,7 code can, therefore, store 50% more data than can a drive that implements the MFM code.

On the magnetic medium in the disk drive (hard-disk or floppy-disk drives), binary data appears as a change in flux (representing a one) or as no change in flux (representing a zero). Because the disk density is limited by the minimum distance between flux transitions, the maximum data density depends on how the data is encoded.

MFM is an RLL code with the designation 1,3;1,2;1. RLL 2,7 code (its full designation is 2,7;1,2;3) allows a minimum of two zeros between each one, and MFM code allows a minimum of one zero. RLL 2,7 code can store as much as 50% more data in a given number of flux changes than can MFM code; therefore, RLL 2,7 code can store as much as 50% more data on a given section of magnetic

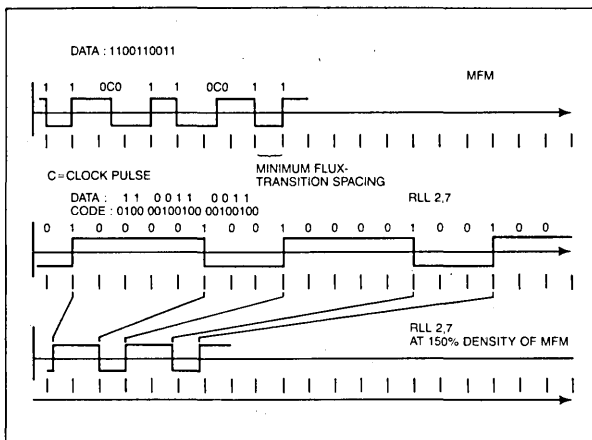


Fig A—Data storage that's 50% more dense is the principal advantage of RLL 2,7 code over the de facto standard MFM code. Because it needs fewer transitions to describe data, the RLL 2,7 code takes up only 67% of the disk storage space that MFM code occupies.

medium.

Consider the example in Fig A. The data stream 1100110011 is represented by eight transitions in MFM code, and by five transitions in RLL 2,7 code. When the disk drive uses the minimum distance between transitions to record the RLL 2,7

code, the RLL 2,7 code takes 67% of the space that the MFM code takes, so it has space available to store 50% more data. In most applications, the actual storage increase is between 35% and 40%, because some disk space is reserved for sector and data-field markers.

In the second cycle (T_1) of the encoding process, the encoder sees that the data bit is 0 and the two look-ahead bits are 1 and 0, respectively. According to the state diagram, when the encoder is in state 2 and sees 010, its output is 00. As before, the encoder then moves to the next state (in this case, state 0) and shifts a new bit into INREG. The rest of the encoding process proceeds similarly.

The decoding process is similar to the encoding process. The decoder, which is described as a state machine (Fig 3), accepts two input bits in RLL 2,7 format and sees the next two coded bits as look-ahead bits. In contrast to the encoder, the decoder produces one output bit for every two input bits.

To implement this encoder and decoder circuitry with

PLDs, you can use PLD-design tools such as CUPL from Personal CAD Systems Inc (San Jose, CA) and Abel from Data I/O Corp (Redmond, WA). These software tools include syntaxes that you can use to describe state machines as well as general logic equations.

To control the rate at which IC_1 (in Fig 4) receives data and code, IC_2 and IC_3 implement the timing signals shown in Fig 5. Three signals (Read, Write, and Code_Clock) from the hard-disk controller and disk-data separator form the basis of the timing signals.

First, the timing circuitry produces a clock signal, Rd_Clk , from the Code_Clock signal that originates at the disk-data separator's FDDAM output. Because the RLL 2,7 code contains two bits for every one bit of

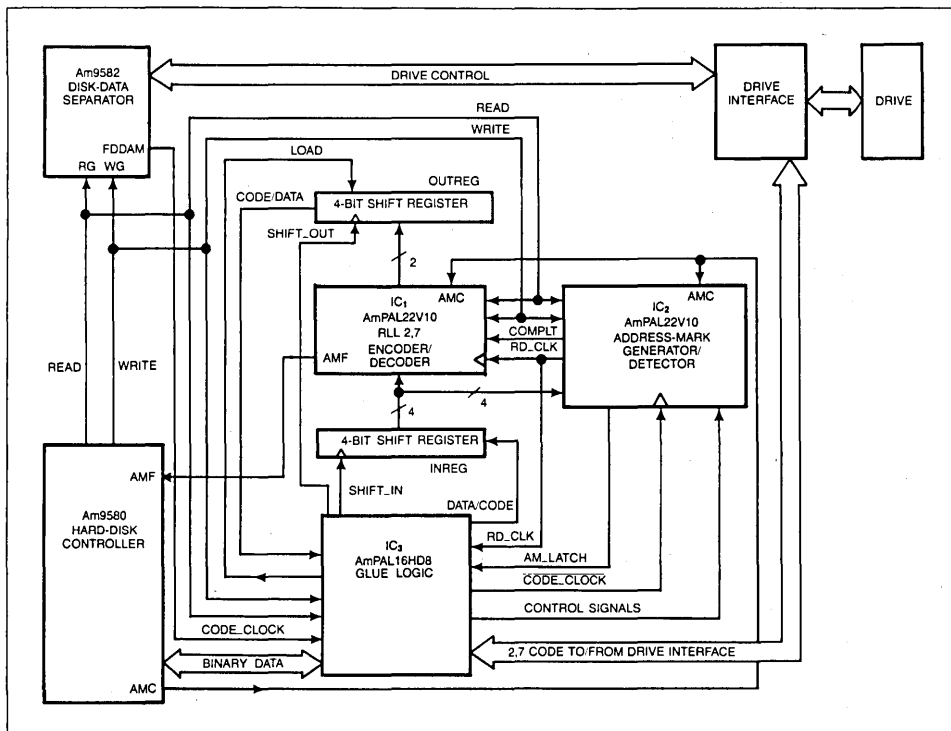


Fig 4—The encoding/decoding circuitry includes three PLDs. IC_1 implements the encoding and decoding state machines. IC_2 monitors IC_1 and provides special timing and input signals. IC_3 implements glue logic and timing signals.

Before the disk-drive system can decode data on the disk, it must synchronize itself with the disk's data clock to find out when the data bits begin.

data, the timing circuitry divides the clock signal for the coded data (Code_Clock), producing a data clock signal, Rd_Clk. The timing circuitry then uses Code_Clock and Rd_Clk to control the timing of shift-register control signals Shift_Out and Shift_In.

To obtain the equations you need to program the PLDs, examine the timing diagrams. The timing diagrams show that the state-machine design requires only a few timing signals to implement the controller.

Because the timing circuitry loads and shifts data produced by IC₁ at the points indicated on the timing diagram in Fig 5, the only clock signal that IC₁ requires to code or decode data in INREG is the Rd_Clk signal.

To control the transfer of data and code, the Shift_In signal latches data or code into INREG, and the Shift_Out signal controls the output of OUTREG. For example, when encoding data, the encoder produces two bits of code on each rising edge of Rd_Clk. Because

Convert RLL 2,7 code to a state machine

You can use a simple algorithmic procedure to convert a code table to an encoding state machine. For each row of the code table, you create a simple 2-state expression for the conversion of the data into the code. Then you combine the expressions into one state machine for the entire table.

For the code in Table 1 of the accompanying article, you consider each data bit and the associated pair of bits of RLL 2,7 code. For example, the first row of the table contains two data bits: one, which is associated with the code 10, and zero, which is associated with the code 00. The state machine for this row begins in state zero (which is arbitrarily assigned) and changes state when it sees that the first data bit is a one and the next bit (the look-ahead bit) is a zero. This change of state, which appears graphically in Fig Aa, results in the output 10.

Note that the state machine must evaluate the look-ahead bit. If this bit is a one instead of a zero, the input data will be 11, corresponding to the second row of the table, so your state machine must generate a 01. For

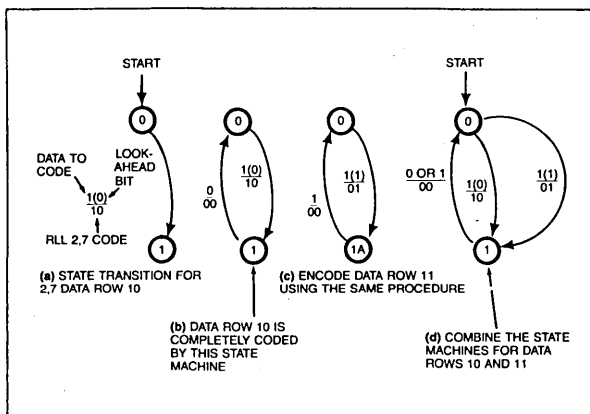


Fig A—Creating a state machine from a code table is a 2-part process. First, you create a simple state machine for each row in the table, and then you combine the state machines into one state machine for the whole table.

this code table, no more than two look-ahead bits are necessary for encoding any data bit.

If the state machine is in state one, and the input data is zero, the state machine produces a 00 output and returns to state zero (Fig Ab.) Note that the data bit encoded in this state is the look-ahead bit from the previous state and that no look-ahead bit is necessary for encoding because the zero is the last bit in data row 10.

You use this procedure to create state machines for the other table rows as well. Fig Ac, for example, contains the state machine for data row 11. Because all the state machines have the same beginning state (state zero), you can combine them to form the complete encoding state machine. Furthermore, the state machines may share other states, as shown in the combination of the two data rows (Fig Ad).

the Load signal is high at the same time that Rd_Clk is high, the bits are loaded into the shift register. Because the frequency of Shift_Out is twice that of Rd_Clk, Shift_Out shifts both encoded bits out before the next encoded bits are loaded. Shift_In presents the next data bit to the encoder at the same time that the second encoded bit is shifted out, starting the next encode cycle.

Before the disk-drive system can decode data on the disk, it must synchronize itself with the data clock from the disk and find out when the data bits begin. To assist the circuitry in synchronization and initialization, you can place synchronization signals, as well as a marker indicating the beginning of data, at the beginning of the RLL code. When the RLL circuitry reads these patterns, it's ready to decode RLL data.

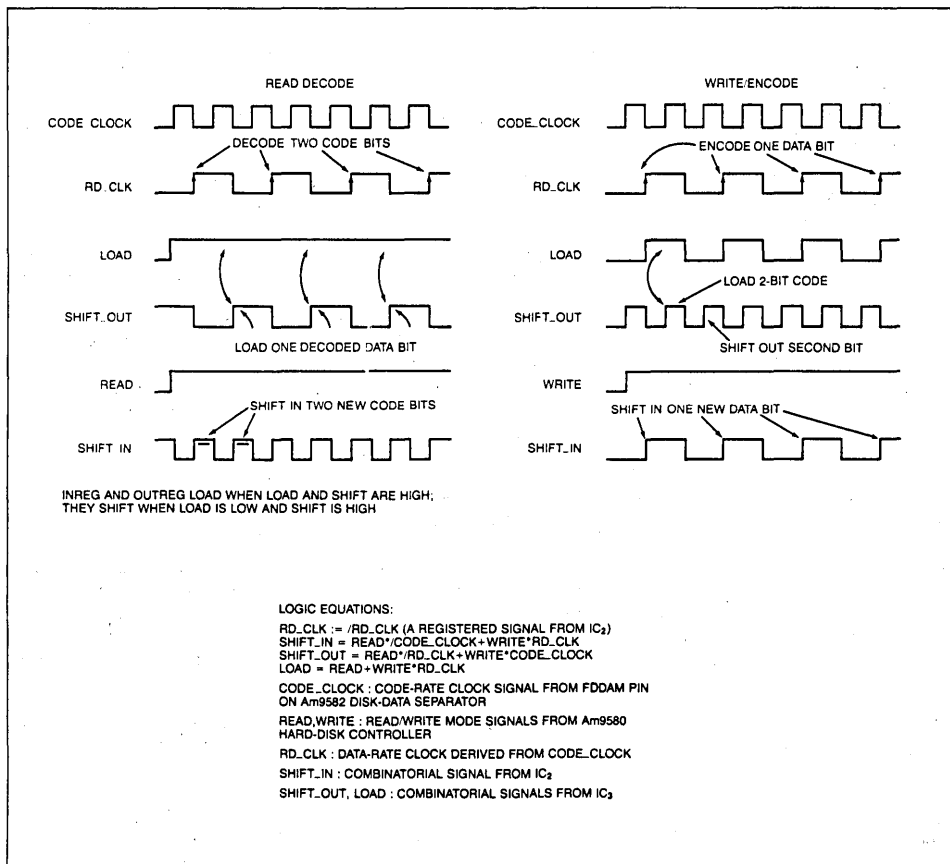


Fig 5—The timing signals from IC₁ and IC₂ control the loading and shifting of code and data in the INREG and OUTREG registers. Note that the clock and shifting signals for the coded bits have twice the frequency of those for the data bits, a situation that corresponds to the 2:1 density ratio between code and data.

If the decoder detects any pattern before it detects the address marker, the circuit resets itself and begins the initialization sequence anew.

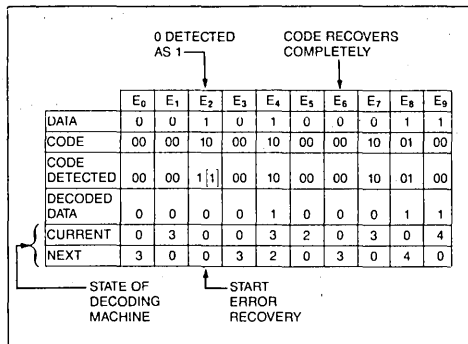


Fig 6—The ability to recover from an error is a useful feature of RLL 2,7 code. An error at cycle E₂ results in incorrectly translated data until the machine recovers fully.

The synchronization field comprises a series of patterns that allow the phase-locked loop in the disk-data separator to synchronize to the frequency of the incoming data. These patterns represent the maximum frequency input. For RLL 2,7 code, that input is 100100, because this code has the fewest permissible zeros between ones. Using the highest possible frequencies minimizes the synchronization time, so the patterns are completed quickly.

A marker that follows the synchronization field indicates the beginning of data. This marker, the address mark, must be distinct from all other code words. For example, you can use a pattern that violates the code rules, such as a string of zeros (this string is called "dc erase" because it removes all flux transitions from the recording medium). Alternatively, you can use a pattern that obeys the code rules but is not a defined code pattern. In the examples in this article, the pattern 00000100 identifies the beginning of data. Although this pattern is not a defined code word, it doesn't violate the RLL 2,7 code rules as long as it's preceded by no more than two zeros in a row.

When the 100100 pattern is synchronizing the circuitry, the circuitry produces the Rd_Clk signal, which is in phase with the pattern. In order for the decoder to operate properly, the Rd_Clk signal must rise with the first one in the pattern and fall with the second one. However, because of the repetitive nature of the pattern, the circuitry could produce a falling edge of Rd_Clk on the first one in the pattern and a rising edge on the second one, in which case the circuitry would not

be in synchronization with the beginning of the data.

The encoding circuitry resolves this synchronization problem by inserting the pattern "0100" eight times between the synchronization field and the address mark. The phase-locked-loop system locks onto the 0100 pattern, and the Rd_Clk control circuitry in IC₂ sets itself to decode the data correctly.

To put the synchronization and marker data into the RLL 2,7 code on the disk, you must design the RLL 2,7 encoder to produce the signals. When the signal WG (from the hard-disk controller IC₃) goes high, the data is encoded to all zeros. The encoder translates the zeros as the synchronization pattern 100100, which is then stored on disk.

Next, the address-mark-control (AMC) signal from the hard-disk controller sets an internal latch in IC₂, producing the output Am_Latch, which forces IC₃ to put ones in the INREG register. The encoding circuitry codes the ones as the string 0100. After the eighth 0100 pattern, IC₂ sets the Complt signal high. On sensing Complt, IC₁ writes the address mark.

The encoding circuitry must have the first four data bits in INREG by the time the address mark is written. IC₁ writes two patterns for the address mark. At the start of the second address-mark pattern, IC₁ sets AMF (Address Mark Found) High and resets Am_Latch low. The assertion of AMF signals the hard-disk controller to begin shifting data into INREG for encoding. Because four Rd_Clk cycles occur while the address-mark pattern is being written, the first four bits are shifted into INREG by the time the second address mark is written.

The decoder circuit has two safeguards that prevent it from identifying the address-mark pattern incorrectly. First, the decoder must detect at least five 0100 patterns before it acknowledges the address-mark patterns. Second, if it detects any pattern before it detects the address mark, the circuit resets itself and begins the initialization sequence.

IC₂ implements these two safeguards by monitoring the RLL code that IC₁ decodes. When the hard-disk controller asserts the RG and AMC signals, IC₂ sets Complt and Am_Latch low. IC₂ sets Complt high when it detects the fifth consecutive 0100 pattern, enabling IC₁ to detect the address mark. If the input to IC₁ is anything other than a 0100 pattern or an address mark, IC₂ sets Complt low, and the initialization begins anew.

If IC₁ successfully detects the address-mark pattern, it sets AMF high. The assertion of AMF causes IC₂ to set Am_Latch high, thus enabling the output of OUT-

REG to send decoded data to the hard-disk controller. Am_Latch remains high as long as RG is high.

RLL 2,7 encoding also provides for error recovery. Whenever a disk-drive system reads code from a disk, the read/write heads or the transmission cables can cause transmission errors. One of the properties of RLL 2,7 code is that any single-bit error (for example, a coded one detected as a zero) will correct itself after a run of at most 16 correctly detected bits.

In short, whenever a 2-bit pattern doesn't match any of the expected patterns for a particular state of the decoding circuitry, the decoding state machine returns to state zero and generates a zero as a translation for the erroneous code bits. Then the decoder shifts the next two code bits into the INREG register and continues decoding from state zero.

In such cases, the decoding state machine can correctly decode coded data, but it may not recover immediately upon returning to state zero. As Fig 6 shows, although the code bits that the decoder detects may be valid, the decoded data is incorrect because the error has forced that state machine into the wrong state. In this example, the decoder doesn't recover until 6 code bits later (in cycle E6), when it again falls into the correct state and accurately decodes the data. **EDN**

Authors' biographies

Arthur Khu is a product planning engineer at Advanced Micro Devices Inc (Sunnyvale, CA). He is involved in the research and development of architectures for advanced programmable-logic devices, and he has developed a general logic compiler for advanced PLDs. Art holds a BS in Math/Computer Science and an MS in Computer Science from Santa Clara University. He lists racquetball and astronomy among his interests.



Rudolph J Sterner, an engineer at Advanced Micro Devices Inc (Sunnyvale, CA), is engaged in the development of disk-drive-related products. Prior to his two years at AMD, Rudy worked at IMI Corp and Sperry Corp. He holds a BSEE from San Jose State University, and in his spare time he enjoys photography.



3.8 BIBLIOGRAPHY

Agrawal, Om. "Advances in Programmable Logic Architecture and Software Solutions," *Journal of Semicustom ICs*, November 1984.

Agrawal, Om and Deepak Mithani. "Controller mit PAL-Programmierung," *Elektronik*, Januar 1986.

Agrawal, Om and Deepak Mithani. "Le contrôleur Am29PL141 programmable par fusibles dans les systèmes microprogrammeur," *Minis et Micros*, Décembre 1985 – Janvier 1986.

Agrawal, Om and Jim Beck. "Hochleistung-PLDs als Ersatz für Semicustom Produkte," *Der Elektroniker*, November 1985.

Agrawal, Om and David Laws. "The Role of Programmable Logic in Systems Design," *VLSI Design*, March 1984.

Agrawal, Om and Kapil Shankar. "Second-Generation Bipolar Devices Offer Faster Speed and Increased Flexibility," *Midcon*, September 1986.

Agrawal, Om and Arthur Khu. "A Smart Single-Chip Fuse Programmable Controller Eases Complex State Machine Design," *Electro* 1986.

Agrawal, Om and Deepak Mithani. "Vereinfachter Entwurf dezentraler microprogrammierbarem Controller-Chip," *Elektronik Informationen*, November 1985.

Agrawal, Om and David Laws. "Welche Rolle spielen PALs?" *Elektronik*, November 1984.

Chang, Don. "Designing a State Machine Using a PAL," *Wescon 1985*, Session 24.

Cole, Bernard. "Field-Programmable Logic: A New Market Force," *Electronics*, pp 25–31, January 27, 1986.

Collett, Ronald. "Programmable Logic Declares War on Gate Arrays," *Digital Design*, pp 33–38, July 1986.

Durwood, Brian and Denny Siu. "Automatic Test Vector Generation and Fault Grading Help to Engineer Testability into PLDs," *Wescon 1985*, Session 24.

Gladstone, Bruce and William D. Ellis. "Using Programmable Logic, Desktop 'Foundry' Lowers Risk of Semicustom Design," *Electronic Design*, October 1985.

Holley, Michael. "IC Advances Expand Circuit Designer's Logic Options," *Computer Design*, pp 77–81, January 15, 1986.

Khu, Arthur. "Section 3.5.4 VME Bus Control Simplified with PLDs," portions reprinted from *EDN*, October 2 and 16, 1986. Copyright 1986 Cahners Publishing Company.

Kahl, Tracy and Bob Osann. "PC Design Tools for Instant Custom Logic," *Wescon 1985*, Session 24.

Kitson, Brad. "Diagnostic Pipeline Register Enhances System Reliability and Testability," *Electro* 1983.

Kitson, Brad and Joshua Rosen. "Super-minicomputer design using PALs," *Electronic Engineering*, March 1984.

Kitson, Brad and Kevin Ow-wing. "Logic Programming Enriches Design Process," *Electronic Design*, March 1984.

Marrin Ken. "PLDs Slow Advance of Gate Arrays in Low-End Designs," *Computer Design*, pp 43–54, February 1, 1986.

Miller, Warren and David Laws. "Fuss-Programmable Logic Spearheads Semicustom Revolution," *Electronic Products*, November 1983.

Miller, Warren and David Laws and Brad Kitson. "Logik-Baustein Kombiniert PAL und Gate-Array-Technik," *Elektronik*, Juni 1984.

Miller, Warren. "On-chip Circuitry Reveals System's Logic States," *Electronic Design*, April 1983.

Oliver, Douglas. "The Benefits of Software-Configured Programmer for Programmable Logic," *Wescon 1985*, Session 25.

Osann, Bob. "Using the PC for Logic Design," *Wescon 1985*, Session 25.

Ow-wing, Kevin. "Diagnostics Registers Simplify System Self-Testing," *Electronic Design*, October 1983.

Ow-wing, Kevin. "Methode de test et de conception de diagnostic," *Minis Et Micros*.

Pellerin, Dave B. and Victor Ehr. "Advanced Software Tools for PLD Design," *Midcon 1985*, Session 19.

Salkow, Steven P. "Design PALs in a High-Performance System," *Wescon 1985*, Session 24.

Smith, David. "Programmable Logic Devices," *EDN*, pp. 94–106, May 15, 1986.

Wang, Tom. "How to Solve the PAL Design Problem," *Wescon 1985*, Session 24.

Yee, Jenny. "Second-Generation Programmable Logic Devices Extend Design Capabilities," *Electro* 1984.

"Programmable Logic Devices, 1986 Technology Forecast," *Electronic Design*, pp 123–129, January 9, 1986.

SECTION 4 — PRODUCT SPECIFICATIONS



- 4.1 Field Programmable Logic Selector Guide**
- 4.2 Glossary**
- 4.3 AmPAL16XX Family 20-Pin IMOX Programmable Array Logic (PAL) Elements**
- 4.4 AmPAL16XXD Family 20-Pin IMOX Ultra High-Speed Programmable Array Logic (PAL) Elements**
- 4.5 AmPAL18P8 20-Pin IMOX Programmable Array Logic**
- 4.6 AmPAL20EG8 IMOX-III ECL Programmable Array Logic**
- 4.7 AmPAL20EV8 IMOX-III ECL Programmable Array Logic**
- 4.8 AmPAL21VT8 24-Pin Dual-Clock Programmable Array Logic**
- 4.9 AmPAL22V10 24-Pin IMOX Programmable Array Logic (PAL)**
- 4.10 AmPAL22V10B 24-Pin IMOX Programmable Array Logic (PAL)**
- 4.11 AmPAL23S8 20-Pin IMOX PAL-Based Sequencer**
- 4.12 AMD 24-Pin Standard PAL Family**
- 4.13 AMD 24-Pin Enhanced PAL Family**



4.14 AMD 24-Pin XOR PAL Family

**4.15 AmPALHC29M16/AmPALHCT29M16 24-Pin
E²-Based CMOS Programmable Array Logic**

**4.16 AmPALHC29MA16/AmPALHCTMA 16 24-Pin
E²-Based CMOS Programmable Array Logic**

4.17 Bipolar PROMs as Programmable Logic Products

4.18 Am29PL141 Fuse Programmable Controller (FPC)

4.18 Am2971 Programmable Event Generator (PEG)

4.1 FIELD PROGRAMMABLE LOGIC SELECTOR GUIDE

Features of PAL Devices

- High speed electrically programmable array logic elements
- User customizable logic patterns, generated in minutes with PROM type programmers
- Improves performance and reduces board area and cost of existing TTL SSI/MSI designs
- Easy to use software design aids available
- Security fuse prevents copying of logic by competitors

Advantages of AMD PAL Devices

- AMD's superior Bipolar technology ensures industry-leading speed/power families of PALs
- Platinum-Silicide fuses and added test words ensure programming yields > 98%
- Post Programming Functional Yield (PPFY) > 99.8%
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parameter testing at the factory through on-board testing circuitry
- Power-up reset simplifies state machine design
- Industry-leading quality guarantees

COMMERCIAL PRODUCTS ONLY

DEVICE NAME	# PINS	MAX I_{CC} mA	MAX f_{MAX} MHz	MAX t_{PD} ns	MIN t_s ns	MAX t_{CO} ns	MIN I_{OL} mA	INPUT STRUCTURE	OUTPUT STRUCTURE	ARRAY SIZE	LOG. PTs	PGMMBLE OE	PGMMBLE OP	LOGIC STRUCTURE
16HD8A 16HD8L 16HD8	20	155 80 155	N/A	25 35 35	N/A	N/A	24 24 24	16 INPUTS: 6 Bidirectional 10 Dedicated	8 OUTPUTS: 8 Dedicated, Active HIGH	32 x 64	64	DED	DED	EIGHT 8-WIDE AND-OR
16H8A 18H8L 16H8	20	155 80 155	N/A	25 35 35	N/A	N/A	24 24 24	16 INPUTS: 6 Bidirectional 10 Dedicated	8 OUTPUTS: 6 Bidirectional and 2 Dedicated, Active HIGH	32 x 64	56 + 8 OE PTs	PRG	DED	EIGHT 7-WIDE AND-OR
16LD8A 16LD8L 16LD8	20	155 80 155	N/A	25 35 35	N/A	N/A	24 24 24	16 INPUTS: 6 Bidirectional 10 Dedicated	8 OUTPUTS: 8 Dedicated, Active LOW	32 x 64	64	DED	DED	EIGHT 8-WIDE AND-OR-INVERT
*16L8D *16L8BL 16L8B *16L8AQ 16L8AL 16L8A 16L8Q 16L8L 16L8	20	180 90 180 45 90 180 45 90 180	N/A	10 15 15 25 25 25 35 35 35	N/A	N/A	24 24 24 12 24 24 12 24 24	16 INPUTS: 6 Bidirectional 10 Dedicated	8 OUTPUTS: 6 Bidirectional, Active LOW 2 Dedicated, Active LOW	32 x 64	56 + 8 OE PTs	PRG	DED	EIGHT 7-WIDE AND-OR-INVERT

FIELD PROGRAMMABLE LOGIC SELECTOR GUIDE

COMMERCIAL PRODUCTS ONLY

DEVICE NAME	# PINS	MAX I _{CC} mA	MAX f _{MAX} MHz	MAX t _{PD} ns	MIN t _S ns	MAX t _{CO} ns	MIN I _{OL} mA	INPUT STRUCTURE	OUTPUT STRUCTURE	ARRAY SIZE	LOG. PTs	PGMMBLE OE	OP	LOGIC STRUCTURE
*16R4D *16R4BL 16R4B *16R4AQ 16R4AL 16R4A 16R4Q 16R4L 16R4	20	180 90 180 45 90 180 45 90 180	55.5 40 40 28.5 28.5 28.5 18 18 18	10 15 15 25 25 25 35 35 35	10 13 13 20 20 20 30 30 30	8 12 12 15 15 15 25 25 25	24 24 24 12 24 24 12 24 24	16 INPUTS: 4 Bidirectional 8 Dedicated 4 Feedback	8 OUTPUTS: 4 Bidirectional 4 Registered	32 x 64	60 + 4 OE PTs	4 PRG 4 DED	DED	FOUR 8-WIDE AND-OR FOUR 7-WIDE AND-OR-INVERT
*16R6D *16R6BL 16R6B *16R6AQ 16R6AL 16R6A 16R6Q 16R6L 16R6	20	180 90 180 45 90 180 45 90 180	55.5 40 40 28.5 28.5 28.5 18 18 18	10 15 15 25 25 25 35 35 35	10 13 13 20 20 20 30 30 30	8 12 12 15 15 15 25 25 25	24 24 24 12 24 24 12 24 24	16 INPUTS: 2 Bidirectional 8 Dedicated 6 Feedback	8 OUTPUTS: 2 Bidirectional 6 Registered	32 x 64	62 + 2 OE PTs	2 PRG 6 DED	DED	SIX 8-WIDE AND-OR TWO 7-WIDE AND-OR-INVERT
*16R8D *16R8BL 16R8B *16R8AQ 16R8AL 16R8A 16R8Q 16R8L 16R8	20	180 90 180 45 90 180 45 90 180	55.5 40 40 28.5 28.5 28.5 18 18 18	N/A	10 13 13 20 20 20 30 30 30	8 12 12 15 15 15 25 25 25	24 24 24 12 24 24 12 24 24	16 INPUTS: 8 Dedicated 8 Feedback	8 OUTPUTS: 8 Registered	32 x 64	64	DED	DED	EIGHT 8-WIDE AND-OR
18P8B 18P8AL 18P8A 18P8Q 18P8L	20	180 90 180 55 90	N/A	15 25 25 35 35	N/A	N/A	24 24 24 12 24	18 INPUTS: 8 Bidirectional 10 Dedicated	8 OUTPUTS: 8 Bidirectional, Active HIGH or LOW	36 x 72	64 + 8 OE PTs	PRG	PRG	EIGHT 8-WIDE AND-OR
10H20EV8 10020EV8 (ECL)	24	220 220	125 125	6 6	4.5 4.5	3.5 3.5	N/A	20 INPUTS: 8 Bidirectional 11 Dedicated 1 INPUT or CLK	8 OUTPUTS: 8 OLMs: Choose REG or COMB, Active HIGH or LOW	40 x 90	80 + AP, AR, & 8 OE PTs	PRG	PRG	VARIABLE PTs: 8-8- 12-12-12-12- 8-8
10H20EG8 10020EG8 (ECL)	24	220 220	125 125	6 6	4.5 4.5	3.5 3.5	N/A	20 INPUTS: 8 Bidirectional 11 Dedicated 1 INPUT or LE	8 OUTPUTS: 8 OLMs: Choose LATCHED or COMB, Active HIGH or LOW	40 x 90	80 + AP, AR, & 8 OE PTs	PRG	PRG	VARIABLE PTs: 8-8- 12-12-12-12- 8-8

COMMERCIAL PRODUCTS ONLY

DEVICE NAME	# PINS	MAX I_{CC} mA	MAX f_{MAX} MHz	MAX t_{PD} ns	MIN t_s ns	MAX t_{CO} ns	MIN I_{OL} mA	INPUT STRUCTURE	OUTPUT STRUCTURE	ARRAY SIZE	LOG. PTs	PGMMBLE OE	PGMMBLE OP	LOGIC STRUCTURE
20L8B 20L8A 20L8AL	24	210 210 105	N/A	15 25 25	N/A	N/A	24 24 24	20 INPUTS: 7 Bidirectional 13 Dedicated	8 OUTPUTS: 7 Bidirectional, Active LOW 1 Dedicated, Active LOW	40 x 72	64 + 8 OE PTs	PRG	DED	EIGHT 7-WIDE AND-OR-INVERT
20L10B 20L10-20 20L10A 20L10AL	24	210 165 165 105	N/A	15 20 25 25	N/A	N/A	24 24 24 24	20 INPUTS: 12 Dedicated 8 Feedback	10 OUTPUTS: 8 Bidirectional 2 Dedicated	40 x 76	66 + 10 OE PTs	PRG	DED	TEN 3-WIDE AND-OR-INVERT
20RP4B 20RP4A 20RP4AL	24	210 210 105	30 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 6 Bidirectional 10 Dedicated 4 Feedback	10 OUTPUTS: 6 Bidirectional 4 REG Inverting	40 x 90	84 + 6 OE PTs	PRG	PRG	SIX (6-2)-WIDE AND-OR FOUR 8-WIDE AND-OR
20RP6B 20RP6A 20RP6AL	24	210 210 105	30 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 4 Bidirectional 12 Dedicated 6 Feedback	10 OUTPUTS: 4 Bidirectional 6 Registered	40 x 90	86 + 4 OE PTs	PRG	PRG	TEN 8-WIDE AND-OR-INVERT
20RP8B 20RP8A 20RP8AL	24	210 210 105	30 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 2 Bidirectional 10 Dedicated 8 Feedback	10 OUTPUTS: 2 Bidirectional 8 Registered	40 x 90	88 + 2 OE PTs	PRG	PRG	TEN 8-WIDE AND-OR
20RP10B 20RP10A 20RP10AL	24	210 210 105	37 25 25	N/A	15 25 25	12 15 15	24 24 24	20 INPUTS: 10 Dedicated 10 Feedback	10 OUTPUTS: 10 Registered	40 x 80	80	PRG	PRG	TEN 8-WIDE AND-OR
20R4B 20R4A 20R4AL	24	210 210 105	37 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 4 Bidirectional 12 Dedicated 4 Feedback	8 OUTPUTS: 4 Bidirectional 4 Registered	40 x 72	68 + 4 OE PTs	PRG	DED	FOUR 8-WIDE AND-OR FOUR 7-WIDE AND-OR-INVERT
20R6B 20R6A 20R6A4	24	210 210 105	37 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 2 Bidirectional 12 Dedicated 6 Feedback	8 OUTPUTS: 2 Bidirectional 6 Registered	40 x 72	70 + 2 OE PTS	PRG	DED	SIX 8-WIDE AND-OR TWO-7-WIDE AND-OR-INVERT
20R8B 20R8A 20R8AL	24	210 210 105	37 25 25	15 25 25	15 25 25	12 15 15	24 24 24	20 INPUTS: 12 Dedicated 8 Feedback	8 OUTPUTS: 8 Registered	40 x 72	72	PRG	DED	EIGHT 8-WIDE AND-OR
20XRP4-20 20XRP4-30 20XRP4-40 20XRP4-30L 20XRP4-40L	24	210 180 180 90 90	30 22.2 14.3 22.2 14.3	20 30 40 30 40	20 30 40 30 40	13 15 30 15 30	24 24 24 24 24	20 INPUTS: 6 Bidirectional 10 Dedicated 4 Feedback	10 OUTPUTS: 6 Bidirectional 4 Registered	40 x 90	84 + 6 OE PTs	PRG	PRG	FOUR (2-6)-WIDE AND-OR-XOR SIX 8-WIDE AND-OR-INVERT

FIELD PROGRAMMABLE LOGIC SELECTOR GUIDE

COMMERCIAL PRODUCTS ONLY

DEVICE NAME	# PINS	MAX I_{CC} mA	MAX f_{MAX} MHz	MAX t_{PD} ns	MIN t_s ns	MAX t_{CO} ns	MIN I_{OL} mA	INPUT STRUCTURE	OUTPUT STRUCTURE	ARRAY SIZE	LOG. PTs	PGMMBLE		LOGIC STRUCTURE
20XRP6-20 20XRP6-30 20XRP6-40 20XRP6-30L 20XRP6-40L	24	210 180 180 90 90	30 22.2 14.3 22.2 14.3	20 30 40 30 40	20 30 40 30 40	13 15 30 15 30	24 24 24 24 24	20 INPUTS: 4 Bidirectional 6 Feedback 10 Dedicated	10 OUTPUTS: 4 Bidirectional 6 Dedicated	40 x 90	86 + 4 OE PTs	PRG	PRG	SIX (2-6)-WIDE AND-OR-XOR FOUR 8-WIDE AND-OR-INVERT
20XRP8-20 20XRP8-30 20XRP8-40 20XRP8-30L 20XRP8-40L	24	210 180 180 90 90	30 22.2 14.3 22.2 14.3	20 30 40 30 40	20 30 40 30 40	13 15 30 15 30	24 24 24 24 24	20 INPUTS: 2 Bidirectional 10 Dedicated 8 Feedback	10 OUTPUTS: 2 Bidirectional 8 Registered	40 x 90	80 + 10 OE PTs	PRG	PRG	EIGHT (2-6)-WIDE AND-OR-XOR TWO 8-WIDE AND-OR-INVERT
20XRP10-20 20XRP10-30 20XRP10-40 20XRP10-30L 20XRP10-40L	24	210 180 180 90 90	30 22.2 14.3 22.2 14.3	N/A	20 30 30 30 40	13 15 15 15 30	24 24 24 24 24	20 INPUTS: 10 Dedicated 10 Feedback	10 OUTPUTS: 10 Registered	40 x 80	80	DED	DED	TEN (2-6)-WIDE AND-OR-XOR
22P10B 22P10A 22P10AL	24	210 210 105	N/A	15 25 25	N/A	N/A	24 24 24	20 INPUTS: 10 Bidirectional 12 Dedicated	10 OUTPUTS: 10 Bidirectional	40 x 90	80 + 10 OE PTs	PRG	PRG	TEN 8-WIDE AND-OR INVERT
*22V10B *22V10AL 22V10A *22V10Q 22V10L 22V10	24	180 90 180 45 90 180	45.5 28.5 28.5 18 18 18	15 25 25 35 35 35	12 20 20 30 30 30	10 15 15 25 25 25	24 24 24 12 24 24	22 INPUTS: 10 Bidirectional 12 Dedicated	10 OUTPUTS: 10 OLMs: Choose REG or COMB, Active HIGH or LOW	44 x 132	120 + AR, SP, & 8 OE PTs	PRG	PRG	VARIABLE PTs: 8-10-12-14- 16-16- 14-12-10-8
22XP10-20 22XP10-30 22XP10-40 22XP10-30L 22XP10-40L	20	210 180 180 90 90	N/A	20 30 40 30 40	N/A	N/A	24 24 24 24 24	22 INPUTS: 10 Bidirectional 12 Dedicated	10 OUTPUTS: 10 Bidirectional, Active HIGH or LOW	40 x 90	80 + 10 OE PTs	PRG	PRG	TEN (2-6)-WIDE AND-OR-XOR-
23S8-20 23S8-25	20	200 200	33 28.5	20 25	17 20	13 15	16 16	23 INPUTS: 9 Dedicated 4 Feedback 4 Programmable 6 Internal Feedback	8 OUTPUTS: (plus 6 BSRs) 4 Registered 4 OLMs: Choose REG or COMB, Active HIGH or LOW	46 x 135	124 + SP, AR, OBS, & 8 OE PTs	PRG	4 PRG, 4 DED	VARIABLE PTs: 8-10-6-8-8- 12-10-10-12- 8-8-6-10-8

COMMERCIAL PRODUCTS ONLY

DEVICE NAME	# PINS	MAX I_{CC} mA	MAX f_{MAX} MHz	MAX t_{PD} ns	MIN t_s ns	MAX t_{CO} ns	MIN I_{OL} mA	INPUT STRUCTURE	OUTPUT STRUCTURE	ARRAY SIZE	LOG. PTs	PGMMBLE OE	PGMMBLE OP	LOGIC STRUCTURE
*HC29M16-35 *HC29M16-45 (CMOS I/O)	24	120 120	20 15	35 45	27 34	23 32	6 6	29 INPUTS: 3 Dedicated 8 Dedicated Feedback	16 OUTPUTS: 16 I/O Logic Macrocells REG/LATCH/COMB Active HIGH or LOW	58 x 188	176 + 8 OE, AR, AP, OBS PRELOAD PTs	PRG	PRG	VARIABLE PTs: 8-8-8-8- 8-8-8-8- 12-12-12-12- 16-16-16-16
*HCT29M16-35 *HCT29M16-45 (TTL I/O)	24	120 120	20 15	35 45	27 34	23 32	6 6	1 CLK/LE/I 1 OE/I 16 I/O LMs						
*HC29MA16-35 *HC29MA16-45 (CMOS I/O)	24	120 120	20 15	35 45	27 34	23 32	6 6	29 INPUTS: 4 Dedicated 1 OE/I 8 Dedicated Feedback	16 OUTPUTS: 16 I/O Logic Macrocells REG/LATCH/COMB Active HIGH or LOW	58 x 184	128 + 16 AR & AP, 16 CLK/LE, 4 OE, OBS & PRELOAD PTs	PRG	PRG	VARIABLE PTs: 5-5-5-5- 5-5-5-5- 9-9-9-9- 13-13-13-13
*HCT29MA16-35 *HCT29MA16-45 (TTL I/O)	24	120 120	20 15	35 45	27 34	23 32	6 6	16 I/O LMs						

KEY TO ABBREVIATIONS:

*	Under Development	t_{CO}	Max. Clock to Output Delay	t_{PD}	Max. Propagation Delay
OE	Output Enable	COMB	Combinatorial	I_{OL}	Min. Output Drive Current
PT	Product Term	OLM	Output Logic Macrocell	LE	Latch Enable
OP	Output Polarity	I/O LM	Input/Output Logic Macrocell	OBS	Observability
REG	Register(ed)	I_{CC}	Input Current	t_s	Min. Setup Time
AR	Asynchronous RESET	N/A	NOT Applicable	BSR	Buried State Register
SP	Synchronous PRESET	f_{MAX}	Max. Operating Frequency;	DED	Dedicated
AP	Asynchronous PRESET		$f_{MAX} = 1/(t_s + t_{CO})$		

DESIGN-AID SOFTWARE TOOLS FOR AMD PAL DEVICES

Vendor	Software	Hardware Platform
Data I/O Corp. 10525 Willows Road N.E. Redmond, WA 98073 (206) 881-6444	ABEL	IBM PC or compatible DEC VAX (VMS, UNIX) Apollo (AEGIS) Sun Microsystems (UNIX)
	DASH/CADAT DASH/ABEL	IBM PC or compatible
ISDATA Haid-und-Neu-Str. 7 D-7500 Karlsruhe West Germany (0721) 693092	LOGIC	IBM PC or compatible DEC VAX (VMS, UNIX) Apollo (AEGIS)
JMC PROMAC Division 2999 Monterey/Salinas Highway Monterey, CA 93940 (408) 373-3607	PALASM	PROMAC P3
Personal CAD Systems Inc. 1290 Parkmoor Avenue San Jose, CA 95126 (408) 971-1300	CUPL	IBM PC or compatible DEC VAX (VMS, UNIX)
	CAE	IBM PC or compatible
MMI (Public Domain)	PALASM	IBM PC or compatible PC (CPM-80) DEC VAX (VMS, UNIX)
AMD (Public Domain)	PLPL	IBM PC or compatible DEC VAX (UNIX)
AMD 901 Thompson Place Sunnyvale CA 94088 (408) 732-2400	AmCUPL	IBM PC or compatible
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043 (415) 968-2900	PERFECT VISTA	IBM PC or compatible

AMD-QUALIFIED PAL PROGRAMMER MODELS

Vendor	Programmer Models	AMD PAL Personality Module	Socket Adapter
DATA I/O Corporation 10525 Willows Rd. N.E. P.O. Box 97046 Redmond, WA 98073-9746 (206) 881-6444	MODELS 100A, 19, 29A, 29B	LOGICPAK	303A - 004 303A - 011A 303A - 011B
	UniSite 40	Not Required	Not Required
DIGILEC, INC. 1602 Lawrence Ave. Suite 113 Ocean, NJ 07712 (210) 493-2420	803	FAM52	DA53, DA55
KONTRON ELECTRONICS, INC. 1230 Charleston Road Mountain View, CA 94039 (800) 227-8834	MODEL-MPP-80S or EPP80	Not Required	SA37
STAG MICROSYSTEMS 528-5 Weddell Drive Sunnyvale, CA 94086 (408) 745-1991	MODEL-PPZ	ZM2200	Not Required
	ZL30A/ZL32	Not Required	Not Required
STRUCTURED DESIGN, INC. 988 Bryant Way Sunnyvale, CA 94087 (408) 737-7131	SD1040 PAL Burner	Not Required	Not Required
VALLEY DATA SCIENCES 2426 Charleston Road Mountain View, CA 94043 (415) 968-2900	VDS 160	Not Required	Not Required
JMC PROMAC DIVISION 2999 Monterey Highway Monterey, CA 93940 (408) 373-3607	PROMAC - P3	Not Required	Not Required

4.2 GLOSSARY

PAL PARAMETER DEFINITION

VOLTAGE

V_{OH} (High-level Output Voltage)

The minimum HIGH logic level guaranteed for all outputs under worst-case operating conditions.

V_{OL} (Low-level Output Voltage)

The maximum LOW logic level guaranteed for all outputs under worst-case operating conditions.

V_{IH} (High-level Input Voltage)

The minimum HIGH level voltage that may be applied to any input that is guaranteed to represent a HIGH logic level.

V_{IL} (Low-level Input Voltage)

The maximum LOW level voltage that may be applied to any input that is guaranteed to represent a LOW logic level.

V_i (Input Clamp Voltage)

The maximum input clamp voltage is a measure of the clamping capability of an input clamp diode provided on every input. The input clamp diode suppresses ringing due to transmission-line impedance mismatching.

CURRENT

I_{OH} (High-level Output Current)

The current forcing condition used to test the HIGH logic level, V_{OH} .

I_{OL} (Low-level Output Current)

The current forcing condition used to test the logic-LOW level, V_{OL} .

I_{IH} (High-level Input Current)

The current measured into an input pin when a HIGH logic level is applied to the input pin.

I_i (Input Current with Maximum Input Voltage)

The current measured into an input pin when the maximum allowable input voltage is applied to the input pin.

I_{IL} (Low-level Input Current)

The current measured out of an input pin when a LOW logic level is applied to the input pin.

I_{OZH} (High-level Leakage Current)

The current measured into a three-stated output pin when a high logic level is applied to the output pin. The three-state condition is overriding a logic-LOW level output state.

I_{OZL} (Low-level Leakage Current)

The current measured out of a three-stated output pin when a LOW logic level is applied to the output pin. The three-state condition is overriding a logic-HIGH level output state.

I_{SC} (Output Short Circuit Current)

The current out of an output pin when $V_{OUT} = 0.5$ V is applied to an output that is in a logic-HIGH level state.

I_{CC} (Power Supply Current)

The current into the V_{CC} terminal of the device when the device is in its maximum power logic condition.

CAPACITANCE

C_{IN} (Input Capacitance)

The capacitance at an input pin at $V_{IN} = 2.0$ V and $f = 1$ MHz.

C_{OUT} (Output Capacitance)

The capacitance at an output pin or I/O pin at $V_{OUT} = 2.0$ V and $f = 1$ MHz.

TIMING PARAMETERS

t_{PD} (Propagation Delay Time)

The t_{PD} parameter specifies the delay time for an output to switch logic levels as measured between a reference level on the input causing the output to change and a reference level on the changing output.

t_{PZX} (Output Enable Time, Three State to Active Output)

The t_{PZX} parameter specifies the delay time for a dedicated enable buffer to switch a registered output from a three-state to an active-HIGH or active-LOW logic level, the standard load pull-up resistor must be disconnected to establish a LOW level during three-state for t_{ZH} measurements.

t_{PXZ} (Output Disable Time, Active Output to Output Disable)

The t_{PXZ} parameter specifies the delay time for a dedicated enable buffer to switch a registered output from an active-HIGH or active-LOW level to a three-state condition. Special measurement levels and loads are specified for these tests to minimize the effect of the test environment.

t_{EA} (Output Enable Time, Three State to Active Output)

The t_{EA} parameter specifies the delay time for a combinatorial or macrocell output to switch from a three-state to an active-HIGH or active-LOW level. The switching path is through normal inputs and special enable product terms associated with the outputs. The standard load pull-up resistor must be disconnected to establish a LOW level during three-state for t_{ZH} measurements.

t_{ER} (Output Disable Time, Active to Three-State Output)

The t_{ER} parameter specifies the delay time for combinatorial or macrocell output to switch from an active-HIGH or active-LOW level to a three-state condition. The switching path is through normal inputs and special enable product terms associated with the outputs. Special measurement levels and loads are specified for these tests to minimize the effect of the test environment.

t_s (Setup Time)

The t_s parameter defines the minimum time a valid data level must precede a clock input edge to assure it is accepted by a storage device (e.g., register).

t_H (Hold Time)

The t_H parameter defines the minimum time a valid data level must be held after a clock input edge to assure it is accepted by a storage device (e.g., register).

t_{CO} (t_{CO1}) (Clock to Output Time, External)

The t_{CO} parameter specifies the delay between the active clock edge and a registered output switching to an active-HIGH or LOW level.

t_{CO2} (Clock to Output Time, Internal)

The t_{CO2} parameter specifies the delay between the active clock edge and a combinatorial output driven by feedback from a switching register.

t_p (t_{p1}) (Clock Period, External)

The t_p parameter specifies the clock period when registered outputs are a function of both internal and external signals.

 t_{p2} (Clock Period, Internal)

The t_{p2} parameter specifies the clock period when registered outputs are a function of internal signals only.

 t_w (Clock Width)

The t_w parameter specifies the minimum clock pulse width, either HIGH or LOW level, that will still allow proper function of registers. In some cases simultaneous minimum width of both HIGH and LOW levels is not allowed. This is true when t_p is greater than two times t_w .

FREQUENCY **f_{MAX} (f_{1MAX}) (Maximum Frequency, External)**

The f_{MAX} parameter is derived from the clock period ($1/t_p$ or $1/t_{p1}$). It is equal to the maximum clocking frequency when registered outputs are a function of both internal and external signals.

 f_{2MAX} (Maximum Frequency, Internal)

The f_{2MAX} parameter is derived from the clock period ($1/t_{p2}$). It is equal to the maximum clocking frequency when registered outputs are a function of internal signals only.

ASYNCHRONOUS RESET TIMING PARAMETERS **t_{AW} (Asynchronous-Reset Pulse Width)**

The t_{AW} parameter specifies the width of the asynchronous reset pulse. (The minimum value for this parameter is the worst case propagation delay minus the best-case propagation delay.)

 t_{AR} (Asynchronous-Reset Recovery Time)

The t_{AR} parameter specifies the time for the asynchronous-reset signal to become inactive.

 t_{AP} (Asynchronous-Reset Propagation Delay)

The t_{AP} parameter specifies the delay between the registered outputs to become reset and the associated asynchronous-reset input.

AmPAL*16XX Family

20-Pin IMOX™ Programmable Array Logic (PAL) Elements

AmPAL*16XX Family

DISTINCTIVE CHARACTERISTICS

- AMD's superior IMOX technology
 - Guarantees $t_{PD} = 15$ ns Max. "B" Versions
- High-Speed, Half-Power ("AL") and Quarter-Power ("Q") versions
- Platinum-silicide fuses and added test words ensure programming yields > 98%
- Post Programming Functional Yields (PPFY) of 99.9%
- PRELOAD feature permits full logical verification
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parametric testing at the factory through on-board testing circuitry
- AMD's industry-leading quality guarantees

GENERAL DESCRIPTION

AMD PAL devices are high-speed, electrically programmable array logic elements. They utilize the familiar sum-of-products (AND-OR) structure allowing users to program custom logic functions to fit most applications precisely. Typically they are a replacement for low-power Schottky SSI/MSI logic circuits, reducing chip count by more than 5 to 1 and greatly simplifying prototyping and board layout.

Seven different devices are available, including both registered and combinatorial devices, in six different speed and

power versions. The very High-Speed "B" versions ($t_{PD} = 15$ ns) run approximately 40% faster than the High-Speed "A" versions ($t_{PD} = 25$ ns). High-Speed, Half-Power "AL" versions ($t_{PD} = 25$ ns, $I_{CC} = 90$ mA) are available, as well as Standard-Speed, Half-Power "L" versions ($t_{PD} = 35$ ns, $I_{CC} = 80$ mA). Quarter-Power "Q" versions ($t_{PD} = 35$ ns, $I_{CC} = 45$ mA) are also available.

Please see the following pages for Block Diagrams.

*Combinatorial functions

PRODUCT SELECTOR GUIDE

AMD PAL Speed/Power Families

Family	t_{PD} ns (Max.)		t_S (1) ns (Min.)		t_{CO} (1) ns (Max.)		I_{CC} (2) mA (Max.)	I_{OL} mA (Min.)	
	C Devices	M Devices	C Devices	M Devices	C Devices	M Devices	C/M Devices	C Devices	M Devices
Very High-Speed ("B") Versions	15	20	13	18	12	15	180	24	12
High-Speed ("A") Versions	25	30	20	25	15	20	155 (2)	24	12
High-Speed, Half-Power ("AL") Versions	25	30	20	25	15	20	90	24	12
Standard Versions	35	40	30	35	25	25	155 (2)	24	12
Half-Power ("L") Versions	35	40	30	35	25	25	80 (2)	24	12
Quarter-Power ("Q") Versions	35	40	30	35	25	25	45	12	8

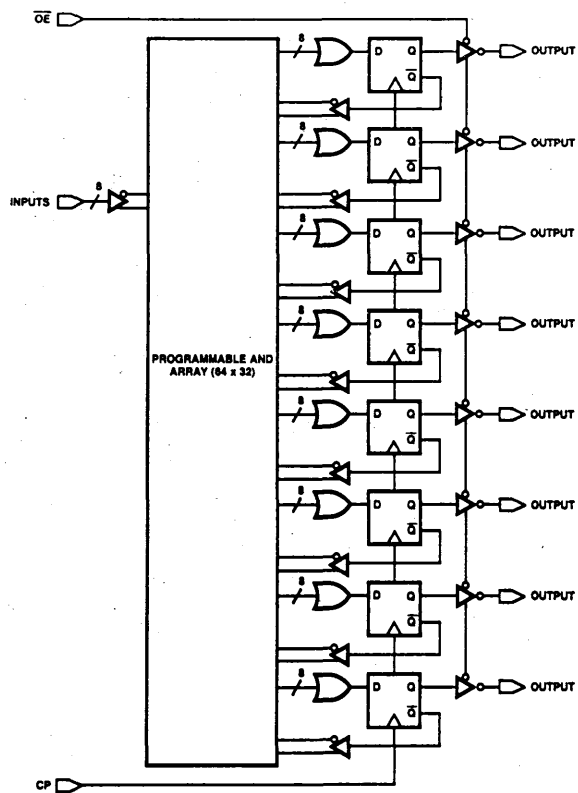
- (1) Sequential functions
(2) Combinatorial functions

AMD PAL FUNCTIONS

Part Number	Array Inputs	Logic	Output Enable	Outputs	Package Pins
16R8	Eight Dedicated, Eight Feedback	Eight 8-Wide AND-OR	Dedicated	Registered Inverting	20
16R6	Eight Dedicated, Six Feedback, Two Bidirectional	Six 8-Wide AND-OR	Dedicated	Registered Inverting	20
		Two 7-Wide AND-OR-INVERT	Programmable	Bidirectional	
16R4	Eight Dedicated, Four Feedback, Four Bidirectional	Four 8-Wide AND-OR	Dedicated	Registered Inverting	20
		Four 7-Wide AND-OR-INVERT	Programmable	Bidirectional	
16L8	Ten Dedicated, Six Bidirectional	Eight 7-Wide AND-OR-INVERT	Programmable	Six Bidirectional Two Dedicated	20
16H8	Ten Dedicated, Six Bidirectional	Eight 7-Wide AND-OR	Programmable	Six Bidirectional Two Dedicated	20
16LD8	Ten Dedicated, Six Bidirectional	Eight 8-Wide AND-OR-INVERT	-	Dedicated	20
16HD8	Ten Dedicated, Six Bidirectional	Eight 8-Wide AND-OR	-	Dedicated	20

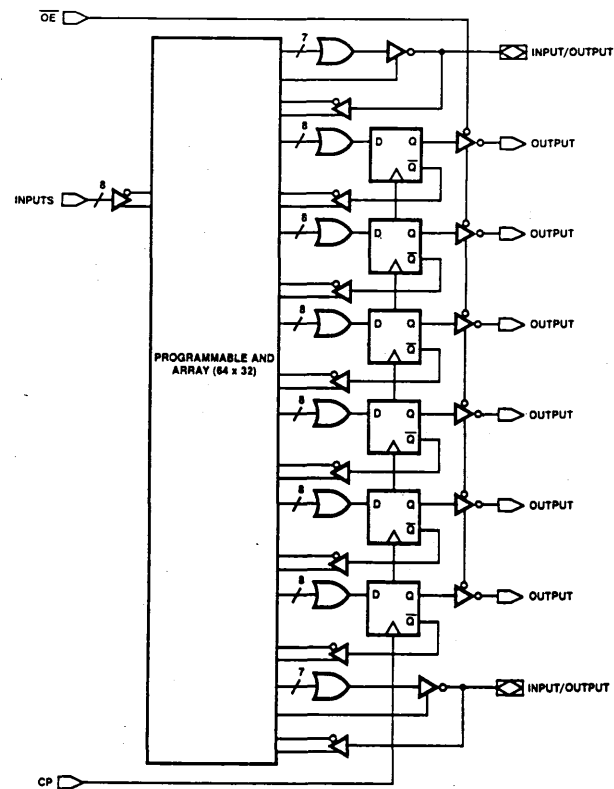
BLOCK DIAGRAMS

AmPAL16R8



BD002720

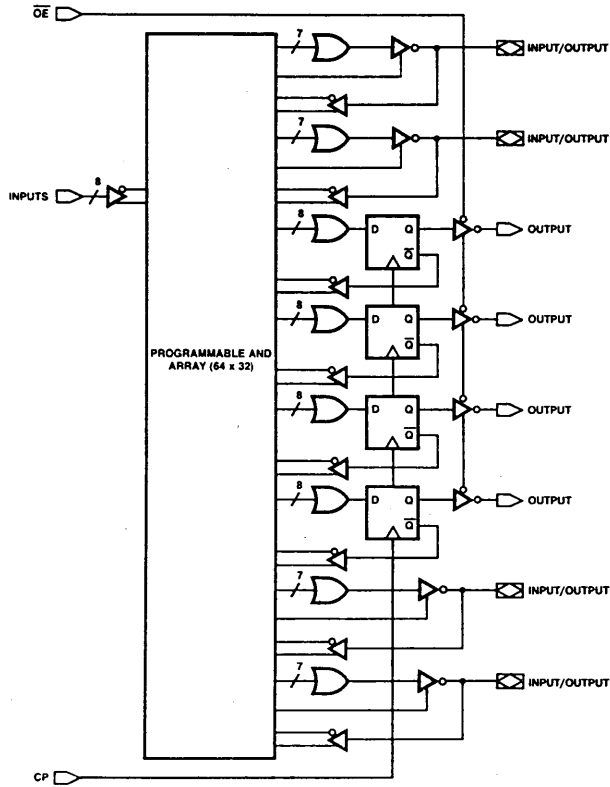
AmPAL16R6



BD002730

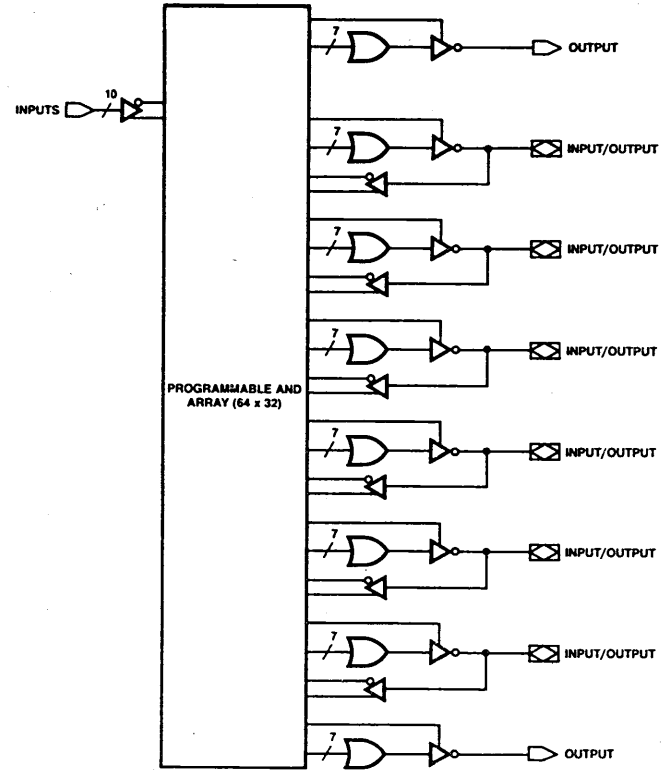
BLOCK DIAGRAMS (Cont'd.)

AmPAL16R4



BD002740

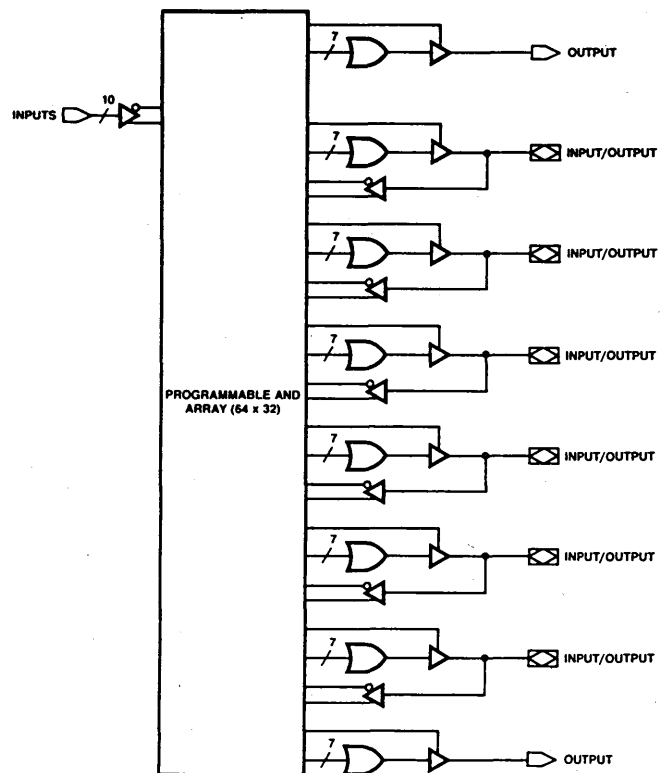
AmPAL16L8



BD002750

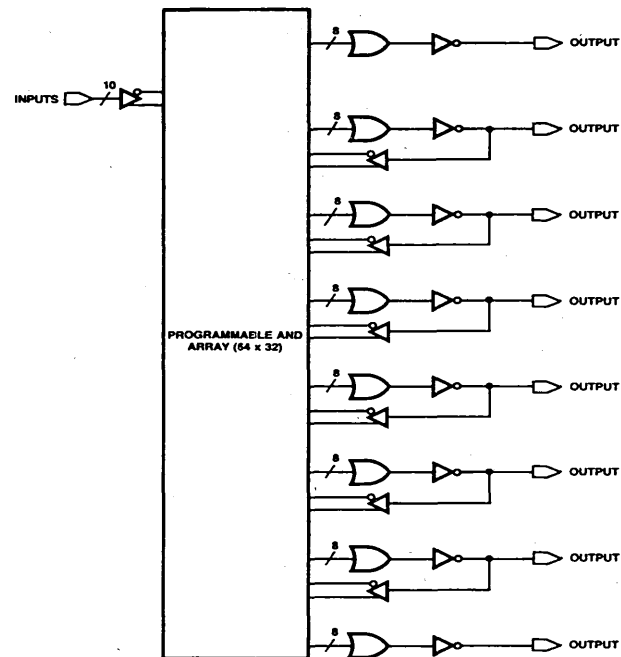
BLOCK DIAGRAMS (Cont'd.)

AmPAL16H8



BD003170

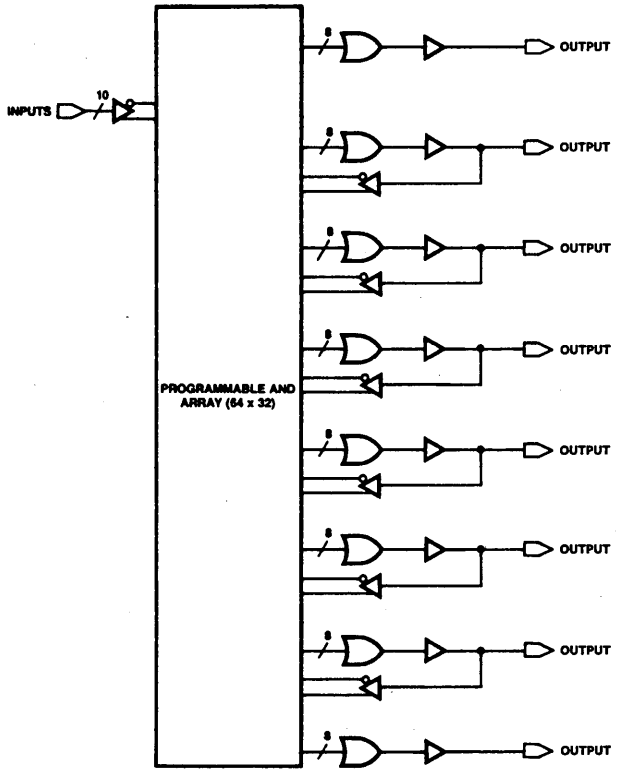
AmPAL16LD8



BD002760

BLOCK DIAGRAMS (Cont'd.)

AmPAL16HD8

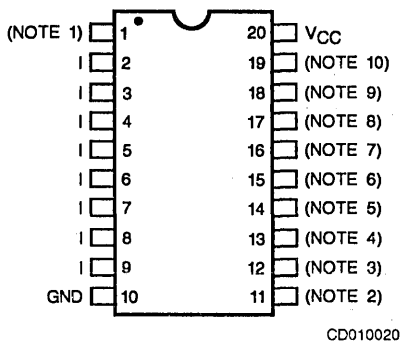


BD003181

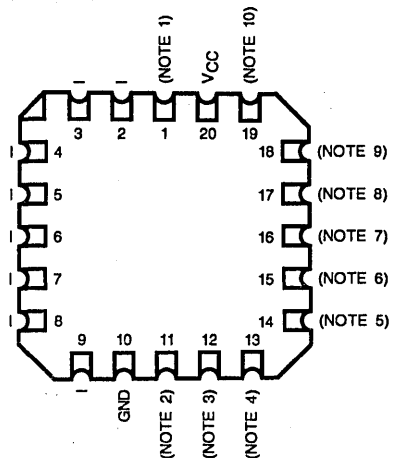
CONNECTION DIAGRAMS

Top View

DIPs*



LCC**



Note: Pin 1 is marked for orientation.

Notes:

	16L8	16R8	16R6	16R4	16H8	16HD8	16LD8
1	I	CLK	CLK	CLK	I	I	I
2	I	OE	OE	OE	I	I	I
3	O	O	I/O	I/O	O	O	O
4	I/O	O	O	I/O	I/O	O	O
5	I/O	O	O	O	I/O	O	O
6	I/O	O	O	O	I/O	O	O
7	I/O	O	O	O	I/O	O	O
8	I/O	O	O	O	I/O	O	O
9	I/O	O	O	I/O	I/O	O	O
10	O	O	I/O	I/O	O	O	O

*Also available in 20-Pin Ceramic Flatpack. Pinouts identical to DIPs.

**Also available in 20-Pin Plastic Leaded Chip Carrier. Pinouts identical to LCC.

PIN DESIGNATIONS

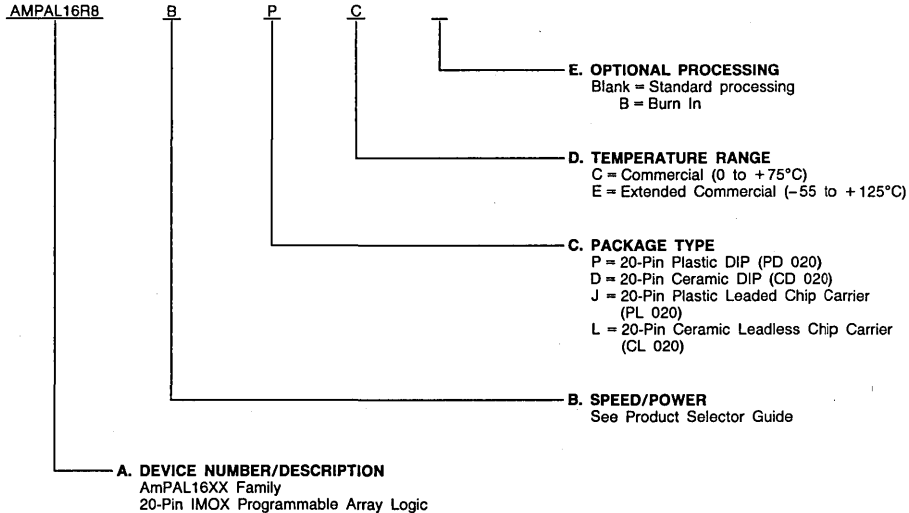
I = Input
 I/O = Input/Output
 O = Output
 VCC = Supply Voltage
 GND = Ground
 CLK = Clock
 OE = Output Enable

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

Valid Combinations	
AMPAL16R8/B/A/AL/L/Q	PC, DC, DCB, DE, JC, LC, LE
AMPAL16R6/B/A/AL/L/Q	
AMPAL16R4/B/A/AL/L/Q	
AMPAL16L8/B/A/AL/L/Q	
AMPAL16H8/A/L	
AMPAL16LD8/A/L	
AMPAL16HD8/A/L	

Valid Combinations

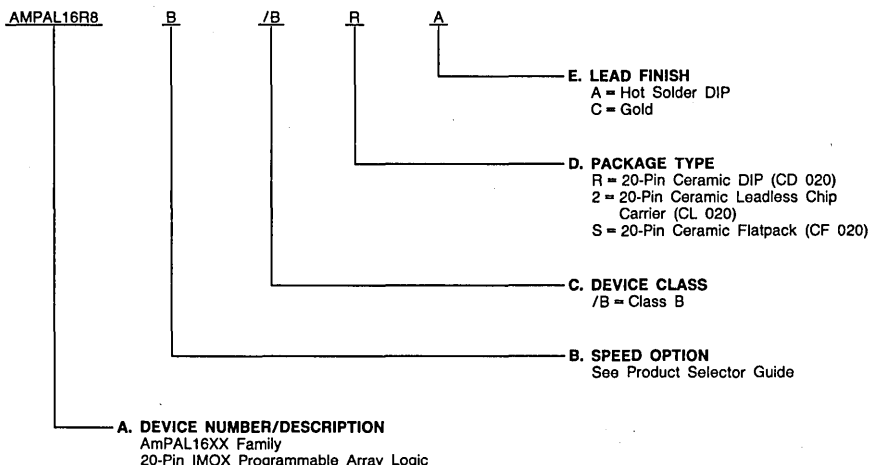
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number**
- B. Speed Option** (if applicable)
- C. Device Class**
- D. Package Type**
- E. Lead Finish**



Valid Combinations	
AMPAL16R8/B/A/AL/L/Q	/BRA, /B2C, /BSA
AMPAL16R6/B/A/AL/L/Q	
AMPAL16R4/B/A/AL/L/Q	
AMPAL16L8/B/A/AL/L/Q	
AMPAL16H8/A/L	
AMPAL16LD8/A/L	
AMPAL16HD8/A/L	

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 consist of Subgroups 1, 2, 3, 4, 9, 10, 11.

DESC Certified PAL Devices

Generic	AMD Part Number	DESC Numbers
16L8	AmPAL16L8A/BRA	8103607RX
	AmPAL16L8A/B2C	81036072X
	AmPAL16L8A/BSA	8103607SX
	AmPAL16L8L/BRA	8103611RX
	AmPAL16L8L/B2C	81036112X
	AmPAL16L8L/BSA	8103611SX
	AmPAL16L8/BRA	8103601RX
	AmPAL16L8/B2C	81036012X
16R8	AmPAL16R8A/BRA	8103608RX
	AmPAL16R8A/B2C	81036082X
	AmPAL16R8A/BSA	8103608SX
	AmPAL16R8L/BRA	8103612RX
	AmPAL16R8L/B2C	81036122X
	AmPAL16R8L/BSA	8103612SX
	AmPAL16R8/BRA	8103602RX
	AmPAL16R8/B2C	81036022X
16R6	AmPAL16R6A/BRA	8103609RX
	AmPAL16R6A/B2C	81036092X
	AmPAL16R6A/BSA	8103609SX
	AmPAL16R6L/BRA	8103613RX
	AmPAL16R6L/B2C	81036132X
	AmPAL16R6L/BSA	8103613SX
	AmPAL16R6/BRA	8103603RX
	AmPAL16R6/B2C	81036032X
16R4	AmPAL16R4A/BRA	8103610RX
	AmPAL16R4A/B2C	81036102X
	AmPAL16R4A/BSA	8103610SX
	AmPAL16R4L/BRA	8103614RX
	AmPAL16R4L/B2C	81036142X
	AmPAL16R4L/BSA	8103614SX
	AmPAL16R4/BRA	8103604RX
	AmPAL16R4/B2C	81036042X

FUNCTIONAL DESCRIPTION

AMD PAL Family Characteristics

All members of the AMD PAL Family have common electrical characteristics and programming procedures. All parts are produced with a fusible link at each input to the AND-gate array, and connections may be selectively removed by applying appropriate voltages to the circuit.

Initially the AND gates are connected, via fuses, to both the TRUE and complement of each input. By selective programming of fuses the AND gates may be "connected" to only the TRUE input (by blowing the complement fuse), to only the complement input (by blowing the TRUE fuse), or to neither type of input (by blowing both fuses) establishing a logical "don't care." When both the TRUE and complement fuses are left intact a logical FALSE results on the output of the AND gate, while all fuses blown results in a logical-TRUE state. The outputs of the AND gates are connected to fixed-OR gates. The only limitations imposed are the number of inputs to the AND gates (up to 16) and the number of AND gates per OR (up to 8).

All parts are fabricated with AMD's fast programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are pre-programmed during manufacturing to ensure extremely high field programming yields (> 98%), and provide extra test paths to achieve excellent parametric correlation.

Power-Up RESET

The registered devices in the AMD PAL family have been designed to reset during system power-up. Following power-up, all registers will be initialized to zero, setting all the outputs to a logic 1. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

PRELOAD

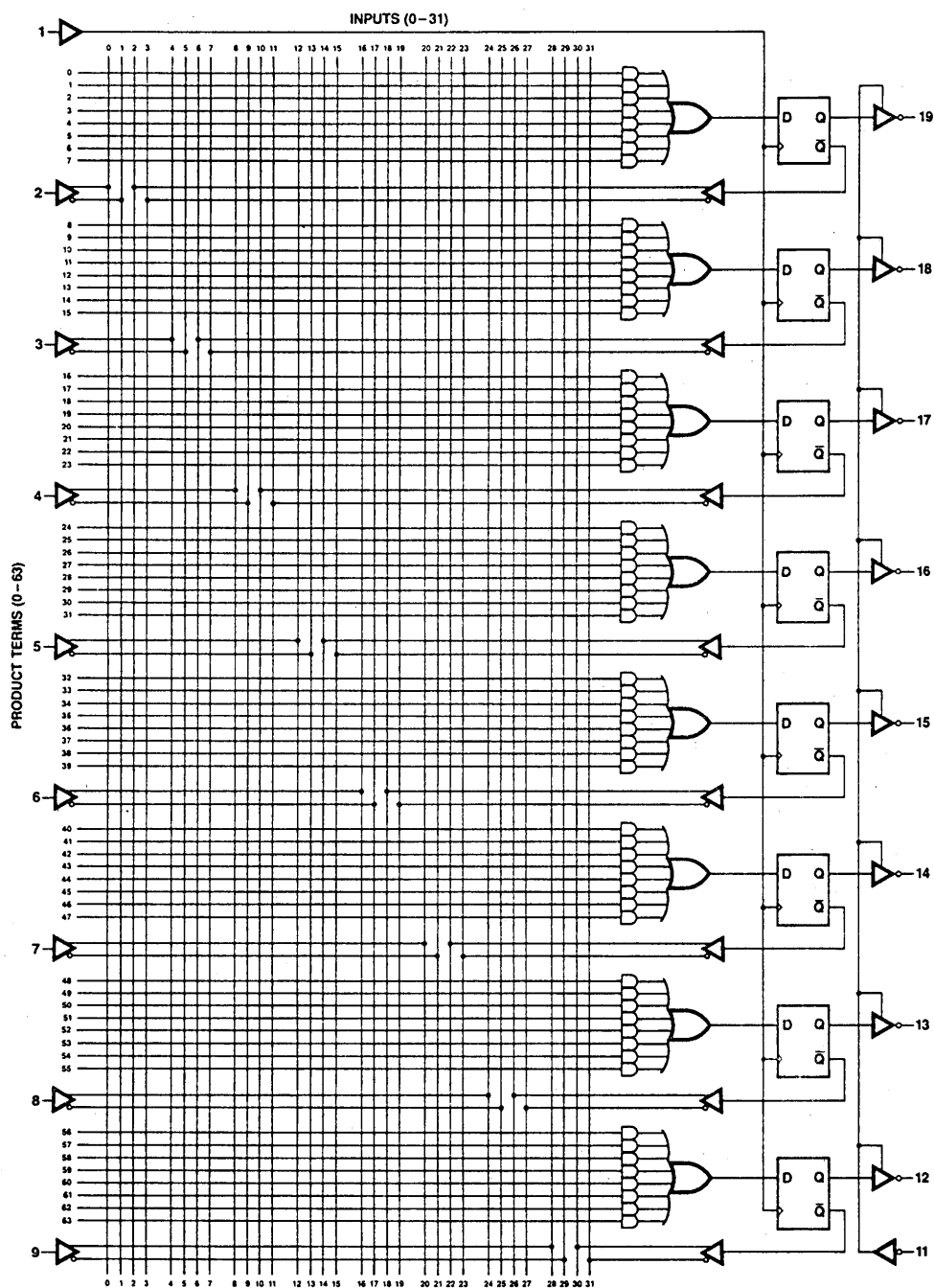
AMD PAL devices are designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality. PRELOAD allows any arbitrary state value to be loaded into the registered output of an AMD PAL device.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to load an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state register "interferes" with the inputs, forcing the machine to go through many transitions before it can reach an arbitrary state value. Therefore the test sequence will be mostly state initialization and not actual testing. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

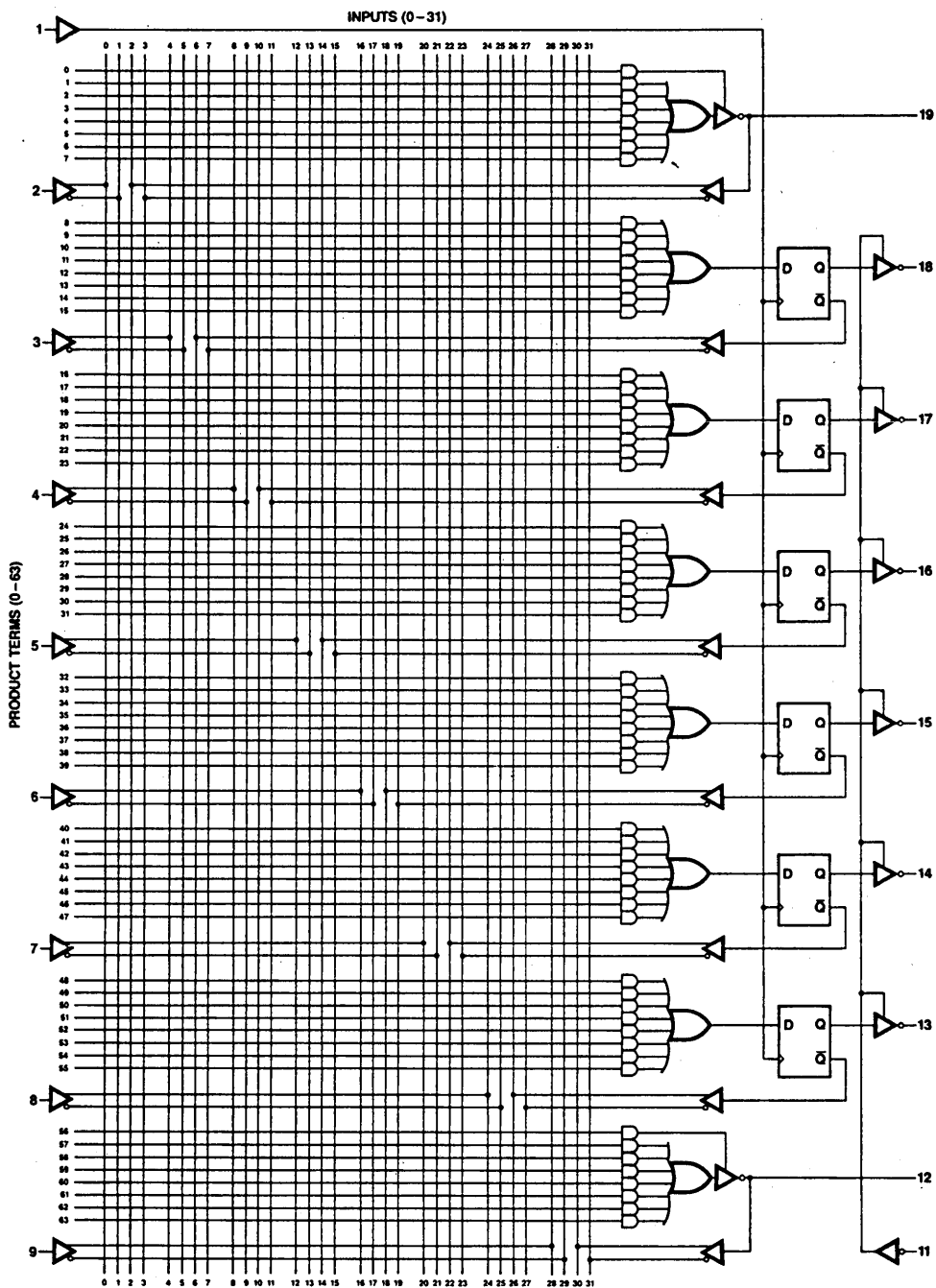
In addition, complete logic verification may become impossible when states that need to be tested cannot be entered with normal state transitions. For example, even though necessary, the state entered when a machine powers up cannot be tested, because it cannot be entered from the main sequence. Similarly, "forbidden" or "don't care" states that are not normally entered need to be tested to ensure that they return to the main sequence.

PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened, and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper in-system operation.



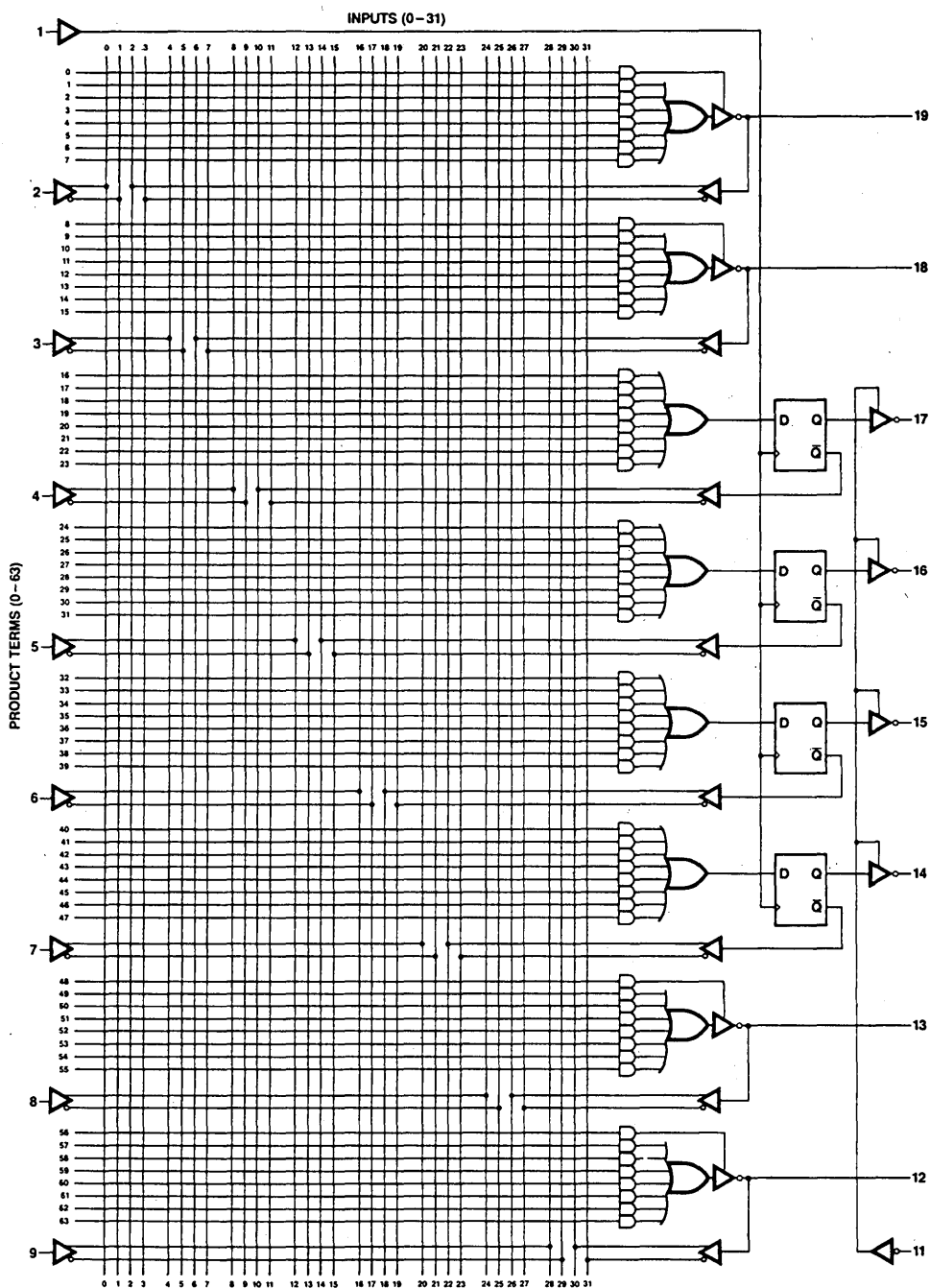
BD002000

Figure 1. AmPAL16R8 Logic Diagram



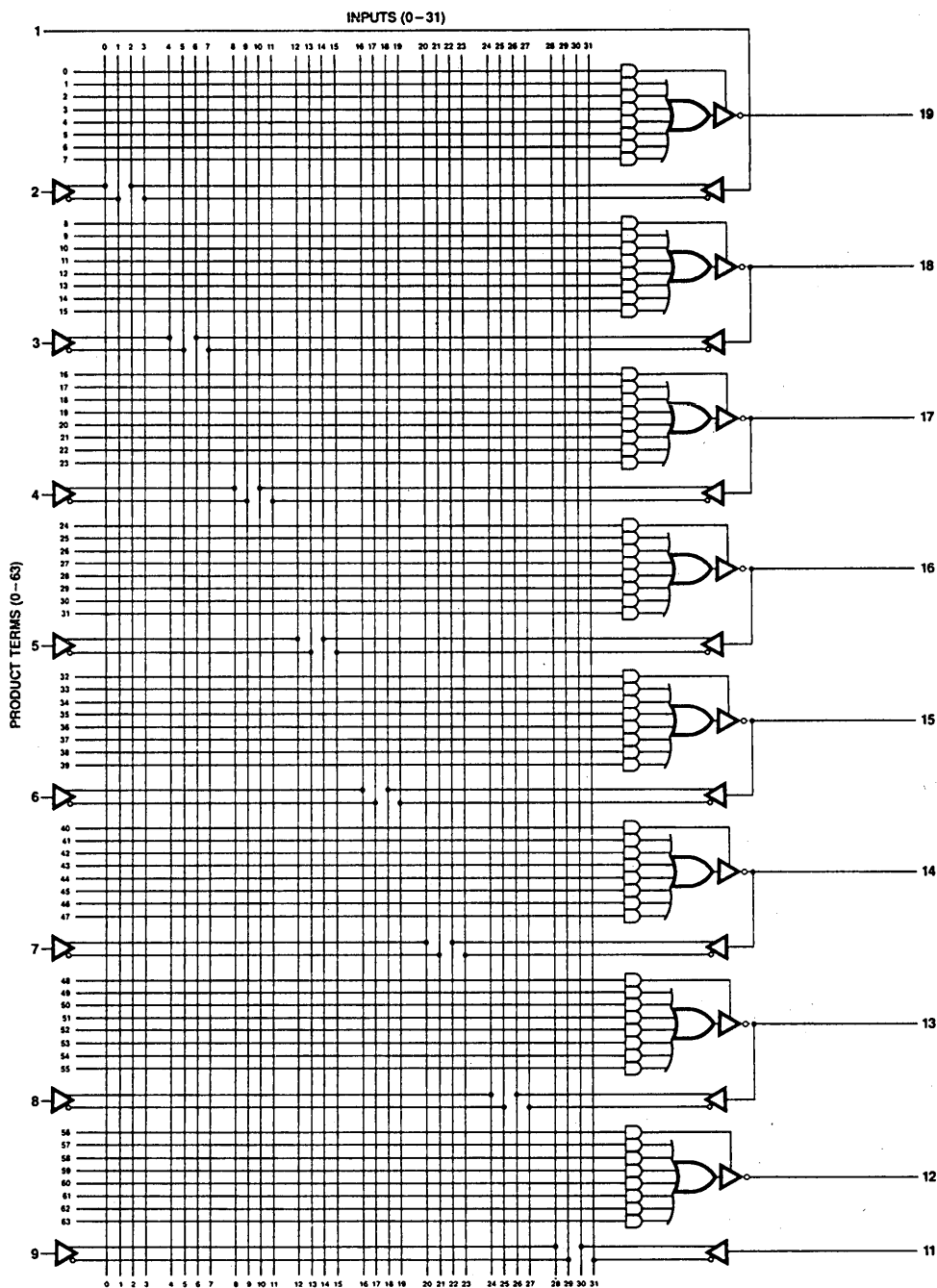
BD001980

Figure 2. AmPAL16R6 Logic Diagram



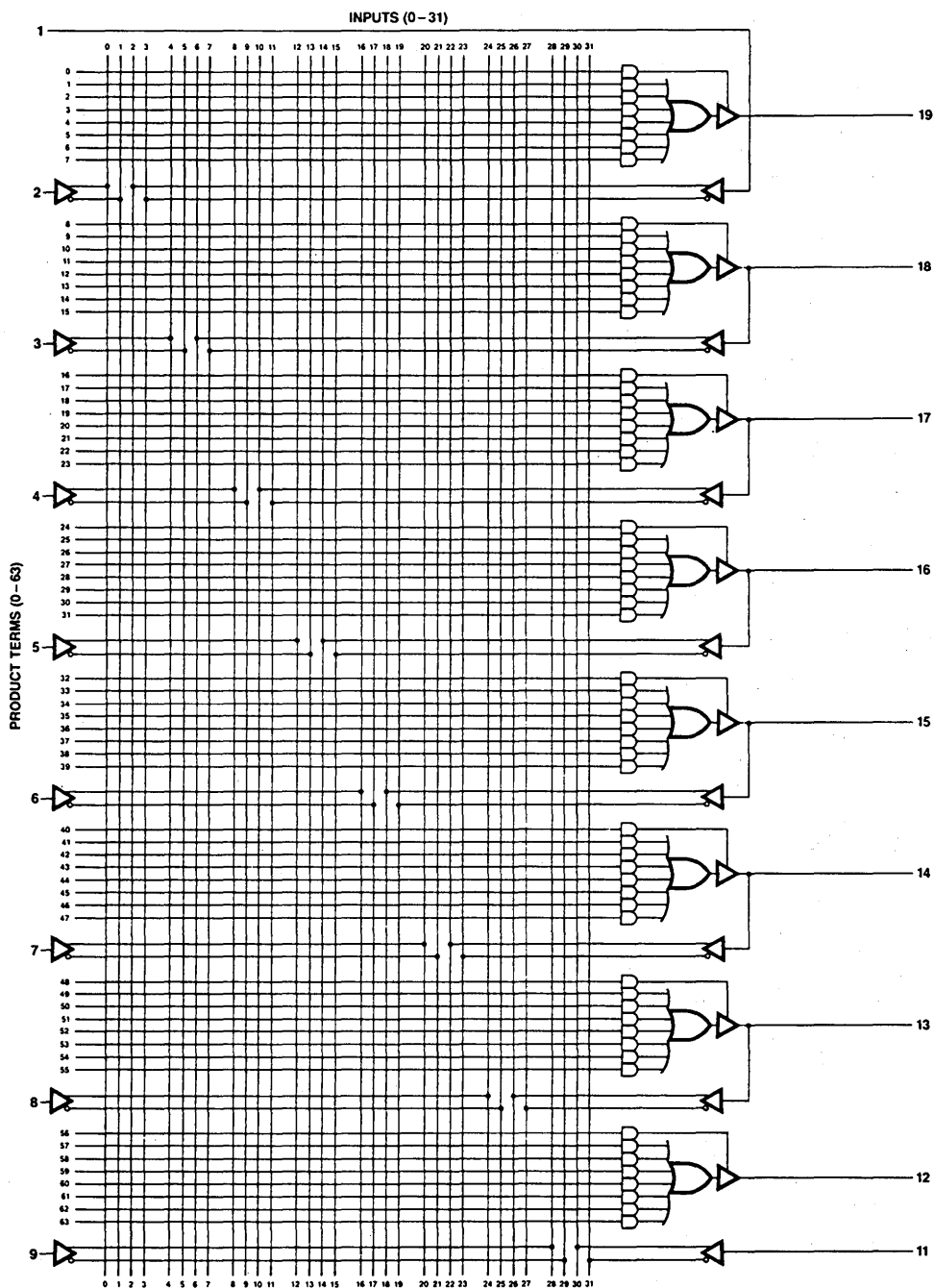
BD001990

Figure 3. AmPAL16R4 Logic Diagram



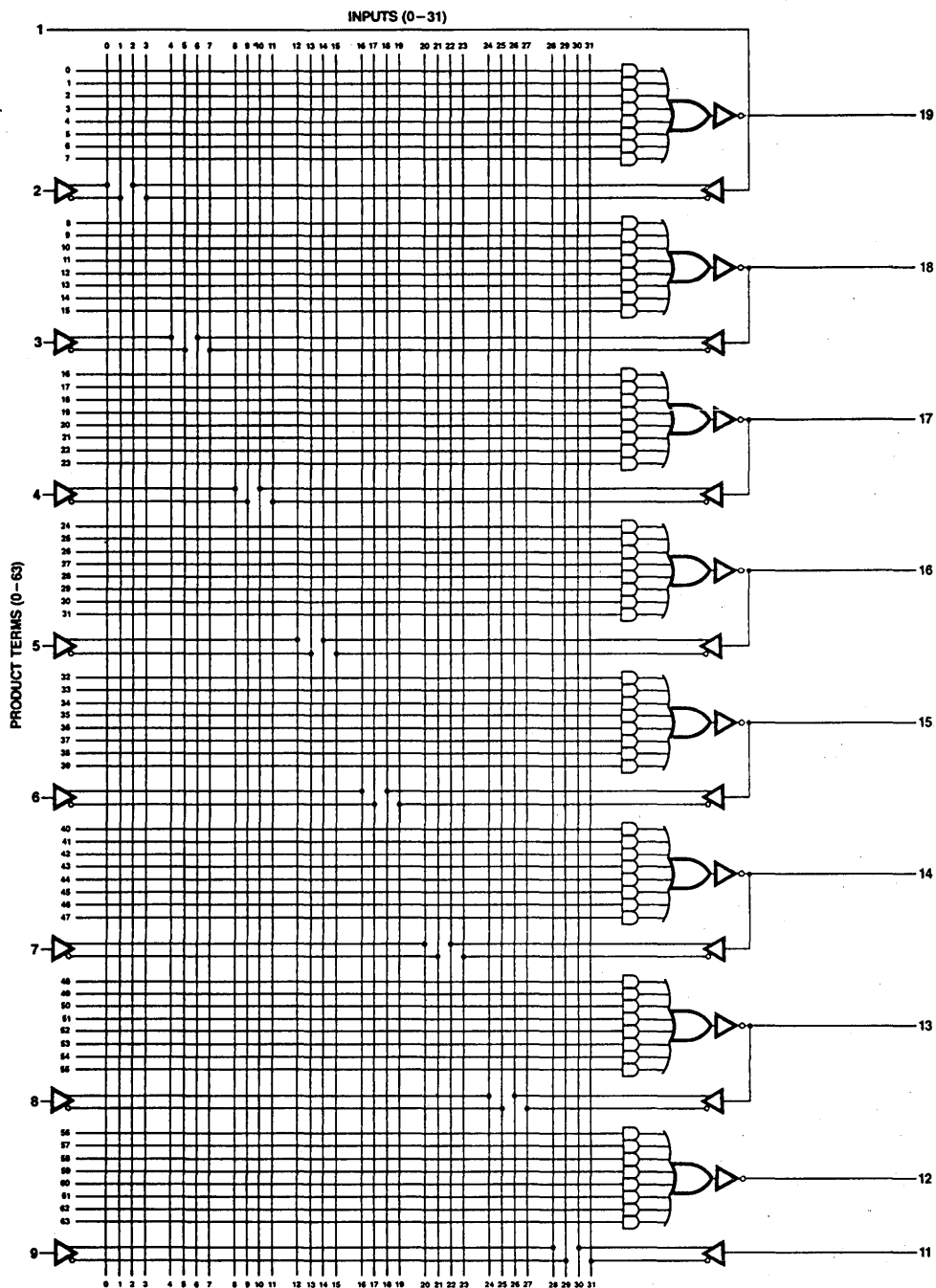
BD002020

Figure 4. AmPAL16L8 Logic Diagram



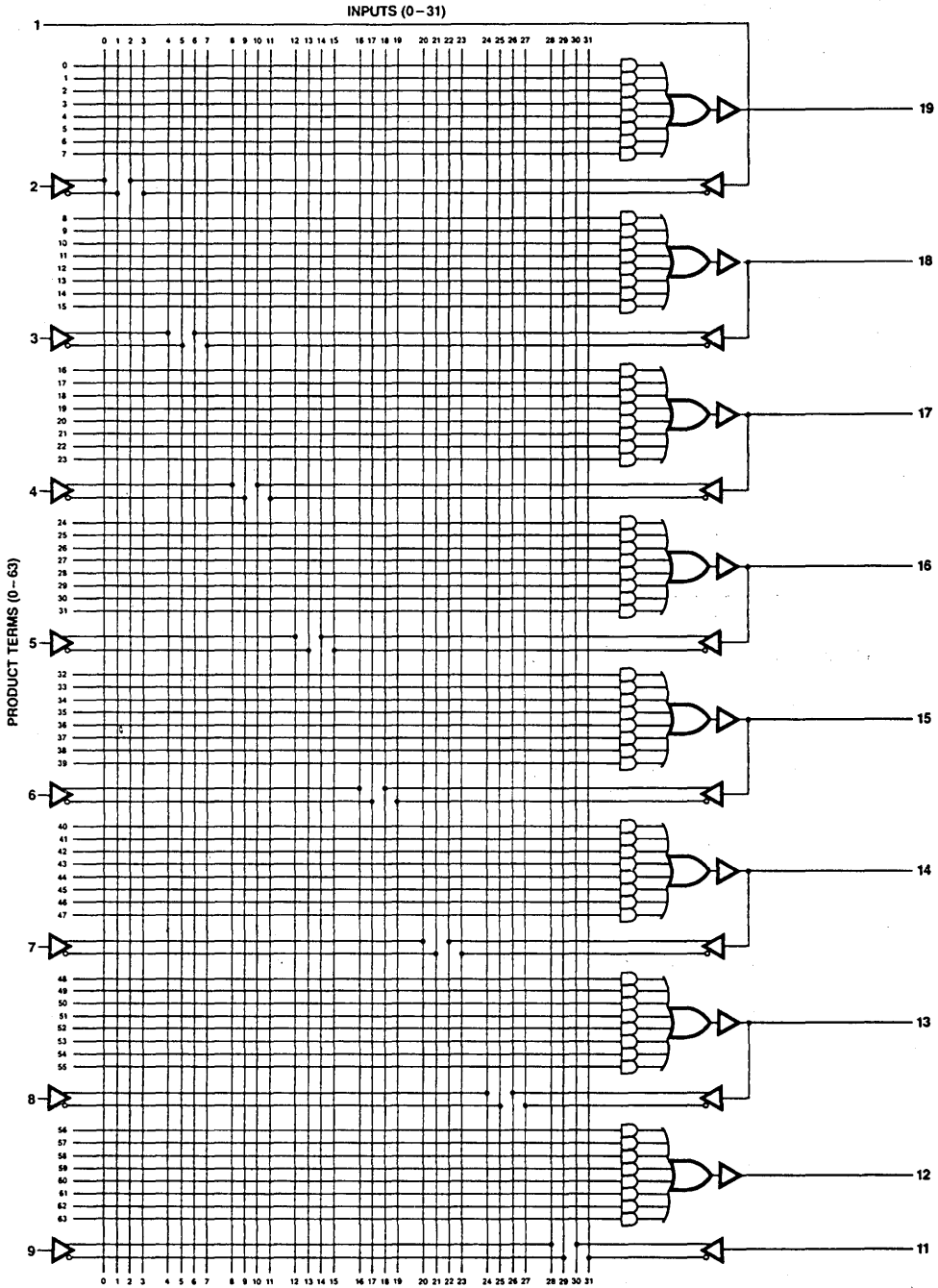
BD002030

Figure 5. AmpAL16H8 Logic Diagram



BD002040

Figure 6. AmPAL16LD8 Logic Diagram



BD002010

Figure 7. AmpAL16HD8 Logic Diagram

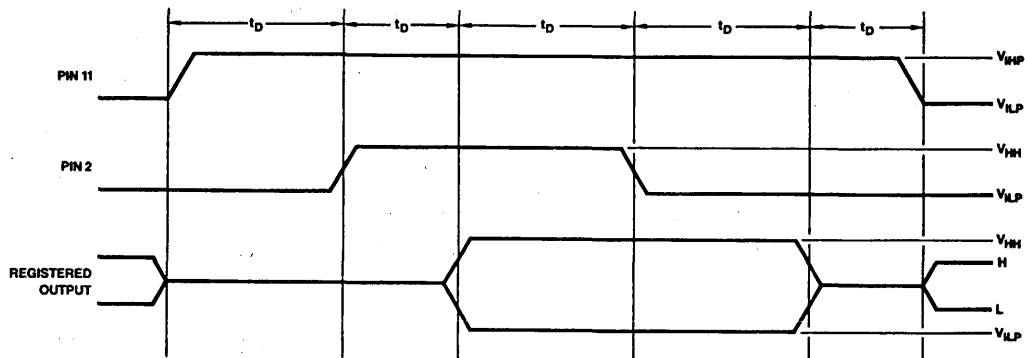
APPLICATIONS

PRELOAD of Registered Outputs

AMD PAL registered outputs are designed with extra circuitry to allow loading each register asynchronously to either a HIGH

or LOW state. This feature simplifies testing since any initial state for the registers can be set to optimize test sequencing.

The pin levels and timing necessary to perform the PRELOAD function are detailed below:



PF001141

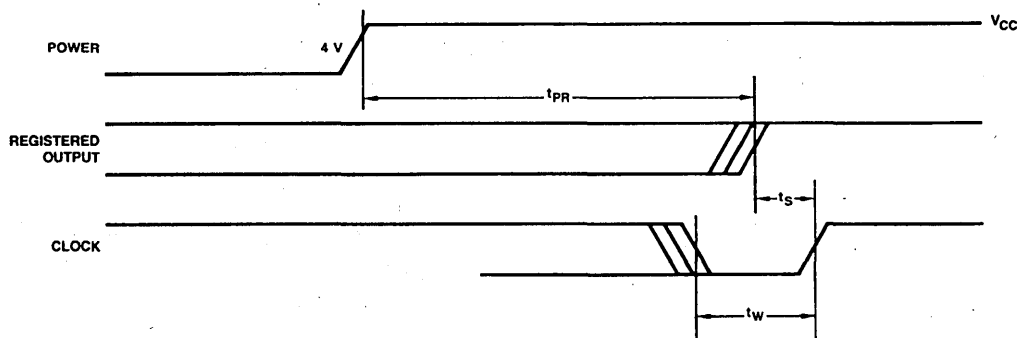
Level forced on registered output pin during PRELOAD cycle	State of the output pin after cycle
V _{HH}	HIGH
0 V to V _{CCH} or OPEN	LOW

Power-Up Reset

The registered devices in the AMD PAL Family have been designed to reset during system power-up. Due to the asynchronous operation of the power-up reset and the wide range of ways V_{CC} can rise to its steady state, two conditions are

required to ensure a valid power-up reset. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from low to high until all applicable input and feedback setup times are met.



WF022300

Parameters	Description	Min.	Typ.	Max.	Units
t _{PR}	Power-Up Reset Time		600	1000	ns
t _S	Input or Feedback Setup Time	See Switching Characteristics			
t _W	Clock Width				

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage to Ground Potential (Pin 20 to Pin 10) Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	21 V
Output Current Into Outputs During Programming (Max Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T _A)	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V
Extended Commercial (E) Devices	
Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V
Military (M) Devices*	
Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3, 4 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions				Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	"Q"	I _{OH} = -2 mA	COM'L	2.4	3.5		V
			All others	I _{OH} = -3.2 mA	COM'L				
				I _{OH} = -2 mA	MIL				
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	"B," "A," "Std." "AL," & "L"	I _{OL} = 24 mA	COM'L			0.5	V
				I _{OL} = 12 mA	MIL				
			"Q"	I _{OL} = 12 mA	COM'L				
				I _{OL} = 8 mA	MIL				
V _{IH} (Note 2)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs				2.0		5.5	V
V _{IL} (Note 2)	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs						0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max., I _{IN} = 0.40 V	"B," "AL," & "Q" "A," "L," & "Std."			-20	-100		μA
						-20	-250		
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V						25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V						1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)				-30	-60	-90	mA
I _{CC}	Power Supply Current	All Inputs = GND, V _{CC} = Max.		16L8A, 16H8A, 16HD8A, 16LD8A, 16L8, 16H8, 16HD8, 16LD8		110	155	mA	
				16L8L, 16H8L, 16HD8L, 16LD8L		55	80		
				16R8B, 16R6B, 16R4B, 16L8B, 16R8A, 16R6A, 16R4A, 16R8, 16R6, 16R4			180		
				16R8L, 16R6L, 16R4L, 16L8AL, 16R8AL, 16R6AL, 16R4AL		60	90		
				16L8Q, 16R8Q, 16R6Q, 16R4Q			45		
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA					-0.9	-1.2	V
I _{OZH} I _{OZL}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V V _O = 0.4 V					100 -100	μA
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz (Note 5)					6		pF
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz (Note 5)					9		

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5V has been chosen to avoid test problems caused by tester ground degradation.

4. I/O pin leakage is the worst case of I_{OZH} or I_I (where X = H or L).

5. These parameters are not 100% tested, but are periodically sampled.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 9, 10, 11 tests unless otherwise noted

COMMERCIAL RANGE

No.	Parameter Symbol	Parameter Description	"B" Version			"A" & "AL" Version			"Std," "L" & "Q" Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 16L8, 16R6, 16R4, 16LD8, 16H8, 16HD8	12		15	17		25	23		35	ns
2	t_{EA}	Input to Output Enable 16L8, 16R6, 16R4, 16H8	12		15	17		25	23		35	ns
3	t_{ER}	Input to Output Disable 16L8, 16R6, 16R4, 16H8	12		15	17		25	23		35	ns
4	t_{PZX}	Pin 11 to Output Enable 16R8, 16R6, 16R4	8		15	12		20	17		25	ns
5	t_{PXZ}	Pin 11 to Output Disable 16R8, 16R6, 16R4	8		15	12		20	17		25	ns
6	t_{CO}	Clock to Output 16R8, 16R6, 16R4	8		12	12		15	17		25	ns
7	t_S	Input or Feedback Setup Time 16R8, 16R6, 16R4	10	13		15	20		20	30		ns
8	t_H	Hold Time 16R8, 16R6, 16R4	-8	0		-10	0		-10	0		ns
9	t_P	Clock Period ($t_S + t_{CO}$)		25			35			55		ns
10	t_W	Clock Width		10			15			25		ns
11	f_{MAX}	Maximum Frequency			40			28.5			18	MHz

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.

2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.

3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

MILITARY RANGE

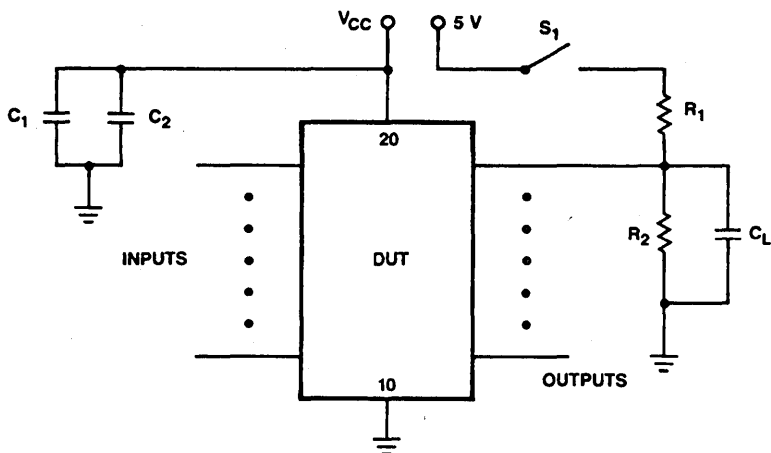
No.	Parameter Symbol	Parameter Description	"B" Version			"A" & "AL" Version			"Std," "L" & "Q" Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 16L8, 16R6, 16R4, 16LD8, 16H8, 16HD8	12		20	17		30	23		40	ns
2	t_{EA}	Input to Output Enable 16L8, 16R6, 16R4, 16H8	12		20	17		30	23		40	ns
3	t_{ER}	Input to Output Disable 16L8, 16R6, 16R4, 16H8	12		20	17		30	23		40	ns
4	t_{PZX}	Pin 11 to Output Enable 16R8, 16R6, 16R4	8		20	12		25	17		25	ns
5	t_{PXZ}	Pin 11 to Output Disable 16R8, 16R6, 16R4	8		20	12		25	17		25	ns
6	t_{CO}	Clock to Output 16R8, 16R6, 16R4	8		15	12		20	17		25	ns
7	t_S	Input or Feedback Setup Time 16R8, 16R6, 16R4	10	18		15	25		20	35		ns
8	t_H	Hold Time 16R8, 16R6, 16R4	-8	0		-10	0		-10	0		ns
9	t_P	Clock Period ($t_S + t_{CO}$)		33			45			60		ns
10	t_W	Clock Width		12			20			25		ns
11	f_{MAX}	Maximum Frequency			30			22			16.5	MHz

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.

2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.

3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

SWITCHING TEST CIRCUIT



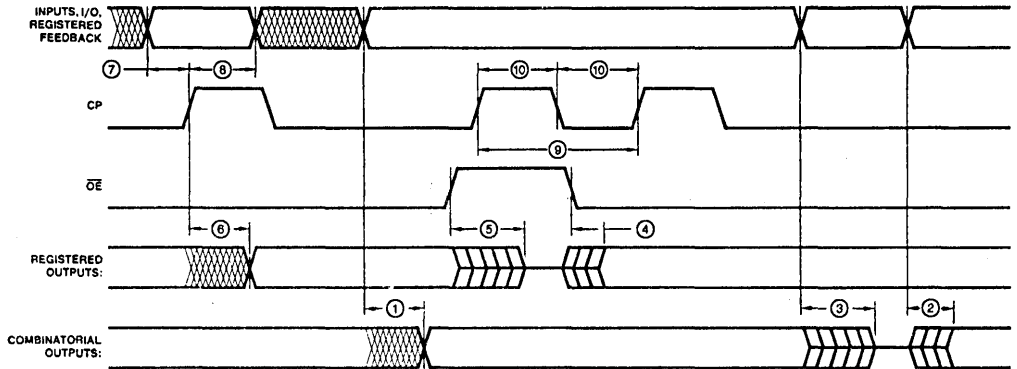
TC003050

Note: C_1 and C_2 are to bypass V_{CC} to ground.

TEST OUTPUT LOADS

Pin Name	"Std," "B," "A," "AL" & "L"		"Q"	
	Commercial	Military	Commercial	Military
R_1	200 Ω	390 Ω	390 Ω	600 Ω
R_2	390 Ω	750 Ω	750 Ω	1200 Ω
C_1	1 μF	1 μF	1 μF	1 μF
C_2	0.1 μF	0.1 μF	0.1 μF	0.1 μF
C_L	50 pF	50 pF	50 pF	50 pF

SWITCHING WAVEFORMS



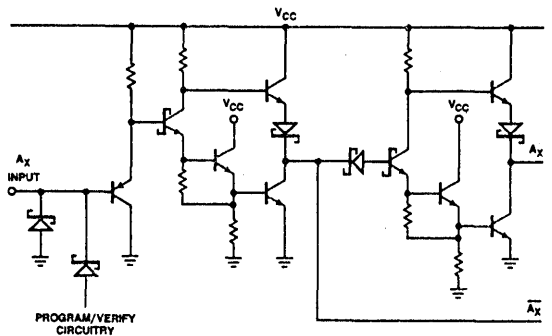
WF002571

KEY TO TIMING DIAGRAM

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

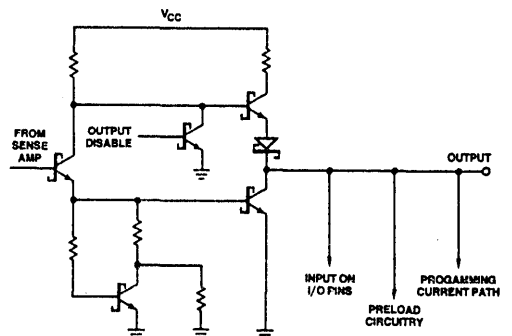
KS000010

INPUT CIRCUITRY



IC000720

OUTPUT CIRCUITRY



IC000730

PROGRAMMING

Each AMD PAL fuse is programmed with a simple sequence of voltages applied to two control pins (1 and 11) and programming voltage pulse applied to the output under programming. Addressing of the 2048 element fuse array is accomplished with normal TTL levels on eight input pins (five select the input line number and three select the product term number). V_{CC} is maintained at a normal level throughout the programming and verify cycle – no extra high levels are required.

The necessary sequence levels for programming any fuse is shown in the Programming Waveforms. The address of each fuse in terms of Input Line Number and Product Term Line Number is defined by the Fuse Address Tables 1 and 2. Current, voltage and timing requirements for each pin are specified in the Programming Parameter Table below.

The 16L8, 16R8, 16R6, 16R4, 16H8, 16LD8 and 16HD8 use identical programming conditions and sequences.

After all programming has been completed, the entire array should be reverified at V_{CCL} and again at V_{CCH} . Reverification can be accomplished by reading all eight outputs in parallel rather than one at a time. The array fuse verification cycle checks that the correct array fuses have been blown and can be sensed by the outputs.

AMD PAL devices have been designed with many internal test features that are used to assure high programming yield and correct logical operation for a correctly programmed part.

An additional fuse is provided on each AMD PAL circuit to prevent unauthorized copying of AMD PAL fuse patterns when design security is desired. Blowing the security fuse blocks entry to the fuse pattern verify mode.

To blow the security fuse:

1. Power up part to V_{CCP}
2. Raise Pin 5 to V_{HH} .
3. Pulse Pin 11 from ground to V_{OP} for a 50 μ sec duration.
4. Perform a normal end-of-programming verify cycle at V_{CCL} and V_{CCH} . All fuse locations should be sensed as blown if the security fuse has been successfully blown.

Note that parts with the security fuse blown may not be returned as programming rejects.

AMD PAL devices normally have high programming yields (> 98%). Programming yield losses are frequently due to poor socket contact, equipment out of calibration or improperly used.

PROGRAMMING PARAMETERS $T_A = 25^\circ C$

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V _{HH}	Control Pin Extra High Level	Pin 1 @ 10-40 mA	10	11	12	V
		Pin 11 @ 10-40 mA	10	11	12	
V _{OP}	Program Voltage Pins 12-19 @ 15-200 mA		18	20	22	V
V _{IHP}	Input HIGH Level During Programming and Verify		2.4	5	5.5	V
V _{ILP}	Input LOW Level During Programming and Verify		0.0	0.3	0.5	V
V _{CCP}	V _{CC} During Programming @ I _{CC} = 50-200 mA		5	5.2	5.5	V
V _{CCL}	V _{CC} During First Pass Verification @ I _{CC} = 50-200 mA		4.1	4.3	4.5	V
V _{CCH}	V _{CC} During Second Pass Verification @ I _{CC} = 50-200 mA		5.4	5.7	6.0	V
V _{Blown}	Successful Blown Fuse Sense Level @ Output	16L8, 16R8, 16R6, 16R4, 16LD8		0.3	0.5	V
		16H8, 16HD8	2.4	3		
V _{OP} /dt	Rate of Output Voltage Change		20		250	V/μs
dV ₁₁ /dt	Rate of Fusing Enable Voltage Change (Pin 11 Rising Edge)		100		1000	V/μs
t _p	Fusing Time First Attempt		40	50	100	μs
	Subsequent Attempts		4	5	10	ms
t _D	Delays Between Various Level Changes		100	200	1000	ns
t _v	Period During which Output is Sensed for V _{Blown} Level				500	ns
V _{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		V _{CCP} - 0.3	V _{CCP}	V _{CCP} + 0.3	V
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	kΩ

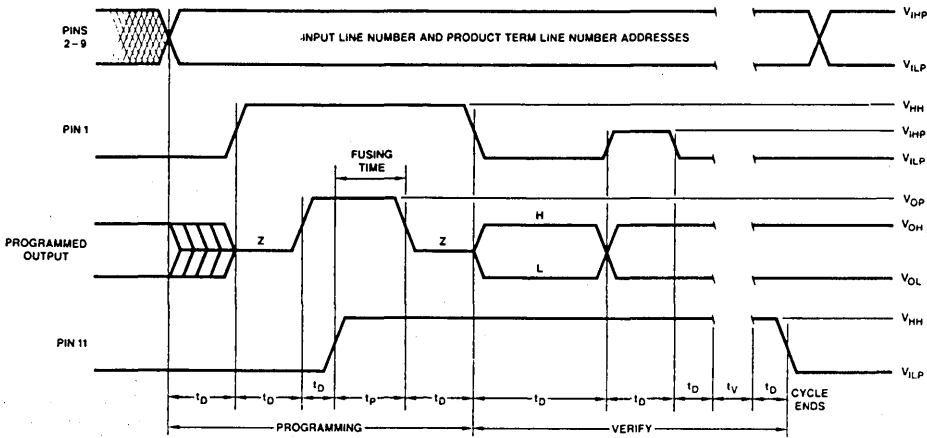
Design Aid Software for AmPAL16XX Family

Name	Vendor	Versions	Notes
ABEL	Data I/O (206) 881-6444	IBM PC VAX/VMS VAX/UNIX	Rev 1.1
CUPL	P-CAD Systems (408) 971-1300	IBM PC VAX/VMS VAX/UNIX CPM 80/86	Rev 2.1
AmCUPL	Advanced Micro Devices (408) 732-2400	IBM PC	Supported by P-CAD Systems

AMD Qualified Programmers

Name	Programmer Model(s)	AMD PAL Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 93052	Systems 19, 29	950-1942-0044	303A-004, Rev 3 or newer
	60	N/A	360A-001, Rev 4 or newer
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086	Model PPZ	2200	On Board
	ZL30	On Board Module (Rev 38 or newer)	
Structured Design 1700 Wyatt Drive Suite 3 Santa Clara, CA 95084	SD-1000J	N/A	On Board
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043	160 Series	N/A	On Board
Digelec 586 Weddell Drive Suite 1 Sunnyvale, CA 94089	803 Series	FAM-52	DA-53
JMC 2999 Monterey Rd. Monterey, CA 93940	PROMAC P3	On-board Module rev 2.0	On Board

PROGRAMMING TIMING DIAGRAM



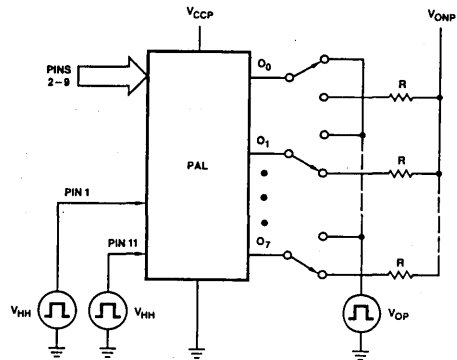
PF001100

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States				
	9	8	7	6	5
0	L	L	L	L	L
1	L	L	L	L	H
2	L	L	L	H	L
3	L	L	L	H	H
4	L	L	L	H	L
5	L	L	L	H	H
6	L	L	L	H	L
7	L	L	L	H	H
8	L	L	L	H	L
9	L	L	L	H	H
10	L	L	L	H	L
11	L	L	L	H	H
12	L	L	L	H	L
13	L	L	L	H	H
14	L	L	L	H	L
15	L	L	L	H	H
16	H	L	L	L	L
17	H	L	L	L	H
18	H	L	L	H	L
19	H	L	L	H	H
20	H	L	L	H	L
21	H	L	H	L	H
22	H	L	H	H	L
23	H	L	H	H	H
24	H	H	L	L	L
25	H	H	L	L	H
26	H	H	L	H	L
27	H	H	L	H	H
28	H	H	H	L	L
29	H	H	H	L	H
30	H	H	H	H	L
31	H	H	H	H	H

L = V_{ILP}
H = V_{IHP}

SIMPLIFIED PROGRAMMING DIAGRAM



PF000380

TABLE 2. PRODUCT TERM ADDRESSING

								Product Term Select Address Pin		
Product Term Line Number								4	3	2
0	8	16	24	32	40	48	56	L	L	L
1	9	17	25	33	41	49	57	L	L	H
2	10	18	26	34	42	50	58	L	H	L
3	11	19	27	35	43	51	59	L	H	H
4	12	20	28	36	44	52	60	H	L	L
5	13	21	29	37	45	53	61	H	L	H
6	14	22	30	38	46	54	62	H	H	L
7	15	23	31	39	47	55	63	H	H	H
Pin 19	Pin 18	Pin 17	Pin 16	Pin 15	Pin 14	Pin 13	Pin 12			
Programming Access and Verify Pin										

L = V_{ILP}
H = V_{IHP}

AmPAL*16XXD Family

20-Pin IMOX™ Ultra High-Speed Programmable Array Logic (PAL) Elements

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- AMD's superior IMOX technology
 - Guarantees $t_{PD} = 10$ ns Max. "D" version
- Very high-speed, half-power ("BL") and high-speed, quarter-power ("AQ") versions
- Platinum-silicide fuses and added test words ensure programming yields > 98%
- Post Programming Functional Yields (PPFY) of 99.9%
- PRELOAD feature permits full logical verification
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parametric testing at the factory through on-board testing circuitry
- AMD's industry-leading quality guarantees

GENERAL DESCRIPTION

AMD PAL devices are high-speed, electrically programmable array logic elements. They utilize the familiar sum-of-products (AND-OR) structure allowing users to program custom logic functions to fit most applications precisely. Typically they are a replacement for low-power Schottky SSI/MSI logic circuits, reducing chip count by more than 5 to 1 and greatly simplifying prototyping and board layout.

Four different devices are available, including both registered and combinatorial devices. These three new speed and power options extend the current AmPAL16XX Family, allowing the designer to precisely match system requirements.

Please see the "AmPAL16XX Family" data sheet (Publication No. 03323D) for Block Diagrams.

PRODUCT SELECTOR GUIDE

AMD PAL Speed/Power Families

Family	t_{PD} ns (Max.)	t_S (1) ns (Max.)	t_{CO} ns (Max.)	I_{CC} (2) mA (Max.)	I_{OL} mA (Min.)
Ultra High-Speed ("D") Versions	10	10	8	180	24
Very High-Speed, Half-Power ("BL") Versions	15	13	12	90	24
High-Speed, Quarter-Power ("AQ") Versions	25	20	15	55	12

- (1) Sequential functions
(2) Combinatorial functions

AMD PAL FUNCTIONS

Part Number	Array Inputs	Logic	Output Enable	Outputs	Package Pins
16R8	Eight Dedicated, Eight Feedback	Eight 8-Wide AND-OR	Dedicated	Registered Inverting	20
16R6	Eight Dedicated, Six Feedback, Two Bidirectional	Six 8-Wide AND-OR	Dedicated	Registered Inverting	20
		Two 7-Wide AND-OR-INVERT	Programmable	Bidirectional	
16R4	Eight Dedicated, Four Feedback, Four Bidirectional	Four 8-wide AND-OR	Dedicated	Registered Inverting	20
		Four 7-Wide AND-OR-INVERT	Programmable	Bidirectional	
16L8	Ten Dedicated, Six Bidirectional	Eight 7-Wide AND-OR-INVERT	Programmable	Six Bidirectional Two Dedicated	20

IMOX is a trademark of Advanced Micro Devices, Inc.

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

Publication # 08486 Rev. A Amendment /0
Issue Date: September 1986

AmPAL*18P8

20-Pin IMOX™ Programmable Array Logic

AmPAL*18P8

DISTINCTIVE CHARACTERISTICS

- Individually programmable output polarity on each output
- Pin compatible superset of most combinatorial 20-pin PALs
- Eight logical product terms per output for increased logic power
- Increased input/output flexibility
 - 18 possible array inputs
 - Eight bidirectional I/Os with individually controllable output enable
- Ultra high-speed version $t_{pd} = 15$ ns maximum
- Superior quality
 - Full AC and DC parametric testing performed on every part
 - Extensive on-chip test circuitry ensures post-programming functional yield (PPFY) of 99.9%
- Platinum-Silicide fuses ensure high programming yield > 98%, fast programming and unsurpassed reliability

GENERAL DESCRIPTION

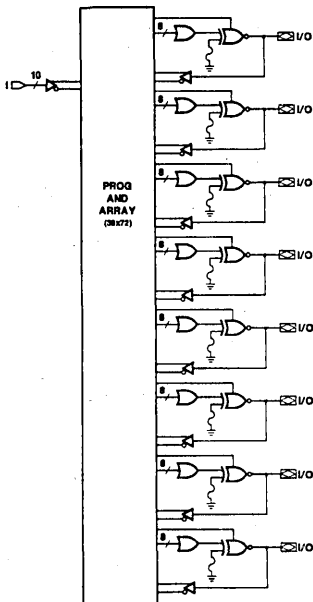
The AmPAL18P8 is an ultra high-performance, functionally enhanced 20-pin Programmable Array Logic element. It utilizes the familiar sum-of-products (AND-OR) structure allowing users to program custom logic functions to precisely fit their application.

The AmPAL18P8 offers significantly enhanced functional capabilities when compared to other combinatorial 20-pin PAL devices. These include two additional bidirectional I/O pins as well as additional product terms (bringing each output to eight logical and one three-state control product

term) for extra logic power. The device also features individually user programmable output polarity, giving the designer the capability to handle both active HIGH and active LOW outputs on the same device.

A wide variety of speed/power selections is available, allowing precise matching to system requirements. The ultra high-speed version offers 15 ns maximum input to output propagation delay, opening up many new applications for the use of programmable logic.

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Family Part No.	AmPAL18P8									
Power Grade	Quarter Power				Half Power				Full Power	
Ordering Part No.	18P8Q		18P8L		18P8AL		18P8A		18P8B	
Speed Grade	Standard Speed				High Speed				Ultra High Speed	
Max. Access Time (ns)	STD	APL	STD	APL	STD	APL	STD	APL	STD	APL
	35	40	35	40	25	30	25	30	15	20
Max. Operating Current (mA)	55		90				180			

STD = AMD "Standard" products

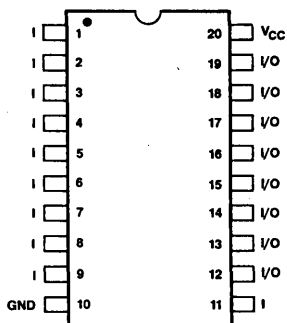
APL = AMD "Approved Products List" products

BD005942

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.
IMOX is a trademark of Advanced Micro Devices, Inc.

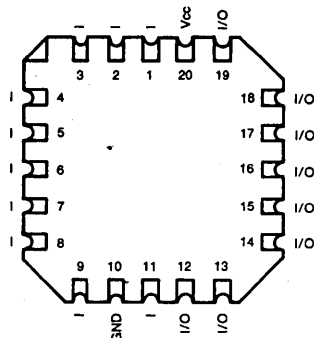
CONNECTION DIAGRAMS

Top View



CD009210

LCC*



CD009220

Note: Pin 1 is marked for orientation.

*Same Pinouts apply for PLCC.

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/Power Option**
- C. Package Type**
- D. Temperature Range**
- E. Optional Processing**

AMPAL18P8

B

P

C

E. OPTIONAL PROCESSING

Blank = Standard processing

D. TEMPERATURE RANGE

C = Commercial (0 to +70°C)

E = Extended Commercial (-55 to +125°C)

C. PACKAGE TYPE

P = 20-Pin Plastic DIP (PD 020)

D = 20-Pin Ceramic DIP (CD 020)

J = 20-Pin Plastic Leadless Chip Carrier (PL 020)

L = 20-Pin Ceramic Leadless Chip Carrier (CL 020)

B. SPEED/POWER OPTION

B = Ultra High Speed/Standard Power

A = High Speed/Standard Power

AL = High Speed/Half Power

L = Standard Speed/Half Power

Q = Standard Speed/Quarter Power

A. DEVICE NUMBER/DESCRIPTION

AmPAL18P8
20-Pin IMOX Programmable
Array Logic

Valid Combinations

Valid Combinations	
AMPAL18P8B	PC, DC, DE, JC, LC, LE
AMPAL18P8A	
AMPAL18P8AL	
AMPAL18P8L	
AMPAL18P8Q	

Valid Combinations

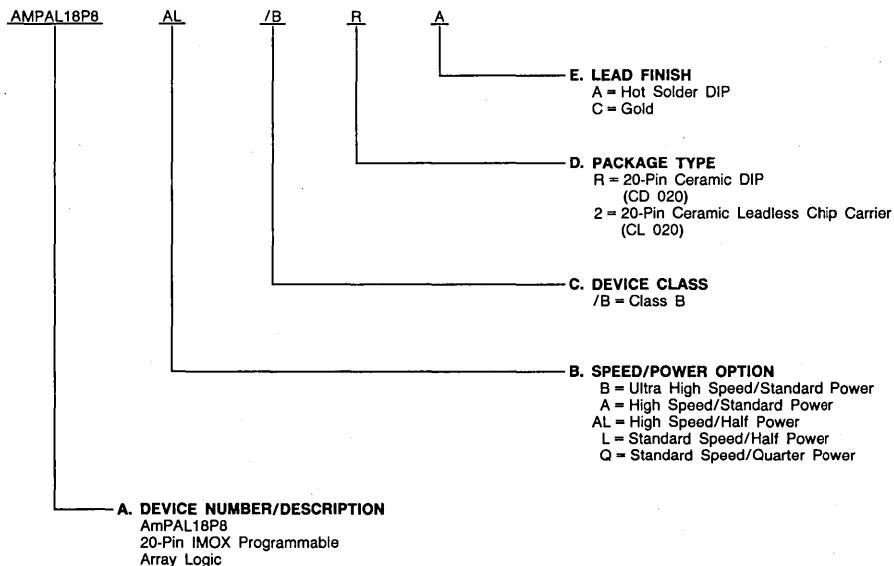
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number**
- B. Speed/Power Option**
- C. Device Class**
- D. Package Type**
- E. Lead Finish**



Valid Combinations	
AMPAL18P8B	/BRA, /B2C
AMPAL18P8A	
AMPAL18P8AL	
AMPAL18P8L	
AMPAL18P8Q	

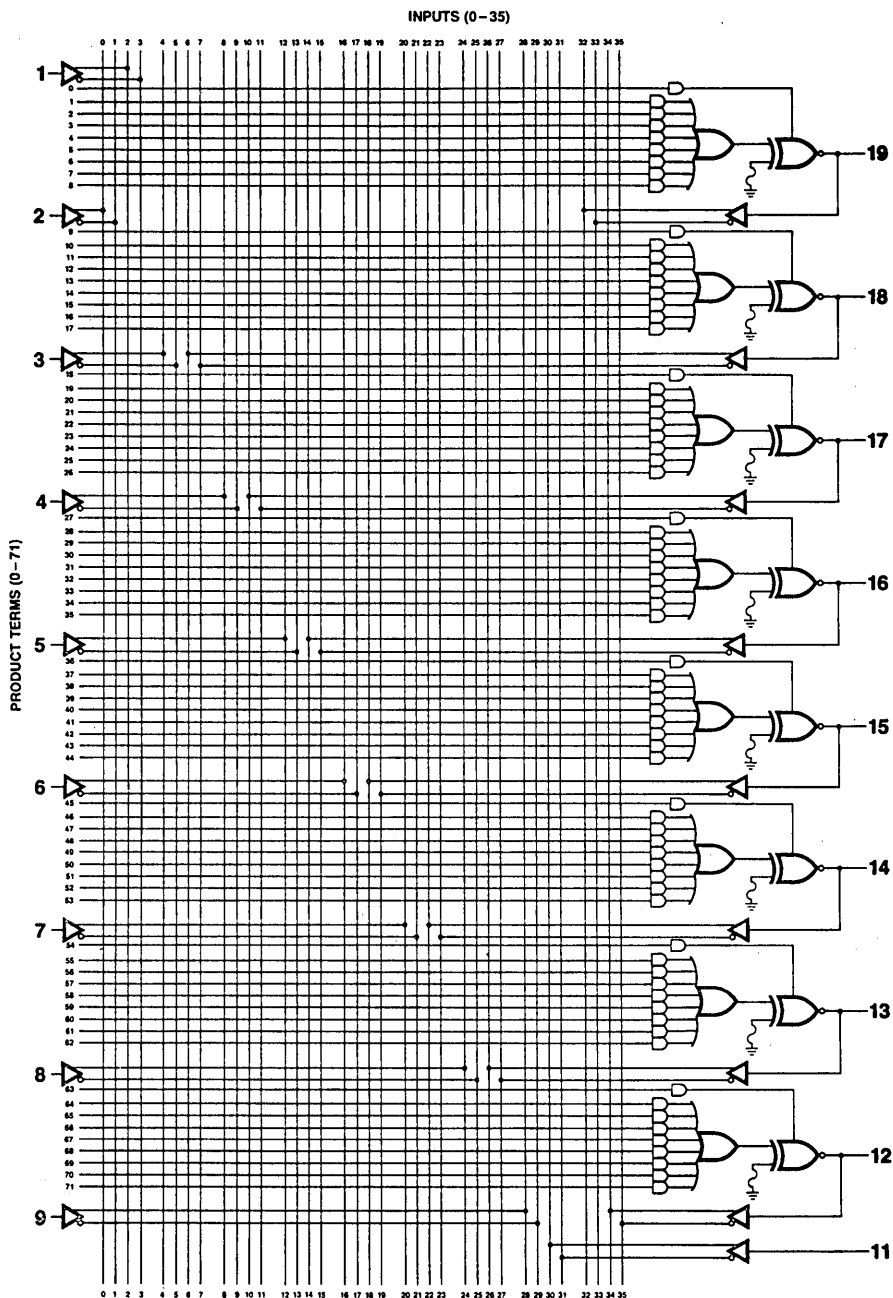
Valid Combinations

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 consist of Subgroups
 1, 2, 3, 7, 8, 9, 10, 11

LOGIC DIAGRAM

**Eighteen Array Inputs**

- 10 dedicated
- 8 bidirectional I/O

Eight 8-Wide AND-OR Structures

- Combinatorial outputs
- Programmable output enable for each output
- Programmable polarity on each output

LD000040

FUNCTIONAL DESCRIPTION

The AmPAL18P8 is a functionally enhanced Programmable Array Logic (PAL) device. The Block Diagram on page one shows the basic architecture of the AmPAL18P8. There are up to eighteen inputs and eight outputs available. The inputs are connected to a programmable AND array which contains 72 logical product terms. Initially the AND gates are connected, via fuses, to both the true and complement of every input. By selective programming of fuses, the AND gates may be "connected" to only the true input (by blowing the complement fuse) or to only the complement input (by blowing the true fuse), or to neither type of input (by blowing both fuses), establishing a logical "don't care." When both the true and complement fuses are left intact, a logical false results on the output of the AND gate. An AND gate with all fuses blown will assume the logical true state.

The AmPAL18P8 has a possible maximum of 18 input pins, two more than previous 20-pin PALs. The extra inputs extend the functional capabilities of the device, which reduces design limitations, making it easier to design with and more flexible.

The AmPAL18P8 can be programmed with more complex logic equations due to the eight product terms and one control term for each output. The control terms also allow for each of the eight bi-directional I/Os to be three-stated, greatly expanding the realm of design possibilities.

The eight bi-directional I/O pins enhance the usefulness of the AmPAL18P8 by allowing for greater complexity of logic equations and hence more logic power.

The AmPAL18P8 also has programmable output polarity, giving the designer the choice of either active HIGH or active

LOW on each of the eight outputs. This simplifies the task of programming the AmPAL18P8 and allows more freedom in optimizing the logic functions. The high-speed version of the AmPAL18P8 boasts 15 ns maximum input-to-output propagation delay, which makes it the fastest TTL-compatible PAL on the market today, and creates new possibilities for the use of programmable logic devices in a wide variety of applications.

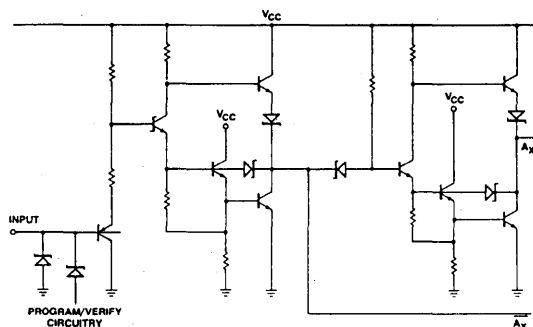
The AmPAL18P8 is manufactured using Advanced Micro Devices' IMOX oxide isolation process. This advanced process permits an increase in density and a decrease in internal capacitance, resulting in the fastest possible programmable logic devices. The AmPAL18P8 is fabricated with AMD's fast-programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern.

Platinum-Silicide was selected as the fuse-link material to achieve a well-controlled melt rate, resulting in large non-conductive gaps that ensure very stable, long-term reliability. Extensive operating testing has proven that this low-field, large gap technology offers the best reliability for fusible link programmable logic.

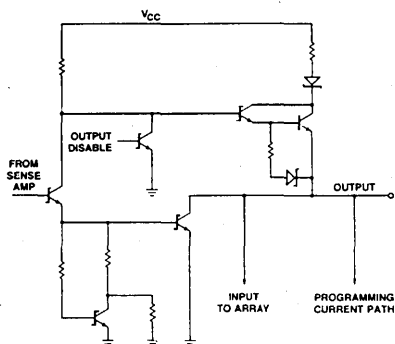
The AmPAL18P8 has been designed with extensive internal test circuitry that allows the programming and operating circuitry in the part to be thoroughly tested at the factory before programming. This assures excellent programming yield and functional performance to data sheet parameters after programming. The Post-Programming Functional Yield (PPFY) for this device is consistently better than 99.9%.

INPUT/OUTPUT DIAGRAMS

Input



Output



IC000803

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage	
with Respect to Ground	-0.5 to +7.0 V
DC Voltage Applied to Outputs	
(except during programming)	-0.5 to +V _{CC} Max.
DC Voltage Applied to	
Outputs During Programming	16 V
Output Current Into Outputs	
During Programming	
(Maximum duration of 1 second)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5.0 mA

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES**Commercial (C) Devices**

Temperature (T _A)	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V

Extended Commercial (E) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Military (M) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions				Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -3.2 mA	18P8A, 18P8B 18P8L, 18P8AL	COM'L	2.4	3.5		Volts
	I _{OH} = -2 mA		18P8Q						
	I _{OH} = -2 mA		(all versions)		MIL				
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = 24 mA	18P8A, 18P8B 18P8L, 18P8AL	COM'L			0.50	Volts
	I _{OL} = 12 mA		18P8Q						
	I _{OL} = 12 mA		A, B, AL, L		MIL				
	I _{OL} = 8 mA		18P8Q						
V _{IH} (Note 2)	Input HIGH level	Guaranteed Input Logical HIGH Voltage for All Inputs				2.0			Volts
V _{IL} (Note 2)	Input LOW level	Guaranteed Input Logical LOW Voltage for All inputs						0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.40 V					-20	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V						25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V						1.0	mA
I _{SC}	Output Short Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)				-30	-60	-90	mA
I _{CC}	Power Supply Current	V _{CC} = Max.	18P8A, 18P8B				180	mA	
			18P8L, 18P8AL				90		
			18P8Q				55		
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA					-0.9	-1.2	Volts
I _{OZH}	Output Leakage Current	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V					100	μA
I _{OZL}			V _O = 0.4 V					-250	
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz					6		pF
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz					9		

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5 V has been chosen to avoid test problems caused by tester ground degradation.

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Units
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz	6	pF
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz	9	

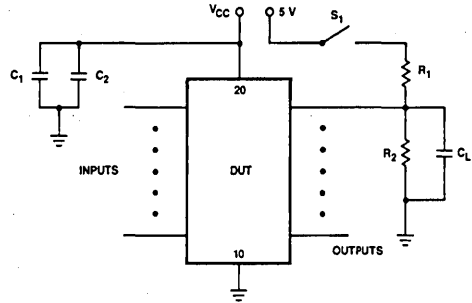
Note: These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING TEST CIRCUIT



TC003050

Note: C_1 and C_2 are to bypass V_{CC} to ground during testing.

TEST OUTPUT LOADS

Power Grade	R_1 (Ω)		R_2 (Ω)		C_L (pF)	C_1 (μ F)	C_2 (μ F)
	STD	APL	STD	APL	STD/APL	STD/APL	STD/APL
18P8B A AL L	200	390	390	750	50	0.1	0.01
18P8Q	390	600	750	1200	50	0.1	0.01

STD = AMD "Standard" products

APL = AMD "Approved Products List" products

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 9, 10, 11 tests unless otherwise noted

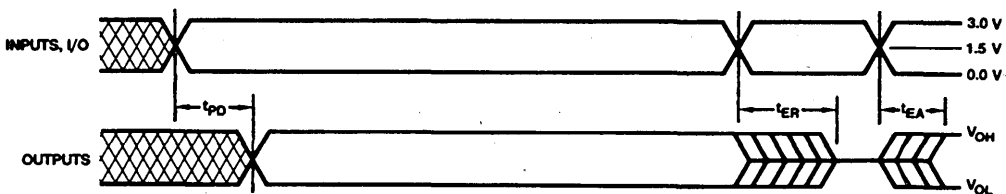
Parameter	Description	Commercial						Military						Units
		18P8B		18P8A/AL		18P8L/Q		18P8B		18P8A/AL		18P8L/Q		
		Typ.	Max.	Typ.	Max.	Typ.	Max.	Typ.	Max.	Typ.	Max.	Typ.	Max.	
t _{PD}	Input to Output Delay	12	15	15	25	25	35	12	20	15	30	25	40	ns
t _{EA}	Input to Output Enable	12	15	15	25	25	35	12	20	15	30	25	40	ns
t _{ER}	Input to Output Disable	12	15	15	25	25	35	12	20	15	30	25	40	ns

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.

2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.

3. For three-state output, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high-impedance to HIGH tests and closed for high-impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high-impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high-impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

SWITCHING WAVEFORM



WF021820

PROGRAMMING

Each AMD PAL fuse is programmed with a simple sequence of voltages applied to two control pins (1 and 11) and a programming voltage pulse applied to the output being programmed. Addressing of the 2600 element fuse array is accomplished with TTL and V_{HH} levels on eight input pins (five select the input line number and three select the product term number). V_{CC} is maintained at a normal level throughout the programming and verify cycle — no extra high V_{CC} levels are required.

The necessary sequence of levels for programming any fuse is shown in the Programming Waveforms. The address of each fuse in terms of Input Line Number and Product Term Line Number is defined by the Fuse Address Tables 1 and 2. Current, voltage and timing requirements for each pin are specified in the Programming Parameter Table on the following page.

After all programming has been completed, the entire array should be reverified at V_{CCL} and again at V_{CCH} . Reverification can be accomplished by reading all eight outputs in parallel rather than one at a time. The array fuse verification cycle

checks that the correct array fuses have been blown and can be sensed by the outputs.

AMD PALs have been designed with many internal test features that are used to assure high programming yield and correct logical operation for a correctly programmed part.

An additional fuse is provided on each AMD PAL circuit to prevent unauthorized copying of AMD PAL fuse patterns when design security is desired. Blowing the security fuse blocks entry to the fuse pattern verify mode.

To blow the security fuse:

1. Power to V_{CCP}
2. Raise Pin 11 to V_{OP}
3. Pulse Pin 5 to V_{HH} for 50 microseconds 10 times
4. Reverify the entire array. A secured device will verify as if all fuses in the array are blown

Note that parts with the security fuse blown may not be returned as programming rejects.

AMD PALs normally have high programming yields (> 98%). Programming yield losses are frequently due to poor socket contact or equipment out of calibration or improperly used.

Design Aid Software for AmPAL18P8

Software Vendor	Software Package	Comments
P-CAD Systems (408) 971-1300	CUPL	
Advanced Micro Devices (408) 732-2400	AmCUPL	Developed and supported by P-CAD Systems
Data I/O (206) 881-6444	ABEL	

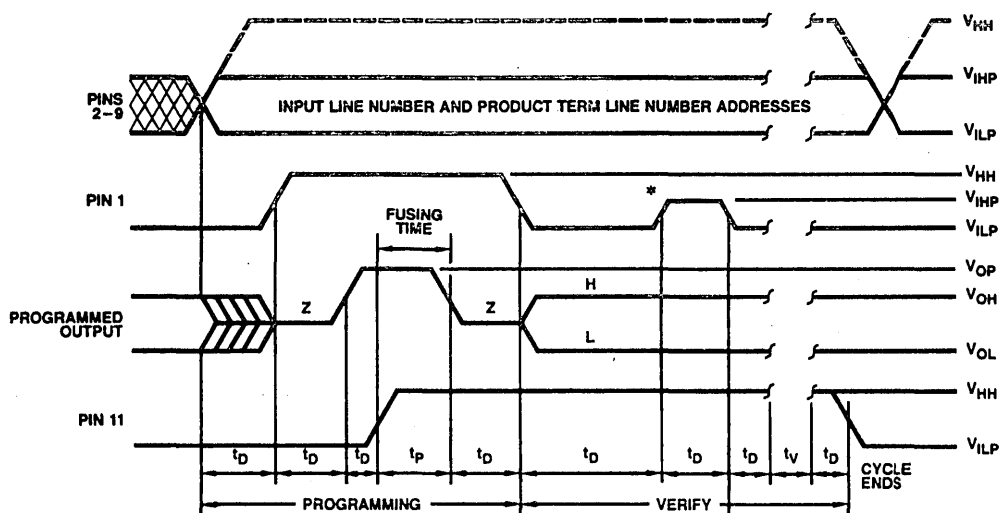
AMD Qualified Programmers

Name	Programmer Model(s)	AMD PAL Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 98052	System 19, 29	950-1942-0044	303A-004, rev 3 or newer
	60	N/A	360A-001, rev 4 or newer
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086	Model PPZ	Revision 19	On Board
	ZL30	On Board Module (rev 38 or newer)	
Structured Design 1700 Wyatt Drive Suite 3 Santa Clara, CA 95084	SD-1000J	Under Development	On Board
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043	160 Series	Under Development	On Board
Digelec 586 Weddell Drive Suite 1 Sunnyvale, CA 94089	803 Series	Under Development	Under Development
JMC 2999 Monterey Rd. Monterey, CA 93940	PROMAC P3	On-Board Module rev 2.0	On Board

PROGRAMMING PARAMETERS $T_A = 25^\circ\text{C}$

Parameters	Description		Min.	Typ.	Max.	Units
V_{HH}	Control Pin Extra High Level	Pins 1 & 11 @ 10–40 mA	10	11	12	Volts
	Address Extra High Level	Pins 4, 5, and 9, @ V_{HH}				
V_{OP}	Program Voltage Pins 12–19 @ 15–200 mA		14	15	16	Volts
V_{IHP}	Input High Level During Programming and Verify		2.4	5	5.5	Volts
V_{ILP}	Input Low Level During Programming and Verify		0.0	0.3	0.5	Volts
V_{CCP}	V_{CC} During Programming @ $I_{CC} = 50–200$ mA		5	5.2	5.5	Volts
V_{CCL}	V_{CC} During First Pass Verification @ $I_{CC} = 50–200$ mA		4.4	4.5	4.6	Volts
V_{CCH}	V_{CC} During Second Pass Verification @ $I_{CC} = 50–200$ mA		5.4	5.5	5.6	Volts
V_{Blown}	Successful Blown Fuse Source Level @ Output			0.3	0.5	Volts
dV_{OP}/dt	Rate of Output Voltage Change		20		250	V/ μsec
dV_{11}/dt	Rate of Fusing Enable Voltage Change (pin 11 rising edge)		20		1000	V/ μsec
t_p	Fusing Time First Attempt		10	50	100	μsec
	Subsequent Attempts (maximum of 8)		4	5	10	msec
t_D	Delays Between Various Level Changes		100	1000	15000	ns
t_v	Period During which Output is Sensed for V_{Blown} Level		100	1000	15000	ns
V_{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		$V_{CCP} - 0.3$	V_{CCP}	$V_{CCP} + 0.3$	Volts
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	k Ω

PROGRAMMING WAVEFORMS



WF021231

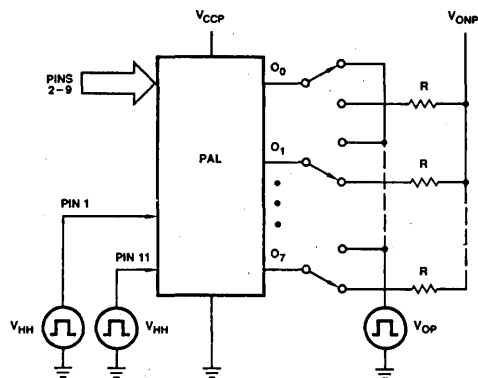
*The Pulse to V_{IHP} on pin 1 is unnecessary for fuse verification on combinatorial parts. It is used as a clock pulse on registered parts and is kept to maintain algorithm compatibility.

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States				
	9	8	7	6	5
0	L	L	L	L	L
1	L	L	L	L	H
2	L	L	L	H	L
3	L	L	L	H	H
4	L	L	H	L	L
5	L	L	H	L	H
6	L	L	H	H	L
7	L	L	H	H	H
8	L	H	L	L	L
9	L	H	L	L	H
10	L	H	L	H	L
11	L	H	L	H	H
12	L	H	H	L	L
13	L	H	H	L	H
14	L	H	H	H	L
15	L	H	H	H	H
16	H	L	L	L	L
17	H	L	L	L	H
18	H	L	L	L	H
19	H	L	L	H	L
20	H	L	H	L	L
21	H	L	H	L	H
22	H	L	H	H	L
23	H	L	H	H	H
24	H	H	L	L	L
25	H	H	L	L	H
26	H	H	L	H	L
27	H	H	L	H	L
28	H	H	H	L	L
29	H	H	H	L	H
30	H	H	H	H	L
31	H	H	H	H	H
32	HH	L	L	L	L
33	HH	L	L	L	H
34	HH	L	L	H	L
35	HH	L	L	H	H
36*	HH	L	H	L	L

L = V_{ILP}
H = V_{IHP}
HH = V_{HH}
*Output polarity.

SIMPLIFIED PROGRAMMING DIAGRAM



LD000050

TABLE 2. PRODUCT TERM ADDRESSING

Product Term Line Number								Product Term Select Address Pin		
								4	3	2
0	9	18	27	36	45	54	63	L	L	L
1	10	19	28	37	46	55	64	L	L	H
2	11	20	29	38	47	56	65	L	H	L
3	12	21	30	39	48	57	66	L	H	H
4	13	22	31	40	49	58	67	H	L	L
5	14	23	32	41	50	59	68	H	L	H
6	15	24	33	42	51	61	69	H	H	L
7	16	26	34	43	52	61	70	H	H	H
8	17	26	35	44	53	62	71	HH	L	L
P	P	P	P	P	P	P	P	HH	L	H
Pin 19	Pin 18	Pin 17	Pin 16	Pin 15	Pin 14	Pin 13	Pin 12	L = V _{ILP} H = V _{IHP} HH = V _{HH}		
Programming Access and Verify Pin										

Output Enable
Logical P.T.s

Output Polarity

AmPAL*10H20EG8/AmPAL10020EG8

IMOX-III™ ECL Programmable Array Logic

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- High-performance $t_{PD} = 6$ ns, $f_{MAX} = 125$ MHz
- Eight user-programmable output logic macrocells for latched or combinatorial operation
- A registered version of the device is available as AmPAL10H20EV8 or AmPAL10020EV8 (see AMD Publication No. 06176A)
- Up to twenty inputs and eight outputs
- Individually user-programmable output polarity
- Variable product-term distribution for increased design flexibility
- Individual product term for output enable
- Asynchronous-RESET and PRESET capability
- Power-up RESET capability
- PRELOAD for improved testability
- Special designed-in test features for full AC and DC testing
- Platinum-silicide fuses ensure high programming yield, fast programming and unsurpassed reliability
- 10KH/100K ECL options
- 50-ohm drive with wired-OR capability
- 24-pin 300 mil DIPs and 28-pin chip carrier packages

GENERAL DESCRIPTION

The AmPAL10H20EG8/AmPAL10020EG8 is an advanced bipolar ECL Programmable Array Logic (PAL) device. It uses the familiar sum-of-products (AND-OR) single array logic structure, allowing users to program custom logic functions. Fabricated with AMD's new advanced bipolar IMOX-III SLOT-isolation process technology and combining the innovative architectural features of the AmPAL22V10, the AmPAL10H20EG8/AmPAL10020EG8 represents the most advanced ECL PAL device available on the market today.

The AmPAL10H20EG8/AmPAL10020EG8 contains up to twenty inputs and eight outputs. It incorporates AMD's unique output logic macrocell (as in the AmPAL22V10), which allows the user to define and program the architecture of each output on an individual basis. Each output can be user-programmable for either latched or combinatorial operation. Each output also has user-programmable output-polarity control, further simplifying the design. The flexibility of the programmable output logic macrocells

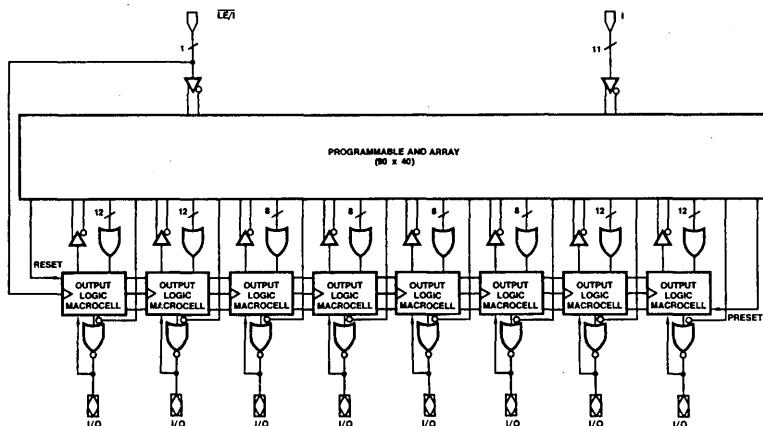
permits the system designer to tailor the device to particular application requirements.

Increased logic power has been built into the AmPAL10H20EG8/AmPAL10020EG8 by providing a variable number of logical product terms per output. Four outputs have twelve logical product terms each, and the other four have eight logical product terms each. This variable allocation of logical product terms allows complex functions to be implemented in a single ECL PAL device. Each output also has a separate output-enable product term.

System operation has been enhanced by the addition of asynchronous-PRESET and RESET product terms for the AmPAL10H20EG8/AmPAL10020EG8. These product terms are common to all latched outputs.

The AmPAL10H20EG8/AmPAL10020EG8 incorporates power-up RESET on all latched outputs. It also has the unique capability to PRELOAD latches to any desired state during testing. PRELOAD permits full logical verification during testing.

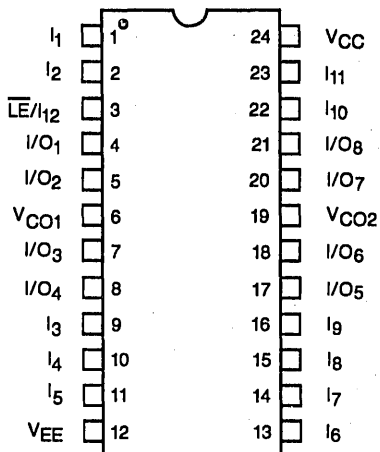
BLOCK DIAGRAM



*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.
IMOX is a trademark of Advanced Micro Devices, Inc.

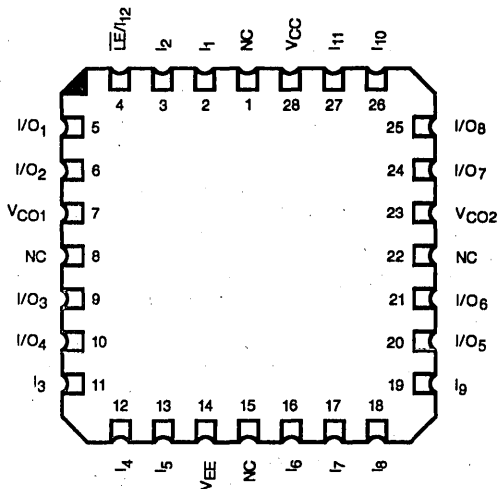
CONNECTION DIAGRAMS

DIPs



CD010170

LCC*



CD010180

*Also available in PLCC. Pinouts identical to LCC.

Note: Pin 1 is marked for orientation.

PIN DESCRIPTION

I₁–I₁₁ Dedicated Input Pins (11)I/O₁–I/O₈ Bidirectional Input/Output Pins (8) $\overline{LE}/I_{12}$ Latch Enable or Input Pin

NC No Connect

V_{CC} Circuit GroundV_{CO1}, V_{CO2} Circuit Ground Pins for Outputs (2)

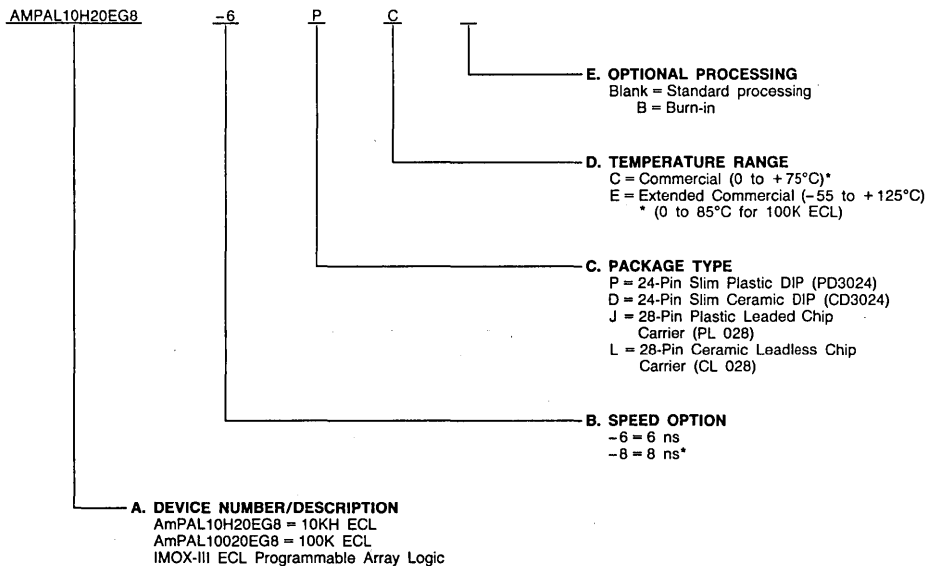
VEE Negative Supply Voltage

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**



*Extended Commercial Range only.

Valid Combinations

AMPAL10H20EG8-6	PC, DC, DCB, JC, LC, LCB
AMPAL10020EG8-6	
AMPAL10H20EG8-8	DE, LE
AMPAL10020EG8-8	

Valid Combinations

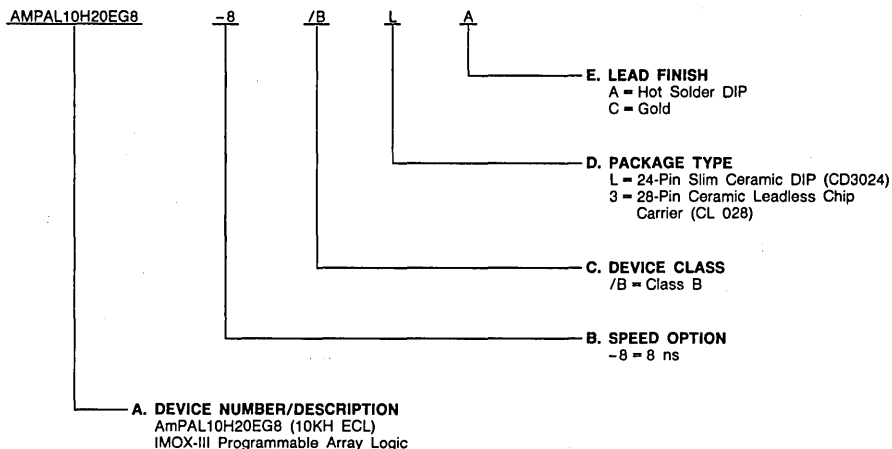
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number**
- B. Speed Option** (if applicable)
- C. Device Class**
- D. Package Type**
- E. Lead Finish**



Valid Combinations	
AMPAL10H20EG8-8	/BLA, /B3C

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.

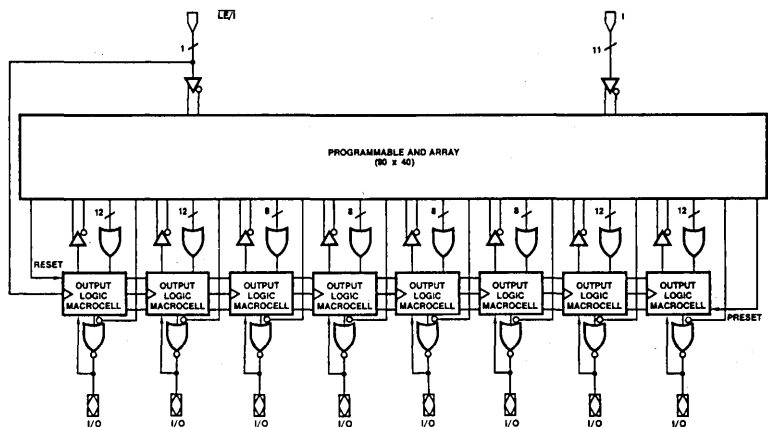
FUNCTIONAL DESCRIPTION

The AmPAL10H20EG8/AmPAL10020EG8 is an advanced bipolar ECL I/O PAL device. It contains a programmable fuse array organized in the familiar sum-of-products (AND-OR) structure.

The block diagram in Figure 1 illustrates the basic architecture of the AmPAL10H20EG8/AmPAL10020EG8. There are up to twenty external inputs and eight outputs. The inputs are connected to a programmable-AND array. Initially, the AND gates are connected, via fuses, to both the true and complement of every input. By selective programming (blowing) of the

fuses, the AND gates may be "connected" to only the true inputs, the complement inputs, or to neither type of input. When both the true and complement fuses are left intact, a logical-FALSE results at the output of the AND gate. An AND gate with all the fuses blown will assume the logical-TRUE state. The outputs of the AND gates are connected to fixed-OR gates.

There are an average of ten product terms per OR gate (output), distributed in a variable fashion. Four outputs have eight product terms each while the other four have twelve product terms each. This variable distribution of product terms allows more complex logical functions to be implemented.



BD006751

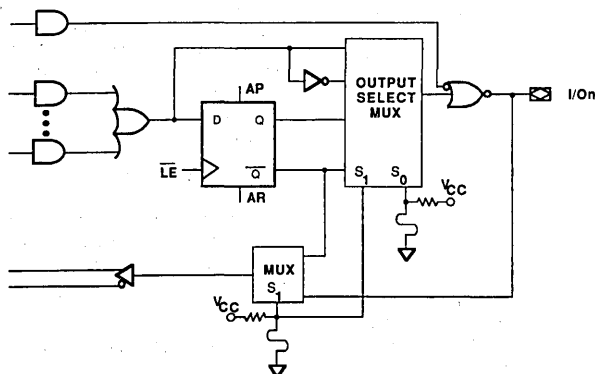
Figure 1. Block Diagram

Output Logic Macrocells

A useful feature of the AmPAL10H20EG8/AmPAL10020EG8 is its versatile programmable output-logic-macrocell structure. It allows the user to program the outputs on an individual basis in a very flexible manner.

The AmPAL10H20EG8/AmPAL10020EG8 output logic macrocell incorporates a transparent latch. As shown in the output logic macrocell diagram, each macrocell contains two

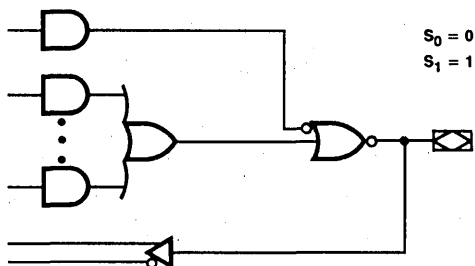
programmable fuses — S_0 and S_1 , for programming the output functions. S_1 controls whether the output will be latched or combinatorial. S_0 controls the output polarity (active-HIGH or active-LOW). Depending on the states of these two fuses, an individual output operates in one of four modes: Latched/Active-LOW, Latched/Active-HIGH, Combinatorial/Active-LOW, and Combinatorial/Active-HIGH. Each output is also provided with a separate output enable product term.



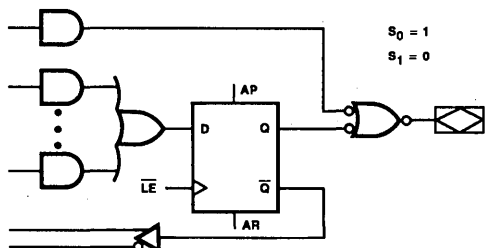
LD000811

Figure 2. AmPAL10H20EG8/AmPAL10020EG8 Output Logic Macrocell

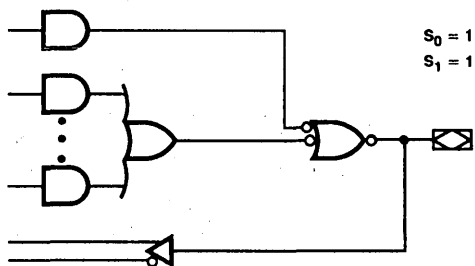
LD000422



LD000431



LD000442



LD000451

S ₁	S ₀	Output Configuration
0	0	Latched/Active-LOW
0	1	Latched/Active-HIGH
1	0	Combinatorial/Active-LOW
1	1	Combinatorial/Active-HIGH

0 = Unblown Fuse
1 = Blown Fuse

Another feature of the AmpPAL10H20EG8/AmpPAL10020EG8 output macrocell structure is the flexibility of its feedback selection. The feedback can be from either the I/O line or the latched output. The feedback multiplexer is also controlled by the S₁ fuse. The feedback path changes with the output-mode selection. If the output is selected to be latched, the feedback is latched. If the output is combinatorial, the feedback is from the I/O line. This feature enables the designer to optimally use the device to meet precise application requirements. Additionally, it also allows him/her to perform control tasks, such as arbitration functions, easily.

Each of the eight output logic macrocells of the AmPAL10H20EG8/AmPAL10020EG8 contains a dedicated product term for output-enable function. When this product term is asserted LOW, the output is forced into a LOW state where it remains until the output-enable product term goes HIGH.

To improve in-system functionality, the AmPAL10H20EG8/AmPAL10020EG8 has additional PRESET and RESET prod-

uct terms. Common asynchronous-RESET and PRESET are provided for all the latches of the AmPAL10H20EG8/AmPAL10020EG8. When the asynchronous PRESET product term is asserted HIGH the output latches are loaded with a HIGH and when the RESET product term is asserted HIGH the output latches are loaded with a LOW. For the RESET/ PRESET to work the latches should be in latched and not transparent mode. These functions are particularly useful for system power-on and reset.

To simplify testing, the AmPAL10H20EG8/AmPAL10020EG8 is designed with PRELOAD circuitry that provides an easy method for testing logical functionality. PRELOAD allows any arbitrary values to be loaded into the PAL device's output latches.

A typical functional-test sequence would be to verify all possible outputs for the device being tested. To verify these transitions requires the ability to set the latches to an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the latch enable is driven transparent to allow logic to determine the

"new state" of the outputs. These outputs can then be checked to validate the design.

Without PRELOAD capability, it is difficult and in some cases impossible to load an arbitrary value into the latches. This can lead to logic-verification sequences which are either incomplete or excessively long. With PRELOAD capability, logic verification sequences can be greatly shortened, hence reducing the test time and the development costs, and guaranteeing proper in-system operation.

Fabrication

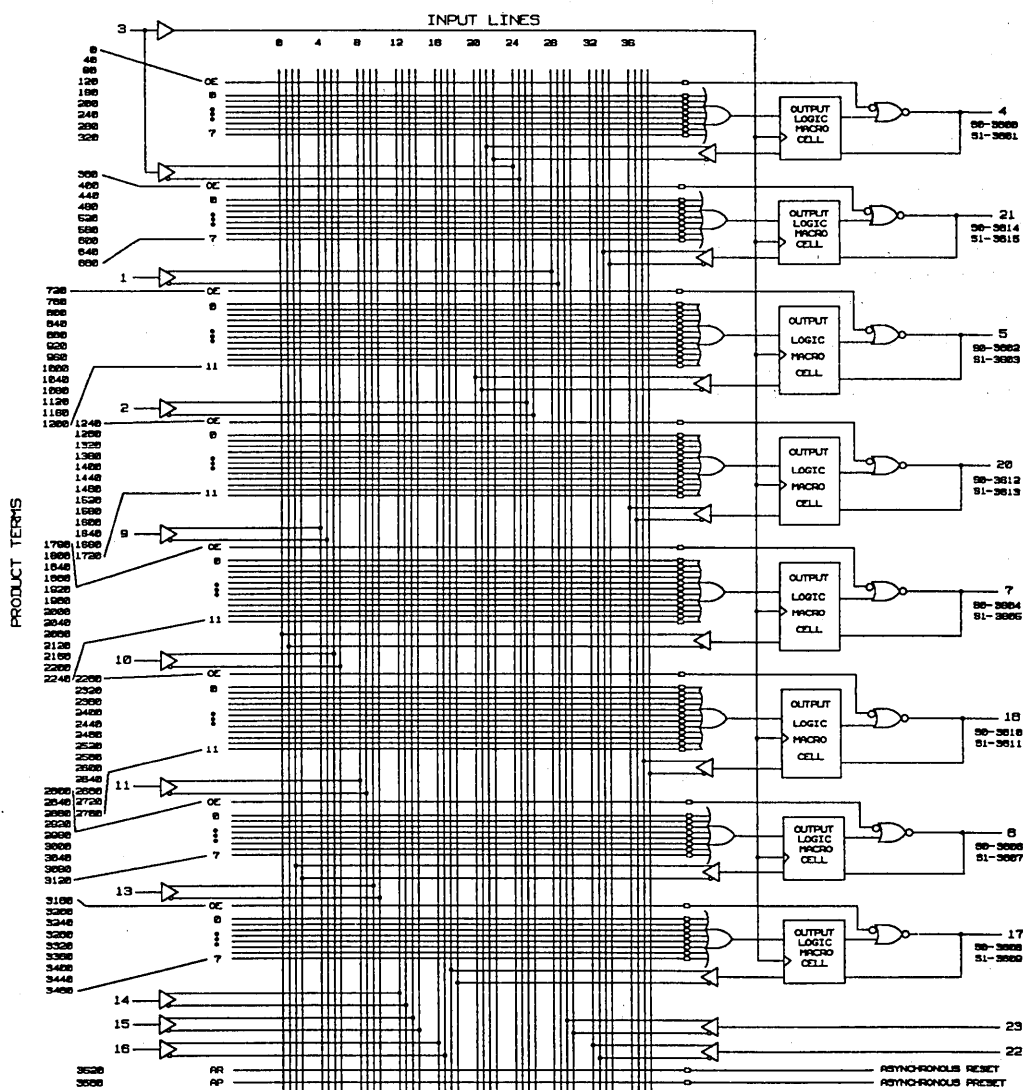
The AmPAL10H20EG8/AmPAL10020EG8 is manufactured using Advanced Micro Devices' new IMOX-III SLOT-isolation process. This advanced process offers increased density and reduced internal capacitance resulting in the fastest possible programmable logic devices.

The AmPAL10H20EG8/AmPAL10020EG8 is fabricated with AMD's fast programming and highly reliable platinum-silicide

fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Platinum-silicide was selected as the fuse-link material to achieve a well controlled melt rate resulting in large non-conductive gaps which ensure very stable, long-term reliability. Extensive operating testing has proven that this low-field, large-gap technology offers the best reliability for fusible-link programmable logic.

Testing

The AmPAL10H20EG8/AmPAL10020EG8 contains many internal test features, including circuitry and extra fuses which allow AMD to test each part before shipping. This ensures extremely high post-programming functional yields. The test fuses are programmed to assure the ability of each part to perform correct programming. There are extra test words which are preprogrammed during manufacturing and tested to ensure correct logical operation and provide extra test paths to achieve excellent parametric correlation.



LD000840

Figure 4. Logic Diagram — JEDEC Fuse Numbering for AmpAL10H20EG8/10020EG8

Note: Pin out for DIP only.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage (V_{EE})	
with Respect to Ground	-8 V to 0 V
DC Input Voltage	
with Respect to Ground	V_{EE} to 0 V
Output Current	
-Continuous	35 mA
-Surge	100 mA

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices — 100K Devices	
Temperature (T_A) Operating Free Air	0 to +85°C
Supply Voltage (V_{EE})	-5.7 to -4.2 V
Commercial (C) Devices — 10KH Devices	
Temperature (T_A) Operating Free Air	0 to +75°C
Supply Voltage (V_{EE})	-5.46 to -4.94 V
Extended Commercial (E) Devices	
Temperature (T_A)	-55°C Min.
Temperature (T_C) Operating Case	+125°C Max.
Supply Voltage (V_{EE})	-5.72 to -4.68 V
Military* (M) Devices	
Temperature (T_A)	-55°C Min.
Temperature (T_C)	+125°C Max.
Supply Voltage (V_{EE})	-5.72 to -4.68 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at $T_C = +25^\circ\text{C}$, $+125^\circ\text{C}$ and -55°C .

DC CHARACTERISTICS over operating range unless otherwise specified

AmPAL10H20EG8 (Commercial)

Parameter Symbols	Parameter Description	Test Conditions			Min.	Max.	Units
V _{OH}	Output HIGH Voltage	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	T _A = 0°C	-1020	-840	mV
				T _A = +25°C	-980	-810	
				T _A = +75°C	-920	-735	
V _{OL}	Output LOW Voltage			T _A = 0°C	-1950	-1630	mV
				T _A = +25°C	-1950	-1630	
				T _A = +75°C	-1950	-1600	
V _{IH}	Input HIGH Voltage	Guaranteed Input HIGH Voltage (Note 3)		T _A = 0°C	-1170	-840	mV
				T _A = +25°C	-1130	-810	
				T _A = +75°C	-1070	-735	
V _{IL}	Input LOW Voltage	Guaranteed Input LOW Voltage (Note 3)		T _A = 0°C	-1950	-1480	mV
				T _A = +25°C	-1950	-1480	
				T _A = +75°C	-1980	-1450	
I _{IH}	Input HIGH Current	V _{IN} = V _{IH} (Max.)		T _A = 0°C		220	μA
				T _A = +25°C		220	
				T _A = +75°C		220	
I _{IL}	Input LOW Current	V _{IN} = V _{IL} (Min.)		T _A = 0°C	0.5		μA
				T _A = +25°C	0.5		
				T _A = +75°C	0.3		
I _{EE}	Power Supply Current	All Inputs and Outputs Open		T _A = 0°C	-230		mA
				T _A = +25°C	-230		
				T _A = +75°C	-230		

Table continues on following page.

Notes: See notes following Military DC characteristics table.

AmPAL10H20EG8 (Military); included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted.

Parameter Symbols	Parameter Description	Test Conditions			Min.	Max.	Units
V _{OH}	Output HIGH Voltage	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{OL}	Output LOW Voltage			T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{IH}	Input HIGH Voltage	Guaranteed Input HIGH Voltage (Note 3)		T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{IL}	Input LOW Voltage	Guaranteed Input LOW Voltage (Note 3)		T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{IH}	Input HIGH Current	V _{IN} = V _{IH} (Max.)		T _A = 0°C	TBD	TBD	μA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{IL}	Input LOW Current	V _{IN} = V _{IL} (Min.)		T _A = 0°C	TBD	TBD	μA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{EE}	Power Supply Current	All Inputs and Outputs Open		T _A = 0°C	TBD	TBD	mA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	

AmPAL10020EG8 (Commercial)

Parameter Symbols	Parameter Description	Test Conditions		Min.	Max.	Units
V _{OH}	Output Voltage HIGH	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	-1025	-880	mV
V _{OL}	Output Voltage LOW			-1810	-1620	mV
V _{OHC}	Output Voltage HIGH	V _{IN} = V _{IH} (Min.) or V _{IL} (Max.)		-1035		mV
V _{OLC}	Output Voltage LOW				-1610	mV
V _{IH}	Input Voltage HIGH	Guaranteed Input Voltage HIGH (Note 3)		-1165	-880	mV
V _{IL}	Input Voltage LOW	Guaranteed Input Voltage LOW (Note 3)		-1810	-1475	mV
I _{IH}	Input Current HIGH	V _{IN} = V _{IH} (Max.)			220	μA
I _{IL}	Input Current LOW	V _{IN} = V _{IL} (Min.)		0.5		μA
I _{EE}	Power Supply Current	All Inputs and Outputs Open		-230		mA

Notes: 1. Guaranteed with transverse air flow exceeding 400 linear F.P.M.

2. The relative values of the specified conditions and limits will be referenced to an algebraic scale. The extremities of the scale are:

"Max." the value closest to positive infinity.

"Min." the value closest to negative infinity.

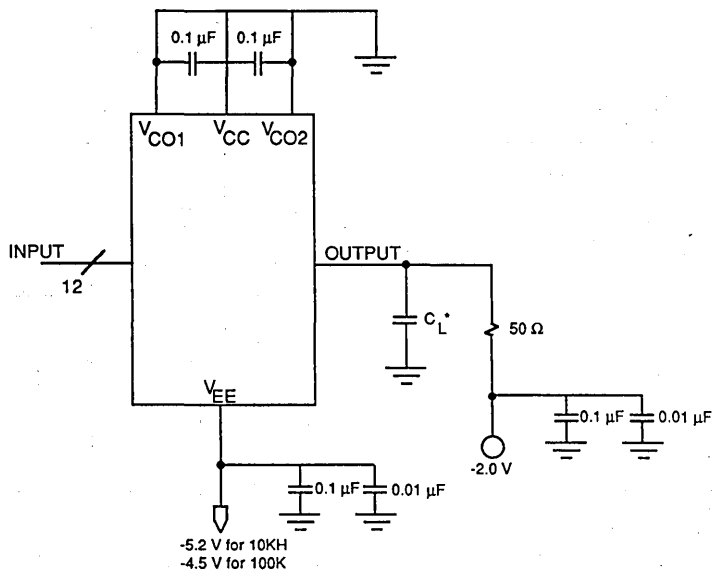
3. These are absolute voltages with respect to the device ground pin and include all overshoots due to system and/or tester noise. Do not attempt to test these values without suitable equipment and fixturing.

SWITCHING CHARACTERISTICS

over operating range unless otherwise specified; included in Group A,
Subgroup 9, 10, 11 tests unless otherwise noted

No.	Parameter Symbol	Parameter Description	Test Conditions	C Devices				E/M Devices				Units
				-6		TBD		-8		TBD		
				Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
1	t _{PD}	Input or Feedback to Output	See Switching Test Circuit		6				8			ns
2	t _{EA}	Input to Output Enable			6				8			ns
3	t _{ER}	Input to Output Disable			6				8			ns
4	t _{LEO}	$\overline{LE}$ to Output			3.5				4.5			ns
5	t _S	Input or Feedback Setup Time			4.5				5.5			ns
6	t _H	Hold Time			0				0			ns
7	t _W	$\overline{LE}$ Width			3				5			ns
8	t _{AW}	Asynchronous RESET/PRESET Width			6				8			ns
9	t _{AR}	Asynchronous RESET/PRESET Recovery Time			6				8			ns
10	t _{AP}	Asynchronous RESET/PRESET to Latched Output RESET/PRESET				6				8		

SWITCHING TEST CIRCUIT

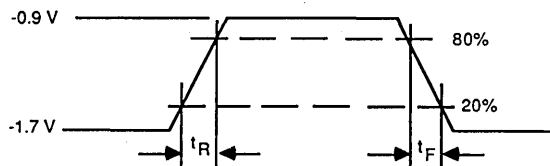


TC003930

* $C_L = 5$ pf for 10KH
 $C_L = 3$ pf for 100KH

SWITCHING TEST WAVEFORM

INPUT PULSE



WF023080

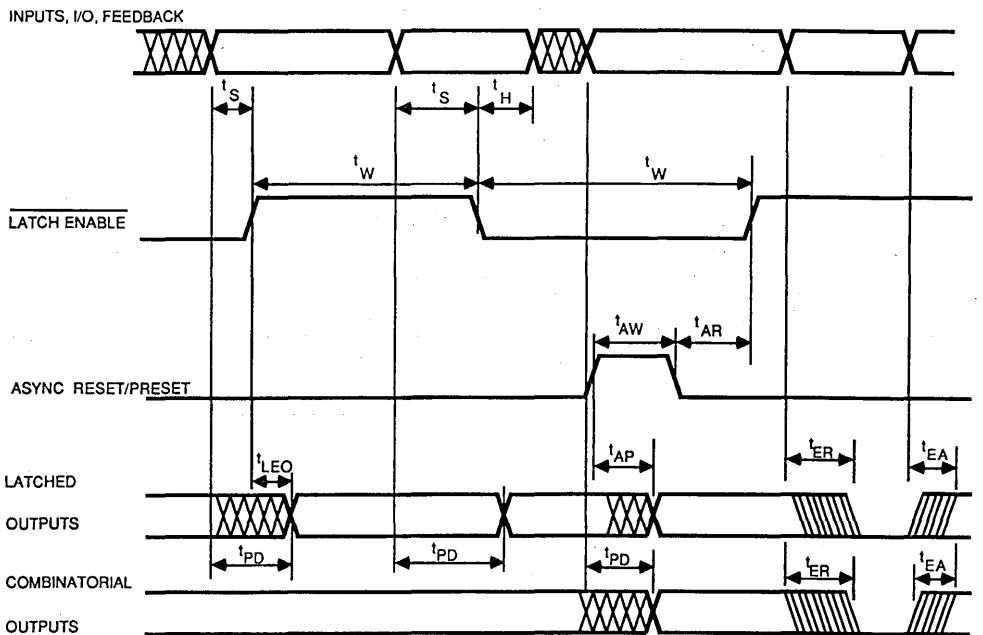
$t_R = t_F = 2.2$ ns Max. for 10KH
 $t_R = t_F = 1.0$ ns Max. for 100K

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS



WF023092

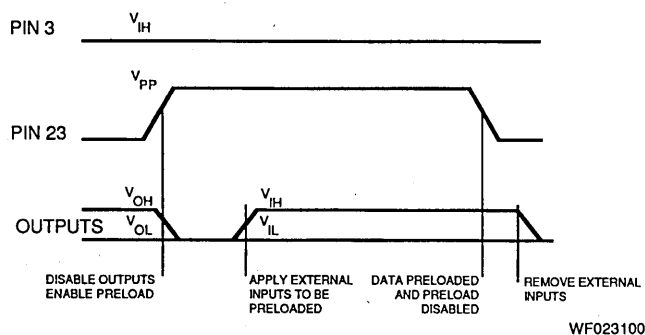
PRELOAD OF LATCHED OUTPUTS

The AmPAL10H20EG8/AmPAL10020EG8 latched outputs are provided with circuitry to allow loading each latch to either a HIGH or LOW state. This simplifies testing as any state can be loaded into the latches to control outputs. The pin levels

and the timing necessary to perform the PRELOAD function are detailed below.

PRELOAD is accessed by applying V_{PP} on pin 23. The data to be preloaded is set on the output pins. Bringing pin 23 back to a logic-LOW level latches the data into the output latches.

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
V_{IH}	Input HIGH Level During PRELOAD and Verify	-1.1	-0.9	-0.7	V
V_{IL}	Input LOW Level During PRELOAD and Verify	-1.85	-1.65	-1.45	V
V_{PP}	PRELOAD Enable Voltage Applied to Pin 23	1.8	2.0	2.2	V



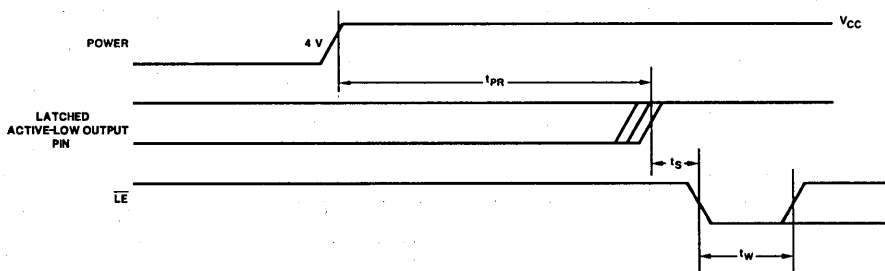
PRELOAD Timing Waveform

POWER-UP RESET

The latches in the AmPAL10H20EG8/AmPAL10020EG8 have been designed with the capability to RESET during system power-up. Following power-up, all latches will be LOW. The output state will depend upon the state of the output buffer and the polarity fuse. This feature provides extra flexibility to the designer. A timing diagram and a parameter table are shown below. Due to the asynchronous operation of the

power-up RESET and the wide range of ways V_{CC} can rise to steady state, two conditions are required to insure a valid power-up RESET. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following RESET, the Latch Enable must not be driven transparent until all applicable input and feedback setup times are met.



Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up RESET Time		600	1000	ns
t_S	Input or Feedback Setup Time	See Switching Characteristics			
t_W	LE Width				

PROGRAMMING AND VERIFICATION

The AmPAL10H20EG8/AmPAL10020EG8 is programmed and verified using AMD's ECL programmable-logic-programming algorithm. The fuse to be programmed is selected by its input line number (array row), its product-term number (array column), and by the output associated with it (one at a time). The levels for addressing the rows and columns are all in ECL (10KH) levels. The fuse is then programmed and verified by applying a simple sequence of voltages as shown in the programming waveform diagram. The values of the programming voltages are shown in the table of programming parameters.

Input line numbers (0 to 39) are addressed using a full decode scheme via ECL levels on pins 9 – 11 and 13 – 15 where pin 15 is the LSB and pin 9 is the MSB. Input line addressing is shown in Table 1. Note that one input line is given to select all the architectural fuses.

Product terms are addressed using a full decode scheme via ECL levels on pins 1, 2, 22 and 23 where pin 23 is the LSB and pin 1 is the MSB. Product terms (column number) addressing is shown in Table 2.

The sequence of signals and events associated with them during programming and verification modes is shown in the Programming Waveforms. The row and column addresses are set up on the corresponding input pins. Then V_{CO1} (pin 6) is brought to a programming voltage of V_{COP} . The normal I/O pins are now disabled and the address decode is enabled. The device is now in a verification mode. V_{CO2} (pin 19) is now brought to the V_{COP} programming voltage. The programming mode is enabled. A high voltage of V_{OP} (Fuse-Enable Voltage) on the output pin corresponding to the fuse to be blown activates the programming voltage, and the fusing is started.

After a certain programming time, the fuse is blown and the programming voltage is removed from V_{CO2} . The device is back in the verification mode. Latch Enable is HIGH during programming. It is then brought LOW to enable the latches and verify for V_{Blown} level (V_{IHP} or V_{ILP}). After verification is done, the programming voltage is removed from V_{CO1} and the part is back in normal mode.

Security Programming

A security fuse is provided on each AmPAL10H20EG8/AmPAL10020EG8 to protect any user proprietary logic design. It is addressed and programmed like any other fuse in the array. Once blown, the circuitry enabling any fuse verification and latch PRELOAD is permanently disabled. The security fuse is verified by verifying the whole fuse array as if every fuse was blown.

Programming Yield

AMD PAL devices have been designed to ensure extremely high programming yields. To help ensure that a part was correctly programmed, once the programming is completed the entire fuse array should be re-verified at both LOW and HIGH V_{CC} . Re-verification can be accomplished by reading all eight outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PAL devices contain many internal features, including circuitry and extra fuses to test the ability of each part to perform programming before shipping, and to assure a correct logical operation for a correctly programmed part. Programming-yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS ($T_A = 25^\circ\text{C}$)

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
V_{IHP}	Input HIGH Level During Programming & Verify	-1.1	-0.9	-0.7	V
V_{ILP}	Input LOW Level During Programming & Verify	-1.85	-1.65	-1.45	V
V_{OP}	Fuse Enable Voltage Applied to Output Being Programmed @ 1 – 25 mA (Additional 80 mA if Output Terminated to 50-ohm Load)	1.8	2.0	2.2	V
V_{COP}	Programming Voltage Applied to V_{CO1} , V_{CO2} @ 15 – 200 mA	14.0	15.0	16.0	V
V_{CO1} & V_{CO2}	Power Supply for the Output Stage Sourcing	-0.1	0	0.1	V
V_{CCP}	V_{CC} During Programming & Verify $I_{CC} = 50 - 250$ mA Sourcing $I_{CC} = 50 - 150$ mA Sinking	-0.3	0	0.3	V
V_{EEP}	VEE During Programming & Verify	-5.4	-5.2	-5.0	V
V_{ONP}	Termination Voltage for Output-Load Resistor	-2.1	-2.0	-1.9	V
R	Output-Load Resistor	47.5	50.0	52.5	Ω
t_p	Fusing Time First Attempt	40	50	100	μs
	Fusing Time Second Attempt	4	5	10	ms
t_D	Delays Between Various Level Changes	100	200	1000	ns
t_v	PRELOAD During Which Output is Sensed for V_{Blown} (V_{IHP} or V_{ILP}) Level			500	ns

TABLE 1. INPUT ADDRESSING

Input Line Number	Address Pin States					
	9	10	11	13	14	15
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	L
3	L	L	L	L	H	H
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	H
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	H
12	L	L	H	H	L	L
13	L	L	H	H	L	H
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	H
18	L	H	L	L	H	L
19	L	H	L	L	H	H
20	H	L	L	L	L	L
21	H	L	L	L	L	H
22	H	L	L	L	H	L
23	H	L	L	L	H	H
24	H	L	L	H	L	L
25	H	L	L	H	L	H
26	H	L	L	H	H	L
27	H	L	L	H	H	H
28	H	L	H	L	L	L
29	H	L	H	L	L	H
30	H	L	H	L	H	L
31	H	L	H	L	H	H
32	H	L	H	H	L	L
33	H	L	H	H	L	H
34	H	L	H	H	H	L
35	H	L	H	H	H	H
36	H	H	L	L	L	L
37	H	H	L	L	L	H
38	H	H	L	L	H	L
39	H	H	L	L	H	H
	RESERVED					
63*	H	H	H	H	H	H

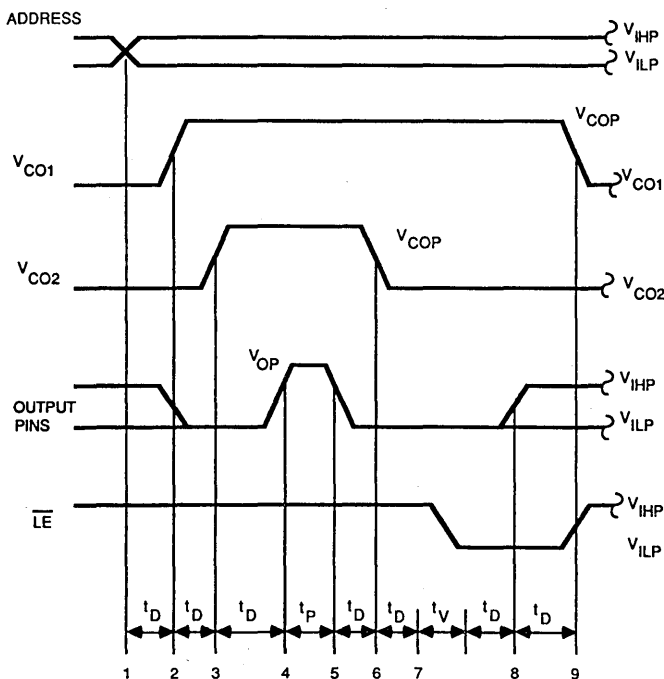
*Architecture Row

TABLE 2. COLUMN NUMBER ADDRESSING

Address				Output Pins							
1	2	22	23	4 OUT1	5 OUT2	7 OUT3	8 OUT4	17 OUT5	18 OUT6	20 OUT7	21 OUT8
L	L	L	L	0	0	0	0	0	0	0	0
L	L	L	H	1	1	1	1	1	1	1	1
L	L	H	L	2	2	2	2	2	2	2	2
L	L	H	H	3	3	3	3	3	3	3	3
L	H	L	L	4	4	4	4	4	4	4	4
L	H	L	H	5	5	5	5	5	5	5	5
L	H	H	L	6	6	6	6	6	6	6	6
L	H	H	H	7	7	7	7	7	7	7	7
H	L	L	L	–	8	8	–	–	8	8	–
H	L	L	H	–	9	9	–	–	9	9	–
H	L	H	L	–	10	10	–	–	10	10	–
H	L	H	H	–	11	11	–	–	11	11	–
H	H	L	L	OE	OE	OE	OE	OE	OE	OE	OE
H	H	L	H	OP	OP	OP	OP	OP	OP	OP	OP
H	H	H	L	R/C	R/C	R/C	R/C	R/C	R/C	R/C	R/C
H	H	H	H	SF	AR	–	–	–	–	AP	–

AR = Async. RESET
 AP = Async. PRESET
 OE = Output Enable
 R/C = Reg/Comb Fuse
 OP = Output Polarity Fuse
 SF = Security Fuse

PROGRAMMING WAVEFORMS



WF023110

AmPAL10H20EG8/AmPAL10020EG8 PROGRAMMING SUPPORT INFORMATION

Hardware Vendor	Programmer Model(s)	Personality Module	Socket Adaptor
Data I/O 10525 Willow Road N.E. Redmond, WA 98052 (206) 881-6444	System 29	Under Development	Not Required
	UNISITE 40	Not Required	Not Required
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94089 (408) 745-1991 or (800) 227-8836	Model PPZ	Under Development	Not Required
	Model ZL30A	Under Development	Not Required

The machines noted above have been qualified by AMD to ensure high programming yields. Check with the factory to determine current status of equipment noted as "Under Development," or for other available models.

Design-Aid Software for AmPAL10H20EG8/AmPAL10020EG8

Software Vendor	Software Package	Comments
P-CAD Systems, Inc. 1290 Parkmoor Ave. San Jose, CA 95126 (408) 971-1300	CUPL	
Advanced Micro Devices, Inc. 901 Thompson Pl. Sunnyvale, CA 94088 (408) 732-2400	AmCUPL	Supported by P-CAD Systems, Inc.
Data I/O 10525 Willow Road N.E. Redmond, WA 98052 (206) 881-6444	ABEL	

AmPAL*10H20EV8/AmPAL10020EV8

IMOX-III™ ECL Programmable Array Logic

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- High-performance $t_{PD} = 6$ ns, $f_{MAX} = 125$ MHz
- Eight user-programmable output logic macrocells for registered or combinatorial operation
- A latched version of the device is available as AmPAL10H20EG8 or AmPAL10020EG8 (see AMD Publication No. 08545A)
- Up to twenty inputs and eight outputs
- Individually user-programmable output polarity
- Variable product-term distribution for increased design flexibility
- Individual product term for output enable
- Asynchronous-RESET and PRESET capability
- Power-up RESET capability
- PRELOAD for improved testability
- Special designed-in test features for full AC and DC testing
- Platinum-silicide fuses ensure high programming yield, fast programming and unsurpassed reliability
- 10KH/100K ECL options
- 50-ohm drive with wired-OR capability
- 24-pin 300 mil DIPs and 28-pin chip carrier packages

GENERAL DESCRIPTION

The AmPAL10H20EV8/AmPAL10020EV8 is an advanced bipolar ECL Programmable Array Logic (PAL) device. It uses the familiar sum-of-products (AND-OR) single array logic structure, allowing users to program custom logic functions. Fabricated with AMD's new advanced bipolar IMOX-III SLOT-isolation process technology and combining the innovative architectural features of the AmPAL22V10, the AmPAL10H20EV8/AmPAL10020EV8 represents the most advanced ECL PAL device available on the market today.

The AmPAL10H20EV8/AmPAL10020EV8 contains up to twenty inputs and eight outputs. It incorporates AMD's unique output logic macrocell (as in the AmPAL22V10), which allows the user to define and program the architecture of each output on an individual basis. Each output can be user-programmable for either registered or combinatorial operation. Each output also has user-programmable output-polarity control, further simplifying the design. The flexibility of the programmable output logic macrocells

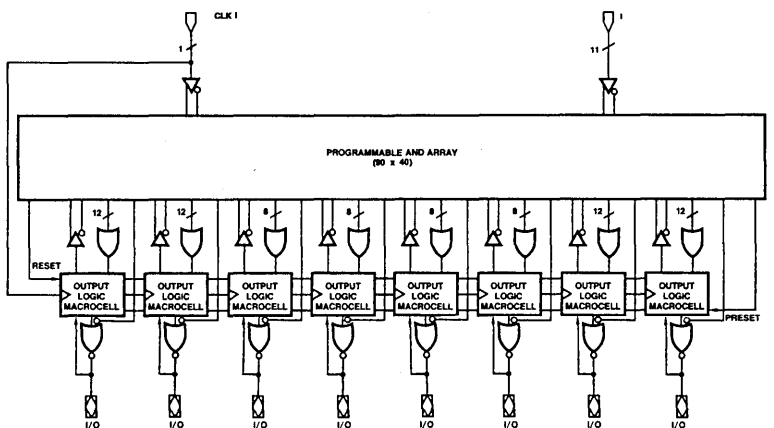
permits the system designer to tailor the device to particular application requirements.

Increased logic power has been built into the AmPAL10H20EV8/AmPAL10020EV8 by providing a variable number of logical product terms per output. Four outputs have twelve logical product terms each, and the other four have eight logical product terms each. This variable allocation of logical product terms allows complex functions to be implemented in a single ECL PAL device. Each output also has a separate output-enable product term.

System operation has been enhanced by the addition of asynchronous-PRESET and RESET product terms for the AmPAL10H20EV8/AmPAL10020EV8. These product terms are common to all registered outputs.

The AmPAL10H20EV8/AmPAL10020EV8 incorporates power-up RESET on all registered outputs. It also has the unique capability to PRELOAD registers to any desired state during testing. PRELOAD permits full logical verification during testing.

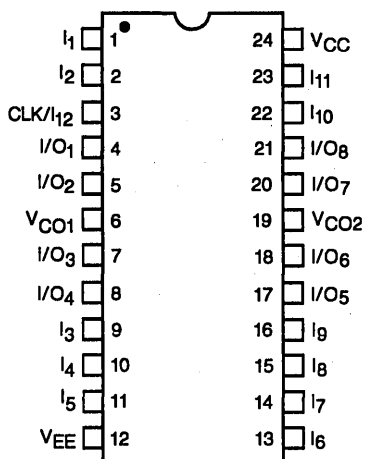
BLOCK DIAGRAM



*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.
IMOX is a trademark of Advanced Micro Devices, Inc.

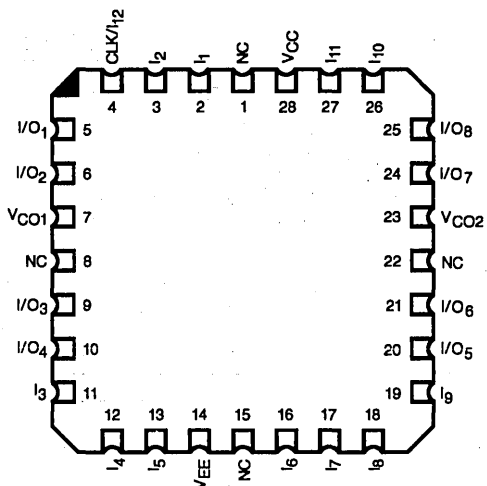
CONNECTION DIAGRAMS

DIPs



CD010171

LCC*



CD010181

*Also available in PLCC. Pinouts identical to LCC.

Note: Pin 1 is marked for orientation.

PIN DESCRIPTION

I₁ - I₁₁ Dedicated Input Pins (11)I/O₁ - I/O₈ Bidirectional Input/Output Pins (8)CLK/I₁₂ Clock or Input Pin

NC No Connect

V_{CC} Circuit GroundV_{CO1}, V_{CO2} Circuit Ground Pins for Outputs (2)V_{EE} Negative Supply Voltage

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

AMPAL10H20EV8

-6

P

C

E. OPTIONAL PROCESSING

Blank = Standard processing
B = Burn-in

D. TEMPERATURE RANGE

C = Commercial (0 to +75°C)*
E = Extended Commercial (-55 to +125°C)
* (0 to 85°C for 100K ECL)

C. PACKAGE TYPE

P = 24-Pin Slim Plastic DIP (PD3024)
D = 24-Pin Slim Ceramic DIP (CD3024)
J = 28-Pin Plastic Leaded Chip Carrier (PL 028)
L = 28-Pin Ceramic Leadless Chip Carrier (CL 028)

B. SPEED OPTION

-6 = 6 ns
-8 = 8 ns*

A. DEVICE NUMBER/DESCRIPTION

AmPAL10H20EV8 = 10KH ECL
AmPAL10020EV8 = 100K ECL
IMOX-III ECL Programmable Array Logic

*Extended Commercial Range only.

Valid Combinations

AMPAL10H20EV8-6	PC, DC, DCB, JC, LC, LCB
AMPAL10020EV8-6	
AMPAL10H20EV8-8	DE, LE
AMPAL10020EV8-8	

Valid Combinations

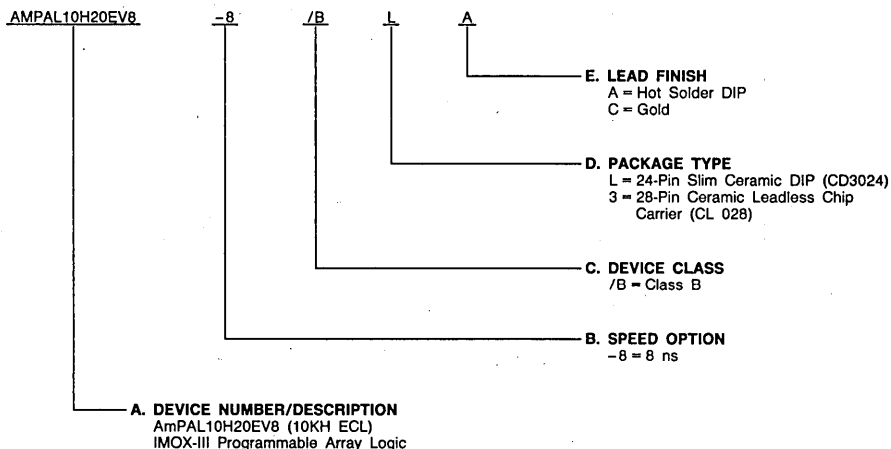
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number
- B. Speed Option (if applicable)
- C. Device Class
- D. Package Type
- E. Lead Finish



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.

Valid Combinations	
AMPAL10H20EV8-8	/BLA, /B3C

Group A Tests

Group A tests consists of Subgroups:
1, 2, 3, 7, 8, 9, 10 & 11.

FUNCTIONAL DESCRIPTION

The AmpPAL10H20EV8/AMPAL10020EV8 is an advanced bipolar ECL I/O PAL device. It contains a programmable fuse array organized in the familiar sum-of-products (AND-OR) structure.

The block diagram in Figure 1 illustrates the basic architecture of the AmpPAL10H20EV8/AMPAL10020EV8. There are up to twenty external inputs and eight outputs. The inputs are connected to a programmable-AND array. Initially, the AND gates are connected, via fuses, to both the true and complement of every input. By selective programming (blowing) of the

fuses, the AND gates may be "connected" to only the true inputs, the complement inputs, or to neither type of input. When both the true and complement fuses are left intact, a logical-FALSE results at the output of the AND gate. An AND gate with all the fuses blown will assume the logical-TRUE state. The outputs of the AND gates are connected to fixed-OR gates.

There are an average of ten product terms per OR gate (output), distributed in a variable fashion. Four outputs have eight product terms each while the other four have twelve product terms each. This variable distribution of product terms allows more complex logical functions to be implemented.

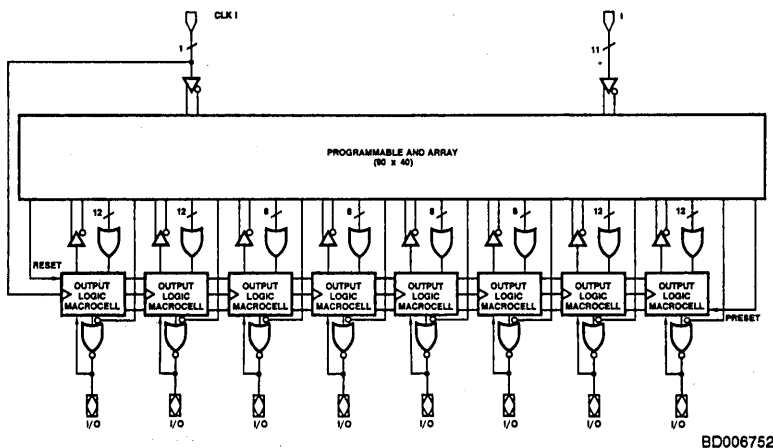


Figure 1. Block Diagram

Output Logic Macrocells

A useful feature of the AmpPAL10H20EV8/AMPAL10020EV8 is its versatile programmable output-logic-macrocell structure. It allows the user to program the outputs on an individual basis in a very flexible manner.

The AmpPAL10H20EV8/AMPAL10020EV8 output logic macrocell incorporates an edge triggered register. As shown in the output logic macrocell diagram, each macrocell contains two

programmable fuses — S_0 and S_1 , for programming the output functions. S_1 controls whether the output will be registered or combinatorial. S_0 controls the output polarity (active-HIGH or active-LOW). Depending on the states of these two fuses, an individual output operates in one of four modes: Registered/Active-LOW, Registered/Active-HIGH, Combinatorial/Active-LOW, and Combinatorial/Active-HIGH. Each output is also provided with a separate output enable product term.

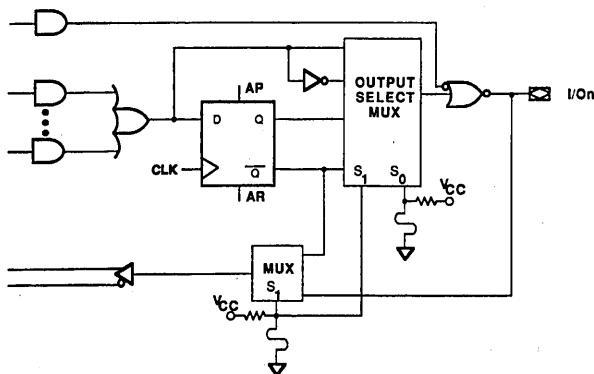


Figure 2. AmpPAL10H20EV8/AMPAL10020EV8 Output Logic Macrocell

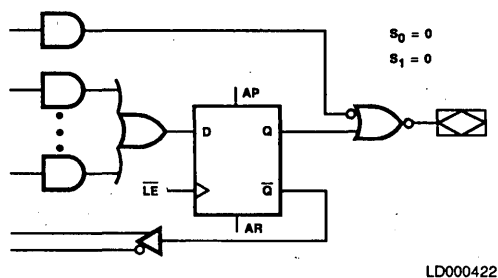


Figure 3-1. Registered/Active-LOW

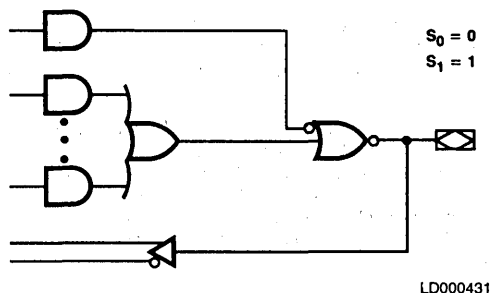


Figure 3-3. Combinatorial/Active-LOW

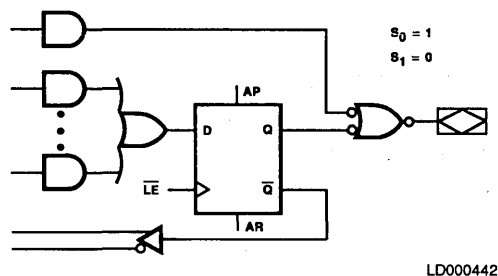


Figure 3-2. Registered/Active-HIGH

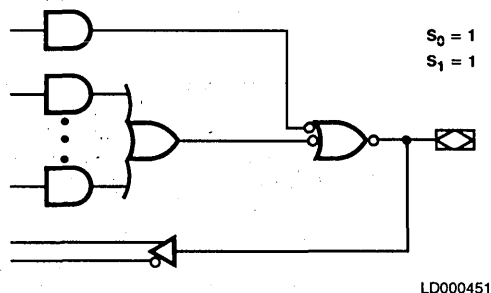


Figure 3-4. Combinatorial/Active-HIGH

S_1	S_0	Output Configuration
0	0	Registered/Active-LOW
0	1	Registered/Active-HIGH
1	0	Combinatorial/Active-LOW
1	1	Combinatorial/Active-HIGH

0 = Unblown Fuse
1 = Blown Fuse

Feedback

Another feature of the AmPAL10H20EV8/AmPAL10020EV8 output macrocell structure is the flexibility of its feedback selection. The feedback can be from either the I/O line or the registered output. The feedback multiplexer is also controlled by the S_1 fuse. The feedback path changes with the output-mode selection. If the output is selected to be registered, the feedback is registered. If the output is combinatorial, the feedback is from the I/O line. This feature enables the designer to optimally use the device to meet precise application requirements. Additionally, it also allows him/her to make complex control state machines for particular design requirements.

Output Enable

Each of the eight output logic macrocells of the AmPAL10H20EV8/AmPAL10020EV8 contains a dedicated product term for output-enable function. When this product term is asserted LOW, the output is forced into a LOW state where it remains until the output-enable product term goes HIGH.

PRESET and RESET

To improve in-system functionality, the AmPAL10H20EV8/AmPAL10020EV8 has additional PRESET and RESET product terms. Common asynchronous-RESET and PRESET are provided for all the registers of the AmPAL10H20EV8/AmPAL10020EV8. When the asynchronous PRESET product term is asserted HIGH the output registers are loaded with a HIGH and when the RESET product term is asserted HIGH the output registers are loaded with a LOW independent of the clock. These functions are particularly useful for system power-on and reset.

PRELOAD

To simplify testing, the AmPAL10H20EV8/AmPAL10020EV8 is designed with PRELOAD circuitry that provides an easy method for testing logical functionality. PRELOAD allows any arbitrary values to be loaded into the PAL device's output registers.

A typical functional-test sequence would be to verify all possible outputs for the device being tested. To verify these transitions requires the ability to set the state registers to an arbitrary "present state" value and to set the device inputs to

any arbitrary "present input" value. Once this is done, the state machine is then clocked into a new state, or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD capability, it is difficult and in some cases impossible to load an arbitrary present state value into the registers. This can lead to logic-verification sequences which are either incomplete or excessively long. With PRELOAD capability, logic verification sequences can be greatly shortened, hence reducing the test time and the development costs, and guaranteeing proper in-system operation.

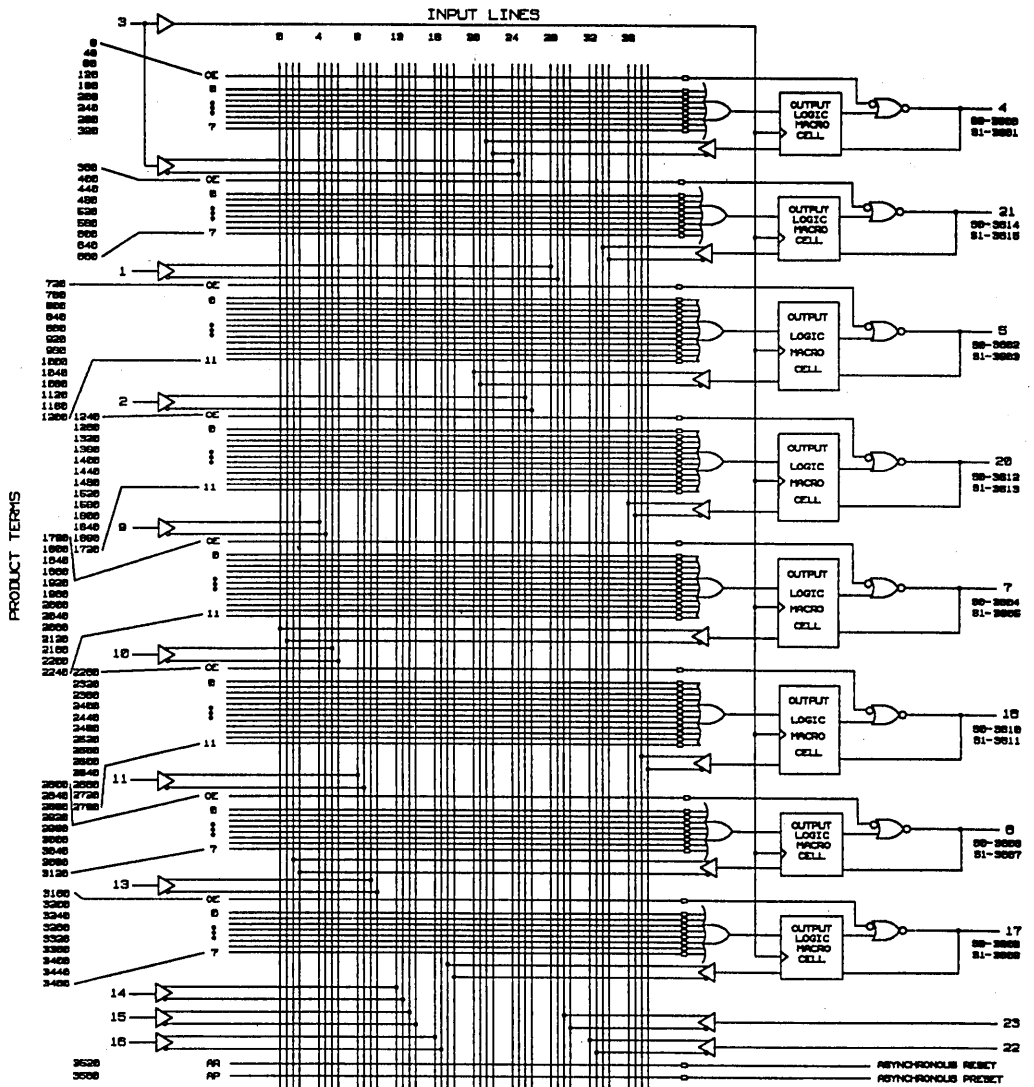
Fabrication

The AmPAL10H20EV8/AmPAL10020EV8 is manufactured using Advanced Micro Devices' new IMOX-III SLOT-isolation process. This advanced process offers increased density and reduced internal capacitance resulting in the fastest possible programmable logic devices.

The AmPAL10H20EV8/AmPAL10020EV8 is fabricated with AMD's fast programming and highly reliable platinum-silicide fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Platinum-silicide was selected as the fuse-link material to achieve a well controlled melt rate resulting in large non-conductive gaps which ensure very stable, long-term reliability. Extensive operating testing has proven that this low-field, large-gap technology offers the best reliability for fusible-link programmable logic.

Testing

The AmPAL10H20EV8/AmPAL10020EV8 contains many internal test features, including circuitry and extra fuses which allow AMD to test each part before shipping. This ensures extremely high post-programming functional yields. The test fuses are programmed to assure the ability of each part to perform correct programming. There are extra test words which are preprogrammed during manufacturing and tested to ensure correct logical operation and provide extra test paths to achieve excellent parametric correlation.



LD000840

Figure 4. Logic Diagram — JEDEC Fuse Numbering for AmPAL10H20EV8/AmPAL10020EV8

Note: Pin out for DIP only.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage (V_{EE})	
with Respect to Ground	-8 V to 0 V
DC Input Voltage	
with Respect to Ground	V_{EE} to 0 V
Output Current	
-Continuous	35 mA
-Surge	100 mA

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices — 100K Devices	
Temperature (T_A) Operating Free Air	0 to +85°C
Supply Voltage (V_{EE})	-5.7 to -4.2 V
Commercial (C) Devices — 10KH Devices	
Temperature (T_A) Operating Free Air	0 to +75°C
Supply Voltage (V_{EE})	-5.46 to -4.94 V
Extended Commercial (E) Devices	
Temperature (T_A)	-55°C Min.
Temperature (T_C) Operating Case	+125°C Max.
Supply Voltage (V_{EE})	-5.72 to -4.68 V
Military* (M) Devices	
Temperature (T_A)	-55°C Min.
Temperature (T_C)	+125°C Max.
Supply Voltage (V_{EE})	-5.72 to -4.68 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at $T_C = +25^\circ\text{C}$, $+125^\circ\text{C}$ and -55°C .

DC CHARACTERISTICS over operating range unless otherwise specified**AmPAL10H20EV8 (Commercial)**

Parameter Symbols	Parameter Description	Test Conditions				Min.	Max.	Units
V _{OH}	Output HIGH Voltage	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	T _A = 0°C	-1020	-840	mV	
				T _A = +25°C	-980	-810		
				T _A = +75°C	-920	-735		
V _{OL}	Output LOW Voltage	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	T _A = 0°C	-1950	-1630	mV	
				T _A = +25°C	-1950	-1630		
				T _A = +75°C	-1950	-1600		
V _{IH}	Input HIGH Voltage	Guaranteed Input HIGH Voltage (Note 3)		T _A = 0°C	-1170	-840	mV	
				T _A = +25°C	-1130	-810		
				T _A = +75°C	-1070	-735		
V _{IL}	Input LOW Voltage	Guaranteed Input LOW Voltage (Note 3)		T _A = 0°C	-1950	-1480	mV	
				T _A = +25°C	-1950	-1480		
				T _A = +75°C	-1980	-1450		
I _{IH}	Input HIGH Current	V _{IN} = V _{IH} (Max.)		T _A = 0°C		220	μA	
				T _A = +25°C		220		
				T _A = +75°C		220		
I _{IL}	Input LOW Current	V _{IN} = V _{IL} (Min.)		T _A = 0°C	0.5		μA	
				T _A = +25°C	0.5			
				T _A = +75°C	0.3			
I _{EE}	Power Supply Current	All Inputs and Outputs Open		T _A = 0°C	-230		mA	
				T _A = +25°C	-230			
				T _A = +75°C	-230			

Table continues on following page.

Notes: See notes following Military DC characteristics table.

AmPAL10H20EV8 (Military); included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted.

Parameter Symbols	Parameter Description	Test Conditions			Min.	Max.	Units
V _{OH}	Output HIGH Voltage	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{OL}	Output LOW Voltage			T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{IH}	Input HIGH Voltage	Guaranteed Input HIGH Voltage (Note 3)		T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
V _{IL}	Input LOW Voltage	Guaranteed Input LOW Voltage (Note 3)		T _A = 0°C	TBD	TBD	mV
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{IH}	Input HIGH Current	V _{IN} = V _{IH} (Max.)		T _A = 0°C	TBD	TBD	μA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{IL}	Input LOW Current	V _{IN} = V _{IL} (Min.)		T _A = 0°C	TBD	TBD	μA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	
I _{EE}	Power Supply Current	All Inputs and Outputs Open		T _A = 0°C	TBD	TBD	mA
				T _A = +25°C	TBD	TBD	
				T _A = +75°C	TBD	TBD	

AmPAL10020EV8 (Commercial)

Parameter Symbols	Parameter Description	Test Conditions		Min.	Max.	Units
V _{OH}	Output Voltage HIGH	V _{IN} = V _{IH} (Max.) or V _{IL} (Min.)	Loading is 50 Ω to -2.0 V	-1025	-880	mV
V _{OL}	Output Voltage LOW			-1810	-1620	mV
V _{OHC}	Output Voltage HIGH	V _{IN} = V _{IH} (Min.) or V _{IL} (Max.)		-1035		mV
V _{OLC}	Output Voltage LOW				-1610	mV
V _{IH}	Input Voltage HIGH	Guaranteed Input Voltage HIGH (Note 3)		-1165	-880	mV
V _{IL}	Input Voltage LOW	Guaranteed Input Voltage LOW (Note 3)		-1810	-1475	mV
I _{IH}	Input Current HIGH	V _{IN} = V _{IH} (Max.)			220	μA
I _{IL}	Input Current LOW	V _{IN} = V _{IL} (Min.)		0.5		μA
I _{EE}	Power Supply Current	All Inputs and Outputs Open		-230		mA

Notes: 1. Guaranteed with transverse air flow exceeding 400 linear F.P.M.

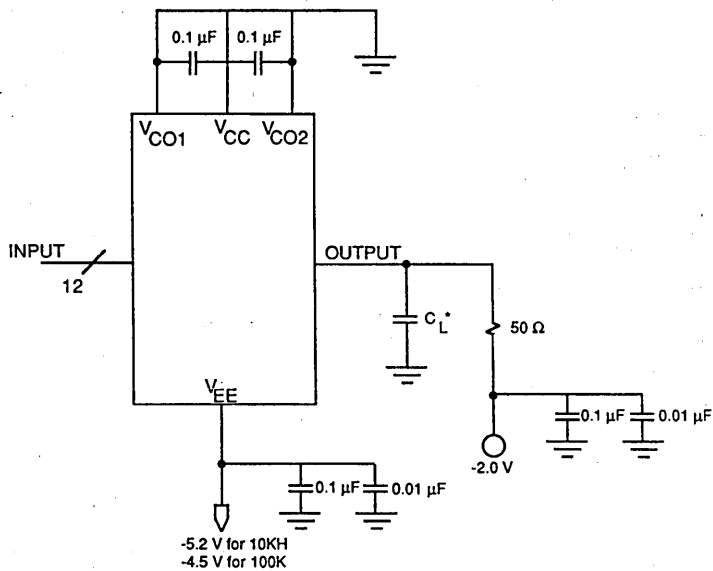
2. The relative values of the specified conditions and limits will be referenced to an algebraic scale. The extremities of the scale are:
 "Max." the value closest to positive infinity.
 "Min." the value closest to negative infinity.

3. These are absolute voltages with respect to the device ground pin and include all overshoots due to system and/or tester noise. Do not attempt to test these values without suitable equipment and fixturing.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 9, 10, 11 tests unless otherwise noted

No.	Parameter Symbol	Parameter Description	Test Conditions	C Devices				E/M Devices				Units
				-6		TBD		-8		TBD		
				Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
1	t _{PD}	Input or Feedback to Output	See Switching Test Circuit		6				8			ns
2	t _{EA}	Input to Output Enable			6				8			ns
3	t _{ER}	Input to Output Disable			6				8			ns
4	t _{CO}	Clock to Output			3.5				4.5			ns
5	t _S	Input or Feedback Setup Time		4.5				5.5				ns
6	t _H	Hold Time		0				0				ns
7	t _P	Clock Period (t _S + t _{CO})		8				10				ns
8	t _W	Clock Width		3				5				ns
9	f _{MAX}	Maximum Frequency			125				100			MHz
10	t _{AW}	Asynchronous RESET/PRESET Width		6				8				ns
11	t _{AR}	Asynchronous RESET/PRESET Recovery Time		6				8				ns
12	t _{AP}	Asynchronous RESET/PRESET to Registered Output RESET/PRESET			6				8			ns

SWITCHING TEST CIRCUIT

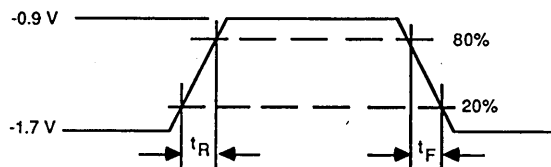


TC003930

* $C_L = 5 \text{ pf}$ for 10KH
 $C_L = 3 \text{ pf}$ for 100KH

SWITCHING TEST WAVEFORM

INPUT PULSE



WF023080

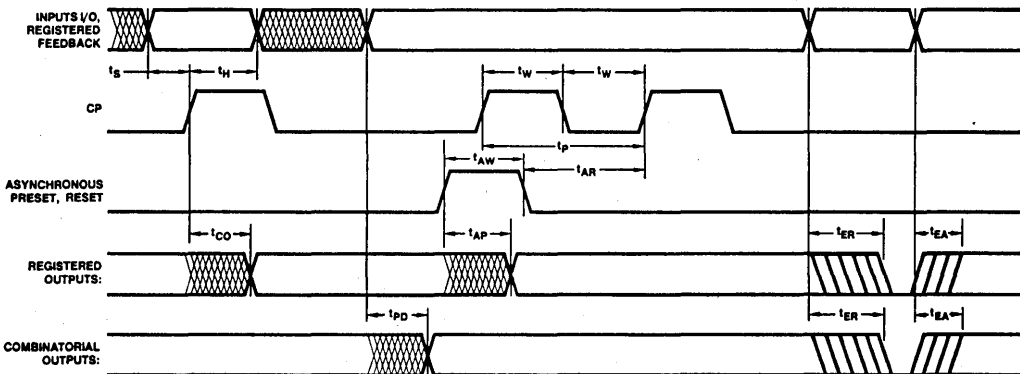
$t_R = t_F = 2.2 \text{ ns Max. for 10KH}$
 $t_R = t_F = 1.0 \text{ ns Max. for 100K}$

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS



WF022284

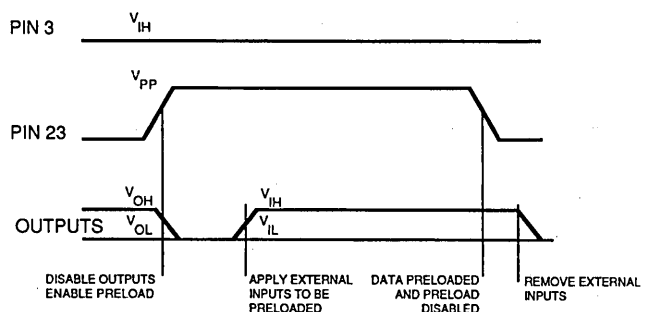
PRELOAD OF REGISTERED OUTPUTS

The AmPAL10H20EV8/AmPAL10020EV8 registered outputs are provided with circuitry to allow loading each register to either a HIGH or LOW state. This simplifies testing as any state can be loaded into the registers to control test sequenc-

ing. The pin levels and the timing necessary to perform the PRELOAD function are detailed below.

PRELOAD is accessed by applying V_{PP} on pin 23. The data to be preloaded is set on the output pins. Bringing pin 23 back to a logic-LOW level loads the data into the output registers.

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
V_{IH}	Input HIGH Level During PRELOAD and Verify	-1.1	-0.9	-0.7	V
V_{IL}	Input LOW Level During PRELOAD and Verify	-1.85	-1.65	-1.45	V
V_{PP}	PRELOAD Enable Voltage Applied to Pin 23	1.8	2.0	2.2	V



WF023100

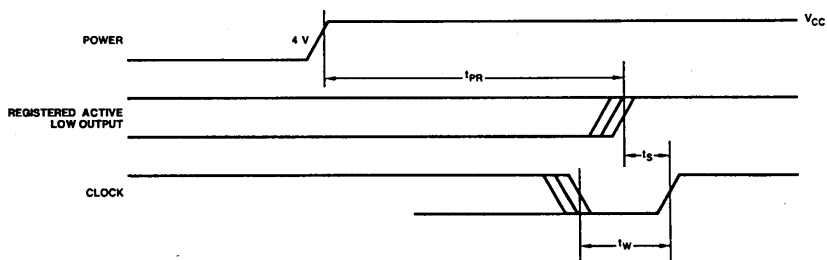
PRELOAD Timing Waveform

POWER-UP RESET

The registers in the AmPAL10H20EV8/AmPAL10020EV8 have been designed with the capability to RESET during system power-up. Following power-up, all registers will be LOW. The output state will depend upon the state of the output buffer and the polarity fuse. This feature provides extra flexibility to the designer. A timing diagram and a parameter table are shown below. Due to the asynchronous operation of

the power-up RESET and the wide range of ways V_{CC} can rise to steady state, two conditions are required to insure a valid power-up RESET. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following RESET, the CLK input must not be driven from LOW-to-HIGH until all applicable input and feedback setup times are met.



WF022301

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up RESET Time		600	1000	ns
t_S	Input or Feedback Setup Time	See Switching Characteristics			
t_W	Clock Width				

PROGRAMMING AND VERIFICATION

The AmPAL10H20EV8/AmPAL10020EV8 is programmed and verified using AMD's ECL programmable-logic-programming algorithm. The fuse to be programmed is selected by its input line number (array row), its product-term number (array column), and by the output associated with it (one at a time). The levels for addressing the rows and columns are all in ECL (10KH) levels. The fuse is then programmed and verified by applying a simple sequence of voltages as shown in the programming waveform diagram. The values of the programming voltages are shown in the table of programming parameters.

Input line numbers (0 to 39) are addressed using a full decode scheme via ECL levels on pins 9 – 11 and 13 – 15 where pin 15 is the LSB and pin 9 is the MSB. Input line addressing is shown in Table 1. Note that one input line is given to select all the architectural fuses.

Product terms are addressed using a full decode scheme via ECL levels on pins 1, 2, 22 and 23 where pin 23 is the LSB and pin 1 is the MSB. Product terms (column number) addressing is shown in Table 2.

The sequence of signals and events associated with them during programming and verification modes is shown in the Programming Waveforms. The row and column addresses are set up on the corresponding input pins. Then V_{CO1} (pin 6) is brought to a programming voltage of V_{COP} . The normal I/O pins are now disabled and the address decode is enabled. The device is now in a verification mode. V_{CO2} (pin 19) is now brought to the V_{COP} programming voltage. The programming mode is enabled. A high voltage of V_{OP} (Fuse-Enable Voltage) on the output pin corresponding to the fuse to be blown activates the programming voltage, and the fusing is started.

After a certain programming time, the fuse is blown and the programming voltage is removed from V_{CO2} . The device is back in the verification mode. A high clock pulse triggers the output registers which are then verified for V_{Blown} level (V_{IHP} or V_{ILP}). After verification is done, the programming voltage is removed from V_{CO1} and the part is back in normal mode.

Security Programming

A security fuse is provided on each AmPAL10H20EV8/AmPAL10020EV8 to protect any user proprietary logic design. It is addressed and programmed like any other fuse in the array. Once blown, the circuitry enabling any fuse verification and register PRELOAD is permanently disabled. The security fuse is verified by verifying the whole fuse array as if every fuse was blown.

Programming Yield

AMD PAL devices have been designed to ensure extremely high programming yields. To help ensure that a part was correctly programmed, once the programming is completed the entire fuse array should be re-verified at both LOW and HIGH V_{CC} . Re-verification can be accomplished by reading all eight outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PAL devices contain many internal features, including circuitry and extra fuses to test the ability of each part to perform programming before shipping, and to assure a correct logical operation for a correctly programmed part. Programming-yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS ($T_A = 25^\circ\text{C}$)

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
V_{IHP}	Input HIGH Level During Programming & Verify	-1.1	-0.9	-0.7	V
V_{ILP}	Input LOW Level During Programming & Verify	-1.85	-1.65	-1.45	V
V_{OP}	Fuse Enable Voltage Applied to Output Being Programmed @ 1 – 25 mA (Additional 80 mA if Output Terminated to 50-ohm Load)	1.8	2.0	2.2	V
V_{COP}	Programming Voltage Applied to V_{CO1} , V_{CO2} @ 15 – 200 mA	14.0	15.0	16.0	V
V_{CO1} & V_{CO2}	Power Supply for the Output Stage Sourcing	-0.1	0	0.1	V
V_{CCP}	V_{CC} During Programming & Verify $I_{CC} = 50 - 250$ mA Sourcing $I_{CC} = 50 - 150$ mA Sinking	-0.3	0	0.3	V
V_{EEP}	V_{EE} During Programming & Verify	-5.4	-5.2	-5.0	V
V_{ONP}	Termination Voltage for Output-Load Resistor	-2.1	-2.0	-1.9	V
R	Output-Load Resistor	47.5	50.0	52.5	Ω
t_p	Fusing Time First Attempt	40	50	100	μs
	Fusing Time Second Attempt	4	5	10	ms
t_D	Delays Between Various Level Changes	100	200	1000	ns
t_V	PRELOAD During Which Output is Sensed for V_{Blown} (V_{IHP} or V_{ILP}) Level			500	ns

TABLE 1. INPUT ADDRESSING

Input Line Number	Address Pin States					
	9	10	11	13	14	15
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	L
3	L	L	L	L	H	H
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	H
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	H
12	L	L	H	H	L	L
13	L	L	H	H	L	H
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	H
18	L	H	L	L	H	L
19	L	H	L	L	H	H
20	H	L	L	L	L	L
21	H	L	L	L	L	H
22	H	L	L	L	H	L
23	H	L	L	L	H	H
24	H	L	L	H	L	L
25	H	L	L	H	L	H
26	H	L	L	H	H	L
27	H	L	L	H	H	H
28	H	L	H	L	L	L
29	H	L	H	L	L	H
30	H	L	H	L	H	L
31	H	L	H	L	H	H
32	H	L	H	H	L	L
33	H	L	H	H	L	H
34	H	L	H	H	H	L
35	H	L	H	H	H	H
36	H	H	L	L	L	L
37	H	H	L	L	L	H
38	H	H	L	L	H	L
39	H	H	L	L	H	H
	RESERVED					
63*	H	H	H	H	H	H

*Architecture Row

TABLE 2. COLUMN NUMBER ADDRESSING

Address				Output Pins							
1	2	22	23	4 OUT1	5 OUT2	7 OUT3	8 OUT4	17 OUT5	18 OUT6	20 OUT7	21 OUT8
L	L	L	L	0	0	0	0	0	0	0	0
L	L	L	H	1	1	1	1	1	1	1	1
L	L	H	L	2	2	2	2	2	2	2	2
L	L	H	H	3	3	3	3	3	3	3	3
L	H	L	L	4	4	4	4	4	4	4	4
L	H	L	H	5	5	5	5	5	5	5	5
L	H	H	L	6	6	6	6	6	6	6	6
L	H	H	H	7	7	7	7	7	7	7	7
H	L	L	L	–	8	8	–	–	8	8	–
H	L	L	H	–	9	9	–	–	9	9	–
H	L	H	L	–	10	10	–	–	10	10	–
H	L	H	H	–	11	11	–	–	11	11	–
H	H	L	L	OE	OE	OE	OE	OE	OE	OE	OE
H	H	L	H	OP	OP	OP	OP	OP	OP	OP	OP
H	H	H	L	R/C	R/C	R/C	R/C	R/C	R/C	R/C	R/C
H	H	H	H	SF	AR	–	–	–	–	AP	–

AR = Async. RESET

AP = Async. PRESET

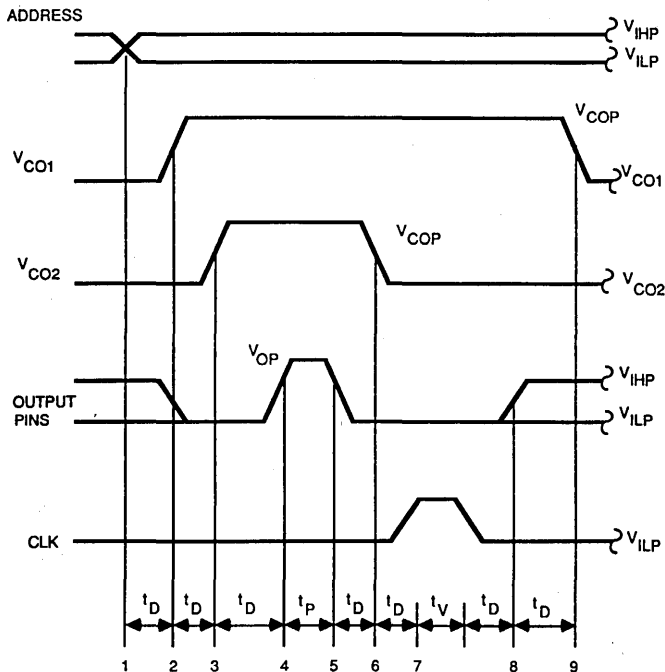
OE = Output Enable

R/C = Reg/Comb Fuse

OP = Output Polarity Fuse

SF = Security Fuse

PROGRAMMING WAVEFORMS



WF023110

AmPAL10H20EV8/AmPAL10020EV8 PROGRAMMING SUPPORT INFORMATION

Hardware Vendor	Programmer Model(s)	Personality Module	Socket Adaptor
Data I/O 10525 Willow Road N.E. Redmond, WA 98052 (206) 881-6444	System 29	Under Development	Not Required
	UNISITE 40	Not Required	Not Required
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94089 (408) 745-1991 or (800) 227-8836	Model PPZ	Under Development	Not Required
	Model ZL30A	Under Development	Not Required

The machines noted above have been qualified by AMD to ensure high programming yields. Check with the factory to determine current status of equipment noted as "Under Development," or for other available models.

Design-Aid Software for AmPAL10H20EV8/AmPAL10020EV8

Software Vendor	Software Package	Comments
P-CAD Systems, Inc. 1290 Parkmoor Ave. San Jose, CA 95126 (408) 971-1300	CUPL	
Advanced Micro Devices, Inc. 901 Thompson Pl. Sunnyvale, CA 94088 (408) 732-2400	AmCUPL	Supported by P-CAD Systems, Inc.
Data I/O 10525 Willow Road N.E. Redmond, WA 98052 (206) 881-6444	ABEL	

AmPAL*21VT8

24-Pin Dual-Clock Programmable Array Logic

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- Enhanced second-generation PAL architecture
- 15 ns maximum propagation delay, $f_{MAX} = 40$ MHz
- Up to twenty-one inputs and eight outputs
- Two banks of four user-programmable output logic macrocells for registered or combinatorial operation
- Individually user-programmable output polarity
- Variable product term distribution for increased design flexibility
- Individual or common clock with programmable polarity for each bank of output logic macrocells
- Two product terms for enable of each output
- Asynchronous-RESET and synchronous-PRESET product terms for each bank of output logic macrocells
- Power-up RESET and PRELOAD capability
- Superior quality
 - Full AC and DC parametric testing performed on every part
 - Exclusive on-chip test circuitry ensures post-programming functional yield (PPFY) of 99.9%.
- Platinum-silicide fuses ensure high programming yield (> 98%), fast programming and unsurpassed reliability

GENERAL DESCRIPTION

The AmPAL21VT8 is an ultra-fast and enhanced second-generation Programmable Array Logic (PAL) device. It uses the familiar sum-of-products (AND-OR) single array logic structure, allowing users to customize logic functions by programming the device for specific applications. Fabricated with AMD's new advanced bipolar IMOX-III SLOT-process technology and incorporating, with enhancements, the innovative architectural features of the AmPAL22V10, the AmPAL21VT8 is an extremely versatile PAL device.

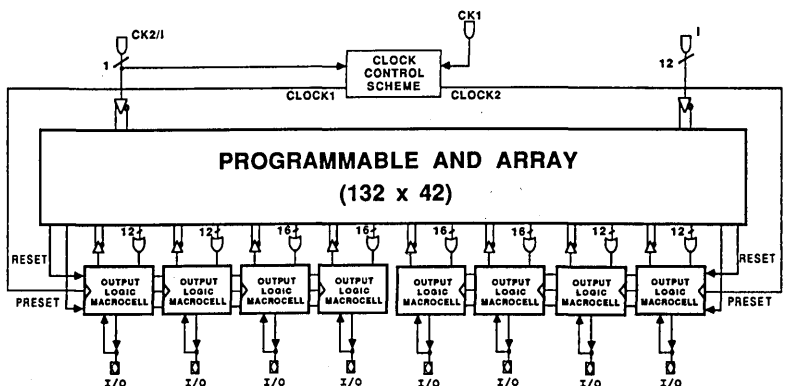
The AmPAL21VT8 contains up to twenty-one array inputs and eight outputs. The eight outputs are output logic macrocells (as in the AmPAL22V10) organized into two banks of four each. Each output logic macrocell is capable of being programmed as "combinatorial" or "registered" with active-HIGH or active-LOW polarity. Each of the two banks of output logic macrocells can be clocked individually or with a common clock. In addition, the individual/common clock has user-programmable polarity. This flexibility permits the system designer to tailor the device to the particular application requirements.

Increased logic power has been built into the AmPAL21VT8 by providing a variable number of logical product terms per output. Four outputs have twelve logical product terms each and the other four have sixteen logical product terms each. This variable allocation of logical product terms allows complex logic functions to be implemented in a single AmPAL21VT8.

The AmPAL21VT8 also offers designers increased flexibility and control over the output enable function. Each output is logically controlled by an OR of two logical product terms. This allows the designer to use more complex control than previously available.

System operation has been enhanced by the addition of synchronous-PRESET and asynchronous-RESET product terms common to all the output logic macrocells in a bank. The AmPAL21VT8 also incorporates power-up RESET on all the registered outputs. It also has the capability to PRELOAD registers to any desired state during testing. This is essential to permit full logic verification during test.

BLOCK DIAGRAM



BD006780

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.
IMOX is a trademark of Advanced Micro Devices, Inc.

Publication # 08602 Rev. A Amendment /0
Issue Date: October 1986



Figure 1. AmPAL21VT8 Output Logic Macrocell

OPERATING RANGES

Temperature (T_A)..... 0 to +75°C
Supply Voltage (V_{CC}) +5.75 V to +5.25 V

AmPAL*22V10

24-Pin IMOX™ Programmable Array Logic (PAL)

DISTINCTIVE CHARACTERISTICS

- Second-generation PAL architecture
- Increased logic power — up to 22 inputs and 10 outputs
- Increased product terms — average 12 per output
- Variable product term distribution improves ease of use
- Each output user programmable for registered or combinatorial operation
- Individually user programmable output polarity
- Extra terms provide logical synchronous PRESET and asynchronous RESET capability
- Comes in standard and high-speed versions — 18 ns typical propagation delay
- TTL-level PRELOAD for improved testability
- Packaged in 24-pin Slim DIP and 28-pin chip carrier packages
- Platinum-Silicide fuses ensure high programming yield, fast programming and unsurpassed reliability
- Full AC and DC testing done at the factory utilizing special designed-in test features

GENERAL DESCRIPTION

The AmPAL22V10 is a second-generation Programmable Array Logic (PAL) device. It utilizes the familiar sum-of-products (AND-OR) logic structure, allowing users to program custom logic functions. The AmPAL22V10 is an extension of the PAL concept. First-generation devices were largely limited to TTL-replacement applications. The AmPAL22V10 permits the development of custom LSI functions of 500 to 800 equivalent gate complexity.

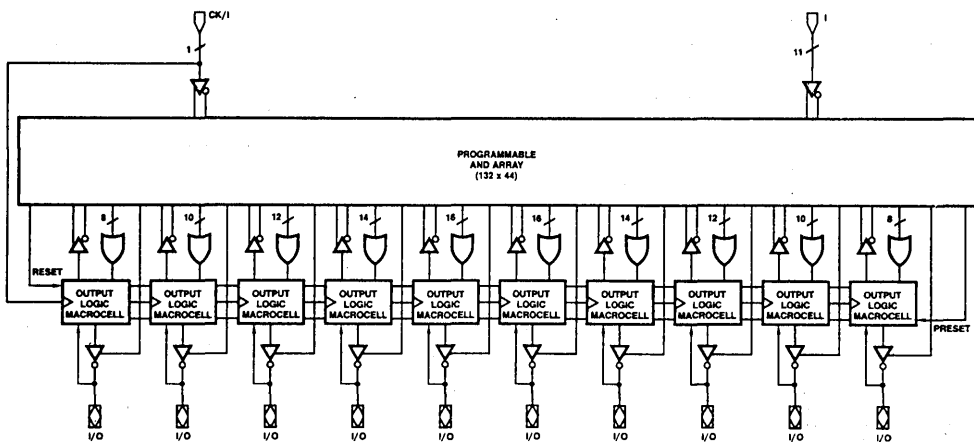
The AmPAL22V10 contains up to 22 inputs and 10 outputs. It incorporates the unique capability of defining and programming the architecture of each output on an individual basis. Each output is user programmable for either registered or combinatorial operation. This allows the designer to optimize the device design, by having only as many registers as needed. In addition each output has user-programmable output polarity, further simplifying design and contributing to the precise application requirements.

Increased logic power has been built into the AmPAL22V10 by increasing the number of product terms from 8-per-output to an average of 12-per-output. Further innovation can be seen in the introduction of variable product term distribution. This technique allocates from 8 to 16 logical product terms to each output (please refer to block diagram for distribution details). This variable allocation of terms allows far more complex functions to be implemented than in previous devices.

System operation has been enhanced by the addition of a synchronous-PRESET and an asynchronous-RESET product term. These terms are common to all output registers.

The AMPAL22V10 also incorporates power-up RESET and the unique capability to PRELOAD the output registers to any desired state during testing. PRELOAD is essential to permit full logical verification during test.

BLOCK DIAGRAM



BD006590

IMOX is a trademark of Advanced Micro Devices, Inc.
PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

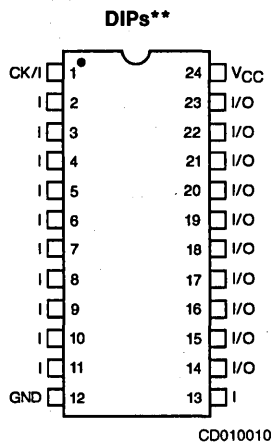
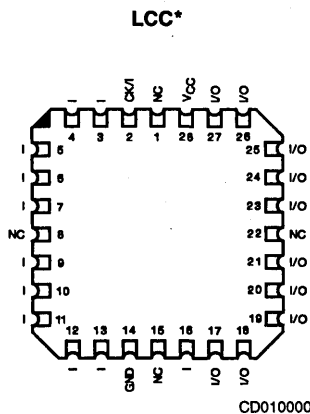
4-83

<u>Publication #</u>	<u>Rev.</u>	<u>Amendment</u>
04162	E	/0
Issue Date: October 1986		

AMPAL*22V10

4

CONNECTION DIAGRAM Top View



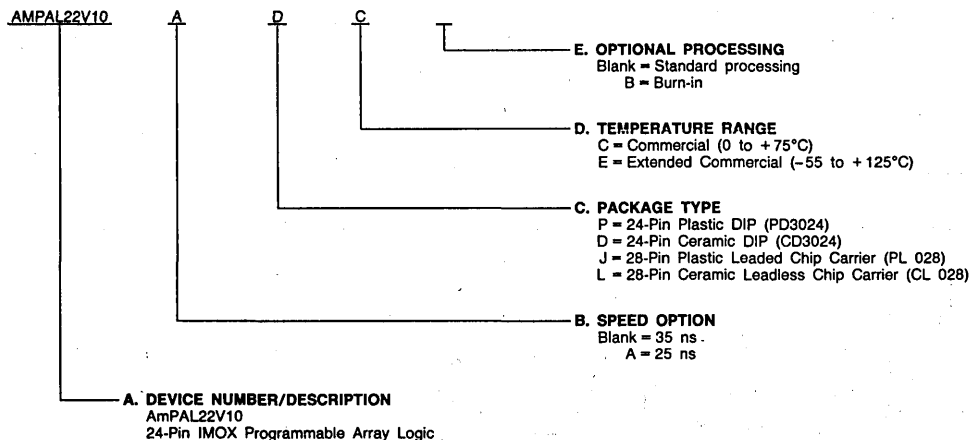
****Also available in Flatpack. Pinouts identical to DIPs.**

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**

- A. Device Number
- B. Speed Option (if applicable)
- C. Package Type
- D. Temperature Range
- E. Optional Processing



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, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

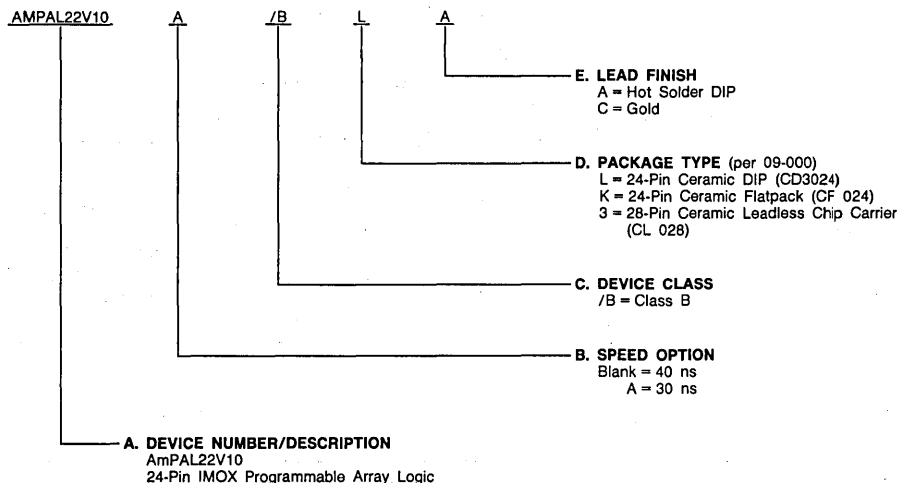
Valid Combinations	
AMPAL22V10	PC, DC, DCB, DE, JC, LC, LE
AMPAL22V10A	

ORDERING INFORMATION

APL 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:

- A. Device Number
- B. Speed Option (if applicable)
- C. Device Class
- D. Package Type
- E. Lead Finish



Valid Combinations	
AMPAL22V10 AMPAL22V10A	/BLA/B3C/BKA

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 consist of Subgroups 1, 2, 3, 7, 8, 9, 10, and 11.

DESC Certified PAL Devices

Generic	AMD Part Number	DESC Numbers
22V10	AmPAL22V10A/BLA	5962-8605301LX
	AmPAL22V10A/B3C	5962-86053013X
	AmPAL22V10A/BKA	5962-8605301KX

FUNCTIONAL DESCRIPTION

The AmPAL22V10 is a second-generation Programmable Array Logic device. It contains a programmable fuse array organized in the familiar sum-of-products (AND-OR) structure.

The block diagram below shows the basic architecture of the AmPAL22V10. There are up to 22 inputs and 10 outputs available. The inputs are connected to a programmable-AND array which contains 120 logical product terms. Initially the AND gates are connected, via fuses, to both the TRUE and complement of every input. By selective programming of fuses the AND gates may be "connected" to only the TRUE input (by blowing the complement fuse), to only the complement input (by blowing the TRUE fuse), or to neither type of input (by blowing both fuses) establishing a logical "don't care." When both the TRUE and complement fuses are left intact, a logical FALSE results on the output of the AND gate. An AND gate with all fuses blown will assume the logical-TRUE state. The outputs of the AND gates are connected to fixed-OR gates. There is an average of 12 product terms per OR gate (output), and as the block diagram shows, variable product term distribution has been implemented. This technique allocates different quantities of logical product terms to different outputs, allowing more complex logical functions to be performed than were previously possible. Up to 16 logical terms can be evaluated in one output in a single clock cycle (no feedback necessary).

Output Logic Macrocells (OLMs)

A dramatic innovation in logic design is the implementation on the AmPAL22V10 of variable output architecture. This allows the user to program on an output-by-output basis the function of the outputs. As shown in the Output Logic Macrocell (OLM) diagram below, each output cell contains two additional fuses (S_0 and S_1). S_1 controls whether the output will be registered or combinatorial. S_0 controls the output polarity (active HIGH or active LOW). Depending on the states of these 2 fuses, an individual output will operate in one of four modes (see logic diagrams on next page). Registered/Active LOW; Registered/Active HIGH; Combinatorial/Active LOW; Combinatorial/Active HIGH. (Note that the feedback path also changes with output mode.) This innovation gives the designer more flexibility and enables him to optimize the device for precise application requirements. It also allows for better device utilization—you only program as many registers as are needed.

PRESET/RESET

To improve in-system functionality, the AmPAL22V10 has additional PRESET and RESET product terms. These terms are connected to all registered outputs. When the synchronous-PRESET product term is asserted (HIGH), the output registers will be loaded with a HIGH on the next LOW-to-HIGH clock transition. When the asynchronous-RESET product term is asserted (HIGH), the output registers will be immediately loaded with a LOW (independent of the clock). These functions are particularly useful for applications such as system power-on and reset.

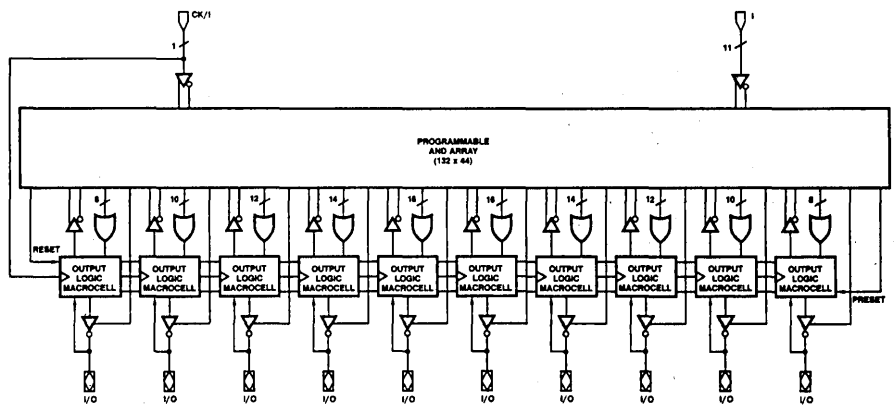


Figure 1. Block Diagram

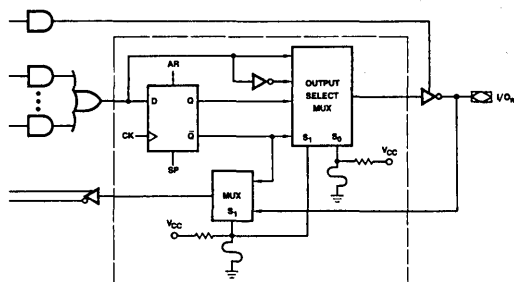


Figure 2. Output Logic Macrocell Diagram

S_1	S_0	Output Configuration
0	0	Registered/Active LOW
0	1	Registered/Active HIGH
1	0	Combinatorial/Active LOW
1	1	Combinatorial/Active HIGH

0 = Unblown Fuse
1 = Blown Fuse

PRELOAD

To simplify testing, the AmPAL22V10 is designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality. PRELOAD allows any arbitrary state value to be loaded into the output registers.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. To verify these transitions requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is then clocked into a new state, or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to load an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state register "interferes" with the inputs, forcing the state machine to go through many state transitions before it can reach an arbitrary state value. Therefore the test sequence will be mostly state initialization and not actual testing. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

In addition, complete logic verification may become impossible when states that need to be tested cannot be entered with normal state transitions. For example, the state which the machine powers up into cannot be tested because it cannot

be entered from the main sequence. Similarly, "forbidden" or don't care states that are not normally entered need to be tested to ensure that they return to the main sequence.

PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened, and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper in-system operation.

Fabrication

The AmPAL22V10 is manufactured using Advanced Micro Devices' IMOX oxide isolation process. This advanced process permits an increase in density and a decrease in internal capacitance resulting in the fastest possible programmable logic devices.

The AmPAL22V10 is fabricated with AMD's fast programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are preprogrammed during manufacturing to ensure extremely high field programming yields (> 98.5%), and provide extra test paths to achieve excellent parametric correlation.

Platinum Silicide was selected as the fuse-link material to achieve a well-controlled melt rate resulting in large nonconductive gaps that ensure very stable, long-term reliability. Extensive operating testing has proven that this low-field, large-gap technology offers the best reliability for fusible link programmable logic.

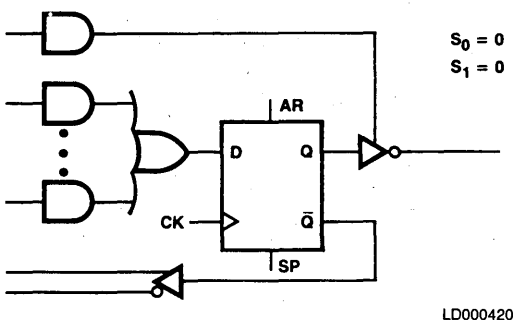


Figure 3-1. Registered/Active LOW

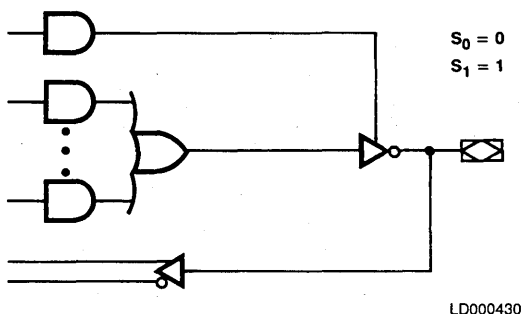


Figure 3-3. Combinatorial/Active LOW

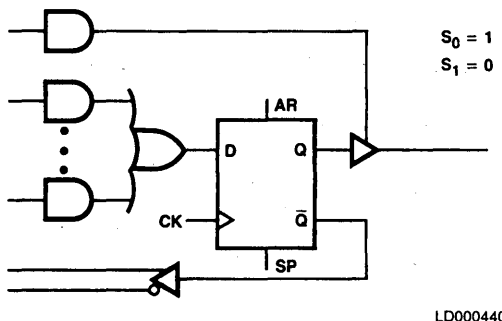


Figure 3-2. Registered/Active HIGH

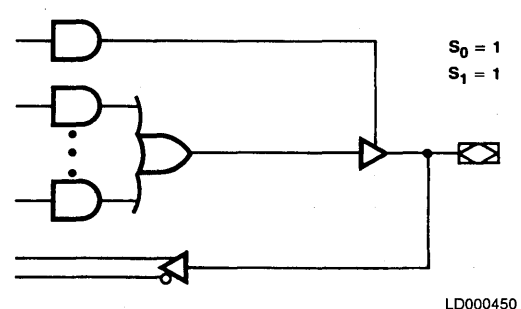
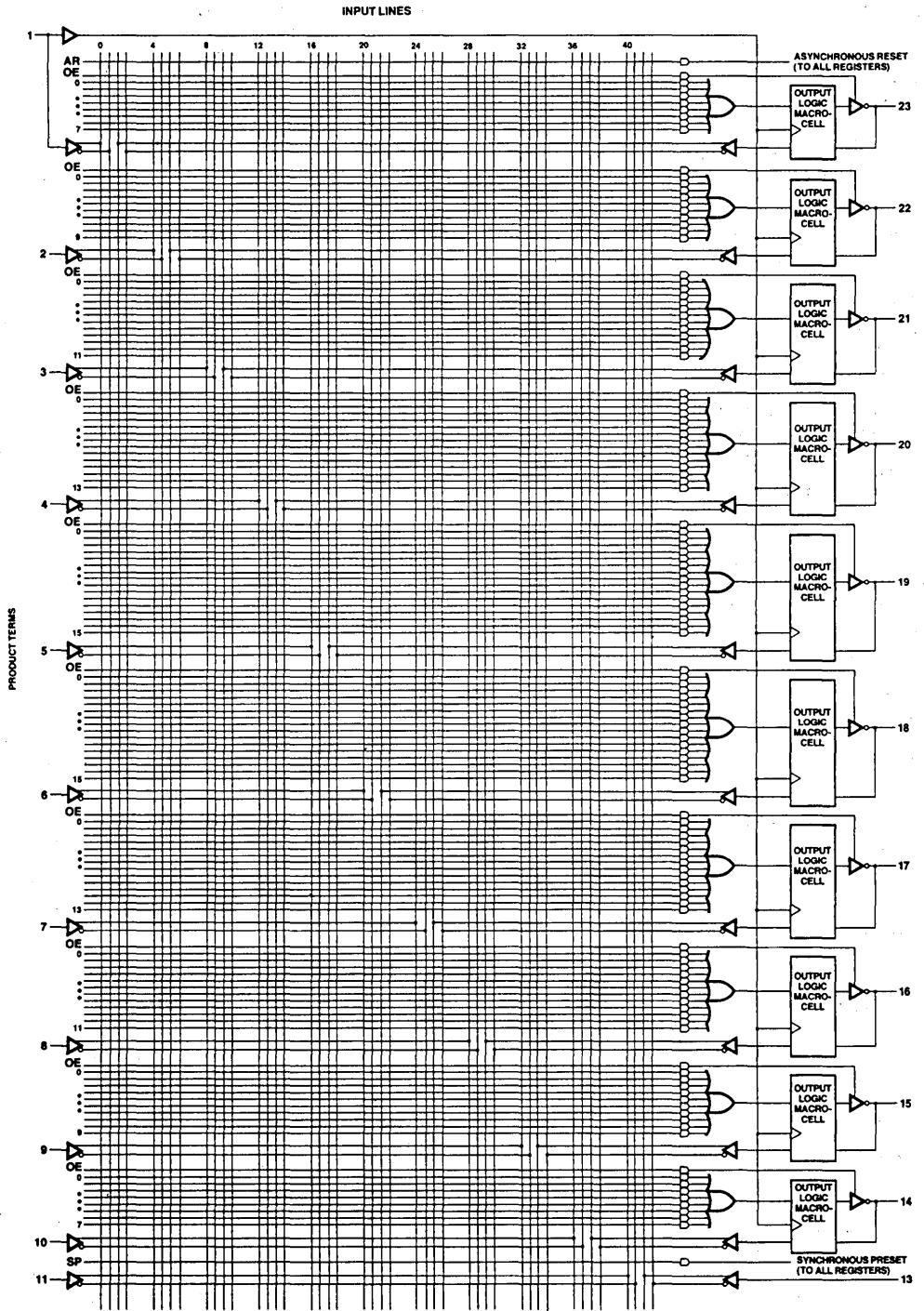


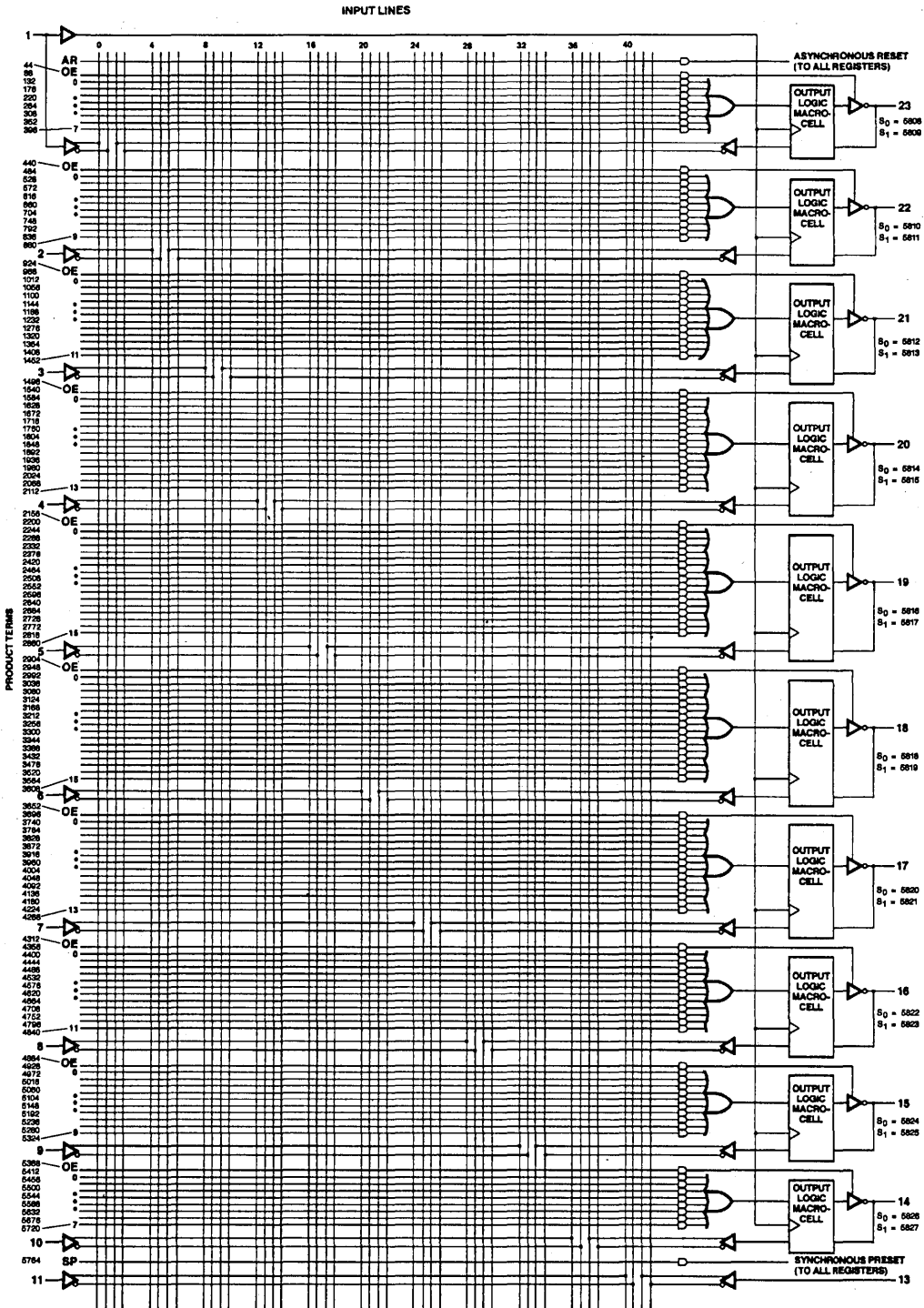
Figure 3-4. Combinatorial/Active HIGH



LD000480

Figure 4. AmPAL22V10* Logic Diagram

*Pinout for DIPs only.



LD000482

Figure 5. Logic Diagram — JEDEC Fuse Numbering

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage to Ground Potential (Pin 24 to Pin 12) Continuous	-0.5 to +7 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	16 V
Output Current into Outputs During Programming (Max. Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5 mA
Ambient Temperature with Power Applied	+125°C

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T _A) Operating Free Air	0°C to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V
Extended Commercial (E) Devices	
Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V
Military (M) Devices*	
Temperature (T _A)	-55°C Min.
Temperature (T _C) Operating Case	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military Product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions		Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -3.2 mA I _{OH} = -2 mA	C Devices E/M Devices	2.4	3.5	Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = 16 mA I _{OL} = 12 mA	C Devices E/M Devices		0.50	Volts
V _{IH} (Note 2)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for all Inputs		2.0			Volts
V _{IL} (Note 2)	Input LOW Level	Guaranteed Input Logical LOW Voltage for all Inputs				0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.40 V			-20	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V				25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V				1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)		-30	-50	-90	mA
I _{CC}	Power Supply Current	V _{CC} = Max.			150	180	mA
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA			-0.9	-1.2	Volts
I _{OZH}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V			100	μA
I _{OZL}			V _O = 0.4 V			-100	
C _{IN} †	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz (Note 5)			11		pF
					6		
C _{OUT} †	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz (Note 5)			9		

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5 V has been chosen to avoid test problems caused by tester ground degradation.

4. I/O pin leakage is the worst case of I_{OZX} or I_{IX} (where X = H or L).

5. Pinout for DIPs only.

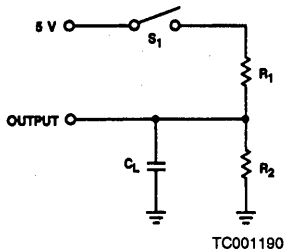
† Not included in Group A tests.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 7, 8, 9, 10, 11 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions	Typ. (Note 1)	C Devices				E/M Devices				Units
				"A"		"Std"		"A"		"Std"		
				Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t _{PD}	Input or Feedback to Non-Registered Output	C Devices R ₁ = 300 R ₂ = 390	18		25		35		30		40	ns
t _{EA}	Input to Output Enable		18		25		35		30		40	ns
t _{ER}	Input to Output Disable		18		25		35		30		40	ns
t _{CO}	Clock to Output		10		15		25		20		25	ns
t _S	Input or Feedback Setup Time		13	20		30		25		35		ns
t _H	Hold Time	E/M Devices R ₁ = 390 R ₂ = 750	-10	0		0		0		0		ns
t _P	Clock Period (t _S + t _{CO})			35		55		45		60		ns
t _W	Clock Width			15		25		20		30		ns
f _{MAX}	Maximum Frequency				28.5		18		22		16.5	MHz
t _{AW}	Asynchronous Reset Width			25		35		30		40		ns
t _{AR}	Asynchronous Reset Recovery Time			25		35		30		40		ns
t _{AP}	Asynchronous Reset to Registered Output Reset				30		40		35		45	ns

- Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.
2. t_{PD} is tested with switch S₁ closed and C_L = 50 pF.
3. For three-state outputs, output enable times are tested with C_L = 50 pF to the 1.5 V level; S₁ is open for high-impedance to HIGH tests and closed for high-impedance to LOW tests. Output disable times are tested with C_L = 5 pF. HIGH to high-impedance tests are made to an output voltage of V_{OH} - 0.5 V with S₁ open; LOW to high-impedance tests are made to the V_{OL} + 0.5 V level with S₁ closed.

SWITCHING TEST CIRCUIT

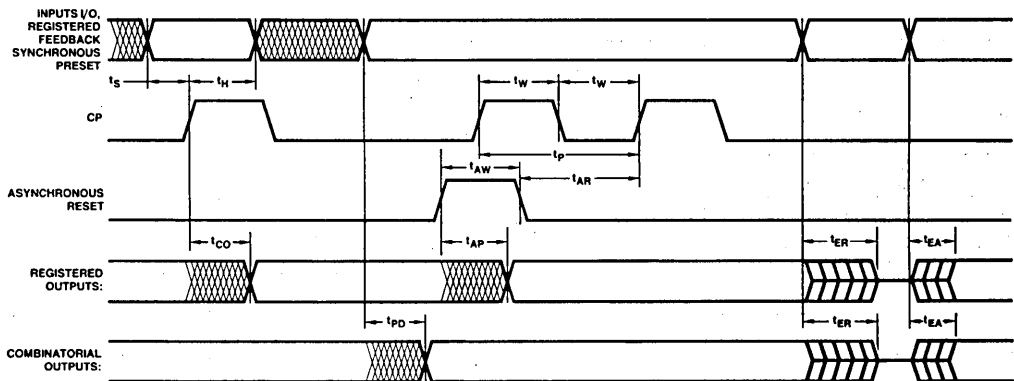


KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DONES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

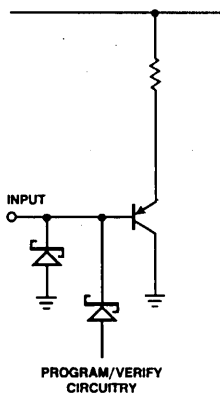
KS000010

SWITCHING WAVEFORMS



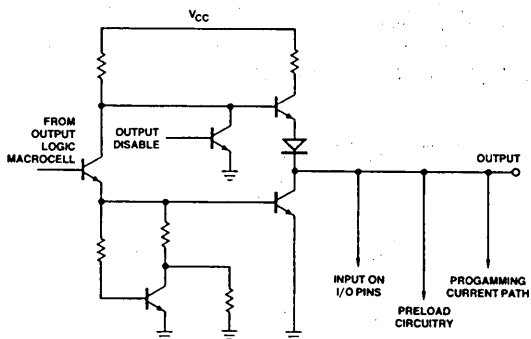
WF022280

INPUT/OUTPUT CURRENT DIAGRAM



IC000892

Input Circuitry



IC000900

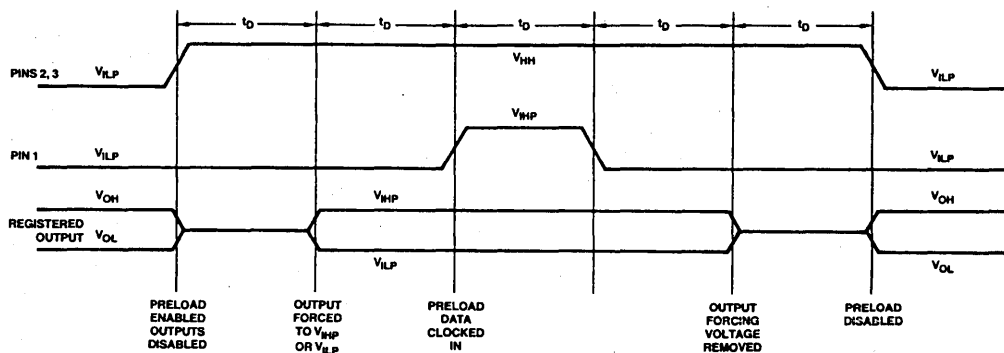
Output Circuitry

PRELOAD OF REGISTERED OUTPUTS

The AmPAL22V10 registered outputs are provided with circuitry to allow loading each register synchronously with either a

HIGH or LOW. This feature will simplify testing since any state can be loaded into the registers to control test sequencing.

The pin levels and timing necessary to perform the PRELOAD function are detailed below. Parameters are listed in the Programming Parameters table (page 12).



WF022293

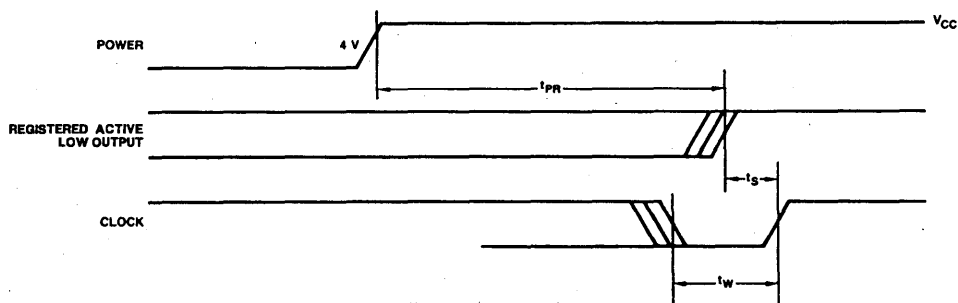
Level forced on registered output pin during PRELOAD cycle	Register Q output state after cycle
V_{IHP}	HIGH
V_{ILP}	LOW

Power-up RESET

The registered devices in the AMD PAL Family have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to LOW. The output state will depend on the polarity of the output buffer. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization. A timing diagram and parameter table are shown below. Due to

the asynchronous operation of the power-up reset and the wide range of ways V_{CC} can rise to its steady state, two conditions are required to insure a valid power-up reset. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.



WF022301

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up Reset Time		600	1000	ns
t_s	Input or Feedback Setup Time	See Switching Characteristics			
t_w	Clock Width				

Programming and Verification

The AmPAL22V10 is programmed and verified using AMD's standard programmable logic programming algorithm. The fuse to be programmed is selected by input line number (array row), product term (array column), and by output (one at a time). The fuse is then programmed and verified by applying a simple sequence of voltages to two control pins (1 and 13).

Input line numbers (0 – 43) are addressed using a full decode scheme via TTL levels on pins 6 – 11 where 6 is the LSB and 11 is the MSB. Even numbered input lines represent the TRUE version of a signal and odd numbered lines represent the complement. Input line addressing is shown in Table 1. Note that input lines 44 – 62 are reserved for further expansion and input line 63 is utilized for selecting the fuses used for programming output polarity and whether the output is registered or combinatorial.

Product terms are addressed using a one-of-24 addressing scheme on pins 2 – 5 where pin 2 is the LSB and 5 is the MSB. Product term addressing is shown in Table 2. Logical product terms (0 – 15) are selected via TTL levels on the four addressing pins. Note that outputs with fewer than 16 product terms will decode blank space for decoding values greater than the number of product terms on that output. Architectural product terms are selected by placing a zener voltage level (V_{HH}) on the MSB (pin 5) and using pins 2 – 4 for an additional eight decoding states (only 5 are used). The specific decoding of architectural features is best shown in Table 2.

Fuse selection by output must be done one output at a time (following control pin 1 going to V_{HH}) as shown in the Programming waveform diagram.

Once fuses have been selected, the simple programming and verification sequence may be completed as shown in the programming waveform diagram. AC and DC requirements for programming are shown in the Programming Parameters table.

Security Fuse Programming

A single fuse is provided on each AmPAL22V10 part to prevent unauthorized copying of PAL fuse patterns. Once blown, the circuitry enabling fuse verification and registered output PRELOAD is permanently disabled.

Programming of the security fuse is the same as an array fuse. Verification of a blown security fuse is accomplished by verifying the whole fuse array as if every fuse is blown.

Programming Yield

AMD PALs have been designed to insure extremely high programming yields (> 98.5%). To help insure that a part was correctly programmed, once programming is completed, the entire fuse array should be reverified at both LOW and HIGH V_{CC} . Reverification can be accomplished by reading all ten outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PALs contain many internal test features, including circuitry and extra fuses which allow AMD to test the ability of each part to perform programming before shipping, to assure high programming yields and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS ($T_A = 25^\circ\text{C}$)

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V_{HH}	Control Pin Extra HIGH Level (Note 1)	Pin 1 @ 10 – 40 mA	10	11	12	Volts
		Pin 13 @ 10 – 40 mA	10	11	12	
V_{OP}	Program Voltage Pins 14 – 23 @ 15 – 200 mA (Note 1)		14	15	16	Volts
V_{IHP}	Input HIGH Level During Programming and Verify		2.4	5	5.5	Volts
V_{ILP}	Input LOW Level During Programming and Verify		0.0	0.3	0.5	Volts
V_{CCP}	V_{CC} During Programming @ $I_{CC} = 50 - 200$ mA		5	5.2	5.5	Volts
V_{CCL}	V_{CC} During First Pass Verification @ $I_{CC} = 50 - 200$ mA		4.4	4.5	4.6	Volts
V_{CCH}	V_{CC} During Second Pass Verification @ $I_{CC} = 50 - 200$ mA		5.4	5.5	5.6	Volts
V_{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	Volts
V_{OP}/dt	Rate of Output Voltage Change		20		250	V/ μ sec
V_{13}/dt	Rate of Fusing Enable Voltage Change (Pin 13 Rising Edge) (Note 1)		100		1000	V/ μ sec
t_P	Fusing Time First Attempt		10	50	100	μ sec
	Subsequent Attempts (Maximum of 8)		4	5	10	msec
t_D	Delays Between Various Level Changes		100	200	15,000	ns
t_V	Period During Which Output is Sensed for V_{Blown} Level				500	ns
V_{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		$V_{CCP} - 0.3$	V_{CCP}	$V_{CCP} + 0.3$	Volts
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	k Ω

Notes: 1. Pinout for DIPs only.

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States						Input Line Number	Input Line Number Address Pin States					
	11	10	9	8	7	6		11	10	9	8	7	6
0	L	L	L	L	L	L	32	H	L	L	L	L	L
1	L	L	L	L	L	H	33	H	L	L	L	L	H
2	L	L	L	L	H	L	34	H	L	L	L	H	L
3	L	L	L	L	H	H	35	H	L	L	L	H	H
4	L	L	L	H	L	L	36	H	L	L	H	L	L
5	L	L	L	H	L	H	37	H	L	L	H	L	H
6	L	L	L	H	H	L	38	H	L	L	H	H	L
7	L	L	L	H	H	H	39	H	L	L	H	H	H
8	L	L	H	L	L	L	40	H	L	H	L	L	L
9	L	L	H	L	L	H	41	H	L	H	L	L	H
10	L	L	H	L	H	L	42	H	L	H	L	H	L
11	L	L	H	L	H	H	43	H	L	H	L	H	H
12	L	L	H	H	L	L							
13	L	L	H	H	L	H							
14	L	L	H	H	H	L							
15	L	L	H	H	H	H							
16	L	H	L	L	L	L							
17	L	H	L	L	L	H							
18	L	H	L	L	H	L							
19	L	H	L	L	H	H							
20	L	H	L	H	L	L							
21	L	H	L	H	L	H							
22	L	H	L	H	H	L							
23	L	H	L	H	H	H							
24	L	H	H	L	L	L							
25	L	H	H	L	L	H							
26	L	H	H	L	H	L							
27	L	H	H	L	H	H							
28	L	H	H	H	L	L							
29	L	H	H	H	L	H							
30	L	H	H	H	H	L							
31	L	H	H	H	H	H	63*	H	H	H	H	H	H

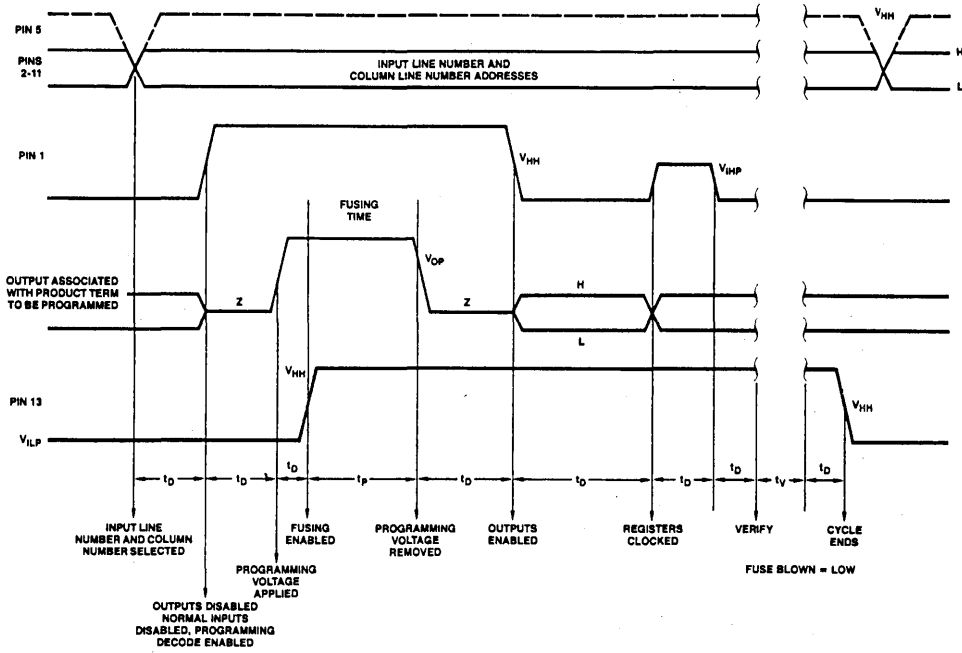
*Architecture row

TABLE 2. COLUMN NUMBER ADDRESSING

Column Number										Column Number Select Address Pin States				Description
										5	4	3	2	
0	0	0	0	0	0	0	0	0	0	L	L	L	L	Logical PTs
1	1	1	1	1	1	1	1	1	1	L	L	L	H	Logical PTs
2	2	2	2	2	2	2	2	2	2	L	L	H	L	Logical PTs
3	3	3	3	3	3	3	3	3	3	L	L	H	H	Logical PTs
4	4	4	4	4	4	4	4	4	4	L	H	L	L	Logical PTs
5	5	5	5	5	5	5	5	5	5	L	H	L	H	Logical PTs
6	6	6	6	6	6	6	6	6	6	L	H	H	L	Logical PTs
7	7	7	7	7	7	7	7	7	7	L	H	H	H	Logical PTs
-	8	8	8	8	8	8	8	8	-	H	L	L	L	Logical PTs
-	9	9	9	9	9	9	9	9	-	H	L	L	H	Logical PTs
-	-	10	10	10	10	10	10	-	-	H	L	H	L	Logical PTs
-	-	11	11	11	11	11	11	-	-	H	L	H	H	Logical PTs
-	-	-	12	12	12	12	-	-	-	H	H	L	L	Logical PTs
-	-	-	13	13	13	13	-	-	-	H	H	L	H	Logical PTs
-	-	-	-	14	14	-	-	-	-	H	H	H	L	Logical PTs
-	-	-	-	15	15	-	-	-	-	H	H	H	H	Logical PTs
OE	OE	OE	OE	OE	OE	OE	OE	OE	OE	HH	L	L	L	Output Enable
P	P	P	P	P	P	P	P	P	P	HH	L	L	H	Output Polarity
R	R	R	R	R	R	R	R	R	R	HH	L	H	L	Register/Non-Register Output
-	AR*	-	-	-	-	-	-	SP**	-	HH	L	H	L	*Asynchronous RESET **Synchronous PRESET
-	SF	-	-	-	-	-	-	-	-	HH	H	L	H	Security Fuse (Special Verify Required)
Pin 23	Pin 22	Pin 21	Pin 20	Pin 19	Pin 18	Pin 17	Pin 17	Pin 15	Pin 14					
Programming Access and Verify Pin														

Legend: L = V_{ILP}H = V_{IHP}HH = V_{HH}

PROGRAMMING WAVEFORMS



WF022580

AmpAL22V10 PROGRAMMING EQUIPMENT INFORMATION

Vendor	Programmer Model (s)	Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 98052 (206) 881-6464	System 19, 29 or 100	Logicpak 715-1983-003 (Rev 4 or Newer)	303A-004 (Rev 3 or Newer)
	Model 60A or 60H	360A-001 (Rev 2 or Newer)	On-Board
JMC PROMAC Division 2999 Monterey/Salinas Highway Monterey, CA 93940 (408) 373-3622	PROMAC P3 (Rev 2.0 or Newer)	On-Board	On-Board
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086 (408) 745-1991 or (800) 227-8836	Model PPZ	Zm2200 (Rev 10 or Newer)	On-Board (Rev 10 or Newer)
	Model ZL30A	On-Board (Rev 38 or Newer)	On-Board (Rev 38 or Newer)
Structured Design 988 Bryant Way Sunnyvale, CA 94087 (408) 988-0725	SD 1040 PAL Burner (Rev V1.02A or Newer)	On-Board	On-Board
Valley Data Sciences 2426 Charleston Road Mt. View, CA 94043 (415) 968-2900	VDS 160 Series	On-Board (Rev 1.03 or Newer)	On-Board
Varix Corporation 122 Spanish Village Suite 608 Dallas, TX 75248 (214) 437-0777	Omni-Programmer SP0-300	Under Development	Under Development
Wavetek-Digelec 1602 Lawrence Ave., Suite 113 Ocean, NJ 07712 (210) 493-2420	803 Series	FAM52 (Rev 5.4 or Newer)	DA55 (Rev B3 or Newer) (Rev V1.02A or Newer)

The machines noted above have been qualified by AMD to insure high programming yields. Check with the factory to determine the current status of vendors noted under development, or other available models.

Design Aid Software for AmpAL22V10

Software Vendor	Software Package	Comments
P-CAD Systems (408) 971-1300	CUPL	
Advanced Micro Devices (408) 732-2400	AmCUPL	Developed and supported by P-CAD Systems
Data I/O (206) 881-6444	ABEL	

AmPAL*22V10B

24-Pin IMOX™ Programmable Array Logic (PAL)

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- B-speed ($t_{PD} = 15$ ns) performance
- Increased logic power — up to 22 inputs and 10 outputs
- Increased product terms — average 12 per output
- Variable product-term distribution improves ease of use
- Each output user-programmable for registered or combinatorial operation
- Individually user-programmable output polarity
- Extra terms provide logical synchronous-PRESET and asynchronous-RESET capability
- Also comes in A-speed Low-Power (L) and Standard-speed Quarter-Power (Q) versions
- TTL level PRELOAD for improved testability
- 24-pin Slim DIP and 28-pin chip carrier packages
- Platinum silicide fuses ensure high programming yield, fast programming and unsurpassed reliability
- Full AC and DC testing done at the factory utilizing special designed-in test features
- 3000-V Input ESD Protection

GENERAL DESCRIPTION

The AmPAL22V10B is a second-generation Programmable Array Logic (PAL) device. It utilizes the familiar sum-of-products (AND-OR) logic structure, allowing users to program custom logic functions. The AmPAL22V10B is an extension of the PAL concept. First-generation devices were largely limited to TTL replacement applications. The AmPAL22V10B permits the development of custom LSI functions of 500 to 800 equivalent gate complexity.

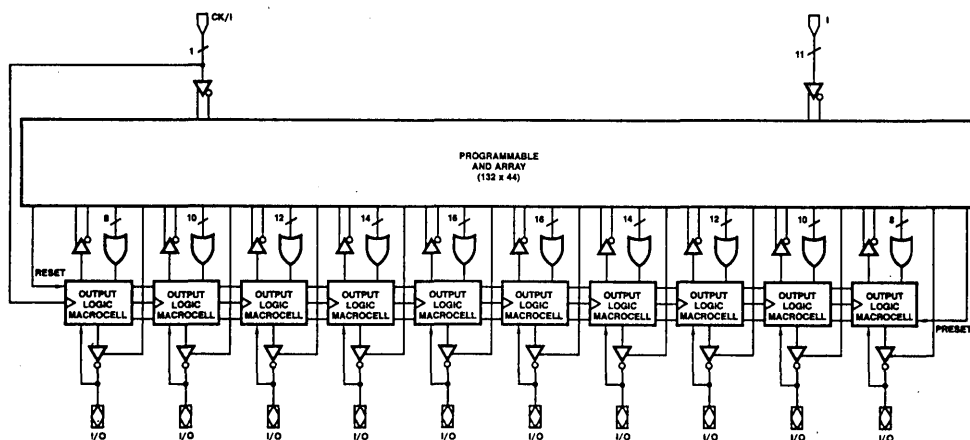
The AmPAL22V10B contains up to 22 inputs and 10 outputs. It incorporates the unique capability of defining and programming the architecture of each output on an individual basis. Each output is user-programmable for either registered or combinatorial operation. This allows the designer to optimize the device design, by having only as many registers as needed. In addition, each output has user-programmable output polarity, further simplifying design and contributing to the precise application requirements.

Increased logic power has been built into the AmPAL22V10B by increasing the number of product terms from 8-per-output to an average of 12-per-output. Further innovation can be seen in the introduction of variable product-term distribution. This technique allocates from 8 to 16 logical product terms to each output (please refer to block diagram for distribution details). This variable allocation of terms allows far more complex functions to be implemented than in previous devices.

System operation has been enhanced by the addition of a synchronous-PRESET and an asynchronous-RESET product term. These terms are common to all output registers.

The AmPAL22V10B also incorporates power-up RESET and the unique capability to PRELOAD the output registers to any desired state during testing. PRELOAD is essential to permit full logical verification during test.

BLOCK DIAGRAM



OPERATING RANGES

Temperature (T_A) Operating Free Air 0°C to $+75^{\circ}\text{C}$
 Supply Voltage (V_{CC}) $+4.75$ to $+5.25$ V

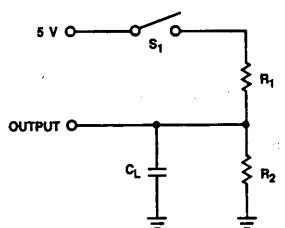
Commercial (C) Devices

SWITCHING CHARACTERISTICS over operating range unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions	Typ. (Note 1)	C Devices				E/M Devices				Units
				"B"		TBD		"B"		TBD		
				Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t _{PD}	Input or Feedback to Non-Registered Output	C Devices R ₁ = 390 R ₂ = 390			15				20			ns
t _{EA}	Input to Output Enable				15				20			ns
t _{ER}	Input to Output Disable				15				20			ns
t _{CO}	Clock to Output				10				15			ns
t _S	Input or Feedback Setup Time			12					17			ns
t _H	Hold Time	E/M Devices R ₁ = 390 R ₂ = 750		0				0				ns
t _P	Clock Period (t _S + t _{CO})			22					32			ns
t _W	Clock Width			11					16			ns
f _{MAX}	Maximum Frequency				45				31			MHz
t _{AW}	Asynchronous Reset Width			15					20			ns
t _{AR}	Asynchronous Reset Recovery Time			15					20			ns
t _{AP}	Asynchronous Reset to Registered Output Reset				20				25			ns

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^{\circ}\text{C}$.2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high-impedance to HIGH tests and closed for high-impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high-impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high-impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

SWITCHING TEST CIRCUIT



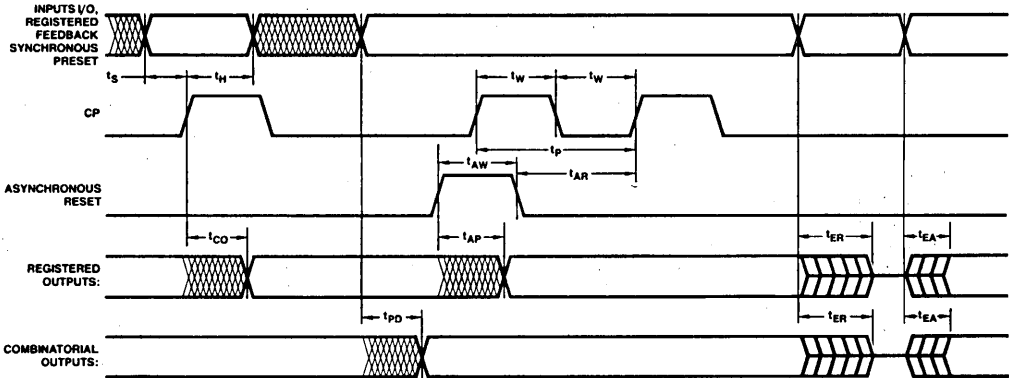
TC001190

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS



WF022280

AmPAL*23S8

20-Pin IMOX™ PAL-Based Sequencer

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- 14 Registers
 - 4 Output Logic Macrocells (OLMs)
 - 4 Output Registers
 - 6 Buried State Registers (BSRs)
- 23 possible array inputs and 8 outputs in a 20-pin package
- 33-MHz external/40-MHz internal cycle time
- Variable product term (PT) distribution for increased design flexibility
- Asynchronous and synchronous outputs supported for both Mealy- and Moore-type state-machine implementations
- Individually user-programmable Output Enable (OE) PTs with polarity control
- PTs for observing the BSRs on 6 of the output pins
- Separate PTs for common Synchronous PRESET and common Asynchronous RESET of all registers
- PRELOAD available on all registers for added test capability
- 99.9% post programming functional yield (PPFY) is due to the complete testability of this and all other AMD PAL devices
- Platinum-Silicide fuse technology produces the most reliable bipolar programmable devices available today

GENERAL DESCRIPTION

The AmPAL23S8 is the first programmable array logic (PAL)-based sequencer device. It utilizes the familiar sum-of-products (AND-OR) logic structure, allowing users to customize logic functions by programming the device for specific applications. The AmPAL23S8 combines the ease of use of the familiar 20-pin PAL devices with the advanced "macrocell" concept introduced in the AmPAL22V10, as well as six Buried State Registers (BSRs).

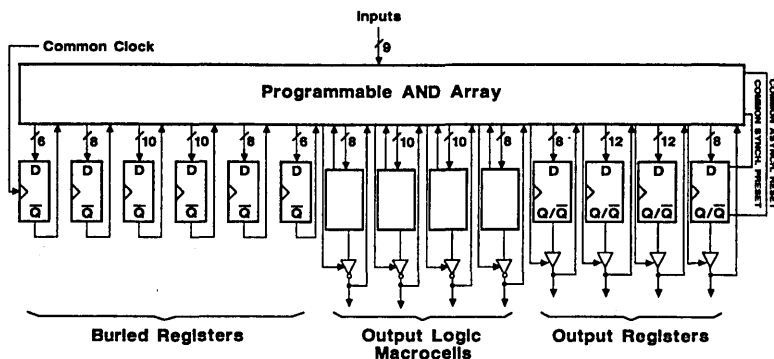
The AmPAL23S8 provides up to twenty-three array inputs and eight outputs. Four of the outputs are Output Logic Macrocells (OLMs) capable of being individually programmed as "combinatorial" or "registered," with active-HIGH or active-LOW polarity on each output. The other four are "registered" outputs, also capable of being programmed for active-HIGH or LOW polarity. All the flexibility on the outputs result in the simplification of logic design. The need to perform "DeMorgan's Law" on equations to have them fit into a PAL device is now a thing of the past. Each of the eight output registers can also be used dynamically as an input or output for greater design flexibility.

The AmPAL23S8 also offers designers increased flexibility and control over Output Enable (OE) functions. Each output is logically controlled by an OE product term (PT), with programmable OE polarity control. This allows the designer to use more complex control than previously available.

The six BSRs provide designers with enhanced logic power for sequencer applications. These registers are not only available to the system designer for use in sequencer applications (without the expense of a valuable I/O pin), but they may also be observed on the output pins during test. The observability of these registers on a programmable logic sequencer adds to the list of features which make this device unique, simple to design with, and simple to debug.

System operation has been enhanced by the addition of Synchronous PRESET and Asynchronous RESET PTs. The AmPAL23S8 also incorporates the unique capability of PRELOADing the eight output registers and the BSRs to any desired state during testing. This is essential to permit full logical verification during test.

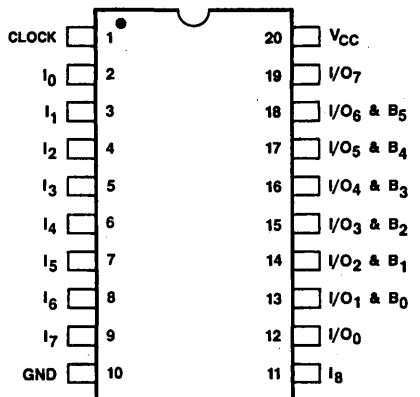
BLOCK DIAGRAM



BD006800

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

CONNECTION DIAGRAM Top View



CD009870

Note: Pin 1 is marked for orientation.

Pin Description

VCC = Supply Voltage

GND = Ground

CLOCK = Clock Pin

I₀ - I₈ = Dedicated Input Pins (9)

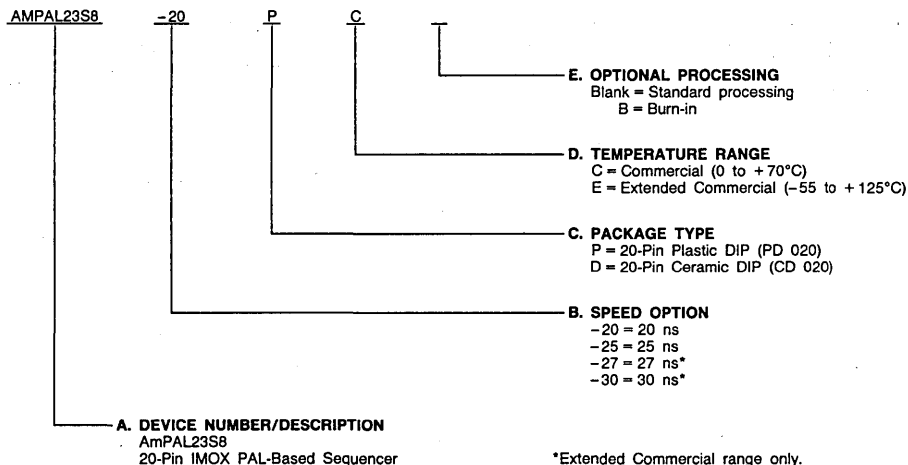
I/O₀ - I/O₇ = Bidirectional I/O Pins (8)

B₀ - B₅ = Observability Pins for BSRs (6)

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



*Extended Commercial range only.

Valid Combinations

Valid Combinations	
AMPAL23S8-20	PC, DC, DCB
AMPAL23S8-25	PC, DC, DCB, DE, DEB
AMPAL23S8-27	DE, DEB
AMPAL23S8-30	

Valid Combinations

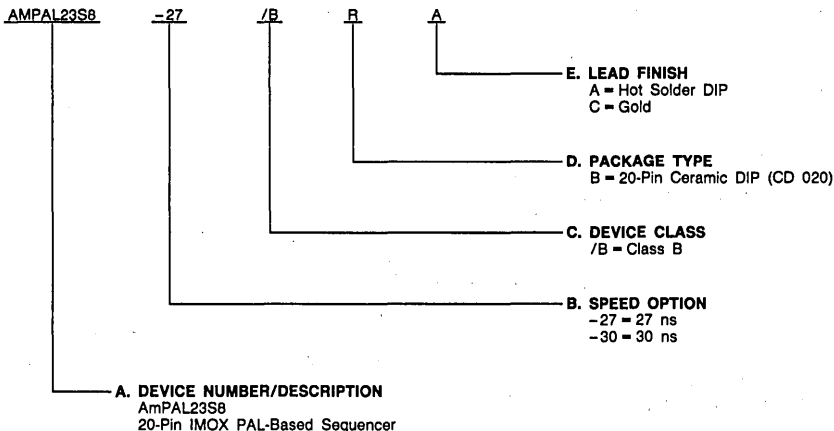
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number**
B. Speed Option (if applicable)
C. Device Class
D. Package Type
E. Lead Finish



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.

Valid Combinations	
AMPAL23S8-27	/BRA
AMPAL23S8-30	

Group A Tests

Group A Tests consist of Subgroups: 1, 2, 3, 7, 8, 9, 10, 11

FUNCTIONAL DESCRIPTION

The AmPAL23S8 is an advanced bipolar programmable array logic (PAL)-based sequencer. It contains a programmable array organized in the familiar sum-of-products structure. The structure of this device makes it particularly ideal for state machine applications. Any design which employs the use of complex state functions is a prime candidate for the AmPAL23S8.

The block diagram on the front cover shows the basic architecture of this device; a maximum of 23 array inputs and 8 outputs are available. The inputs are connected to a programmable AND array containing 135 product terms (PTs), of which 124 are logical PTs and 11 are control PTs. Before programming, the AND gates are connected to both the true and complement of every input. By selectively programming fuses, the AND gates may be connected to only the true input, the complement input, or to neither type of input, establishing a logical "don't care". When both the true and complement fuses are left intact, a logical FALSE results on the output of the AND gate. An AND gate with all fuses blown will assume the logical TRUE state. The outputs of the AND gates are connected to OR gates.

Variable Product Term (PT) Distribution

The number of AND gates assigned to each OR gate varies in a fixed manner for each output as shown in the logic diagram (Figure 5). The OR-gate outputs feed dedicated registers and macrocells. Each OR gate averages approximately ten PTs for output registers and macrocells. This gives the capability of using from eight to twelve logical PTs on one output in a single clock cycle (no feedback necessary). Buried state registers (BSRs) have an average of eight PTs per OR gate, providing the capability of using from six to ten logical PTs in one BSR in a single clock cycle.

Variable Output Architecture: Output Logic Macrocells (OLMs)

An innovation in logic design is the implementation on the AmPAL23S8 of variable output architecture on four of the outputs. These Output Logic Macrocells (OLMs) are user programmable for a great deal of design flexibility. Each of the four OLMs can be independently programmed for eight distinct configurations. The outputs can be either "registered" or "combinatorial;" they can also be individually programmed for active-HIGH or active-LOW polarity. Finally, the feedback paths which feed through the multiplexer back to the AND array can be programmed so that they originate either from the register or from the I/O pin. From the feedback multiplexer both the true and complement of the output going back to the array are available. All possible configurations of the OLMs are illustrated in Figures 4-1 through 4-8. For maximum flexibility, selection of output polarity and feedback path are kept independent of each other.

Output Registers

In addition to the four OLMs on the AmPAL23S8, there are also four output registers. The data on the output registers may be fed back to the array. When the output is disabled, the pin may be used as an external input. Since each of the eight outputs can obtain feedback from the pins associated with them, all eight of them provide the advantage of being usable dynamically as either inputs or outputs, significantly increasing design flexibility and possibilities.

Buried State Registers (BSRs)

One of the key features of the AmPAL23S8 is the six observable Buried State Registers (BSRs). All BSR outputs are fed directly back to the AND array for state machine implementation. In this manner, the advantage of having six

extra registers for state machine implementation is not paid for in output pins. The contents of each of the BSRs is observable on an associated pin through the use of the user-programmable observability product term.

The extensive user-programmable flexibility enhances the usefulness of this device for different types of state machine implementations. The possibility exists to create both the Mealy and Moore type of design in the same device.

Programmable Output Polarity

Each output has a user-programmable output polarity fuse which, when blown, indicates that the output will be active HIGH, and when intact, active LOW. The obvious benefit of this enhancement is the increased flexibility of design. With the choice of output polarity, there is no need to DeMorganize equations to fit the device, allowing for more efficient designs both in terms of the amount of time spent in design as well as effective utilization of the device.

For further enhancement of the increased logic power of the AmPAL23S8, each output has a PT to control Output Enable (OE) with programmable polarity.

PRESET/RESET

To improve functionality at the system level, the AmPAL23S8 has additional RESET and PRESET PTs. One PT controls Output Register and BSR PRESET, and one PT controls the RESET for these registers. When the Synchronous PRESET PT is asserted (HIGH), all registers are loaded with a HIGH on the next LOW-to-HIGH clock transition. When the Asynchronous RESET PT is asserted, all registers are immediately loaded with a LOW, independent of the clock. These functions are particularly useful for applications such as system power-up and RESET.

PRELOAD

In order to simplify testing, the AmPAL23S8 is designed with PRELOAD circuitry that provides an easy method for testing logical functionality. PRELOAD allows any arbitrary "present state" values to be loaded into the OLMs, BSRs and Output Registers of this device. OLM Registers and BSRs are PRELOADED in separate cycles, allowing them to be PRELOADED with different values. Logic verification sequences can be significantly shortened, and all possible state sequences tested, reducing test time and development costs, and guaranteeing proper functionality in system.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. To verify these transitions requires the ability to set the state registers to an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is then clocked into a new state, or "next state," which can be checked to validate the transition from the "present state." In this way, any state transition can be checked.

It is obvious that to attempt the debugging of a design using BSRs without the benefit of PRELOAD capability would be quite difficult. The combination of this feature and the BSRs being observable is virtually indispensable for efficient and trouble-free state machine design.

Observability

This extra ease of debugging the design comes from the use of the Observability (OBS) PT. When the OBS PT is selected, it disables six of the Output Registers and Macrocell buffers, and enables the BSR buffers onto the output pins associated with them, pins 13 through 18. When the OBS PT is not selected, the Output Registers and Macrocell buffers are enabled and the BSR buffers are disabled. When all the fuses

for this PT are intact, (i.e., OBS is not selected), the data from the BSRs will not be visible on the output pins, and the Output Registers and OLMs will be enabled.

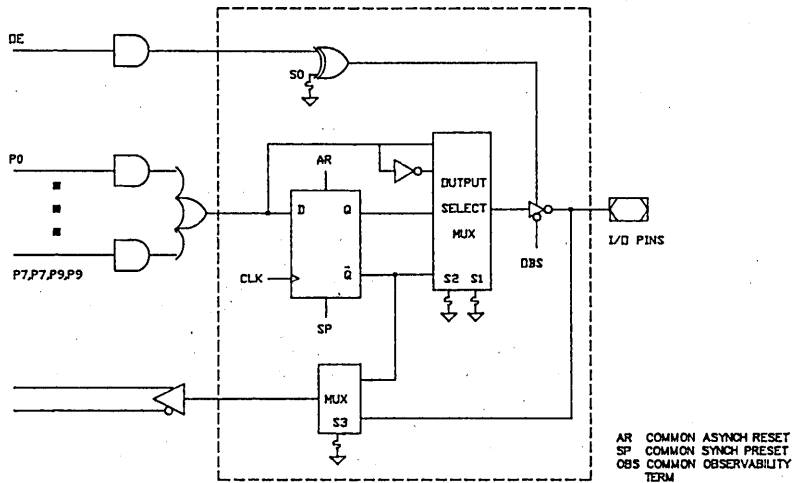
Processing and Fuse Technology

The AmPAL23S8 is manufactured using Advanced Micro Devices' IMOX oxide isolation process. This advanced process permits an increase in density and a decrease in internal capacitance resulting in the fastest possible programmable logic devices.

The AmPAL23S8 is fabricated with AMD's fast programming, highly reliable Platinum-Silicide fuse technology. Utilizing an

easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are preprogrammed during manufacturing to ensure extremely high field programming yields (> 98%), and provide extra test paths to achieve excellent parametric correlation.

Platinum-Silicide was selected as the fuse link material to achieve a well controlled melt rate resulting in large, nonconductive gaps that ensure very stable, long term reliability. Extensive operating testing has proven that this low-field, large-gap technology offers the best reliability for fusible link programmable logic.



LD000172

S0	OUTPUT ENABLE POLARITY
0	ENABLED HIGH
1	ENABLED LOW

S1	S2	S3	OUTPUT CONFIGURATION
0	0	0	ACTIVE LOW/REG/REG FEEDBACK
0	0	1	ACTIVE LOW/REG/IO FEEDBACK
0	1	0	ACTIVE LOW/COMB/REG FEEDBACK
0	1	1	ACTIVE LOW/COMB/IO FEEDBACK
1	0	0	ACTIVE HIGH/REG/REG FEEDBACK
1	0	1	ACTIVE HIGH/REG/IO FEEDBACK
1	1	0	ACTIVE HIGH/COMB/REG FEEDBACK
1	1	1	ACTIVE HIGH/COMB/IO FEEDBACK

0 = UNBLOWN FUSE

1 = BLOWN FUSE

Figure 1. Output Logic Macrocell (OLM)

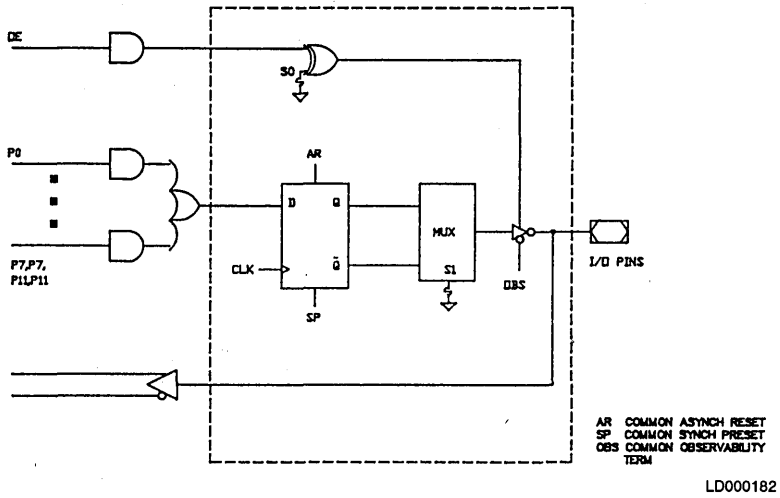


Figure 2. Output Register With Polarity

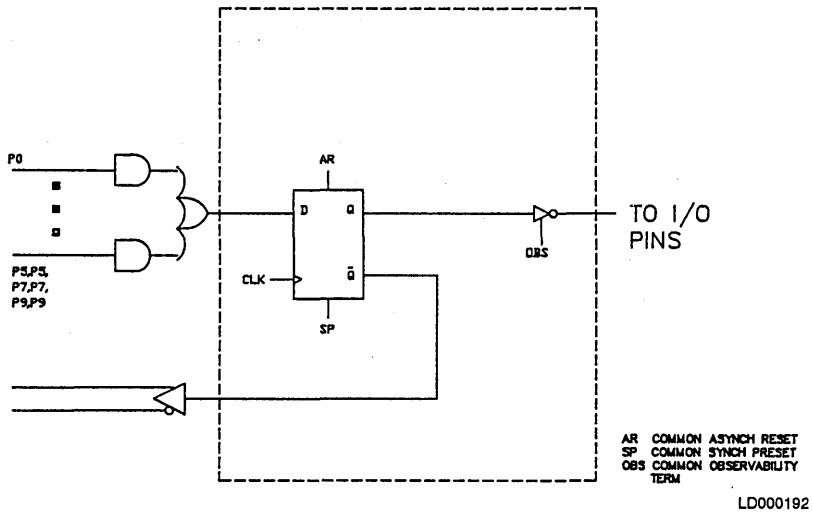
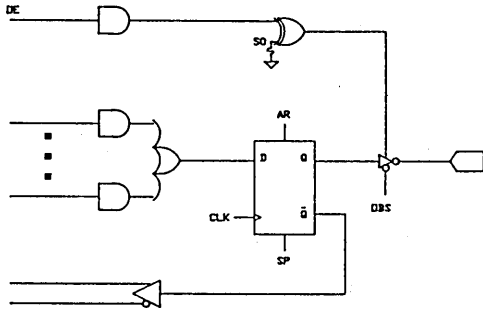


Figure 3. Buried State Register (BSR)

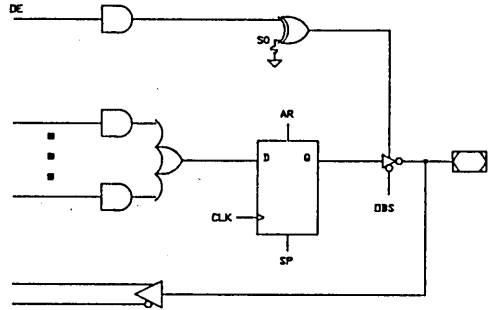
Figure 4. Possible Configurations of the Output Logic Macrocells (OLMs)



LD000200

OUTPUT
ACTIVE LOW S1 = 0
REGISTERED S2 = 0
FEEDBACK
REGISTERED S3 = 0

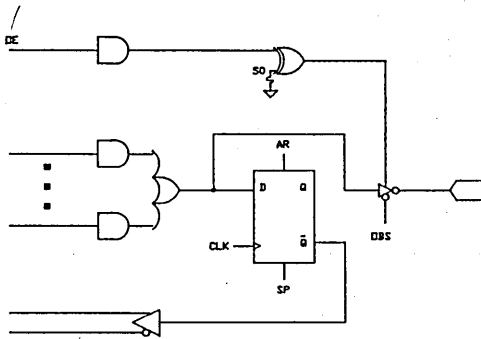
Figure 4-1.



LD000210

OUTPUT
ACTIVE LOW S1 = 0
REGISTERED S2 = 0
FEEDBACK
I/O PIN S3 = 1

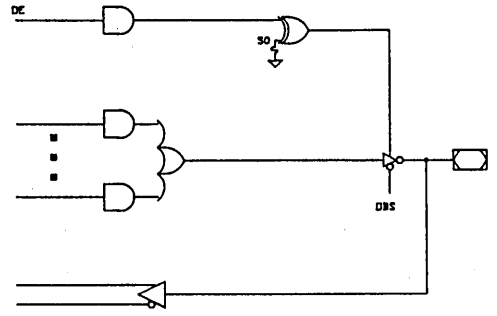
Figure 4-2.



LD000220

OUTPUT
ACTIVE LOW S1 = 0
COMBINATORIAL S2 = 1
FEEDBACK
REGISTERED S3 = 0

Figure 4-3.

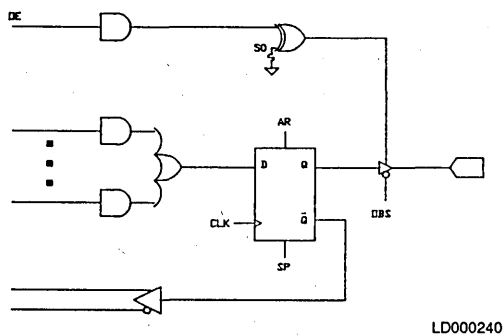


LD000230

OUTPUT
ACTIVE LOW S1 = 0
COMBINATORIAL S2 = 1
FEEDBACK
I/O PIN S3 = 1

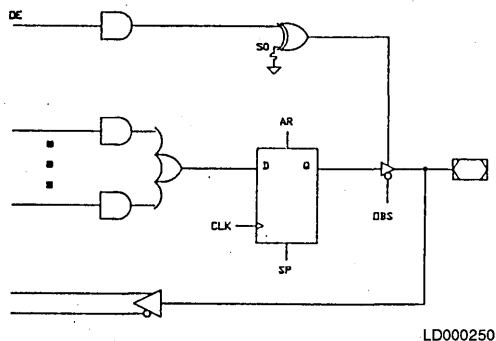
Figure 4-4.

Figure 4. Possible Configurations of the Output Logic Macrocells (OLMs) (Cont'd.)



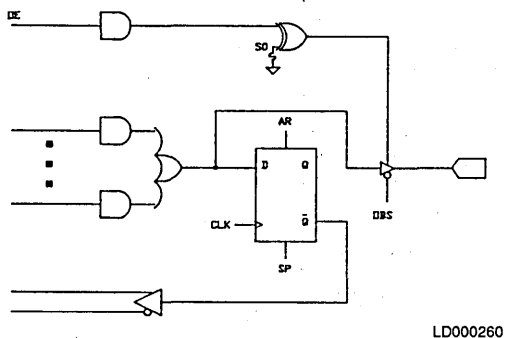
OUTPUT
ACTIVE HIGH S1 = 1
REGISTERED S2 = 0
FEEDBACK REGISTERED S3 = 0

Figure 4-5.



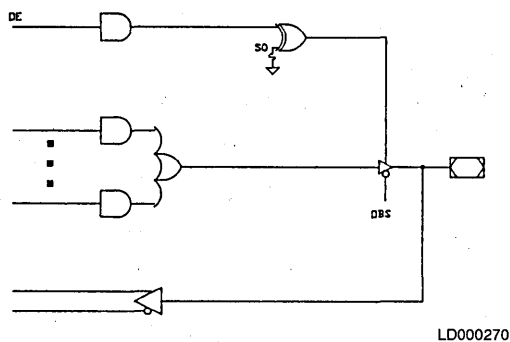
OUTPUT
ACTIVE HIGH S1 = 1
REGISTERED S2 = 0
FEEDBACK I/O PIN S3 = 1

Figure 4-6.



OUTPUT
ACTIVE HIGH S1 = 1
COMBINATORIAL S2 = 1
FEEDBACK REGISTERED S3 = 0

Figure 4-7.



OUTPUT
ACTIVE HIGH S1 = 1
COMBINATORIAL S2 = 1
FEEDBACK I/O PIN S3 = 1

Figure 4-8.

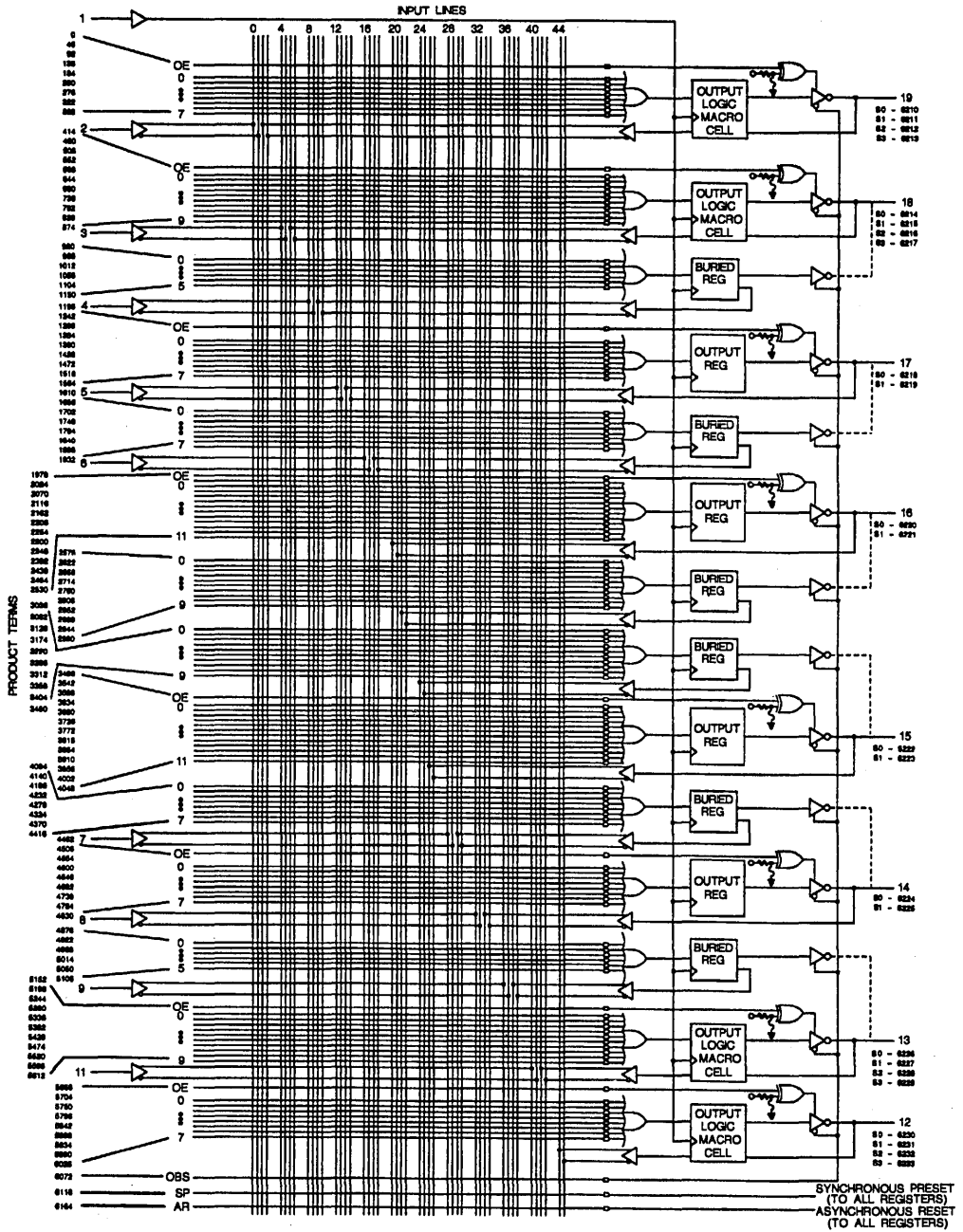
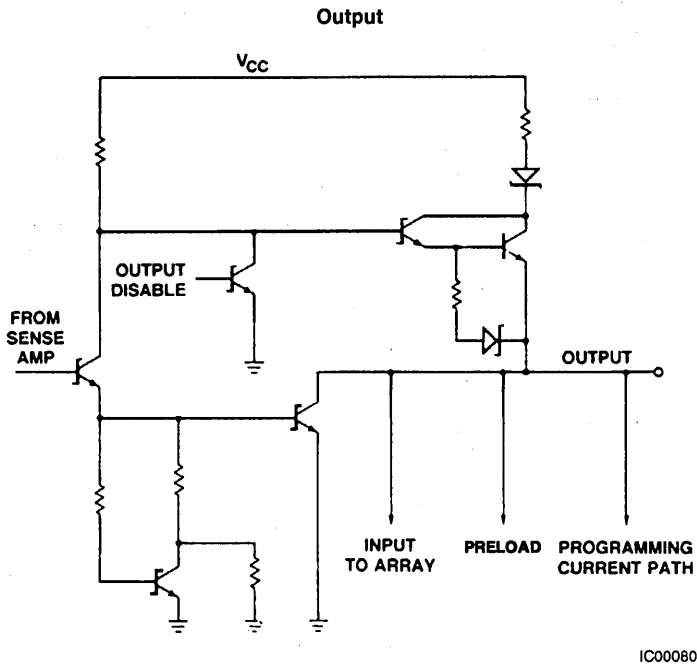
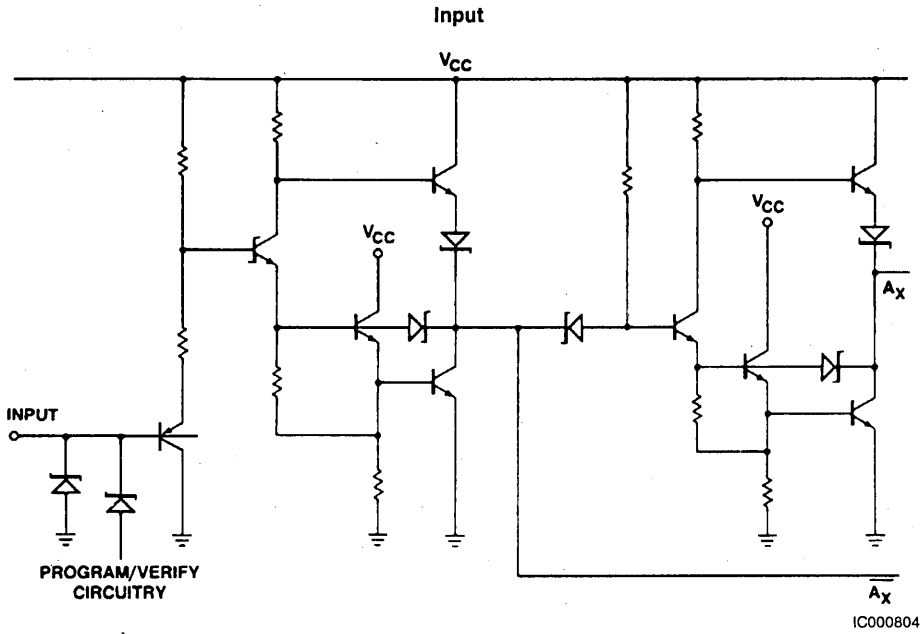


Figure 4. Logic Diagram — JEDEC Fuse Numbering for AmPAL23S8

INPUT/OUTPUT DIAGRAMS



ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Ambient Temperature with Power Applied	+125°C
Supply Voltage to Ground Potential Continuous (Pin 20 to Pin 10)	-0.5 to +7.0 V
DC Voltage Applied to Outputs (except during programming)	-0.5 to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	16 V
Output Current into Outputs During Programming (Maximum duration of 1 second)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5.0 mA

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES**Commercial (C) Devices**

Temperature (T _A) Operating Free Air	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V

Extended Commercial (E) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C) Operating Case	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Military (M) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions		Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -3.2 mA C Devices	2.4	2.5	0.50	Volts
			I _{OH} = -2 mA E/M Devices				
V _{OL}	Output LOW Voltage		I _{OL} = 16 mA C Devices				
			I _{OL} = 12 mA E/M Devices				
V _{IH} (Note 2)	Input HIGH level	Guaranteed Input Logical HIGH Voltage for All Inputs		2.0			Volts
V _{IL} (Note 2)	Input LOW level	Guaranteed Input Logical LOW Voltage for All Inputs				0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.40 V			-10	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V				25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V				1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)		-30	-45	-90	mA
I _{CC}	Power/Supply Current	All Inputs = GND, V _{CC} = Max.				200	mA
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA			-0.9	-1.2	Volts
I _{OZH}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V			100	μA
I _{OZL}			V _O = 0.4 V			-100	

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5 V has been chosen to avoid test problems caused by tester ground degradation.

4. I/O pin leakage is the worst case of I_{OZX} or I_{IX} (where X = H or L).

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions		Typ.	Units
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz	Pins 1, 11	10	pF
			Other Pins	6	
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz		9	

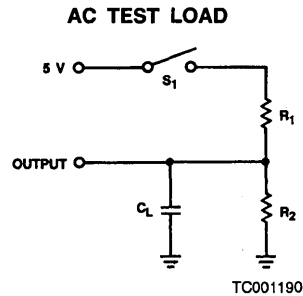
Note: These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

KEY TO SWITCHING WAVEFORMS

SWITCHING TEST CIRCUIT

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010



TEST OUTPUT LOADS			
C Devices		E/M Devices	
R ₁	R ₂	R ₁	R ₂
300	390	390	750

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 7, 8, 9, 10, 11 tests unless otherwise noted

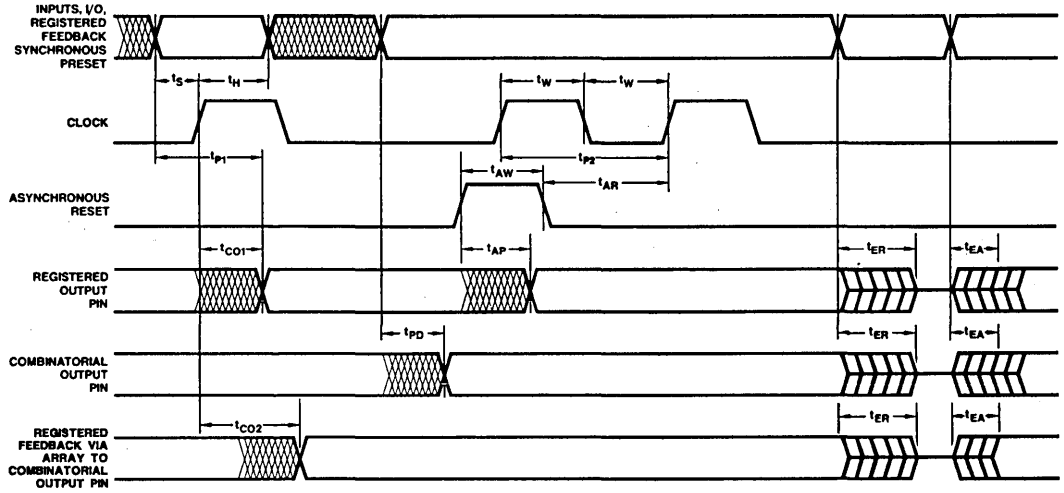
No.	Parameter Symbol	Parameter Description	Test Conditions	Typ (Note 1)	C Devices				E/M Devices				Units
					-20		-25		-27		-30		
					Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
1	t _{PD}	Input or Feedback to Non-Registered Output	C Devices R ₁ = 300 R ₂ = 390			20		25		27		30	ns
2	t _{EA}	Input to Output Enable				25		28		30		33	ns
3	t _{ER}	Input to Output Disable				25		28		30		33	ns
4	t _{CO1}	Clock to Output				13		15		18		20	ns
5	t _{CO2}	Reg. Feedback through Array to Combinatorial Output, Relative to External Clock				28		35		40		45	ns
6	t _S	Input or Feedback Setup Time				17		20		22		25	ns
7	t _H	Hold Time				0		0		0		0	ns
8	t _{P1}	Clock Period (t _S + t _{CO1})				30		35		40		45	ns
9	t _{P2}	Minimum Setup Time (Reg. Feedback to Reg. Input Internal Path)				25		30		35		40	ns
10	t _W	Clock Width				12		15		17		20	ns
11	f _{1MAX}	Maximum Frequency (1/t _W)	E/M Devices R ₁ = 390 R ₂ = 750			33		28.5		25		22.5	MHz
12	f _{2MAX}	Maximum Frequency (1/t _{P2})				40		33		28.5		25	MHz
13	t _{AW}	Asynchronous Reset Width				20		25		27		30	ns
14	t _{AR}	Asynchronous Reset Recovery Time				20		25		27		30	ns
15	t _{AP}	Asynchronous Reset to Registered Output Reset				25		30		32		35	ns

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. t_{PD} is tested with switch S₁ closed and C_L = 50 pF.

3. For three-state outputs, output enable times are tested with C_L = 50 pF to the 1.5 V level; S₁ is open for high-impedance to HIGH tests and closed for high-impedance to LOW tests. Output disable times are tested with C_L = 5 pF. HIGH to high-impedance tests are made to an output voltage of V_{OH} - 0.5 V with S₁ open; LOW to high-impedance tests are made to the V_{OL} + 0.5 V level with S₁ closed.

SWITCHING WAVEFORM

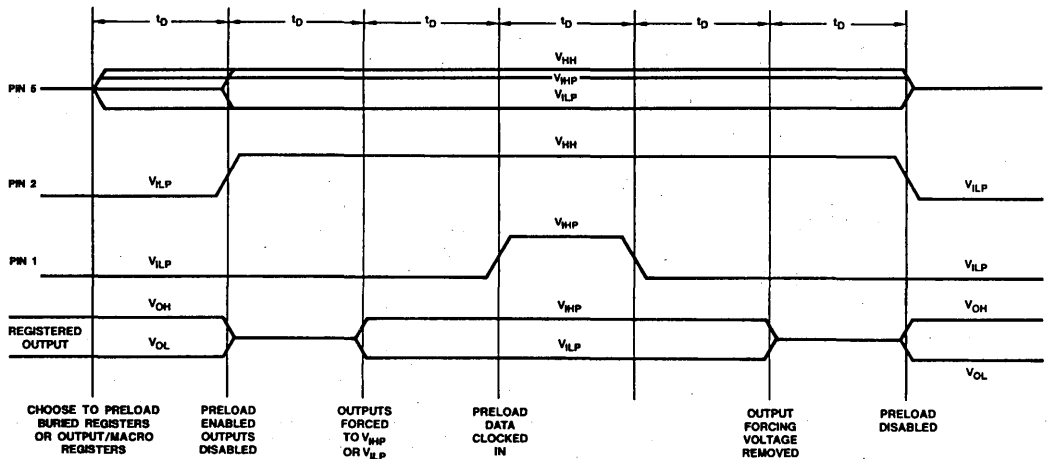


WF022285

PRELOAD OF OUTPUT/MACROCELL REGISTERS OR BURIED STATE REGISTERS

All AmPAL23S8 registers are provided with circuitry to allow loading each register synchronously with a HIGH or LOW. Output/macrocell registers and buried state registers are

PRELOADED in separate cycles allowing output/macrocell registers and buried state registers to be PRELOADED with different values. PRELOAD will simplify testing since any state can be loaded into the registers to control testing sequences. The pin levels and timing necessary to perform the PRELOAD function are detailed below. Parameters are listed in the Programming Parameters table.



WF022292

Register Selection	Pin 5
Buried	V_{HH}
Output/Macro	$\leq V_{IHP}$

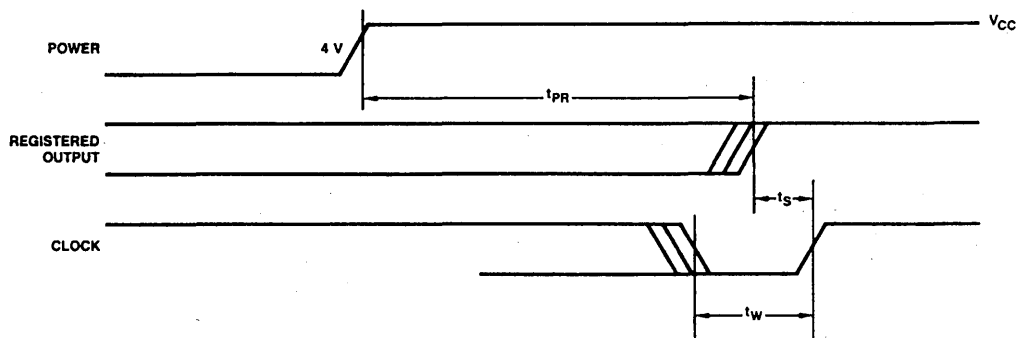
Level Forced on Register Output Pin During PRELOAD Cycle	Register State After Cycle
V_{IHP}	HIGH
V_{ILP}	LOW

POWER-UP RESET

The registered devices in the AMD PAL Family have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to LOW. The output state will depend on the polarity of the output buffer. This feature provides an extra flexibility to the designer and is especially valuable in simplifying state-machine initialization. A timing diagram and parameter table follow. Due to the

asynchronous operation of the power-up reset and the wide range of ways V_{CC} can rise to its steady state, two conditions are required to ensure a valid power-up reset. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.



WF022300

Parameter Symbol	Parameter Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up Reset Time		600	1000	ns
t_S	Input or Feedback Setup Time	See Switching Characteristics			
t_W	Clock Width				

Programming and Verification

The AmPAL23S8 is programmed and verified using AMD's standard programmable logic programming algorithm. The fuse to be programmed is selected by input line number (array row), product term (array column), and by one output at a time. The fuse is then programmed and verified by applying a simple sequence of voltages to two control pins (1 and 11).

Input line numbers (0 through 46) are addressed using a full decode scheme via TTL levels on pins 6 through 9, where 6 is the least significant bit (LSB) and 9 is the most significant bit (MSB). Even numbered input lines represent the "true" version of a signal, and odd numbered lines represent the "complement." Input line addressing is shown in Table 1. Note that input lines 47 through 53 are reserved for further expansion. Also, input line 46 is employed in the selection of the fuses used for programming output polarity, feedback selection, and "registered" versus "combinatorial" output selection.

Product terms (PTs) are addressed with the use of a 1-of-26 addressing scheme on pins 2 through 5, where pin 2 is the LSB and pin 5 is the MSB. PT addressing is shown in Table 2. Note that the outputs with unassigned fuse numbers in Table 2 will decode blank space for those decoding values. Logical PTs and architectural PTs are selected by the same addressing scheme. The specific decoding of architectural features is best shown in Table 2.

Fuse selection by output must be done one output at a time (following control pin 1 going to V_{HH}) as shown in the Programming Waveform diagram.

Once fuses have been selected, the simple programming and verification sequence may be completed as shown in the Programming Waveform diagram. AC and DC requirements for programming are shown in the Programming Parameters table. A detailed description of the programming algorithm is provided.

Programming Yield

AMD PALs have been designed to ensure extremely high programming yields (> 98.5%). To help ensure that a part was correctly programmed once programming is completed, the entire fuse array should be re-verified at both low and high V_{CC} . Re-verification can be accomplished by reading all eight outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PALs contain many internal test features—including circuitry which allows AMD to test the ability of each part to perform programming before shipping, to assure high programming yields and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper use of equipment.

PROGRAMMING PARAMETERS ($T_A = 25^\circ\text{C}$)

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V _{HH}	Control Pin Extra High Level	Pins 1 & 11 @ 10–60 mA	10	11	12	V
	Input Extra High Level During Programming & Verify	Pins 3, 4, 7, 8, & 9	10	11	12	
V _{OP}	Program Voltage Pins 12–19 @ 15–200 mA		14	15	16	V
V _{IHP}	Input High Level During Programming and Verify		2.4	5.0	5.5	V
V _{ILP}	Input Low Level During Programming and Verify		0.0	0.3	0.5	V
V _{CCP}	V _{CC} During Programming (V _{CC} = 100–200 mA)		5.0	5.2	5.5	V
V _{CCL}	V _{CC} During First Pass Verification		4.4	4.5	4.6	V
V _{CCH}	V _{CC} During Second Pass Verification		5.4	5.5	5.6	V
V _{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	V
dV _{OP} /dt	Rate of Output Voltage Change		20		250	V/ μs
dV _I /dt	Rate of Control Voltage Change	Pins 1 & 11	20		1000	V/ μs
t _p	First Time Fusing Attempt		10	50	100	μs
	Subsequent Attempts (maximum of 8)		4	5	10	ms
t _d	Delays Between Various Level Changes		100	1000	15,000	ns
t _v	Period During which Output is Sensed for V _{Blown} Level		100	1000	15,000	ns
V _{ONP}	Pull-Up Voltage On Outputs Not Being Programmed (During Programming)		5.0	V _{CCP}	5.5	V
I _{ONP}	Current Limit Into Outputs Not Being Programmed (During Programming)		1.0	2.0	3.0	mA

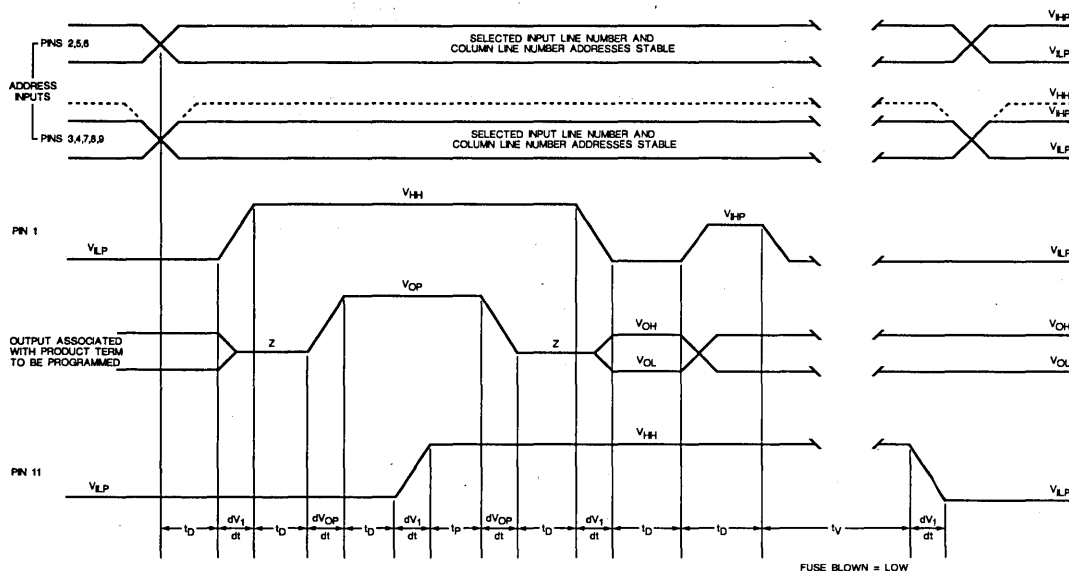
TTL PAL Programming Procedure

Fusing Technique

AMD's PAL devices have been designed to use a programming algorithm which minimizes the requirements on the programmer while allowing the circuit to blow the Platinum-Silicide fuse links quickly and reliably. Specifically, the following sequence of events must take place:

1. V_{CCP} power is applied to the device;
2. The appropriate input line number and column line number addresses are selected;
3. Outputs and normal inputs are disabled; programming decode is enabled;
4. The programming voltage is applied to one output;
5. The fuse enable voltage is raised to enable a high-threshold voltage gate. This action gates the current flow through the proper fuse, resulting in an open fuse in a few microseconds;
6. The output voltage is lowered (the programming voltage is removed);
7. The outputs are enabled;
8. The registers are clocked to enable the sensed bit to the output;
9. Verify that the fuse has been blown (fuse blown = LOW). In the unlikely event that the fuse does not verify as blown, a maximum of eight subsequent attempts with much longer pulses can be applied to the fuse at a higher duty cycle;
10. At the conclusion of programming, the device should be verified for correct data at all addresses, with two V_{CC} supply voltages, $V_{CCH} = 5.5$ V, and $V_{CCL} = 4.5$ V.

PROGRAMMING WAVEFORMS



WF022310

AmPAL23S8 Programming Support Information

Hardware Vendor	Programmer Model(s)	Personality Module	Socket Adaptor
Data I/O 10525 Willow Road N.E. Redmond, WA 98052	System 19, 29, or 100	Under Development	Under Development
	Model 60A or 60H	Under Development	Under Development
Stag Microsystems 528-5 Weddel Drive Sunnyvale, CA 94086	Model PPZ	Under Development	Under Development
	Model ZL30	Under Development	Under Development

The machines noted above have been qualified by AMD to ensure high programming yields. Check with the factory to determine current status of equipment noted as "Under Development," or for other available models.

Design Aid Software for AmPAL23S8

Software Vendor	Software Package	Comments
P-CAD Systems (408) 971-1300	CUPL	
Advanced Micro Devices (408) 732-2400	AmCUPL	Developed and supported by P-CAD Systems
Data I/O (206) 881-6444	ABEL	

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address State Pins				Input Line Number	Input Line Number Address State Pins			
	9	8	7	6		9	8	7	6
0	L	L	L	L	27	L	HH	H	H
1	L	L	L	H	28	H	HH	L	L
2	L	L	H	L	29	H	HH	L	H
3	L	L	H	H	30	H	HH	H	L
4	L	H	L	L	31	H	HH	H	H
5	L	H	L	H	32	L	L	HH	L
6	L	H	H	L	33	L	L	HH	H
7	L	H	H	H	34	L	H	HH	L
8	H	L	L	L	35	L	H	HH	H
9	H	L	L	H	36	H	L	HH	L
10	H	L	H	L	37	H	L	HH	H
11	H	L	H	H	38	H	H	HH	L
12	H	H	L	L	39	H	H	HH	H
13	H	H	L	H	40	HH	HH	L	L
14	H	H	H	L	41	HH	HH	L	H
15	H	H	H	H	42	HH	HH	H	L
16	HH	L	L	L	43	HH	HH	H	H
17	HH	L	L	H	44	HH	L	HH	L
18	HH	L	H	L	45	HH	L	HH	H
19	HH	L	H	H	46*	HH	H	HH	L
20	HH	H	L	L	47	•			
21	HH	H	L	H	48	•			
22	HH	H	H	L	49	•			
23	HH	H	H	H	50	RESERVED			
24	L	HH	L	L	51	•			
25	L	HH	L	H	52	•			
26	L	HH	H	L	53	•			

Legend: L = V_{LP} H = V_{HP} HH = V_{HH}

*Architecture Row

TABLE 2. COLUMN NUMBER ADDRESSING

Column Number								Column Number Select Address Pin States				Description
OLM	OLM	REG	REG	REG	REG	OLM	OLM					
0	0	0	0	0	0	0	0	H	L	L	L	Logical PTs
1	1	1	1	1	1	1	1	H	L	L	H	Logical PTs
2	2	2	2	2	2	2	2	H	L	H	L	Logical PTs
3	3	3	3	3	3	3	3	H	L	H	H	Logical PTs
4	4	4	4	4	4	4	4	H	H	L	L	Logical PTs
5	5	5	5	5	5	5	5	H	H	L	H	Logical PTs
6	6	6	6	6	6	6	6	H	H	H	L	Logical PTs
7	7	7	7	7	7	7	7	H	H	H	H	Logical PTs
	8		8	8		8		H	HH	L	L	Logical PTs
OBS	9		9	9		9		H	HH	L	H	Observability
F	F		10	10		F	F	H	HH	H	L	Feedback Select
R	R		11	11		R	R	H	HH	H	H	Register/Combinatorial
OP	OP	OP	OP	OP	OP	OP	OP	H	L	HH	L	Output Polarity
OE	OE	OE	OE	OE	OE	OE	OE	H	L	HH	H	Output Enable (OE)
OEP	OEP	OEP	OEP	OEP	OEP	OEP	OEP	H	H	HH	L	OE Polarity
	SP					AR		H	H	HH	H	SP = Sync. PRESET AR = Async. RESET
Pin 12	Pin 13	Pin 14	Pin 15	Pin 16	Pin 17	Pin 18	Pin 19	Pin 5	Pin 4	Pin 3	Pin 2	
Buried State Registers												
	0	0	0	0	0	0		L	L	L	L	Logical PTs
	1	1	1	1	1	1		L	L	L	H	Logical PTs
	2	2	2	2	2	2		L	L	H	L	Logical PTs
	3	3	3	3	3	3		L	L	H	H	Logical PTs
	4	4	4	4	4	4		L	H	L	L	Logical PTs
	5	5	5	5	5	5		L	H	L	H	Logical PTs
		6	6	6	6			L	H	H	L	Logical PTs
		7	7	7	7			L	H	H	H	Logical PTs
			8	8				L	HH	L	L	Logical PTs
			9	9				L	HH	L	H	Logical PTs
Pin 12	Pin 13	Pin 14	Pin 15	Pin 16	Pin 17	Pin 18	Pin 19	Pin 5	Pin 4	Pin 3	Pin 2	
Programming Access & Verify Pins												

Legend: L = V_{ILP}
H = V_{IHP}
HH = V_{HH}

AMD Standard 24-Pin PAL* Family

24-Pin IMOX™ Programmable Array Logic (PAL) Elements

PRELIMINARY

AMD Standard 24-Pin PAL* Family

DISTINCTIVE CHARACTERISTICS

- AMD's superior IMOX technology
– Guarantees $t_{PD} = 15$ ns max
- Programming yields > 98% are realized via platinum-silicide fuse technology and the use of added test words
- Post Programming Functional Yield (PPFY) of 99.9%
- PRELOAD feature permits full logical verification
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parametric testing at the factory through on-board testing circuitry
- > 3000V ESD input protection per pin
- JEDEC-Standard LCC and PLCC pinout

GENERAL DESCRIPTION

AMD Standard 24-pin PAL devices are high-speed, electrically programmable array logic elements. They utilize the familiar sum-of-products (AND-OR) structure allowing users to program custom logic functions to fit most applications precisely. Typically they are a replacement for low-power Schottky SSI/MSI logic circuits, reducing chip count by more than 5 to 1 and greatly simplifying prototyping and board layout.

Five different devices are available, including both registered and combinatorial devices. All devices have user-programmable output polarity on all outputs. A variety of speed options allow the designer maximum flexibility in matching precise system requirements. The Product Selector Guide below shows the available speed options. The second table gives details about the functionality of the five available devices.

Please see the following pages for Block Diagrams.

PRODUCT SELECTOR GUIDE

AMD PAL Speed/Power Families

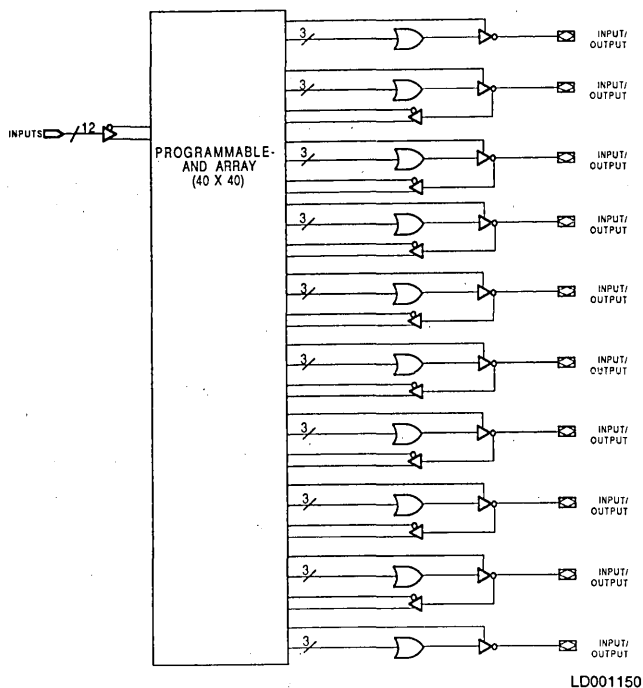
Family	t_{PD} ns (Max.)		t_s ns (Min.)		t_{CO} ns (Max.)		I_{CC} mA (Max.)	I_{OL} mA (Min.)	
	C Devices	M Devices	C Devices	M Devices	C Devices	M Devices	C/M Devices	C Devices	M Devices
Very High-Speed ("B") Versions	15	20	15	20	12	13	210	24	12
High-Speed ("A") Versions	25	30	25	30	15	20	210*	24	12
High-Speed, Half-Power ("AL") Versions	25	30	25	30	15	20	105	24	12
-20 & -25 Versions (AmPAL20L10 only)	20	25	20	25	13	15	165	24	12

*Except AmPAL20L10 ($I_{CC} = 165$ mA)

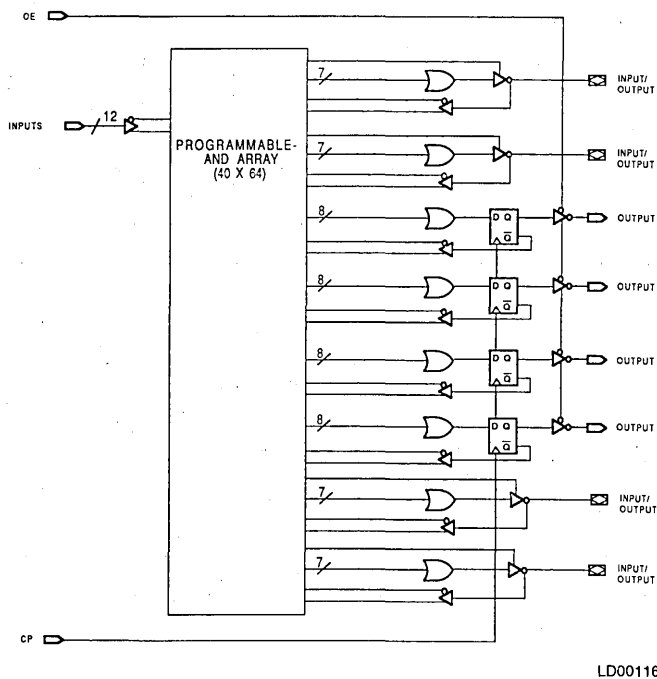
Part Number	Array Inputs	Logic	Output Enable	Outputs	Package Pins
20L10	12 Dedicated, 8 Bidirectional	Ten (3)-Wide AND-OR	Programmable	8 Bidirectional 2 Dedicated	24
20R4	12 Dedicated, 4 Feedback, 4 Bidirectional	Four (8)-Wide AND-OR	Dedicated	Registered	24
		Four (7)-Wide AND-OR	Programmable	Bidirectional	
20R6	12 Dedicated, 6 Feedback, 2 Bidirectional	Six (8)-Wide AND-OR	Dedicated	Registered	24
		Two (7)-Wide AND-OR	Programmable	Bidirectional	
20R8	12 Dedicated, 8 Feedback	Eight (8)-Wide AND-OR	Dedicated	Registered	24
20L8	14 Dedicated, 6 Feedback	Eight (7)-Wide AND-OR	Programmable	6 Bidirectional 2 Dedicated	24

BLOCK DIAGRAMS

AmPAL20L10

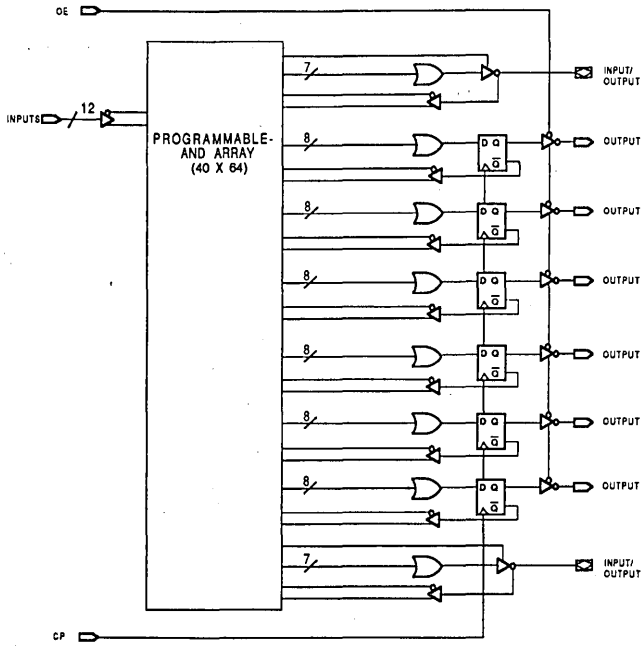


AmPAL20R4



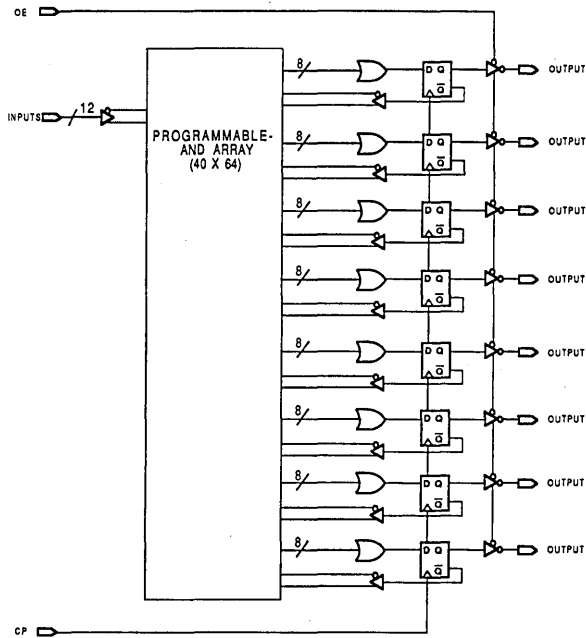
BLOCK DIAGRAMS (Cont'd.)

AmPAL20R6



LD001170

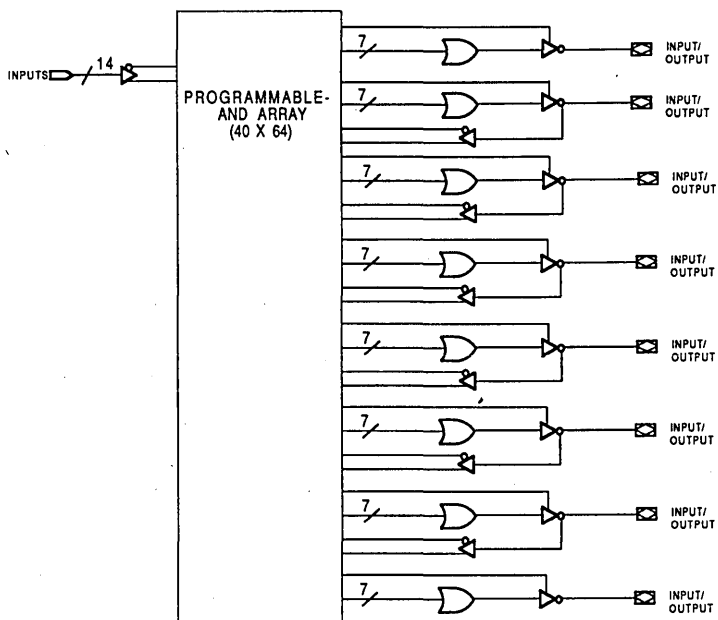
AmPAL20R8



LD001180

BLOCK DIAGRAMS (Cont'd.)

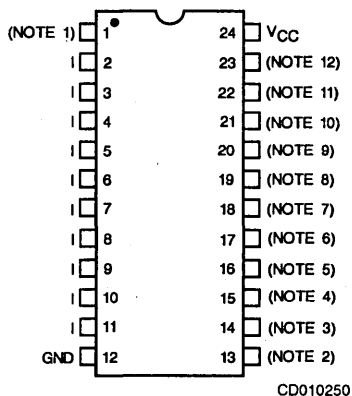
AmPAL20L8



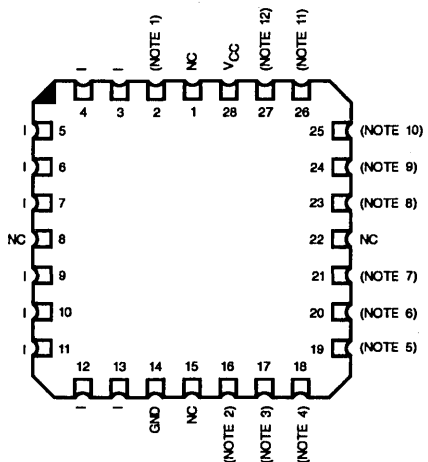
LD001190

CONNECTION DIAGRAMS Top View

DIPs*



LCC**



Note: Pin 1 is marked for orientation.

Notes:

	22L10	20R4	20R6	20R8	20L8
1	I	CLK	CLK	CLK	I
2	I	OE	OE	OE	I
3	O	I	I	I	I
4	I/O	I/O	I/O	O	O
5	I/O	I/O	O	O	I/O
6	I/O	O	O	O	I/O
7	I/O	O	O	O	I/O
8	I/O	O	O	O	I/O
9	I/O	O	O	O	I/O
10	I/O	I/O	O	O	I/O
11	I/O	I/O	I/O	O	O
12	O	I	I	I	I

*Also available in 24-Pin Ceramic Flatpack. Pinouts identical to DIPs.

**Also available in 28-Pin Plastic Leaded Chip Carrier. Pinouts identical to LCC.

PIN DESIGNATIONS

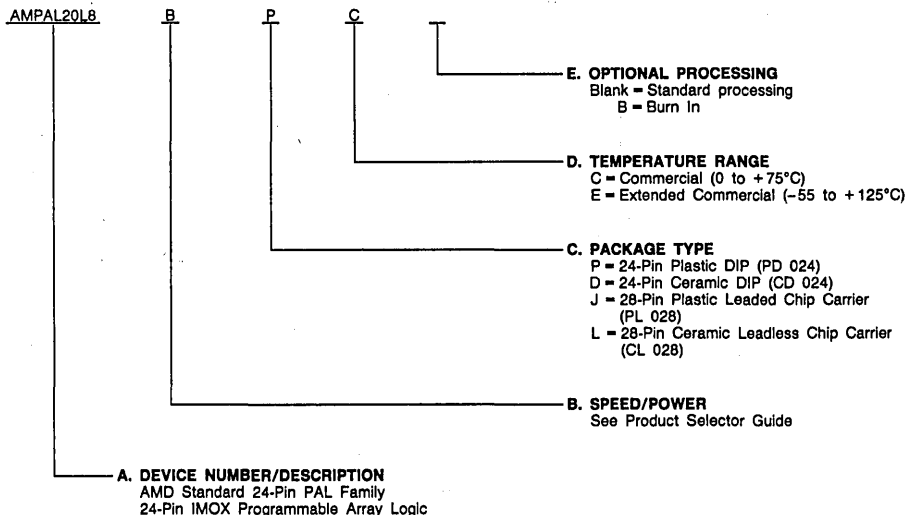
I = Input
 I/O = Input/Output
 O = Output
 VCC = Supply Voltage
 GND = Ground
 CLK = Clock
 OE = Output Enable
 NC = No Connect

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	
AMPAL20L10B/-20/A/AL	PC, DC, DCB, DE, JC, LC, LE
AMPAL20R4B/A/AL	
AMPAL20R6B/A/AL	
AMPAL20R8B/A/AL	
AMPAL20L8B/A/AL	

Valid Combinations

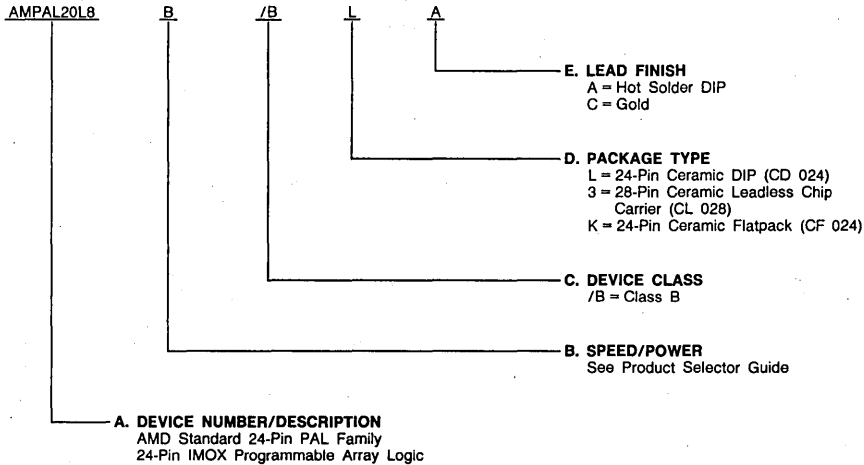
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL 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) for APL products is formed by a combination of:

- A. Device Number
- B. Speed Option (if applicable)
- C. Device Class
- D. Package Type
- E. Lead Finish



Valid Combinations	
AMPAL20L10B/-25/A/AL	/BLA, /B3C, /BKA
AMPAL20R4B/A/AL	
AMPAL20R6B/A/AL	
AMPAL20R8B/A/AL	
AMPAL20L8B/A/AL	

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 consist of Subgroups
1, 2, 3, 7, 8, 9, 10, & 11

FUNCTIONAL DESCRIPTION

AMD Standard 24-Pin PAL Family Characteristics

All members of the AMD Standard 24-Pin PAL Family have common electrical characteristics and programming procedures. All parts are produced with a fusible link at each input to the AND gate array, and connections may be selectively removed by applying appropriate voltages to the circuit.

Initially the AND gates are connected, via fuses, to both the true and complement of each input. By selective programming of fuses the AND gates may be "connected" to only the true input (by blowing the complement fuse), to only the complement input (by blowing the true fuse), or to neither type of input (by blowing both fuses) establishing a logical "don't care." When both the TRUE and complement fuses are left intact a logical false results on the output of the AND gate, while all fuses blown results in a logical-TRUE state. For combinatorial outputs, the AND gates are connected to fixed-OR gates whose outputs become device outputs. For registered outputs, the AND gates are connected to fixed-OR gates whose outputs become output register inputs.

All parts are fabricated with AMD's fast programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are pre-programmed during manufacturing to insure extremely high field programming yields (> 98%), and provide extra test paths to achieve excellent parametric correlation.

Power-Up Reset

The registered devices in the AMD PAL family have been designed to reset during system power-up. Following power-up, all registers will be initialized to zero, setting all the outputs to a logic 1. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

PRELOAD

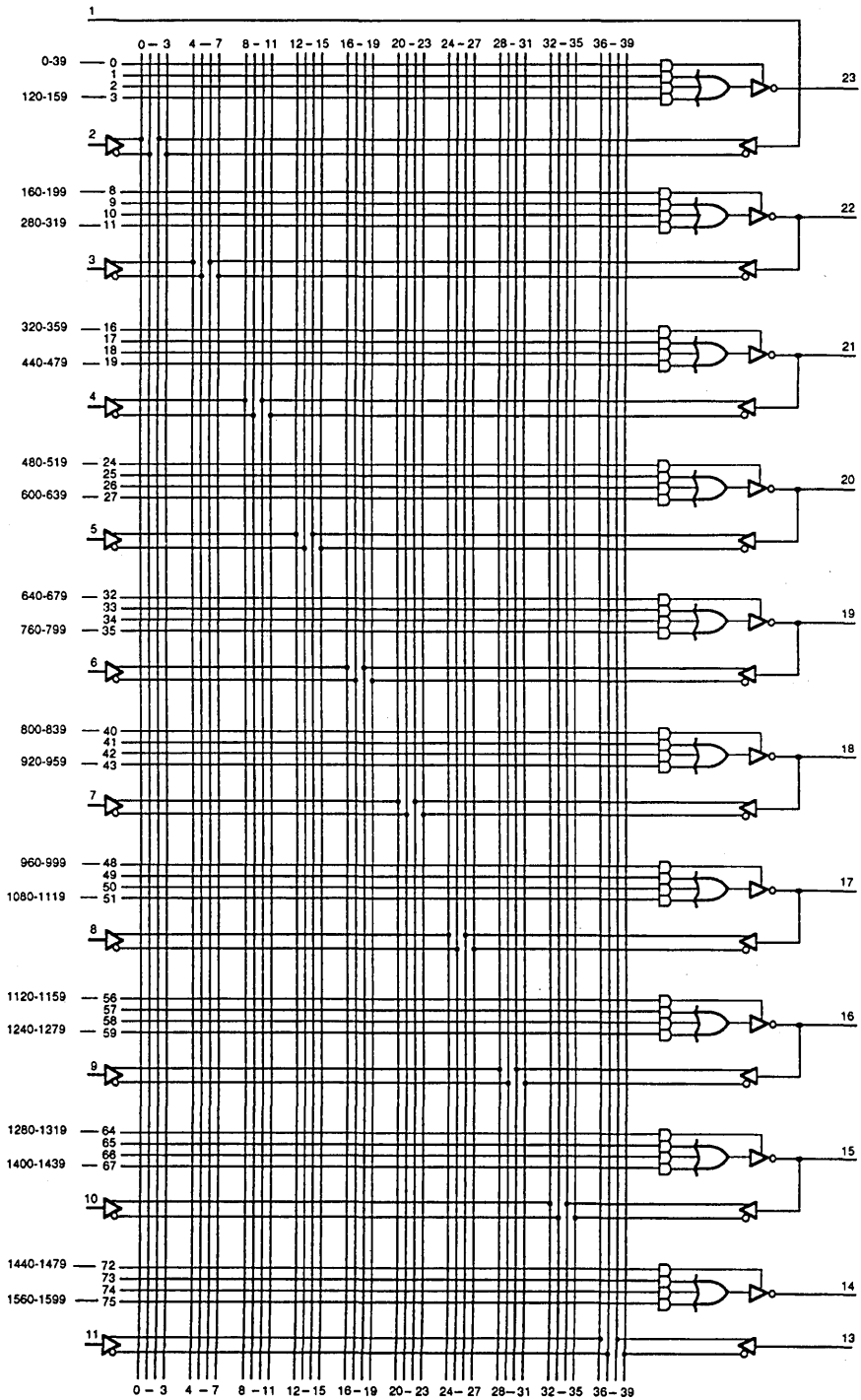
AMD PAL devices are designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality. PRELOAD allows any arbitrary state value to be loaded into the registered output of an AMD PAL device.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to load an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state register "interferes" with the inputs, forcing the machine to go through many transitions before it can reach an arbitrary state value. Therefore the test sequence will be mostly state initialization and not actual testing. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

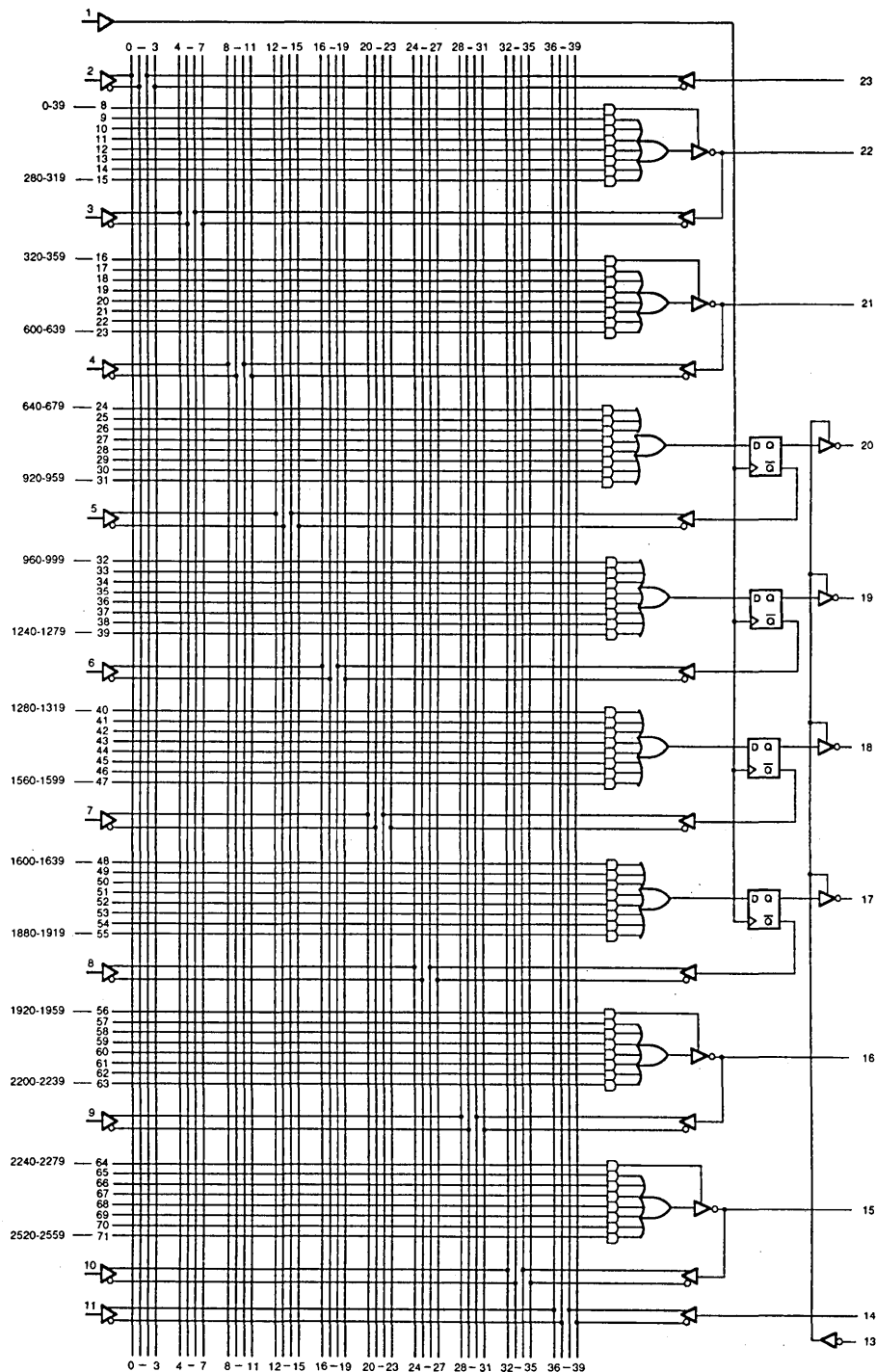
In addition, complete logic verification may become impossible when states that need to be tested cannot be entered with normal state transitions. For example, even though necessary, the state entered when a machine powers up cannot be tested, because it cannot be entered from the main sequence. Similarly, "forbidden" or don't care states that are not normally entered need to be tested to ensure that they return to the main sequence.

PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened, and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper in-system operation.



LD001210

Figure 1. AmPAL20L10 Logic Diagram and JEDEC Fuse Numbering



LD001220

Figure 2. AmPAL20R4 Logic Diagram and JEDEC Fuse Numbering

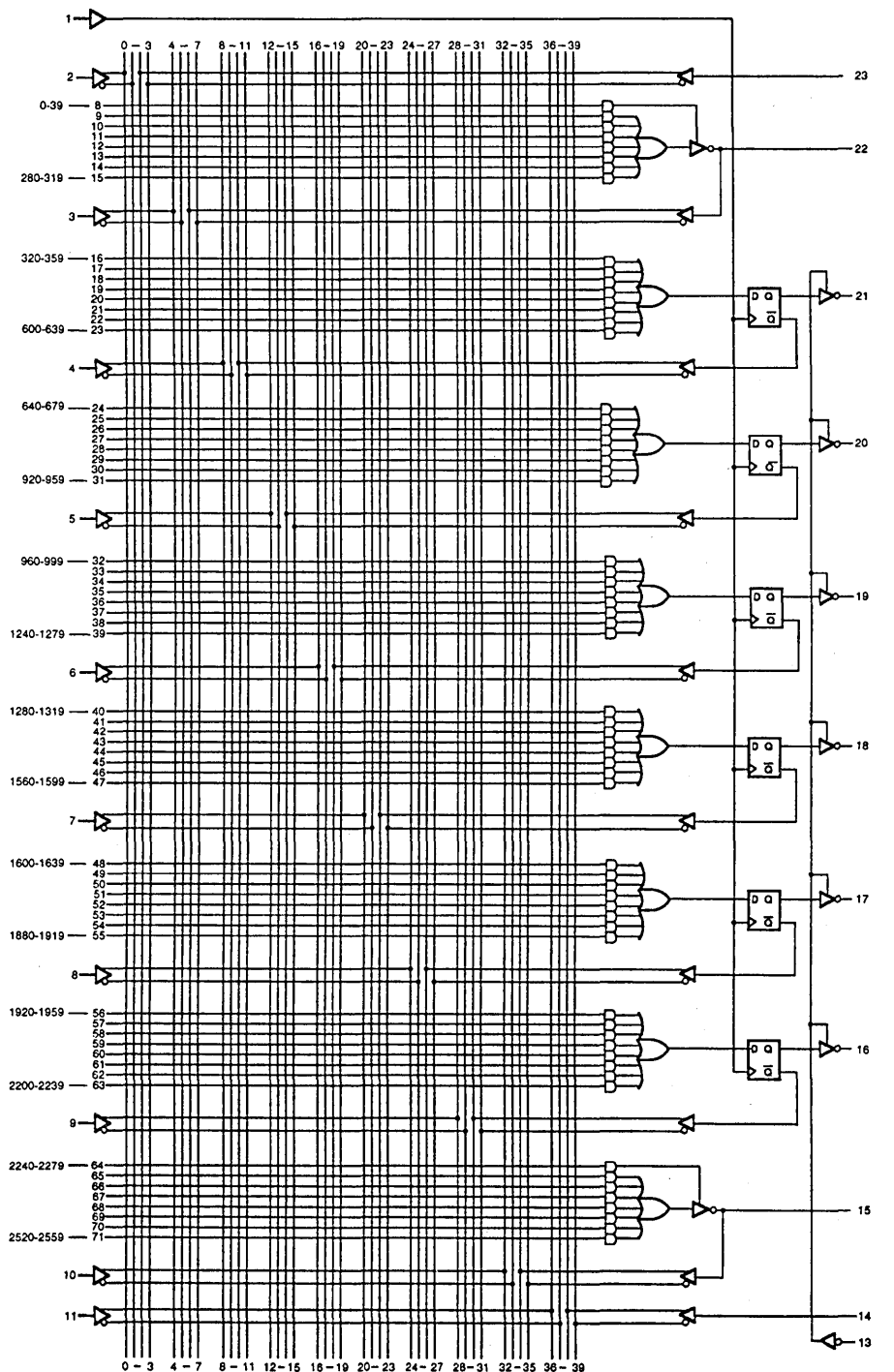
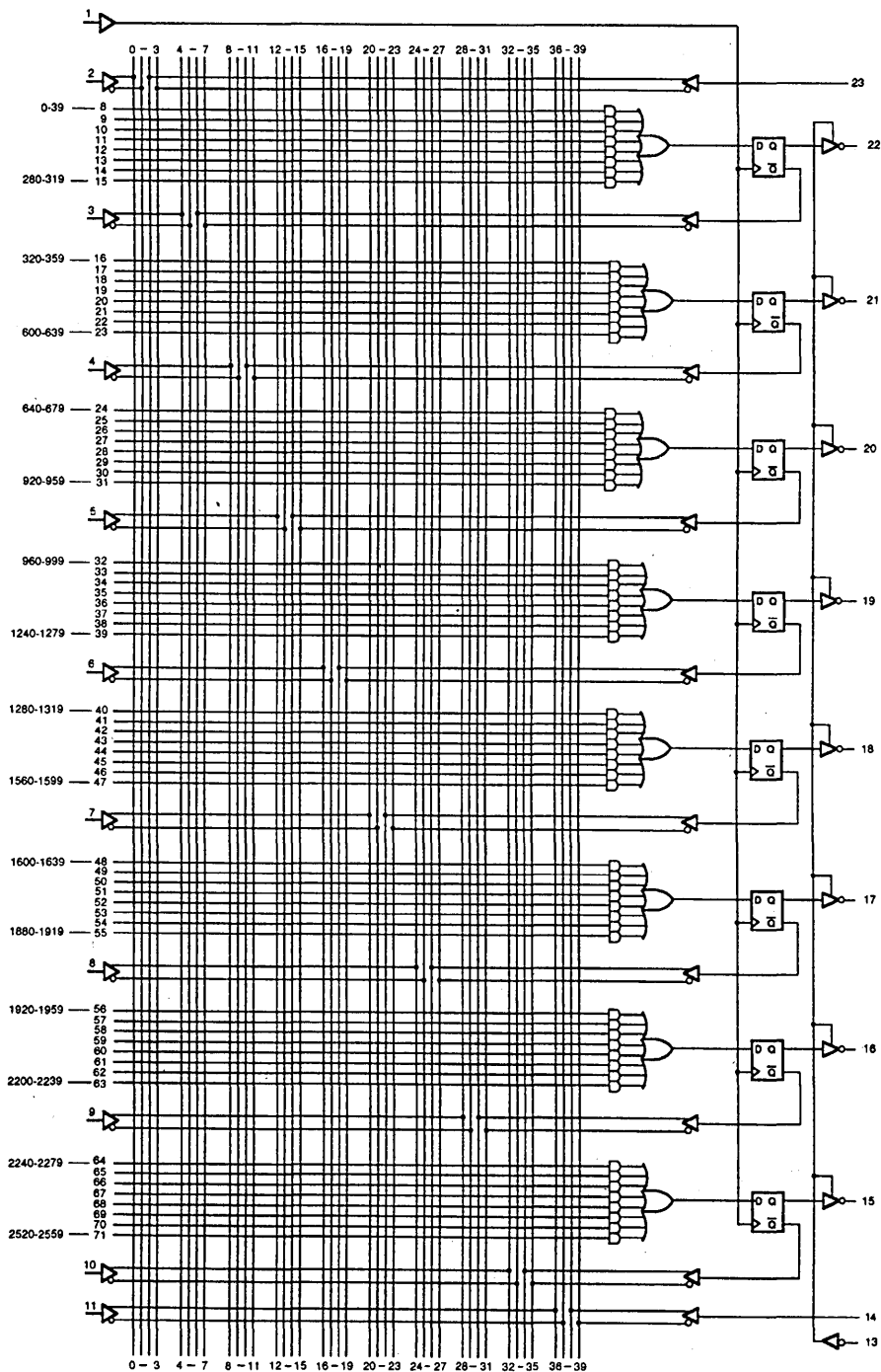
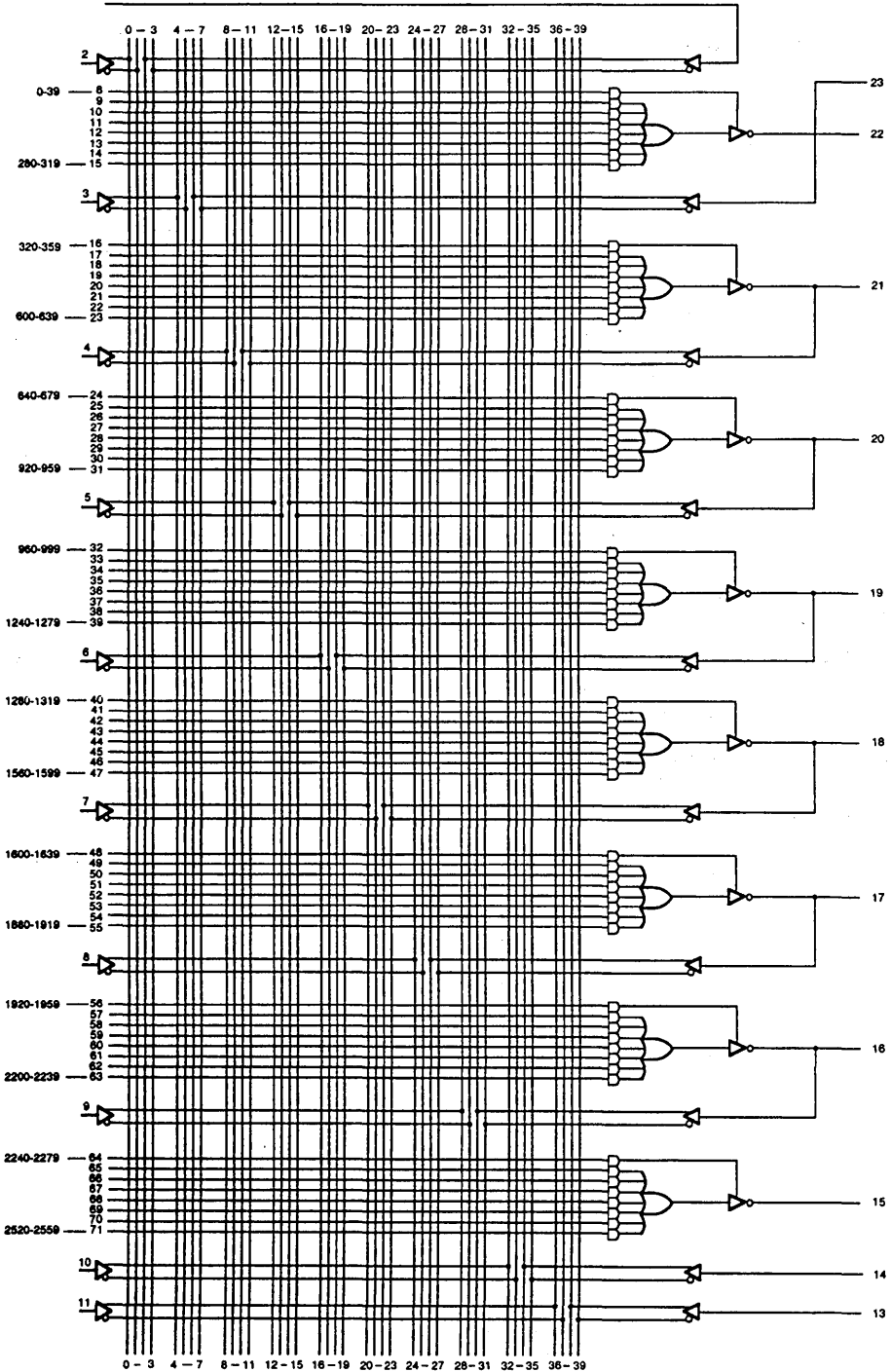


Figure 3. AmpAL20R6 Logic Diagram and JEDEC Fuse Numbering



LD001230

Figure 4. AmPAL20R8 Logic Diagram and JEDEC Fuse Numbering



LD001300

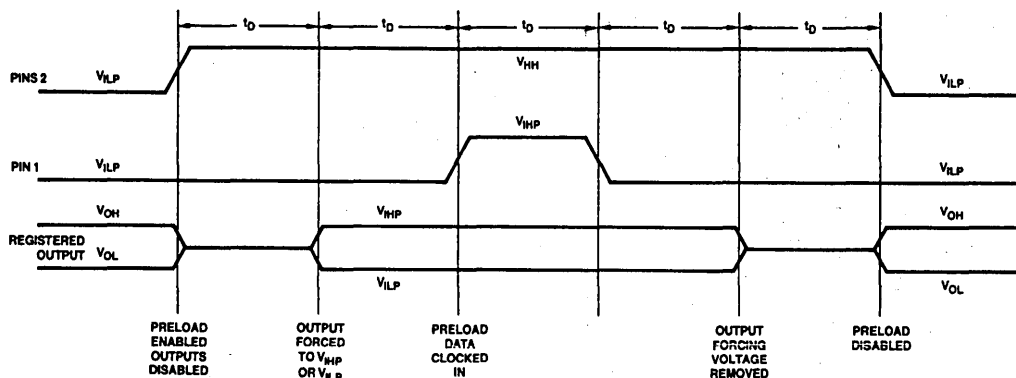
Figure 5. AmPAL20L8 Logic Diagram and JEDEC Fuse Numbering

PRELOAD of Registered Outputs

The AMD Standard 24-Pin PAL devices incorporate circuitry to allow loading each register synchronously to either a HIGH or

LOW state. This feature simplifies testing since any initial state for the registers can be set to optimize test sequencing.

The pin levels and timing necessary to perform the PRELOAD function are detailed below:



WF022294

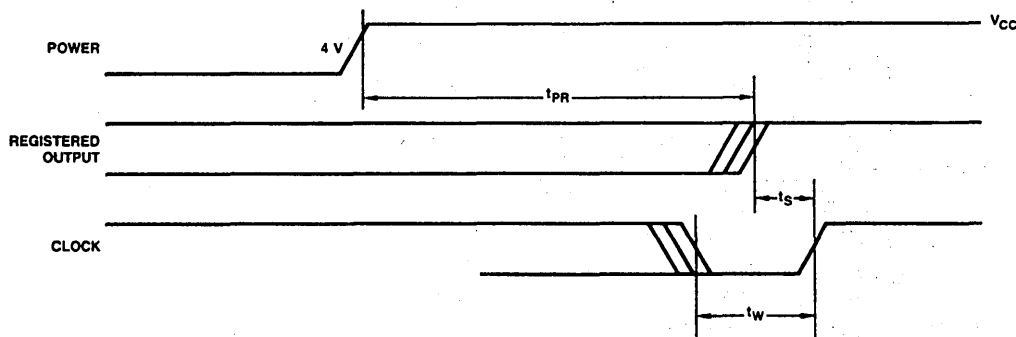
Level forced on registered output pin during PRELOAD cycle	Register Q output state after cycle
VIHP	HIGH
VILP	LOW

Power-Up Reset

The registered devices in the AMD Standard 24-Pin PAL Family have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to LOW. The output state will be HIGH. This feature provides flexibility to the designer and is especially valuable in simplifying state-machine initialization. A timing diagram and parameter table are shown below. Due to the asynchronous

operation of the power-up RESET and the wide range of ways V_{CC} can rise to its steady state, two conditions are required to ensure a valid power-up RESET. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.



WF022300

Parameters	Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up Reset Time		600	1000	ns
t_s	Input or Feedback Setup Time	See Switching Characteristics			
t_w	Clock Width				

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage to Ground Potential (Pin 24 to Pin 12) Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	16 V
Output Current Into Outputs During Programming (Max Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices

Temperature (T _A)	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V

Extended Commercial (E) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Military (M) Devices*

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions		Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -3.2 mA COM'L I _{OH} = -2 mA MIL	2.4	3.5		V
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = -24 mA COM'L I _{OL} = -12 mA MIL			0.5	V
V _{IH} (Note 2)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs		2.0		5.5	V
V _{IL} (Note 2)	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs				0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max., I _{IN} = 0.40 V			-20	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V				25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V				1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)		-30	-60	-90	mA
I _{CC}	Power Supply Current	V _{CC} = Max.	20L10	COM'L B -20, A AL	MIL B -25, A AL	210 165 105	mA
			20R4, 20R6, 20R8, 20L8	B A AL	B A AL	210 210 105	
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA			-0.9	-1.2	V
I _{OZH}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V			100	μA
I _{OZL}			V _O = 0.4 V			-100	

- Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.
2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.
3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.
V_{OUT} = 0.5V has been chosen to avoid test problems caused by tester ground degradation.
4. I/O pin leakage is the worst-case of I_{OZH} or I_I (where X = H or L).

CAPACITANCE*

Parameter Symbol	Parameter Description	Test Conditions		Typ.	Units
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz	Pins 1, 13 Others	11 6	pF
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz		9	

*These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

SWITCHING CHARACTERISTICS

over operating range unless otherwise specified; included in Group A Subgroup 9, 10, 11 tests unless otherwise noted

COMMERCIAL RANGE

No.	Parameter Symbol	Parameter Description	B Versions			-20 Versions (Note 4)			A & AL Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 20L10, 20R4, 20R6, 20L8			15			20			25	ns
2	t_{EA}	Input to Output Enable 20L10, 20R4, 20R6, 20L8			18			20			25	ns
3	t_{ER}	Input to Output Disable 20L10, 20R4, 20R6, 20L8			15			20			25	ns
4	t_{PZX}	Pin 13 to Output Enable 20R4, 20R6, 20R8			15						20	ns
5	t_{PXZ}	Pin 13 to Output Disable 20R4, 20R6, 20R8			12						20	ns
6	t_{CO}	Clock to Output 20R4, 20R6, 20R8			12						15	ns
7	t_S	Input or Feedback Setup Time 20R4, 20R6, 20R8		15							25	ns
8	t_H	Hold Time 20R4, 20R6, 20R8		0							0	ns
9	t_P	Clock Period ($t_S + t_{CO}$)		27							40	ns
10	t_{WL}/t_{WH}	Clock Width		10/12							15/15	ns
11	f_{MAX}	Maximum Frequency			37.0						25.0	MHz

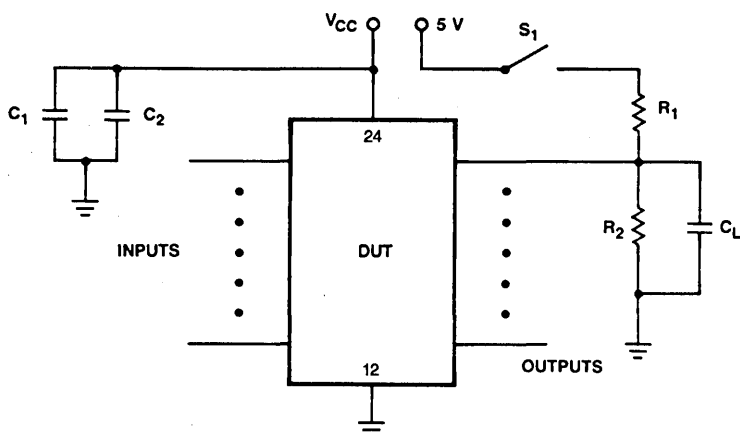
- Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.
 2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.
 3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.
 4. AmPAL20L10 only.

MILITARY RANGE

No.	Parameter Symbol	Parameter Description	B Versions			-25 Versions (Note 4)			A & AL Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 20L10, 20R4, 20R6, 20L8			20			25			30	ns
2	t_{EA}	Input to Output Enable 20L10, 20R4, 20R6, 20L8			25			25			30	ns
3	t_{ER}	Input to Output Disable 20L10, 20R4, 20R6, 20L8			20			25			30	ns
4	t_{PZX}	Pin 13 to Output Enable 20R4, 20R6, 20R8			20						25	ns
5	t_{PXZ}	Pin 13 to Output Disable 20R4, 20R6, 20R8			20						25	ns
6	t_{CO}	Clock to Output 20R4, 20R6, 20R8			15						20	ns
7	t_S	Input or Feedback Setup Time 20R4, 20R6, 20R8		20							30	ns
8	t_H	Hold Time 20R4, 20R6, 20R8		0							0	ns
9	t_P	Clock Period ($t_S + t_{CO}$)		35							50	ns
10	t_{WL}/t_{WH}	Clock Width		12/12							20/20	ns
11	f_{MAX}	Maximum Frequency			28.6						20.0	MHz

- Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.
 2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.
 3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.
 4. AmPAL20L10 only.

SWITCHING TEST CIRCUIT

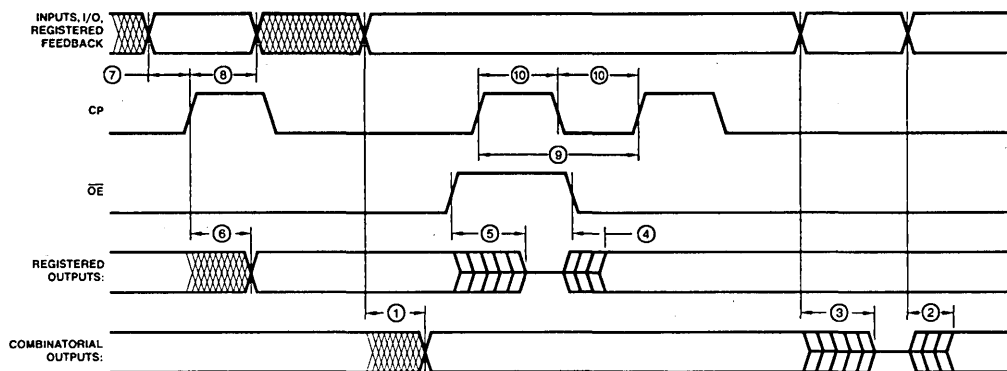


TC003051

Note: C_1 and C_2 are to bypass V_{CC} to ground.

TEST OUTPUT LOADS		
Pin Name	Commercial	Military
R_1	200 Ω	390 Ω
R_2	390 Ω	750 Ω
C_1	1 μF	1 μF
C_2	0.1 μF	0.1 μF
C_L	50 pF	50 pF

SWITCHING WAVEFORMS



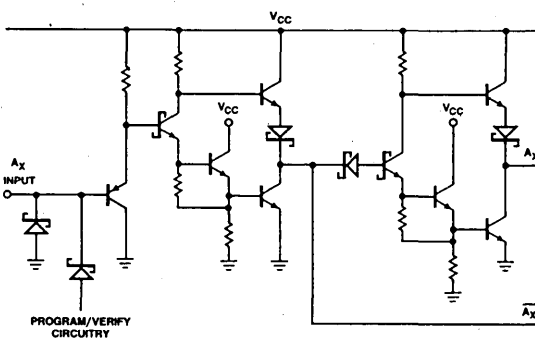
WF002571

KEY TO TIMING DIAGRAM

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

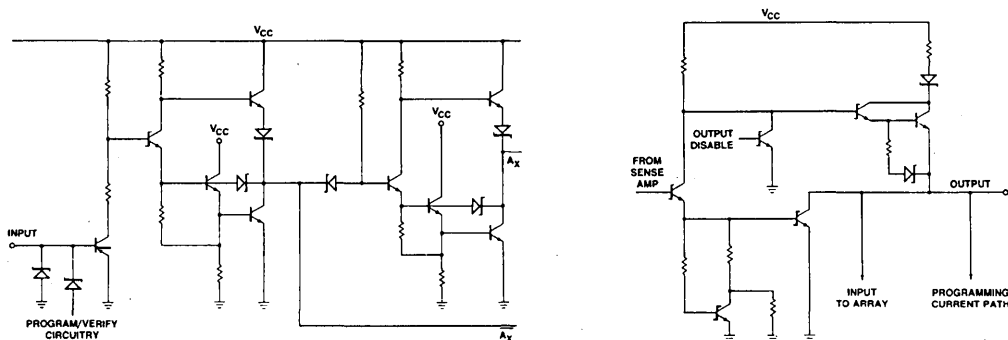
KS000010

INPUT CIRCUITRY



IC000720

OUTPUT CIRCUITRY



IC000801

Programming and Verification

AMD Standard 24-Pin PAL Family devices are programmed and verified using AMD's standard programmable logic algorithm. The fuse to be programmed is selected by input line number (array row), product term (array column), and by output (one at a time). The fuse is then programmed and verified by applying a simple sequence of voltages to two control pins (1 and 13).

Input line numbers are addressed using a full decode scheme via TTL levels on pins 6-11 where 6 is the LSB and 11 is the MSB. Even-numbered input lines represent the true version of a signal and odd-numbered lines represent the complement. Input line addressing is shown in Table 1.

Product terms are addressed using a 1-of-16 addressing scheme on pins 2-5 where pin 2 is the LSB and 5 is the MSB. Product term addressing is shown in Table 2. Logical product terms are selected via TTL levels on the four addressing pins.

Fuse selection by output must be done one output at a time (following control pin 1 going to V_{HH}), as shown in the programming timing diagram.

Once fuses have been selected, the simple programming and verification sequence may be completed as shown in the programming timing diagram. AC and DC requirements for programming are shown in the programming parameter table.

Security Fuse Programming

A single fuse is provided on each device to prevent unauthorized copying of PAL fuse patterns. Once blown, the circuitry enabling fuse verification and registered output PRELOAD is permanently disabled.

Programming of the security fuse is the same as an array fuse. Verification of a blown security fuse is accomplished by verifying the whole fuse array as if every fuse is blown.

Programming Yield

AMD PAL devices have been designed to ensure extremely high programming yields (> 98%). To help ensure that a part was correctly programmed, once programming is completed, the entire fuse array should be verified at both LOW and HIGH V_{CC} . Reverification can be accomplished by reading all ten outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PAL devices contain many internal test features, including circuitry and extra fuses which allow AMD to test the ability of each part to perform programming before shipping, to assure high programming yields and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS $T_A = 25^\circ\text{C}$

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V_{HH}	Control Pin Extra High Level	Pin 1 @ 5-10 mA	10	11	12	V
		Pin 13 @ 5-10 mA	10	11	12	
V_{OP}	Program Voltage Pins 14-23 @ 15-200 mA		14	15	16	V
V_{IHP}	Input HIGH Level During Programming and Verify		2.4	5	5.5	V
V_{ILP}	Input LOW Level During Programming and Verify		0.0	0.3	0.5	V
V_{CCP}	V_{CC} During Programming @ $I_{CC} = 50-275$ mA		5	5.2	5.5	V
V_{CCL}	V_{CC} During First Pass Verification @ $I_{CC} = 50-275$ mA		4.4	4.5	4.6	V
V_{CCH}	V_{CC} During Second Pass Verification @ $I_{CC} = 50-275$ mA		5.4	5.5	5.6	V
V_{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	V
V_{OP}/dt	Rate of Output Voltage Change		20		250	V/ μ s
dV_{13}/dt	Rate of Fusing Enable Voltage Change (Pin 13 Rising Edge)		100		1000	V/ μ s
t_p	Fusing Time First Attempt		40	50	100	μ s
	Subsequent Attempts		4	5	10	ms
t_D	Delays Between Various Level Changes		100	200	1000	ns
t_V	Period During which Output is Sensed for V_{Blown} Level				500	ns
V_{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		$V_{CCP} - 0.3$	V_{CCP}	$V_{CCP} + 0.3$	V
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	k Ω

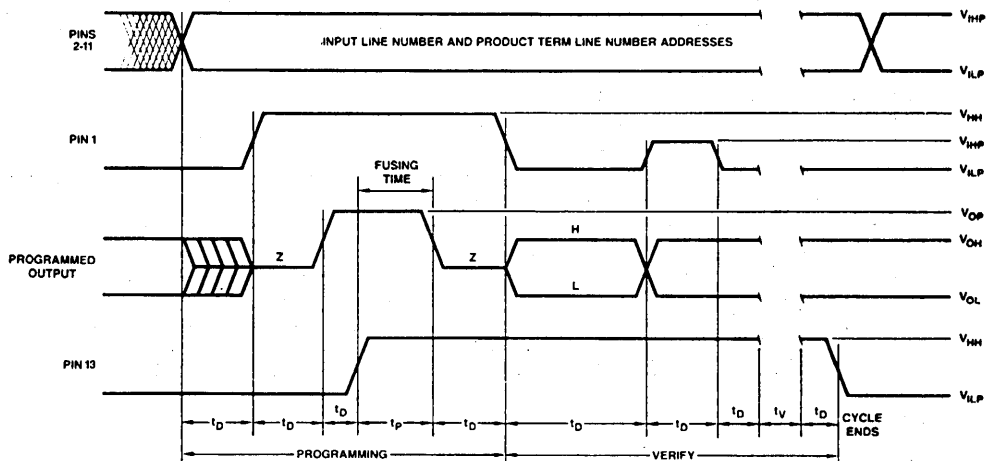
Design Aid Software for AMD Standard 24-Pin PAL Family

Name	Vendor	Versions	Notes
ABEL	Data I/O (206) 881-6444	IBM PC VAX/VMS VAX/UNIX	
CUPL	P-CAD Systems (408) 971-1300	IBM PC VAX/VMS VAX/UNIX CPM 80/86	
AmCUPL	Advanced Micro Devices (408) 732-2400	IBM PC	Supported by P-CAD Systems

AMD Qualified Programmers

Name	Programmer Model(s)	AMD PAL Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 93052	Systems 19, 29	950-1942-0044	303A-011A
	60	N/A	Under Development
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086	Model PPZ	Under Development	On Board
	ZL30	Under Development	
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043	160 Series	Under Development	On Board

PROGRAMMING TIMING DIAGRAM



PF001101

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States					
	6	7	8	9	10	11
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	L
3	L	L	L	L	H	H
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	H
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	H
12	L	L	H	H	L	L
13	L	L	H	H	L	H
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	H
18	L	H	L	L	H	L
19	L	H	L	L	H	H
20	L	H	L	H	L	L
21	L	H	L	H	L	H
22	L	H	L	H	H	L
23	L	H	L	H	H	H
24	L	H	H	L	L	L
25	L	H	H	L	L	H
26	L	H	H	L	H	L
27	L	H	H	L	H	H
28	L	H	H	H	L	L
29	L	H	H	H	L	H
30	L	H	H	H	H	L
31	L	H	H	H	H	H
32	H	L	L	L	L	L
33	H	L	L	L	L	H
34	H	L	L	L	H	L
35	H	L	L	L	H	H
36	H	L	L	H	L	L
37	H	L	L	H	L	H
38	H	L	L	H	H	L
39	H	L	L	H	H	H

L = V_{ILP} H = V_{IHP}

TABLE 2-1. PRODUCT TERM ADDRESSING (AmPAL20L10)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	8	16	24	32	40	48	56	64	72
L	L	L	H	1	9	17	25	33	41	49	57	65	73
L	L	H	L	2	10	18	26	34	42	50	58	66	74
L	L	H	H	3	11	19	27	35	43	51	59	67	75
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 SF = Security Fuse

TABLE 2-2. PRODUCT TERM ADDRESSING (AmPAL20R4)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	
L	L	L	L	-	8	16	24	32	40	48	56	64	
L	L	L	H	-	9	17	25	33	41	49	57	65	
L	L	H	L	-	10	18	26	34	42	50	58	66	
L	L	H	H	-	11	19	27	35	43	51	59	67	
L	H	L	L	-	12	20	28	36	44	52	60	68	
L	H	L	H	-	13	21	29	37	45	53	61	69	
L	H	H	L	-	14	22	30	38	46	54	62	70	
L	H	H	H	-	15	23	31	39	47	55	63	71	
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 SF = Security Fuse

TABLE 2-3. PRODUCT TERM ADDRESSING (AmpAL20R6)

Product Term Select Address Pin				Programming Access and Verify Pin								
5	4	3	2	23	22	21	20	19	18	17	16	15
L	L	L	L	-	8	16	24	32	40	48	56	64
L	L	L	H	-	9	17	25	33	41	49	57	65
L	L	H	L	-	10	18	26	34	42	50	58	66
L	L	H	H	-	11	19	27	35	43	51	59	67
L	H	L	L	-	12	20	28	36	44	52	60	68
L	H	L	H	-	13	21	29	37	45	53	61	69
L	H	H	L	-	14	22	30	38	46	54	62	70
L	H	H	H	-	15	23	31	39	47	55	63	71
H	H	H	H	SF								
L = VILP H = VIHPP OE = Output Enable SF = Security Fuse				Logical/Architectural Product Term Line Number								

TABLE 2-4. PRODUCT TERM ADDRESSING (AmpAL20R8)

Product Term Select Address Pin				Programming Access and Verify Pin								
5	4	3	2	23	22	21	20	19	18	17	16	15
L	L	L	L	-	8	16	24	32	40	48	56	64
L	L	L	H	-	9	17	25	33	41	49	57	65
L	L	H	L	-	10	18	26	34	42	50	58	66
L	L	H	H	-	11	19	27	35	43	51	59	67
L	H	L	L	-	12	20	28	36	44	52	60	68
L	H	L	H	-	13	21	29	37	45	53	61	69
L	H	H	L	-	14	22	30	38	46	54	62	70
L	H	H	H	-	15	23	31	39	47	55	63	71
H	H	H	H	SF								
L = VILP H = VIHPP OE = Output Enable SF = Security Fuse				Logical/Architectural Product Term Line Number								

TABLE 2-5. PRODUCT TERM ADDRESSING (AmpAL20L8)

Product Term Select Address Pin				Programming Access and Verify Pin								
5	4	3	2	23	22	21	20	19	18	17	16	15
L	L	L	L	-	8	16	24	32	40	48	56	64
L	L	L	H	-	9	17	25	33	41	49	57	65
L	L	H	L	-	10	18	26	34	42	50	58	66
L	L	H	H	-	11	19	27	35	43	51	59	67
L	H	L	L	-	12	20	28	36	44	52	60	68
L	H	L	H	-	13	21	29	37	45	53	61	69
L	H	H	L	-	14	22	30	38	46	54	62	70
L	H	H	H	-	15	23	31	39	47	55	63	71
H	H	H	H	SF								

L = V_{ILP}
 H = V_{IHP}
 OE = Output Enable
 SF = Security Fuse

Logical/Architectural Product Term Line Number

AMD Enhanced 24-Pin PAL* Family

24-Pin IMOX™ Programmable Array Logic (PAL) Elements

PRELIMINARY

AMD Enhanced 24-Pin PAL* Family

DISTINCTIVE CHARACTERISTICS

- AMD's superior IMOX technology
 - Guarantees $t_{pd} = 15$ ns max
- Individually programmable output polarity on each output
- Eight logical product terms per output
- Programming yields > 98% are realized via platinum-silicide fuse technology and the use of added test words
- Post Programming Functional Yield (PPFY) of 99.9%
- PRELOAD feature permits full logical verification
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parametric testing at the factory through on-board testing circuitry
- > 3000V ESD input protection per pin
- JEDEC-Standard LCC and PLCC pinout

GENERAL DESCRIPTION

AMD Enhanced 24-pin PAL devices are high-speed, electrically programmable array logic elements. They utilize the familiar sum-of-products (AND-OR) structure allowing users to program custom logic functions to fit most applications precisely. Typically they are a replacement for low-power Schottky SSI/MSI logic circuits, reducing chip count by more than 5 to 1 and greatly simplifying prototyping and board layout. Additional product terms, two additional outputs, and programmable output polarity are enhancements over industry-standard 24-pin PAL devices.

Five different devices are available, including both registered and combinatorial devices. All devices have user-programmable output polarity on all outputs. A variety of speed options allow the designer maximum flexibility in matching precise system requirements. The Product Selector Guide below shows the available speed options. The second table gives details about the functionality of the five available devices.

Please see the following pages for Block Diagrams.

PRODUCT SELECTOR GUIDE

AMD PAL Speed/Power Families

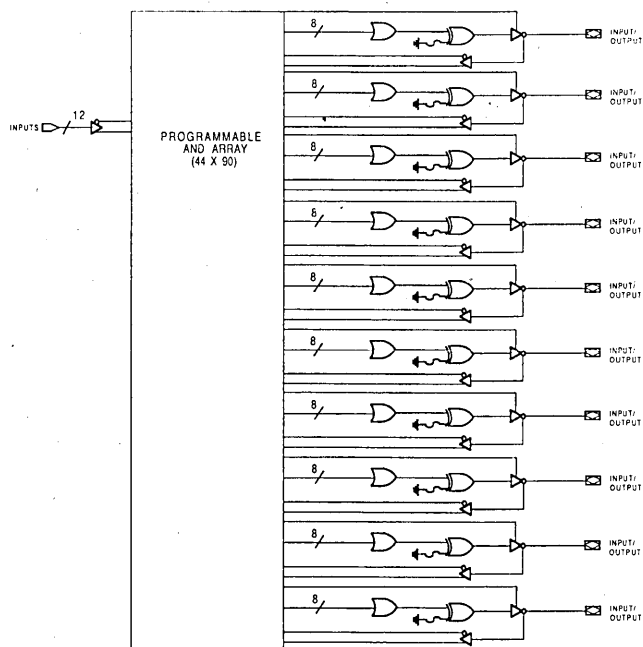
Family	t_{pd} ns (Max.)		t_s ns (Min.)		t_{co} ns (Max.)		I_{CC} mA (Max.)	I_{OL} mA (Min.)	
	C Devices	M Devices	C Devices	M Devices	C Devices	M Devices	C/M Devices	C Devices	M Devices
Very High-Speed ("B") Versions	15	20	15	20	12	15	210	24	12
High-Speed ("A") Versions	25	30	25	30	15	20	210	24	12
High-Speed, Half-Power ("AL") Versions	25	30	25	30	15	20	105	24	12

Part Number	Array Inputs	Logic	Output Enable	Outputs/Polarity	Package Pins
22P10	12 Dedicated, 10 Bidirectional	Ten (8)-Wide AND-OR	Programmable	Bidirectional/Programmable	24
20RP4	10 Dedicated, 4 Feedback, 6 Bidirectional	Four (8)-Wide AND-OR	Dedicated	Registered/Programmable	24
		Six 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20RP6	10 Dedicated, 6 Feedback, 4 Bidirectional	Six (8)-Wide AND-OR	Dedicated	Registered/Programmable	24
		Four 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20RP8	10 Dedicated, 8 Feedback, 2 Bidirectional	Eight (8)-Wide AND-OR	Dedicated	Registered/Programmable	24
		Two 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20RP10	10 Dedicated, 10 Feedback	Ten (8)-Wide AND-OR	Dedicated	Registered/Programmable	24

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

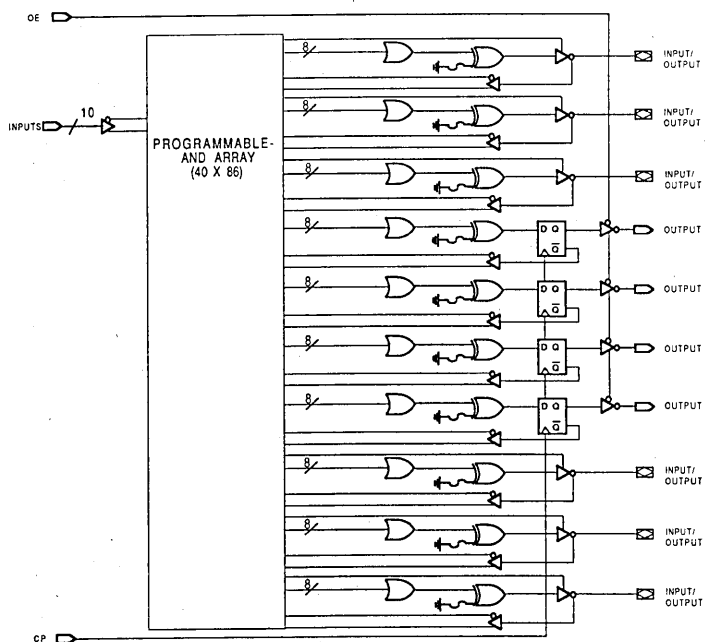
BLOCK DIAGRAMS

AmPAL22P10



LD001100

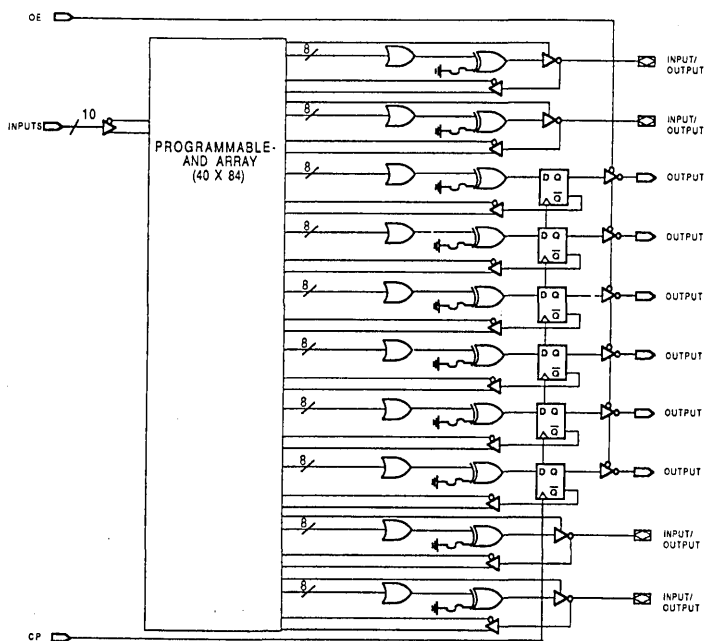
AmPAL20RP4



LD001110

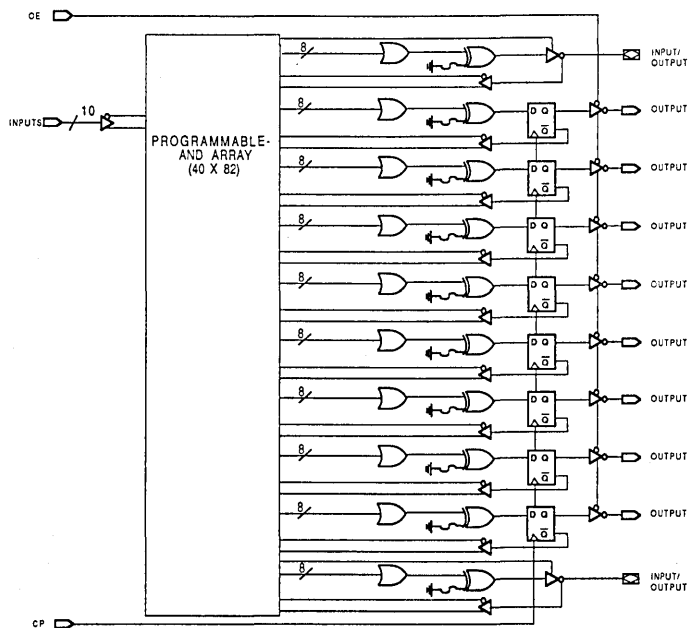
BLOCK DIAGRAMS (Cont'd.)

AmPAL20RP6



LD001120

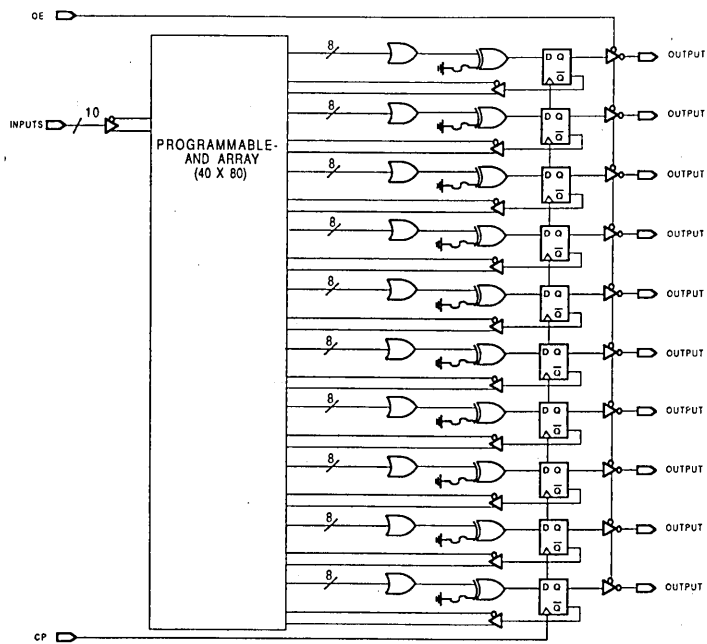
AmPAL20RP8



LD001130

BLOCK DIAGRAMS (Cont'd.)

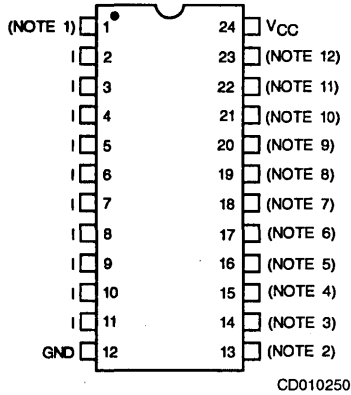
AmPAL20RP10



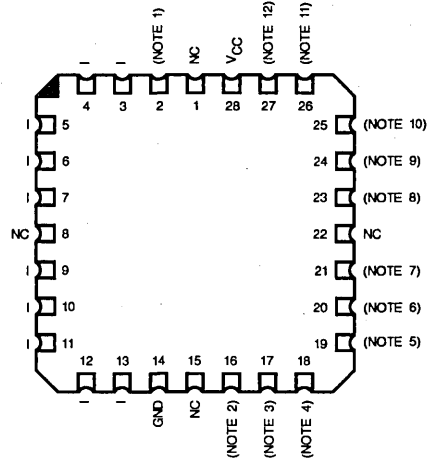
LD001140

CONNECTION DIAGRAMS Top View

DIPs*



LCC**



Note: Pin 1 is marked for orientation.

Notes:

	22P10	20RP4	20RP6	20RP8	20RP10
1	I	CLK	CLK	CLK	CLK
2	I	OE	OE	OE	OE
3	I/O	I/O	I/O	I/O	O
4	I/O	I/O	I/O	O	O
5	I/O	I/O	O	O	O
6	I/O	O	O	O	O
7	I/O	O	O	O	O
8	I/O	O	O	O	O
9	I/O	O	O	O	O
10	I/O	I/O	O	O	O
11	I/O	I/O	I/O	O	O
12	I/O	I/O	I/O	I/O	O

*Also available in 24-Pin Ceramic Flatpack. Pinouts identical to DIPs.

**Also available in 28-Pin Plastic Leaded Chip Carrier. Pinouts identical to LCC.

PIN DESIGNATIONS

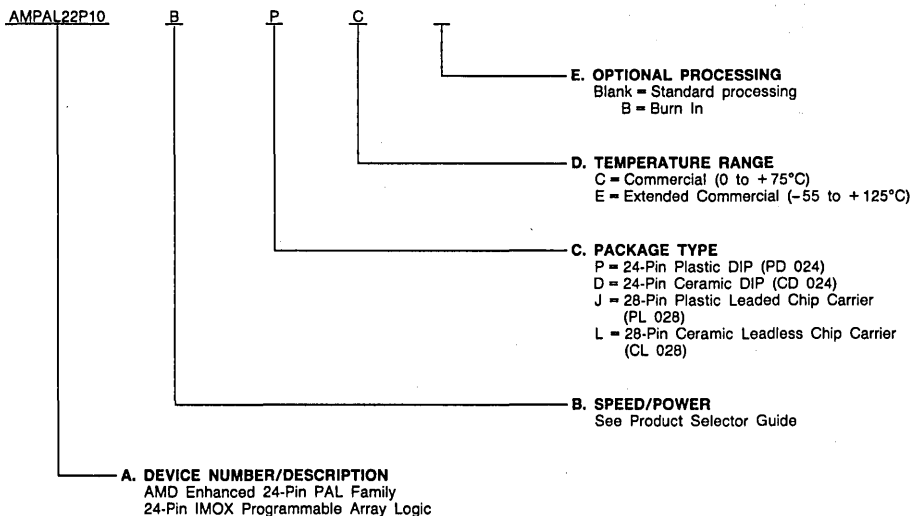
I = Input
 I/O = Input/Output
 O = Output
 V_{CC} = Supply Voltage
 GND = Ground
 CLK = Clock
 OE = Output Enable
 NC = No Connect

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**



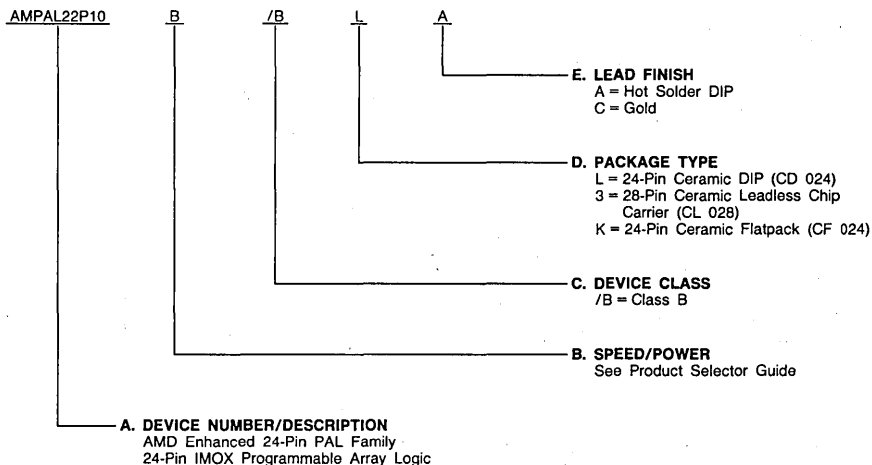
Valid Combinations	
AMPAL22P10B/A/AL	PC, DC, DCB, DE, JC, LC, LE
AMPAL20RP4B/A/AL	
AMPAL20RP6B/A/AL	
AMPAL20RP8B/A/AL	
AMPAL20RP10B/A/AL	

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, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)**APL 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) for APL products is formed by a combination of:

A. Device Number**B. Speed Option (if applicable)****C. Device Class****D. Package Type****E. Lead Finish****Valid Combinations**

Valid Combinations	
AMPAL22P10B/A/AL	/BLA, /B3C, /BKA
AMPAL20RP4B/A/AL	
AMPAL20RP6B/A/AL	
AMPAL20RP8B/A/AL	
AMPAL20RP10B/A/AL	

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 consist of Subgroups
1, 2, 3, 7, 8, 9, 10, & 11

FUNCTIONAL DESCRIPTION

AMD Enhanced 24-Pin PAL Family Characteristics

All members of the AMD Enhanced 24-Pin PAL Family have common electrical characteristics and programming procedures. All parts are produced with a fusible link at each input to the AND gate array, and connections may be selectively removed by applying appropriate voltages to the circuit.

Initially the AND gates are connected, via fuses, to both the true and complement of each input. By selective programming of fuses the AND gates may be "connected" to only the true input (by blowing the complement fuse), to only the complement input (by blowing the true fuse), or to neither type of input (by blowing both fuses) establishing a logical "don't care." When both the true and complement fuses are left intact a logical false results on the output of the AND gate, while all fuses blown results in a logical true state. For combinatorial outputs, the AND gates are connected to fixed-OR gates whose outputs become device outputs. For registered outputs, the AND gates are connected to fixed-OR gates whose outputs become output register inputs.

All parts are fabricated with AMD's fast programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are pre-programmed during manufacturing to insure extremely high field programming yields (> 98%), and provide extra test paths to achieve excellent parametric correlation.

Power-Up Reset

The registered devices in the AMD PAL family have been designed to reset during system power-up. Following power-up, all registers will be initialized to zero, setting all the outputs to a logic 1. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

PRELOAD

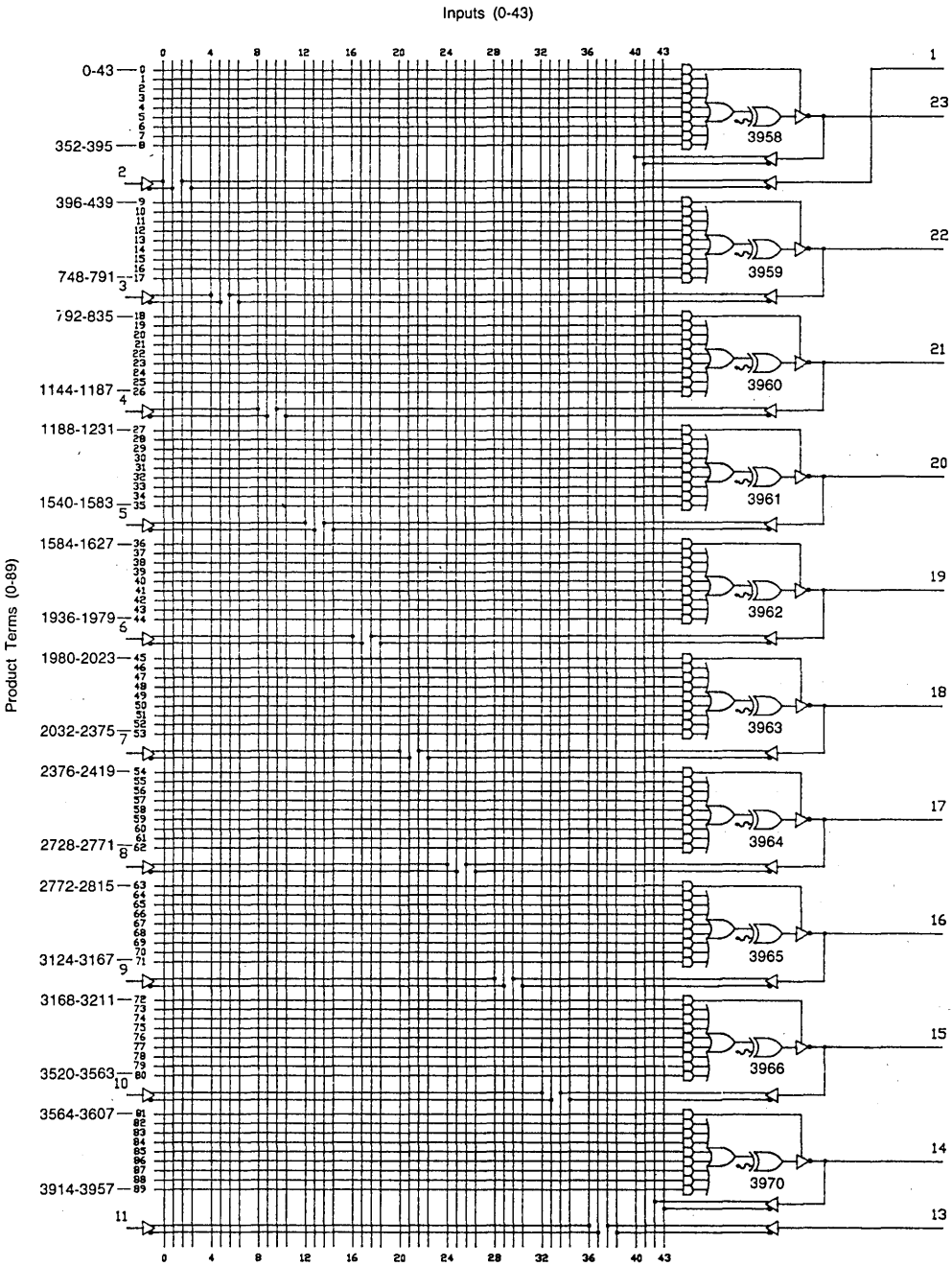
AMD PAL devices are designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality. PRELOAD allows any arbitrary state value to be loaded into the registered output of an AMD PAL device.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to load an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state register "interferes" with the inputs, forcing the machine to go through many transitions before it can reach an arbitrary state value. Therefore the test sequence will be mostly state initialization and not actual testing. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

In addition, complete logic verification may become impossible when states that need to be tested cannot be entered with normal state transitions. For example, even though necessary, the state entered when a machine powers up cannot be tested, because it cannot be entered from the main sequence. Similarly, "forbidden" or don't care states that are not normally entered need to be tested to ensure that they return to the main sequence.

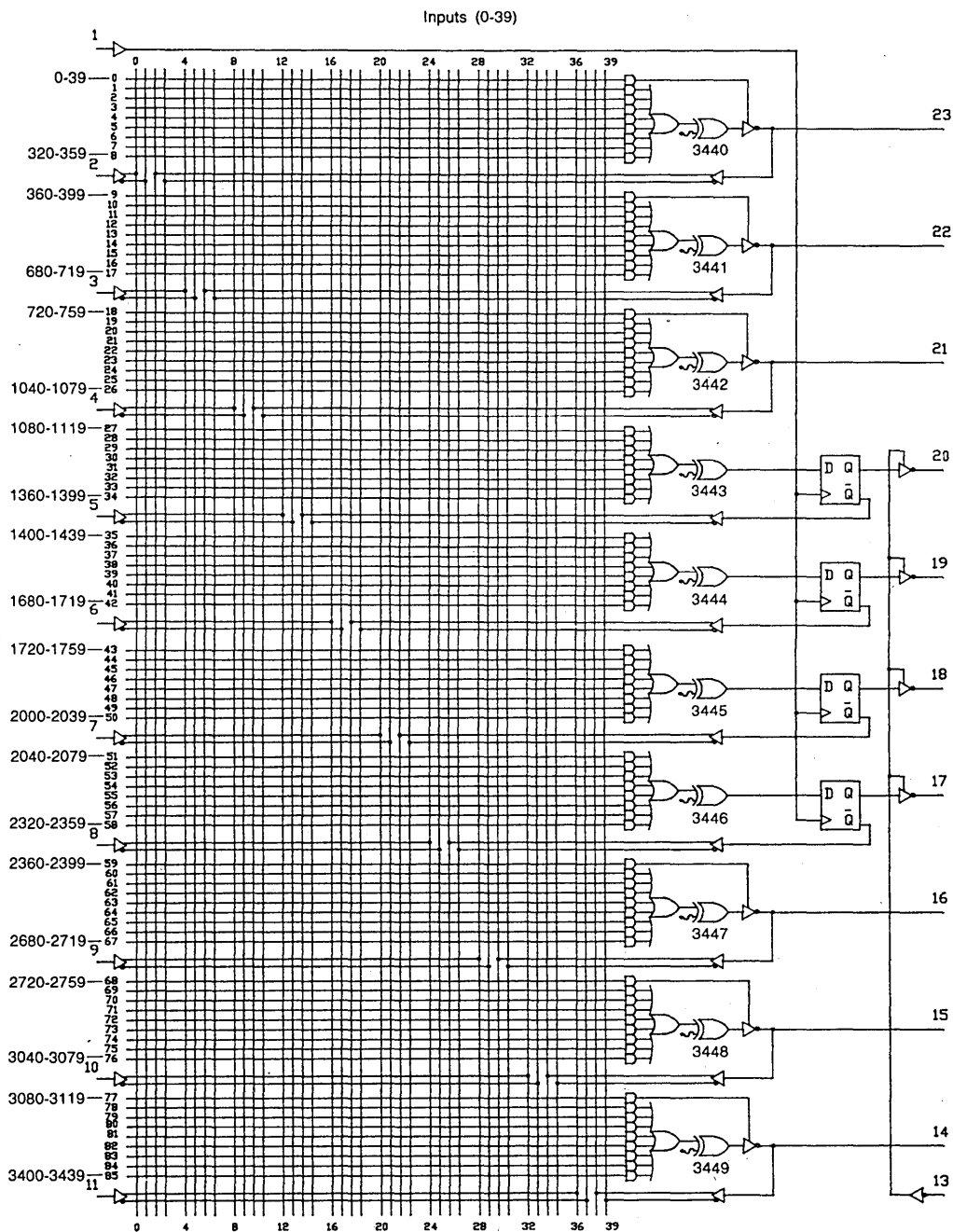
PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened, and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper in-system operation.



LD001270

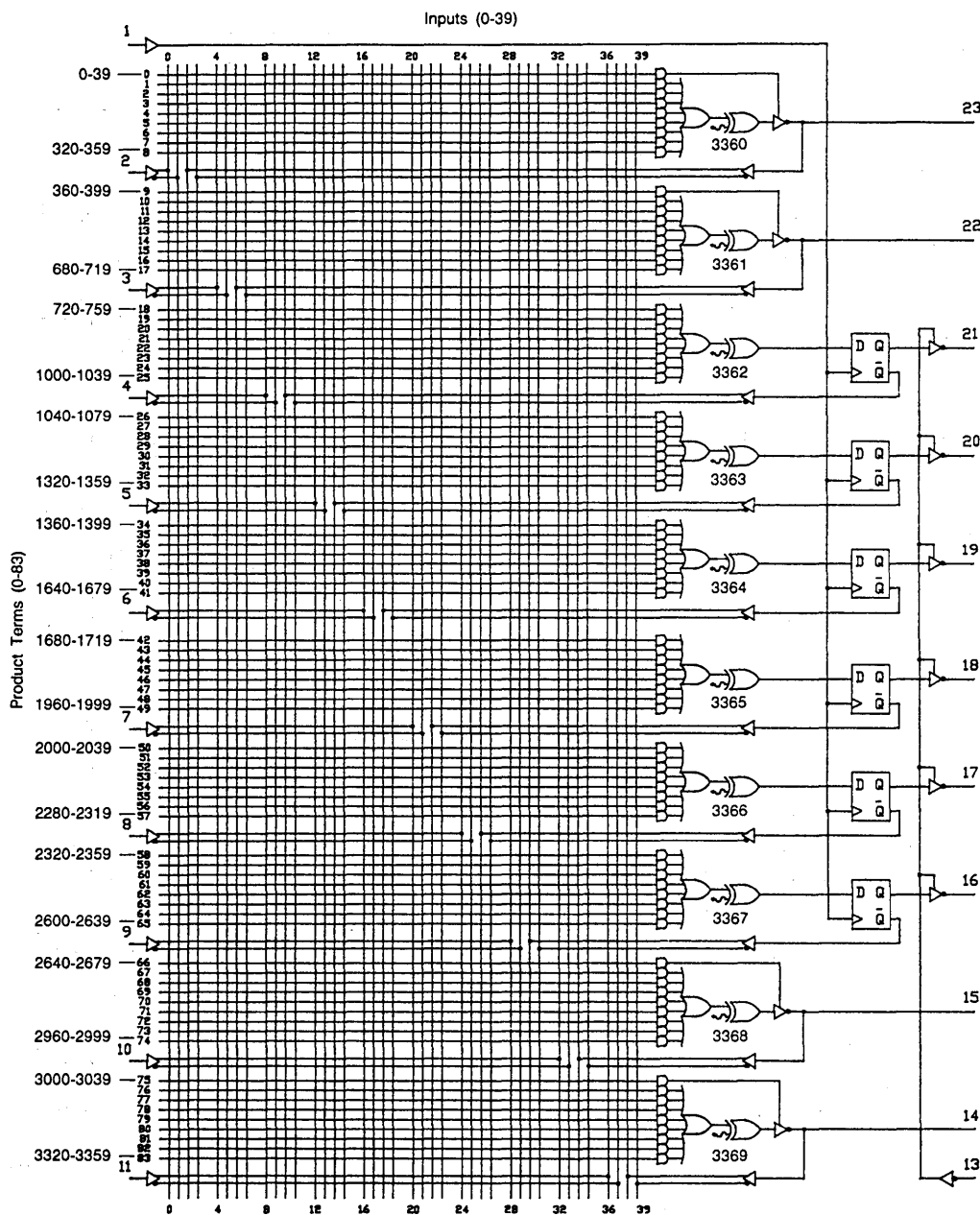
Figure 1. AmpPAL22P10 Logic Diagram and JEDEC Fuse Numbering

Product Terms (0-85)



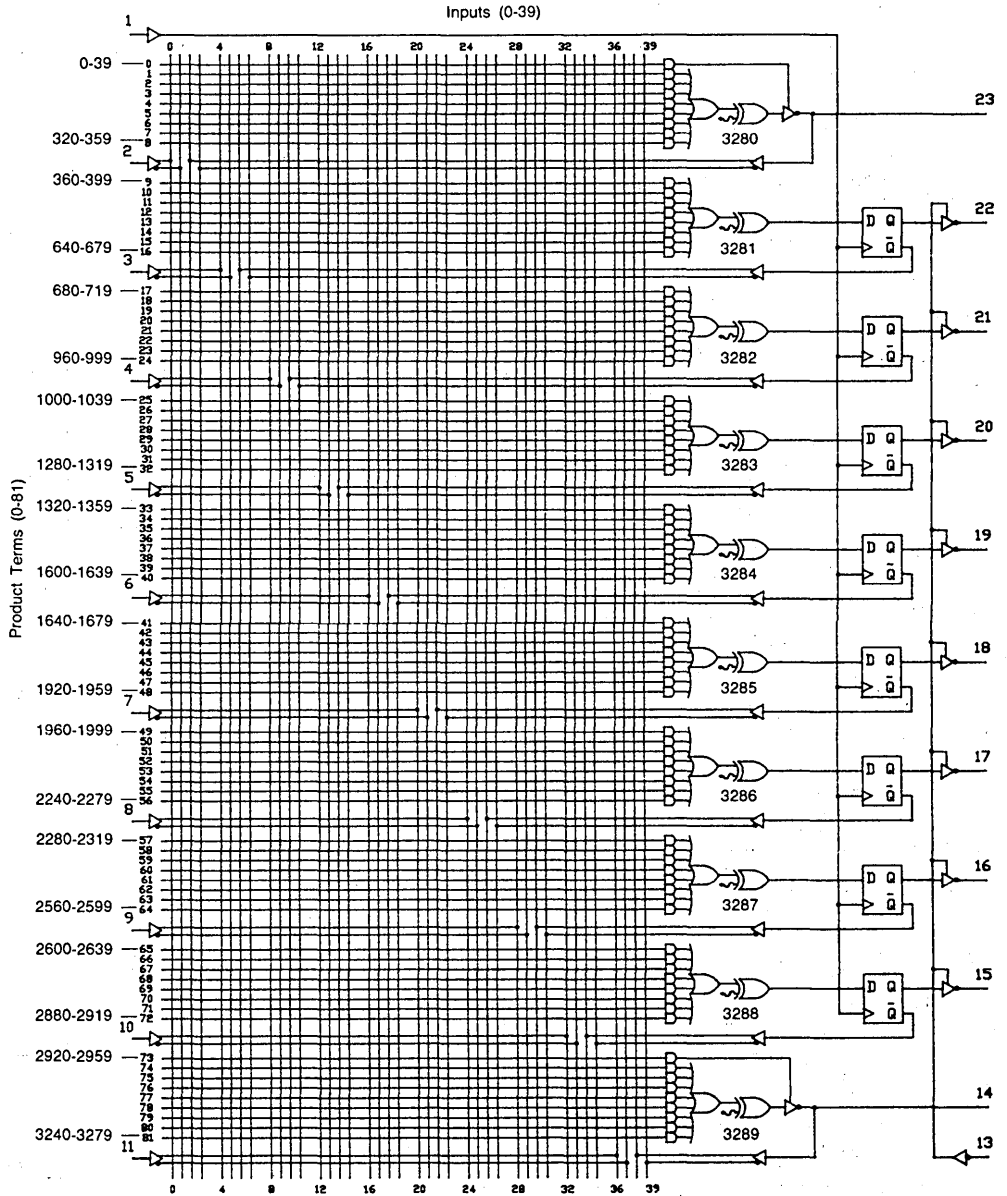
LD001260

Figure 2. AmPAL20RP4 Logic Diagram and JEDEC Fuse Numbering



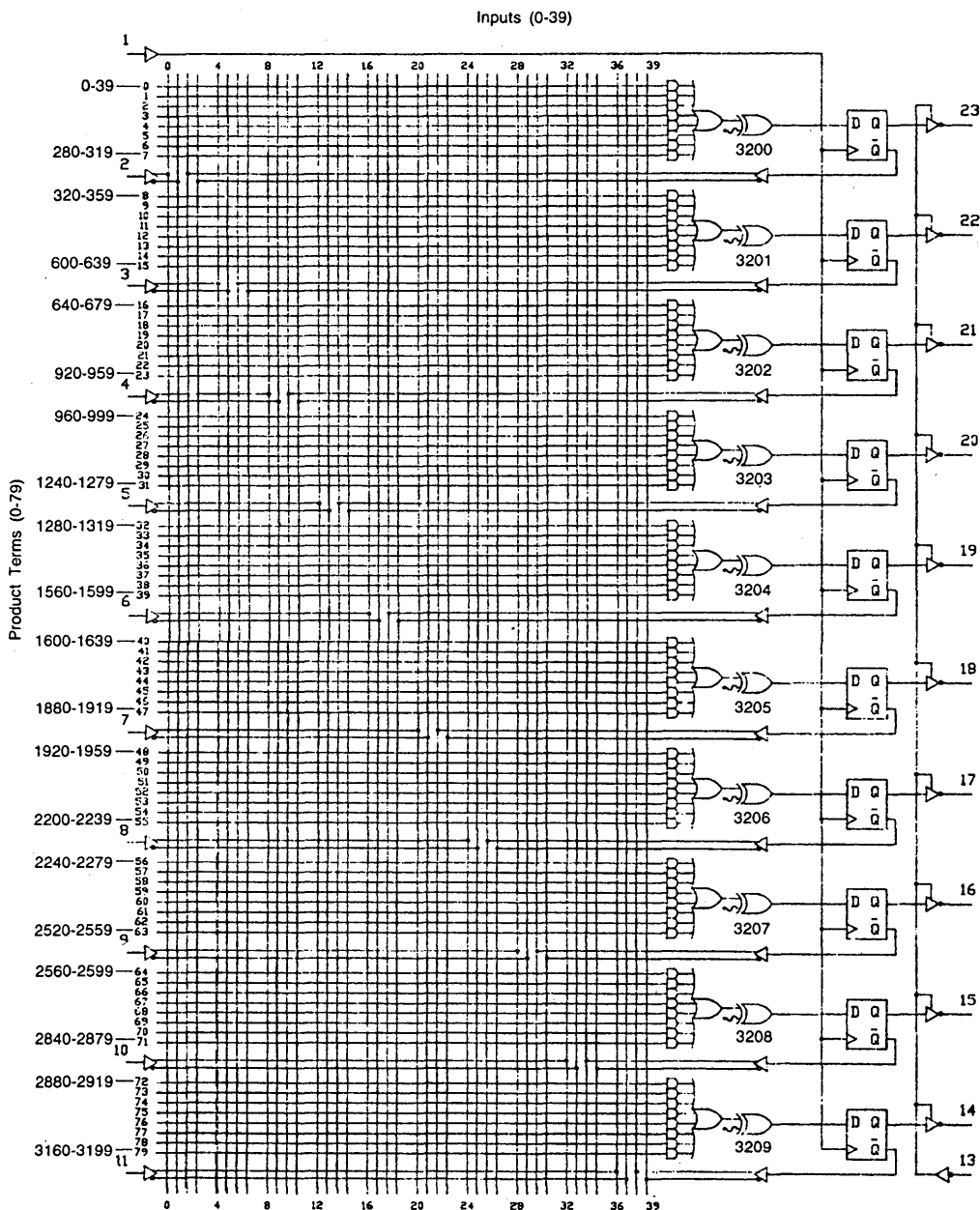
LD001250

Figure 3. AmPAL20RP6 Logic Diagram and JEDEC Fuse Numbering



LD001240

Figure 4. AmPAL20RP8 Logic Diagram and JEDEC Fuse Numbering



LD001280

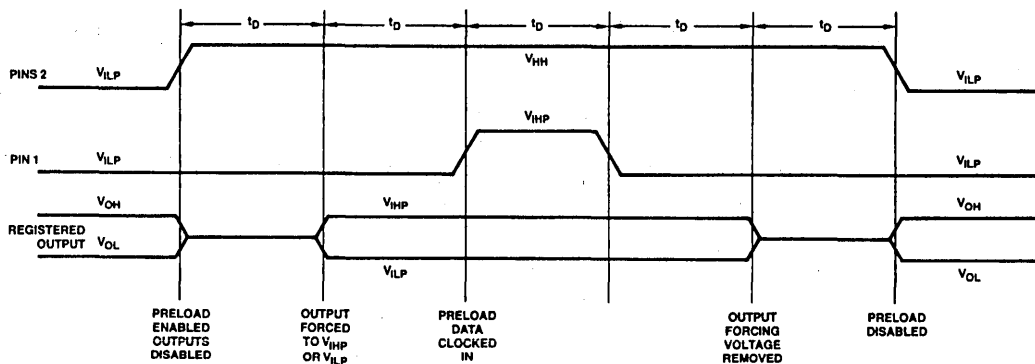
Figure 5. AMPAL20RP10 Logic Diagram and JEDEC Fuse Numbering

PRELOAD of Registered Outputs

The AMD Enhanced 24-pin PAL devices incorporate circuitry to allow loading each register synchronously to either a HIGH

or LOW state. This feature simplifies testing since any initial state for the registers can be set to optimize test sequencing.

The pin levels and timing necessary to perform the PRELOAD function are detailed below:



WF022294

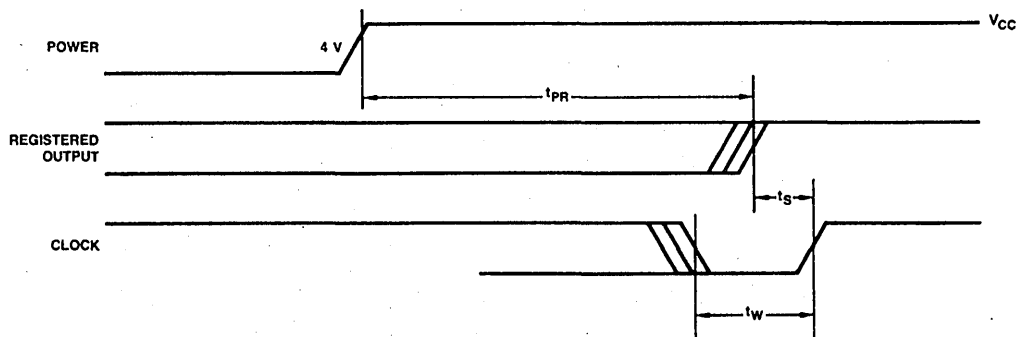
Level forced on registered output pin during PRELOAD cycle	Register Q output state after cycle
V_{IHP}	HIGH
V_{ILP}	LOW

Power-Up Reset

The registered devices in the AMD Enhanced 24-Pin PAL Family have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to LOW. The output state will be HIGH. This feature provides flexibility to the designer and is especially valuable in simplifying state-machine initialization. A timing diagram and parameter table are shown below. Due to the asynchronous

operation of the power-up RESET and the wide range of ways V_{CC} can rise to its steady state, two conditions are required to ensure a valid power-up RESET. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.



WF022300

Parameters	Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up Reset Time		600	1000	ns
t_S	Input or Feedback Setup Time	See Switching Characteristics			
t_W	Clock Width				

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage to Ground Potential (Pin 24 to Pin 12) Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	16 V
Output Current Into Outputs During Programming (Max Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES**Commercial (C) Devices**

Temperature (T _A)	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V

Extended Commercial (E) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Military (M) Devices*

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions	Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -3.2 mA COM'L I _{OH} = -2 mA MIL	2.4	3.5	V
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = -24 mA COM'L I _{OL} = -12 mA MIL		0.5	V
V _{IH} (Note 2)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs		2.0	5.5	V
V _{IL} (Note 2)	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs			0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max., I _{IN} = 0.40 V		-20	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V			25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V			1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 3)	-30	-60	-90	mA
I _{CC}	Power Supply Current	V _{CC} = Max. B, A -AL			210 105	mA
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -1.8 mA		-0.9	-1.2	V
I _{OZH}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.7 V V _O = 0.4 V		100 -100	μA

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5V has been chosen to avoid test problems caused by tester ground degradation.

4. I/O pin leakage is the worst case of I_{OZX} or I_{IX} (where X = H or L).

CAPACITANCE*

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Units
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz	Pins 1, 13 Others	11 6
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz	9	pF

*These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A Subgroup 9, 10, 11 tests unless otherwise noted

COMMERCIAL RANGE

No.	Parameter Symbol	Parameter Description	"B" Version			"A" & "AL" Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 22P10, 20RP4, 20RP6, 20RP8			15			25	ns
2	t_{EA}	Input to Output Enable 22P10, 20RP4, 20RP6, 20RP8			18			25	ns
3	t_{ER}	Input to Output Disable 22P10, 20RP4, 20RP6, 20RP8			15			25	ns
4	t_{PZX}	Pin 13 to Output Enable 20RP4, 20RP6, 20RP8, 20RP10			15			20	ns
5	t_{PXZ}	Pin 13 to Output Disable 20RP4, 20RP6, 20RP8, 20RP10			12			20	ns
6	t_{CO}	Clock to Output 20RP4, 20RP6, 20RP8, 20RP10			12			15	ns
7	t_S	Input or Feedback Setup Time 20RP4, 20RP6, 20RP8, 20RP10		15			25		ns
8	t_H	Hold Time 20RP4, 20RP6, 20RP8, 20RP10		0			0		ns
9	t_P	Clock Period ($t_S + t_{CO}$)		27			40		ns
10	t_{WL}/t_{WH}	Clock Width		10/12			15/15		ns
11	f_{MAX}	Maximum Frequency			37.0			25.0	MHz

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.

2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.

3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

MILITARY RANGE

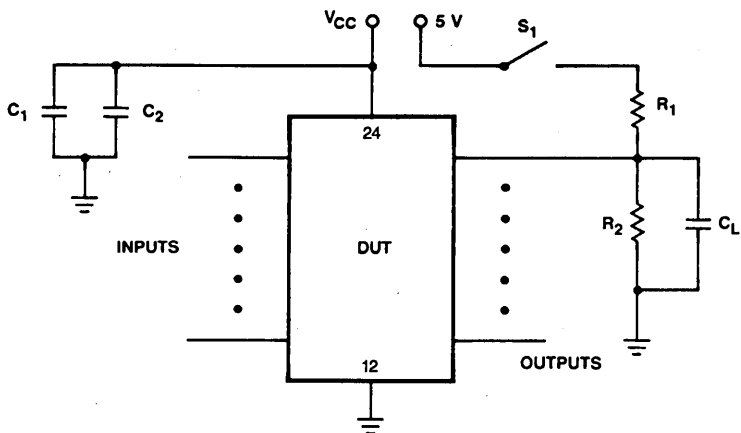
No.	Parameter Symbol	Parameter Description	"B" Version			"A" & "AL" Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t_{PD}	Input or Feedback to Non-Registered Output 22P10, 20RP4, 20RP6, 20RP8			20			30	ns
2	t_{EA}	Input to Output Enable 22P10, 20RP4, 20RP6, 20RP8			25			30	ns
3	t_{ER}	Input to Output Disable 22P10, 20RP4, 20RP6, 20RP8			20			30	ns
4	t_{PZX}	Pin 13 to Output Enable 20RP4, 20RP6, 20RP8, 20RP10			20			25	ns
5	t_{PXZ}	Pin 13 to Output Disable 20RP4, 20RP6, 20RP8, 20RP10			20			25	ns
6	t_{CO}	Clock to Output 20RP4, 20RP6, 20RP8, 20RP10			15			20	ns
7	t_S	Input or Feedback Setup Time 20RP4, 20RP6, 20RP8, 20RP10		20			30		ns
8	t_H	Hold Time 20RP4, 20RP6, 20RP8, 20RP10		0			0		ns
9	t_P	Clock Period ($t_S + t_{CO}$)		35			50		ns
10	t_{WL}/t_{WH}	Clock Width		12/12			20/20		ns
11	f_{MAX}	Maximum Frequency			28.6			20.0	MHz

Notes: 1. Typical limits are at $V_{CC} = 5.0$ V and $T_A = 25^\circ\text{C}$.

2. t_{PD} is tested with switch S_1 closed and $C_L = 50$ pF.

3. For three-state outputs, output enable times are tested with $C_L = 50$ pF to the 1.5 V level; S_1 is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with $C_L = 5$ pF. HIGH to high impedance tests are made to an output voltage of $V_{OH} - 0.5$ V with S_1 open; LOW to high impedance tests are made to the $V_{OL} + 0.5$ V level with S_1 closed.

SWITCHING TEST CIRCUIT

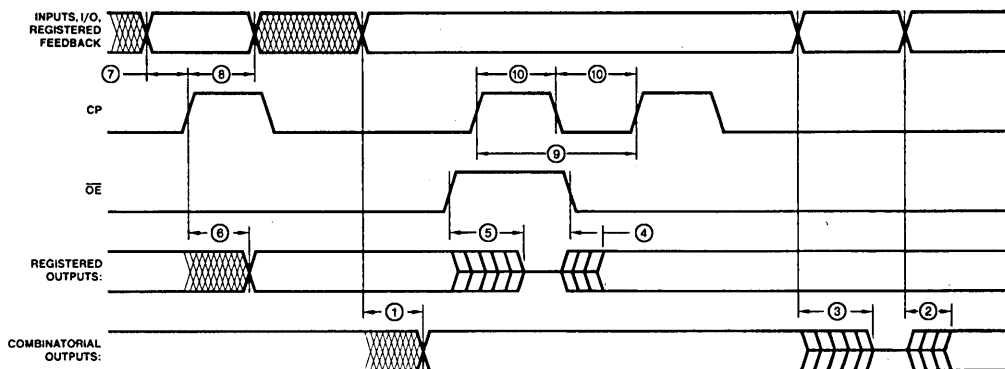


TC003051

Note: C_1 and C_2 are to bypass V_{CC} to ground.

TEST OUTPUT LOADS		
Pin Name	Commercial	Military
R_1	200 Ω	390 Ω
R_2	390 Ω	750 Ω
C_1	1 μF	1 μF
C_2	0.1 μF	0.1 μF
C_L	50 pF	50 pF

SWITCHING WAVEFORMS



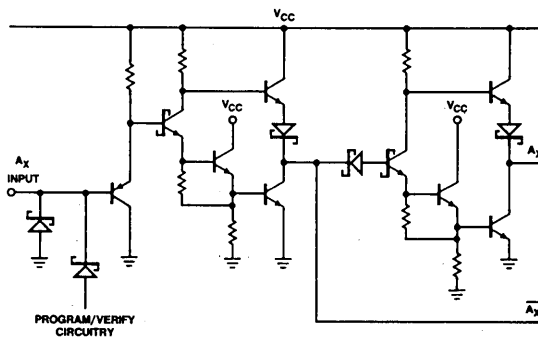
WF002571

KEY TO TIMING DIAGRAM

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

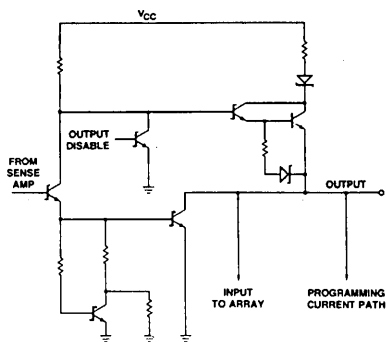
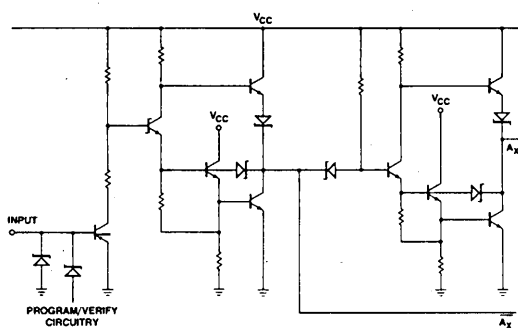
KS000010

INPUT CIRCUITRY



IC000720

OUTPUT CIRCUITRY



IC000801

Programming and Verification

AMD Enhanced 24-Pin PAL Family devices are programmed and verified using AMD's standard programmable logic algorithm. The fuse to be programmed is selected by input line number (array row), product term (array column), and by output (one at a time). The fuse is then programmed and verified by applying a simple sequence of voltages to two control pins (1 and 13).

Input line numbers are addressed using a full decode scheme via TTL levels on pins 6-11 where 6 is the LSB and 11 is the MSB. Even-numbered input lines represent the true version of a signal and odd-numbered lines represent the complement. Input line addressing is shown in Table 1. Note that input line 63 is utilized for selecting the fuses used for programming output polarity.

Product terms are addressed using a 1-of-16 addressing scheme on pins 2-5 where pin 2 is the LSB and 5 is the MSB. Product term addressing is shown in Table 2. Logical and architectural product terms are selected via TTL levels on the four addressing pins.

Fuse selection by output must be done one output at a time (following control pin 1 going to V_{HH}), as shown in the programming timing diagram.

Once fuses have been selected, the simple programming and verification sequence may be completed as shown in the programming timing diagram. AC and DC requirements for programming are shown in the programming parameter table.

Security Fuse Programming

A single fuse is provided on each device to prevent unauthorized copying of PAL fuse patterns. Once blown, the circuitry enabling fuse verification and registered output PRELOAD is permanently disabled.

Programming of the security fuse is the same as an array fuse. Verification of a blown security fuse is accomplished by verifying the whole fuse array as if every fuse is blown.

Programming Yield

AMD PAL devices have been designed to ensure extremely high programming yields (> 98%). To help ensure that a part was correctly programmed, once programming is completed, the entire fuse array should be verified at both LOW and HIGH V_{CC} . Reverification can be accomplished by reading all ten outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PAL devices contain many internal test features, including circuitry and extra fuses which allow AMD to test the ability of each part to perform programming before shipping, to assure high programming yields and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS $T_A = 25^\circ\text{C}$

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V _{HH}	Control Pin Extra High Level	Pin 1 @ 5-10 mA	10	11	12	V
		Pin 13 @ 5-10 mA	10	11	12	
V _{OP}	Program Voltage Pins 14-23 @ 15-200 mA		14	15	16	V
V _{IHP}	Input HIGH Level During Programming and Verify		2.4	5	5.5	V
V _{ILP}	Input LOW Level During Programming and Verify		0.0	0.3	0.5	V
V _{CCP}	V _{CC} During Programming @ I _{CC} = 50-275 mA		5	5.2	5.5	V
V _{CCL}	V _{CC} During First Pass Verification @ I _{CC} = 50-275 mA		4.4	4.5	4.6	V
V _{CCH}	V _{CC} During Second Pass Verification @ I _{CC} = 50-275 mA		5.4	5.5	5.6	V
V _{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	V
V _{OP} /dt	Rate of Output Voltage Change		20		250	V/μs
dV _{I3} /dt	Rate of Fusing Enable Voltage Change (Pin 13 Rising Edge)		100		1000	V/μs
t _p	Fusing Time First Attempt		40	50	100	μs
	Subsequent Attempts		4	5	10	ms
t _D	Delays Between Various Level Changes		100	200	1000	ns
t _V	Period During which Output is Sensed for V _{Blown} Level				500	ns
V _{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		V _{CCP} - 0.3	V _{CCP}	V _{CCP} + 0.3	V
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	kΩ

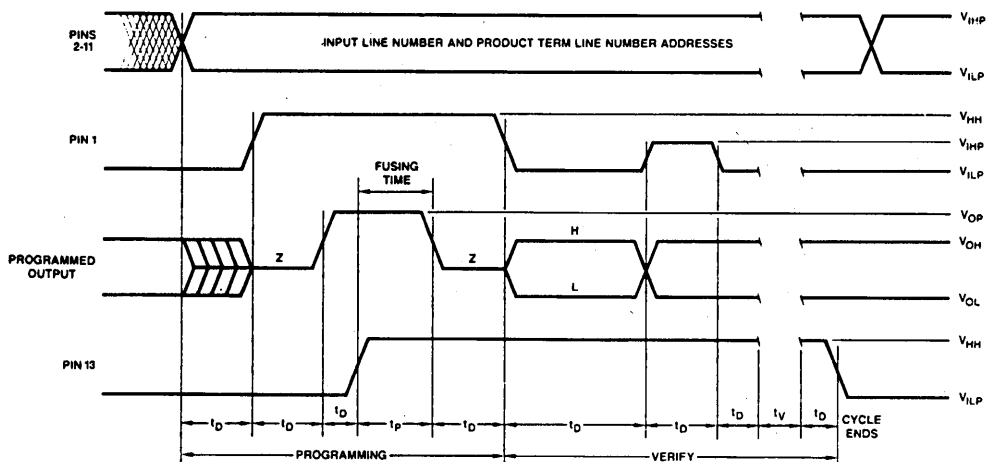
Design Aid Software for AMD Enhanced 24-Pin PAL Family

Name	Vendor	Versions	Notes
ABEL	Data I/O (206) 881-6444	IBM PC VAX/VMS VAX/UNIX	
CUPL	P-CAD Systems (408) 971-1300	IBM PC VAX/VMS VAX/UNIX CPM 80/86	
AmCUPL	Advanced Micro Devices (408) 732-2400	IBM PC	Supported by P-CAD Systems

AMD Qualified Programmers

Name	Programmer Model(s)	AMD PAL Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 93052	Systems 19, 29	950-1942-0044	303A-011A
	60	N/A	Under Development
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086	Model PPZ	Under Development	On Board
	ZL30	Under Development	
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043	160 Series	Under Development	On Board

PROGRAMMING TIMING DIAGRAM



PF001101

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States					
	6	7	8	9	10	11
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	H
3	L	L	L	L	H	L
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	H
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	H
12	L	L	H	H	L	L
13	L	L	H	H	L	H
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	H
18	L	H	L	L	H	L
19	L	H	L	L	H	H
20	L	H	L	H	L	L
21	L	H	L	H	L	H
22	L	H	L	H	H	L
23	L	H	L	H	H	H
24	L	H	H	L	L	L
25	L	H	H	L	L	H
26	L	H	H	L	H	L
27	L	H	H	L	H	H
28	L	H	H	H	L	L
29	L	H	H	H	L	H
30	L	H	H	H	H	L
31	L	H	H	H	H	H
32	H	L	L	L	L	L
33	H	L	L	L	L	H
34	H	L	L	L	H	L
35	H	L	L	L	H	H
36	H	L	L	H	L	L
37	H	L	L	H	L	H
38	H	L	L	H	H	L
39	H	L	L	H	H	H
40*	H	L	H	L	L	L
41*	H	L	H	L	L	H
42*	H	L	H	L	H	L
43*	H	L	H	L	H	H
63**	H	H	H	H	H	H

* Used for AmPAL22P10 only

** Used for programming polarity

L = V_{ILP}H = V_{IHP}

TABLE 2-1. PRODUCT TERM ADDRESSING (AmPAL22P10)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	27	36	45	54	63	72	81
L	L	L	H	1	10	19	28	37	46	55	64	73	82
L	L	H	L	2	11	20	29	38	47	56	65	74	83
L	L	H	H	3	12	21	30	39	48	57	66	75	84
L	H	L	L	4	13	22	31	40	49	58	67	76	85
L	H	L	H	5	14	23	32	41	50	59	68	77	86
L	H	H	L	6	15	24	33	42	51	60	69	78	87
L	H	H	H	7	16	25	34	43	52	61	70	79	88
H	L	L	L	8	17	26	35	44	53	62	71	80	89
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	L	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-2. PRODUCT TERM ADDRESSING (AmPAL20RP4)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	27	35	43	51	59	68	77
L	L	L	H	1	10	19	28	36	44	52	60	69	78
L	L	H	L	2	11	20	29	37	45	53	61	70	79
L	L	H	H	3	12	21	30	38	46	54	62	71	80
L	H	L	L	4	13	22	31	39	47	55	63	72	81
L	H	L	H	5	14	23	32	40	48	56	64	73	82
L	H	H	L	6	15	24	33	41	49	57	65	74	83
L	H	H	H	7	16	25	34	42	50	58	66	75	84
H	L	L	L	8	17	26	-	-	-	-	67	76	85
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-3. PRODUCT TERM ADDRESSING (AmPAL20RP6)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	26	34	42	50	58	66	75
L	L	L	H	1	10	19	27	35	43	51	59	67	76
L	L	H	L	2	11	20	28	36	44	52	60	68	77
L	L	H	H	3	12	21	29	37	45	53	61	69	78
L	H	L	L	4	13	22	30	38	46	54	62	70	79
L	H	L	H	5	14	23	31	39	47	55	63	71	80
L	H	H	L	6	15	24	32	40	48	56	64	72	81
L	H	H	H	7	16	25	33	41	49	57	65	73	82
H	L	L	L	8	17	—	—	—	—	—	—	74	83
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = V_{ILP}
 H = V_{IHP}
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-4. PRODUCT TERM ADDRESSING (AmPAL20RP8)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	17	25	33	41	49	57	65	73
L	L	L	H	1	10	18	26	34	42	50	58	66	74
L	L	H	L	2	11	19	27	35	43	51	59	67	75
L	L	H	H	3	12	20	28	36	44	52	60	68	76
L	H	L	L	4	13	21	29	37	45	53	61	69	77
L	H	L	H	5	14	22	30	38	46	54	62	70	78
L	H	H	L	6	15	23	31	39	47	55	63	71	79
L	H	H	H	7	16	24	32	40	48	56	64	72	80
H	L	L	L	8	—	—	—	—	—	—	—	—	81
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = V_{ILP}
 H = V_{IHP}
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-5. PRODUCT TERM ADDRESSING (AmPAL20RP10)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	8	16	24	32	40	48	56	64	72
L	L	L	H	1	9	17	25	33	41	49	57	65	73
L	L	H	L	2	10	18	26	34	42	50	58	66	74
L	L	H	H	3	11	19	27	35	43	51	59	67	75
L	H	L	L	4	12	20	28	36	44	52	60	68	76
L	H	L	H	5	13	21	29	37	45	53	61	69	77
L	H	H	L	6	14	22	30	38	46	54	62	70	78
L	H	H	H	7	15	23	31	39	47	55	63	71	79
H	L	L	L	-	-	-	-	-	-	-	-	-	-
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = V_{ILP}
 H = V_{IHP}
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

AMD 24-Pin XOR PAL* Family

24-Pin IMOX™ Programmable Array Logic (PAL) Elements

PRELIMINARY

AMD 24-Pin XOR PAL* Family

DISTINCTIVE CHARACTERISTICS

- AND-OR-XOR logic structure
- AMD's superior IMOX technology
 - Guarantees $t_{PD} = 20$ ns max
- Individually programmable output polarity on each output
- Eight logical product terms per output
- Programming yields > 98% are realized via platinum-silicide fuse technology and the use of added test words
- Post Programming Functional Yield (PPFY) of 99.9%
- PRELOAD feature permits full logical verification
- Reliability assured through more than 70 billion fuse hours of life testing with no failures
- Full AC and DC parametric testing at the factory through on-board testing circuitry
- > 3000V ESD input protection per pin
- JEDEC-Standard LCC and PLCC pinout

GENERAL DESCRIPTION

AMD 24-pin XOR PAL devices are high-speed, electrically programmable array logic elements. They utilize the familiar sum-of-products (AND-OR-XOR) structure allowing users to program custom logic functions to fit most applications precisely. Typically they are a replacement for low-power Schottky SSI/MSI logic circuits that require an exclusive-OR function, reducing chip count by more than 5 to 1 and greatly simplifying prototyping and board layout.

Five different devices are available, including both registered and combinatorial devices. All devices have user-programmable output polarity on all outputs. A variety of speed options allow the designer maximum flexibility in matching precise system requirements. The Product Selector Guide below shows the available speed options. The second table gives details about the functionality of the five available devices.

Please see the following pages for Block Diagrams.

PRODUCT SELECTOR GUIDE

AMD PAL Speed/Power Families

Family	t_{PD} ns (Max.)		t_S ns (Min.)		t_{CO} ns (Max.)		I_{CC} mA (Max.)	I_{OL} mA (Min.)	
	C Devices	M Devices	C Devices	M Devices	C Devices	M Devices	C/M Devices	C Devices	M Devices
Very High-Speed (-20 & -25) Versions	20	25	20	25	13	15	210	24	12
High-Speed (-30 & -35) Versions	30	35	30	35	15	25	180	24	12
High-Speed, Half-Power (-30L & -35L) Versions	30	35	30	35	15	25	90	24	12
Standard (-40 & -45) Versions	40	45	40	45	30	35	180	24	12
Half-Power (-40L & -45L) Versions	40	45	40	45	30	35	90	24	12

Part Number	Array Inputs	Logic	Output Enable	Outputs/Polarity	Package Pins
22XP10	12 Dedicated, 10 Bidirectional	Ten (2-6)-Wide AND-OR-XOR	Programmable	Bidirectional/Programmable	24
20XRP4	10 Dedicated, 4 Feedback, 6 Bidirectional	Four (2-6)-Wide AND-OR-XOR	Dedicated	Registered/Programmable	24
		Six 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20XRP6	10 Dedicated, 6 Feedback, 4 Bidirectional	Six (2-6)-Wide AND-OR-XOR	Dedicated	Registered/Programmable	24
		Four 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20XRP8	10 Dedicated, 8 Feedback, 2 Bidirectional	Eight (2-6)-Wide AND-OR-XOR	Dedicated	Registered/Programmable	24
		Two 8-Wide AND-OR	Programmable	Bidirectional/Programmable	
20XRP10	10 Dedicated, 10 Feedback	Ten (2-6)-Wide AND-OR-XOR	Dedicated	Registered/Programmable	24

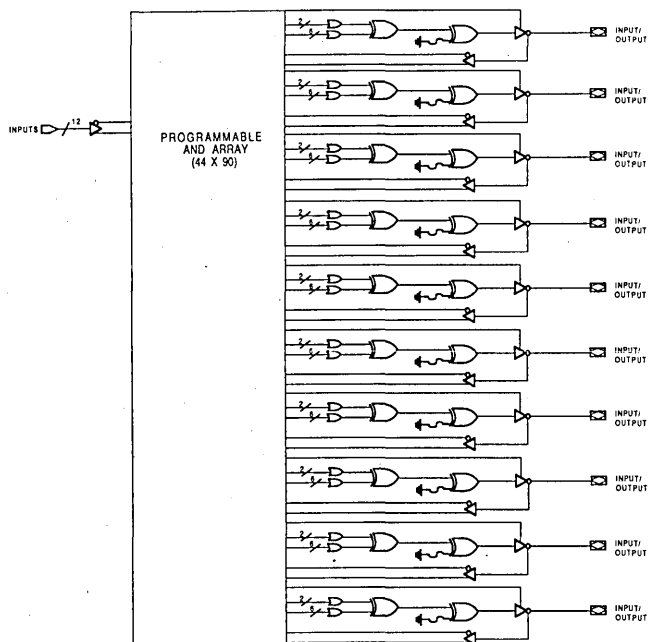
IMOX is a trademark of Advanced Micro Devices, Inc.

*PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

Publication # 08655 Rev. A Amendment /0
Issue Date: October 1986

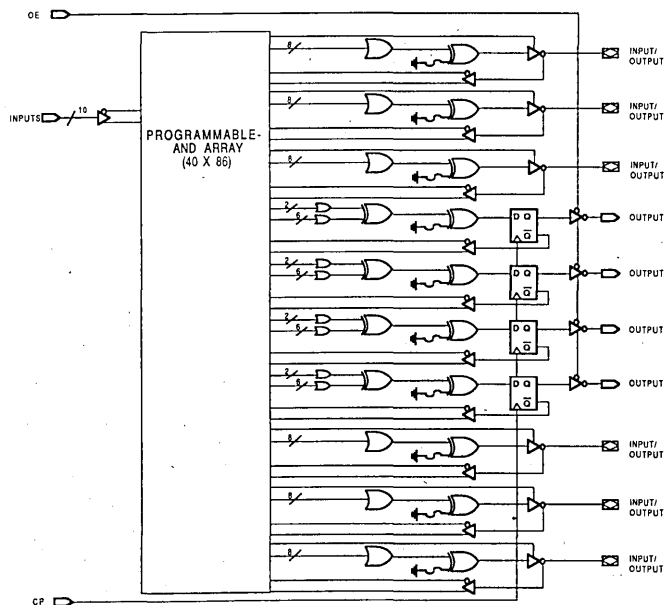
BLOCK DIAGRAMS

AmPAL22XP10



LD000900

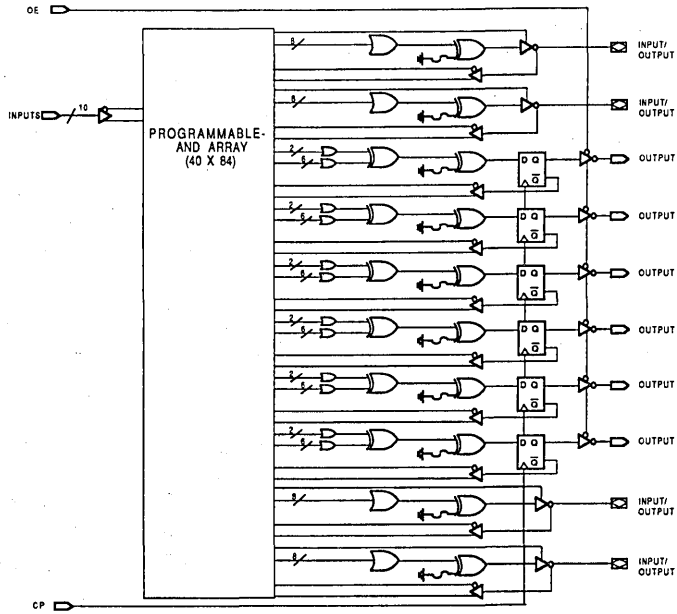
AmPAL20XRP4



LD000940

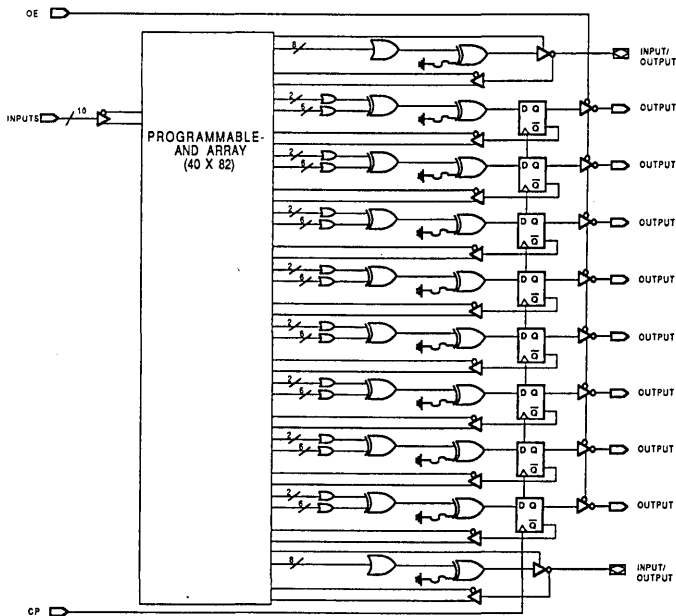
BLOCK DIAGRAMS (Cont'd.)

AmPAL20XRP6



LD000920

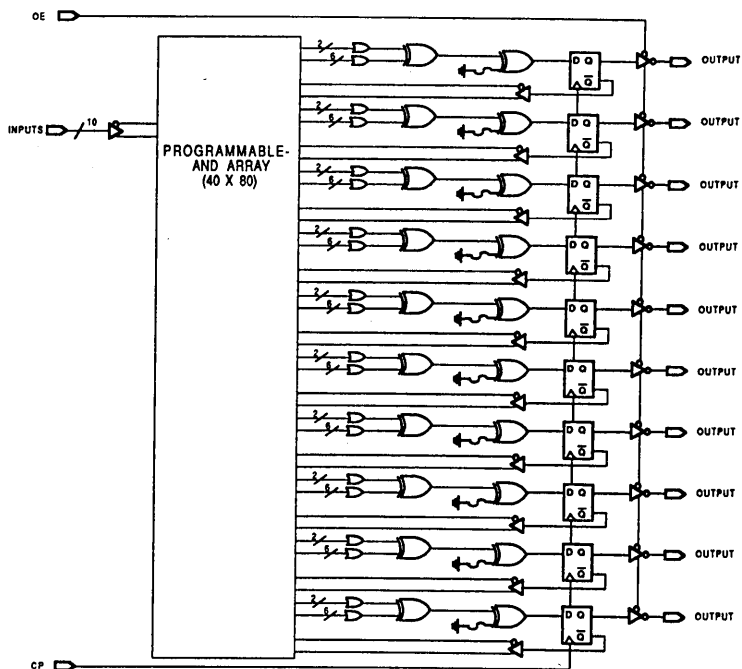
AmPAL20XRP8



LD000930

BLOCK DIAGRAMS (Cont'd.)

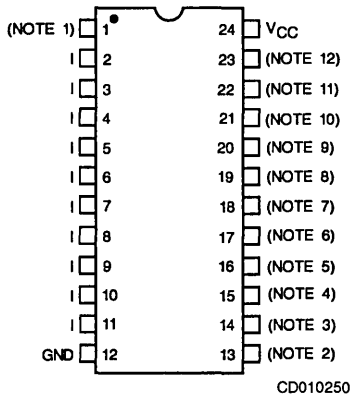
AmPAL20XRP10



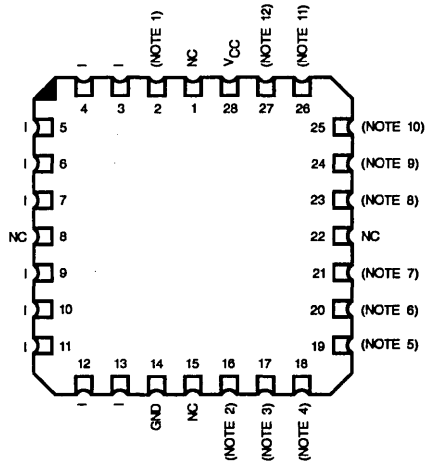
LD000910

CONNECTION DIAGRAMS Top View

DIPs*



LCC**



Note: Pin 1 is marked for orientation.

Notes:

	22XP10	20XRP4	20XRP6	20XRP8	20XRP10
1	I	CLK	CLK	CLK	CLK
2	I	OE	OE	OE	OE
3	I/O	I/O	I/O	I/O	O
4	I/O	I/O	I/O	O	O
5	I/O	I/O	O	O	O
6	I/O	O	O	O	O
7	I/O	O	O	O	O
8	I/O	O	O	O	O
9	I/O	O	O	O	O
10	I/O	I/O	O	O	O
11	I/O	I/O	I/O	O	O
12	I/O	I/O	I/O	I/O	O

*Also available in 24-Pin Ceramic Flatpack. Pinouts identical to DIPs.

**Also available in 28-Pin Plastic Leaded Chip Carrier. Pinouts identical to LCC.

PIN DESIGNATIONS

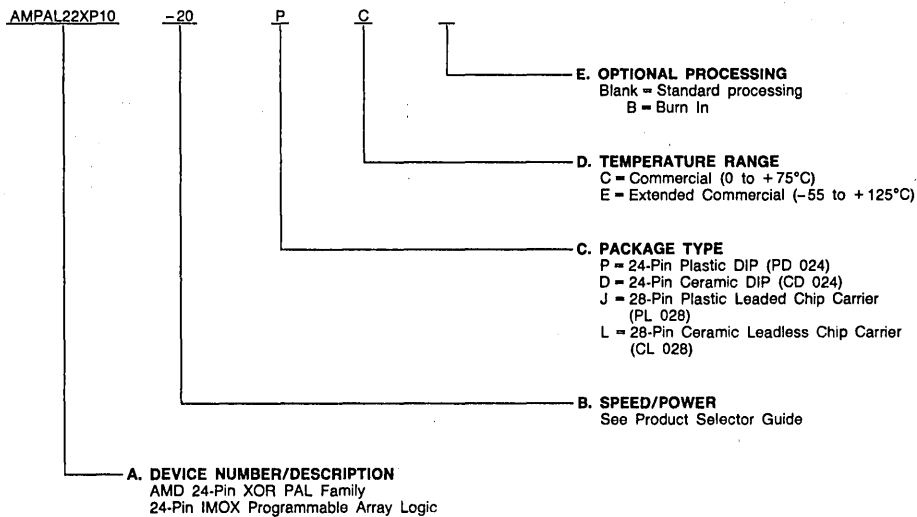
I = Input
I/O = Input/Output
O = Output
VCC = Supply Voltage
GND = Ground
CLK = Clock
OE = Output Enable
NC = No Connect

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

Valid Combinations	
AMPAL22XP10-20/-30/-30L/-40/-40L	PC, DC, DCB, DE, JC, LC, LE
AMPAL20XRP4-20/-30/-30L/-40/-40L	
AMPAL20XRP6-20/-30/-30L/-40/-40L	
AMPAL20XRP8-20/-30/-30L/-40/-40L	
AMPAL20XRP10-20/-30/-30L/-40/-40L	

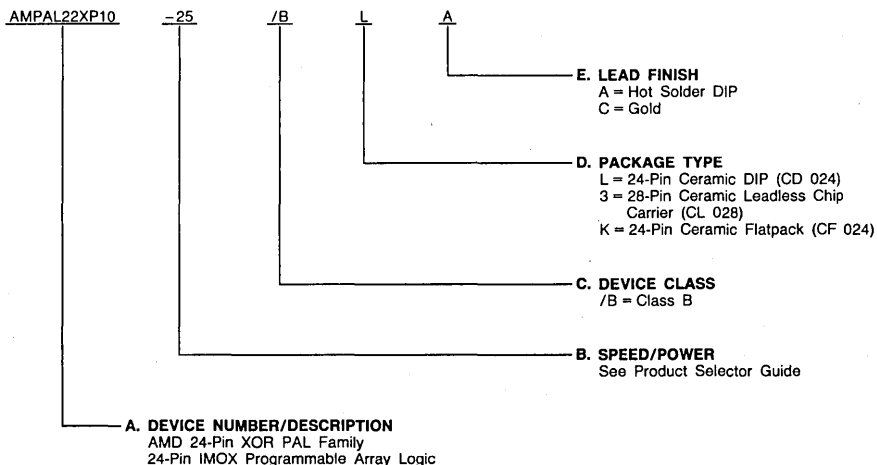
Valid Combinations

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, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)**APL 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) for APL products is formed by a combination of:

- A. Device Number**
B. Speed Option (if applicable)
C. Device Class
D. Package Type
E. Lead Finish

**Valid Combinations**

AMPAL22XP10-25/-35/-35L/-45/-45L	/BLA, /B3C, /BKA
AMPAL20XRP4-25/-35/-35L/-45/-45L	
AMPAL20XRP6-25/-35/-35L/-45/-45L	
AMPAL20XRP8-25/-35/-35L/-45/-45L	
AMPAL20XRP10-25/-35/-35L/-45/-45L	

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 consist of Subgroups
 1, 2, 3, 7, 8, 9, 10, & 11

FUNCTIONAL DESCRIPTION

AMD 24-Pin XOR PAL Family Characteristics

All members of the AMD 24-Pin XOR PAL Family have common electrical characteristics and programming procedures. All parts are produced with a fusible link at each input to the AND gate array, and connections may be selectively removed by applying appropriate voltages to the circuit.

Initially the AND gates are connected, via fuses, to both the true and complement of each input. By selective programming of fuses the AND gates may be "connected" to only the true input (by blowing the complement fuse), to only the complement input (by blowing the true fuse), or to neither type of input (by blowing both fuses) establishing a logical "don't care." When both the true and complement fuses are left intact a logical false results on the output of the AND gate, while all fuses blown results in a logical true state. On the AmPAL22XP10 device, the AND gates are connected to fixed (2-6) OR-XOR structures whose outputs become device outputs. The remaining four (registered) devices function as follows: for combinatorial outputs, the AND gates are connected to fixed-OR gates whose outputs become device outputs. For registered outputs, the AND gates are connected to fixed (2-6) OR-XOR structures whose outputs become output register inputs.

All parts are fabricated with AMD's fast programming, highly reliable Platinum-Silicide Fuse technology. Utilizing an easily implemented programming algorithm, these products can be rapidly programmed to any customized pattern. Extra test words are pre-programmed during manufacturing to insure extremely high field programming yields (> 98%), and provide extra test paths to achieve excellent parametric correlation.

Power-Up Reset

The registered devices in the AMD PAL family have been designed to reset during system power-up. Following power-up, all registers will be initialized to zero, setting all the outputs to a logic 1. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

PRELOAD

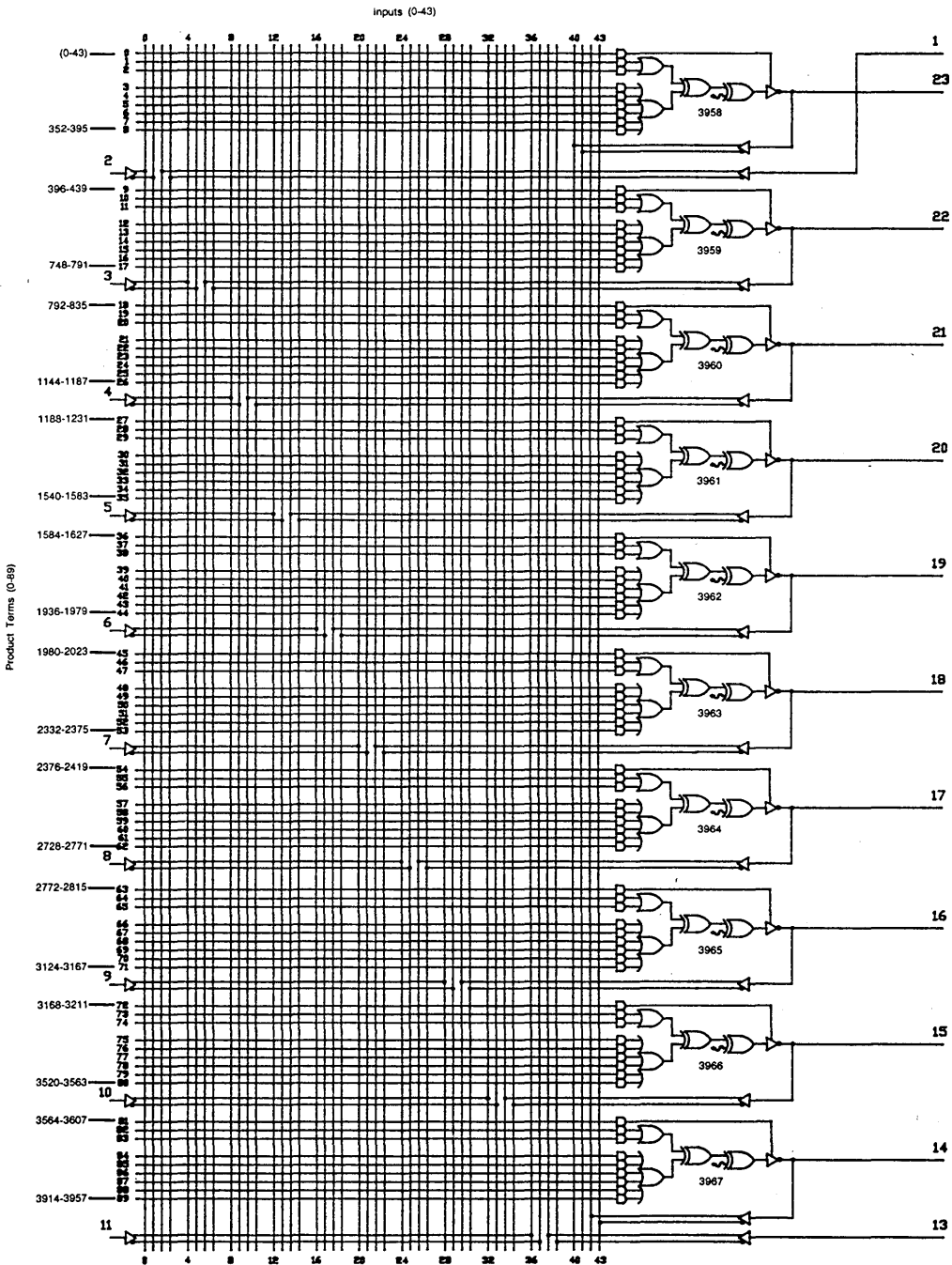
AMD PAL devices are designed with unique PRELOAD circuitry that provides an easy method of testing registered devices for logical functionality. PRELOAD allows any arbitrary state value to be loaded into the registered output of an AMD PAL device.

A typical functional test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the device inputs to any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state or "next state." The next state is then checked to validate the transition from the present state. In this way any state transition can be checked.

Without PRELOAD, it is difficult and in some cases impossible to load an arbitrary present state value. This can lead to logic verification sequences that are either incomplete or excessively long. Long test sequences result when the feedback from the state register "interferes" with the inputs, forcing the machine to go through many transitions before it can reach an arbitrary state value. Therefore the test sequence will be mostly state initialization and not actual testing. The test sequence becomes excessively long when a state must be reentered many times to test a wide variety of input combinations.

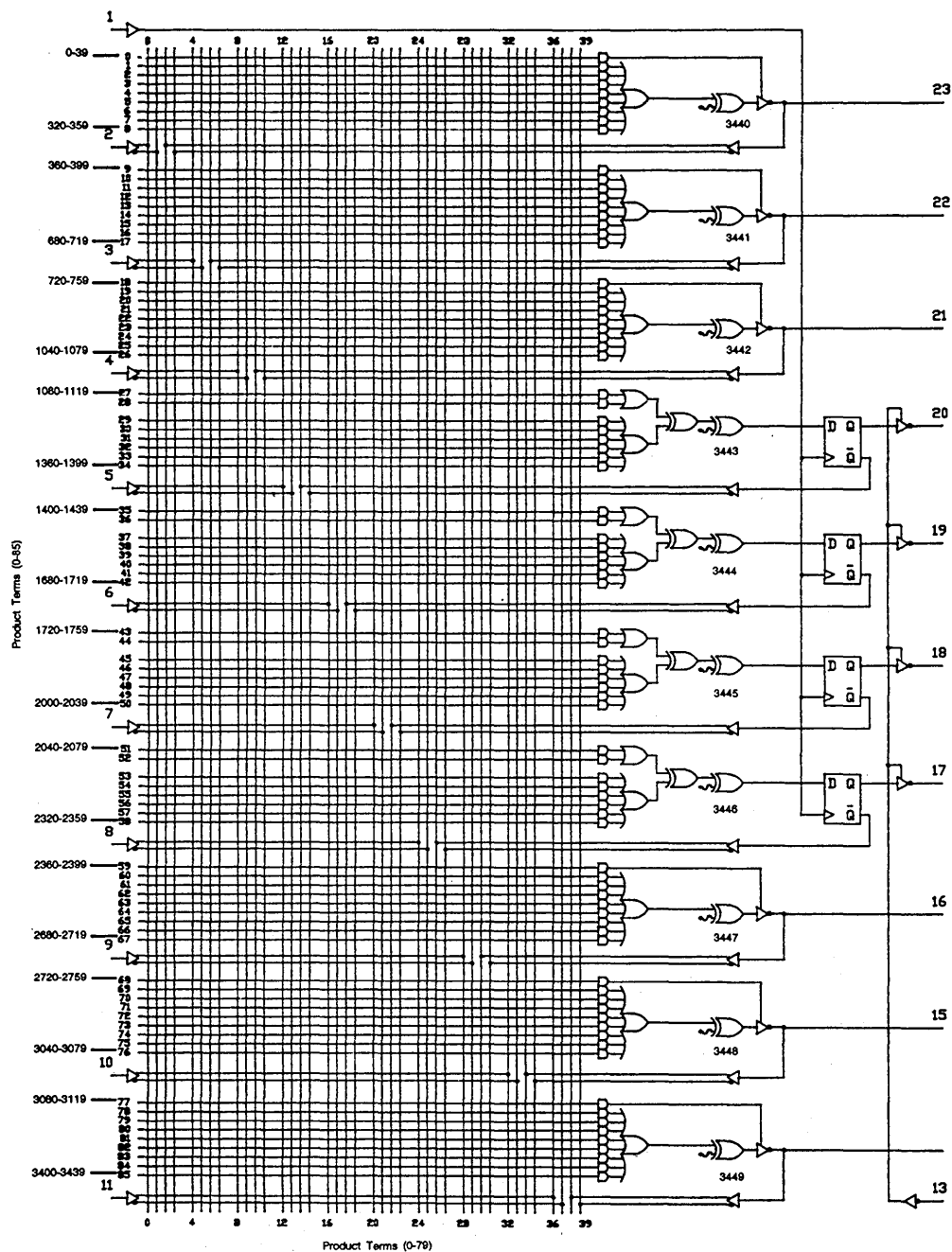
In addition, complete logic verification may become impossible when states that need to be tested cannot be entered with normal state transitions. For example, even though necessary, the state entered when a machine powers up cannot be tested, because it cannot be entered from the main sequence. Similarly, "forbidden" or don't care states that are not normally entered need to be tested to ensure that they return to the main sequence.

PRELOAD eliminates these problems by providing the capability to go directly to any desired arbitrary state. Thus test sequences may be greatly shortened, and all possible states can be tested, greatly reducing test time and development costs, and guaranteeing proper in-system operation.



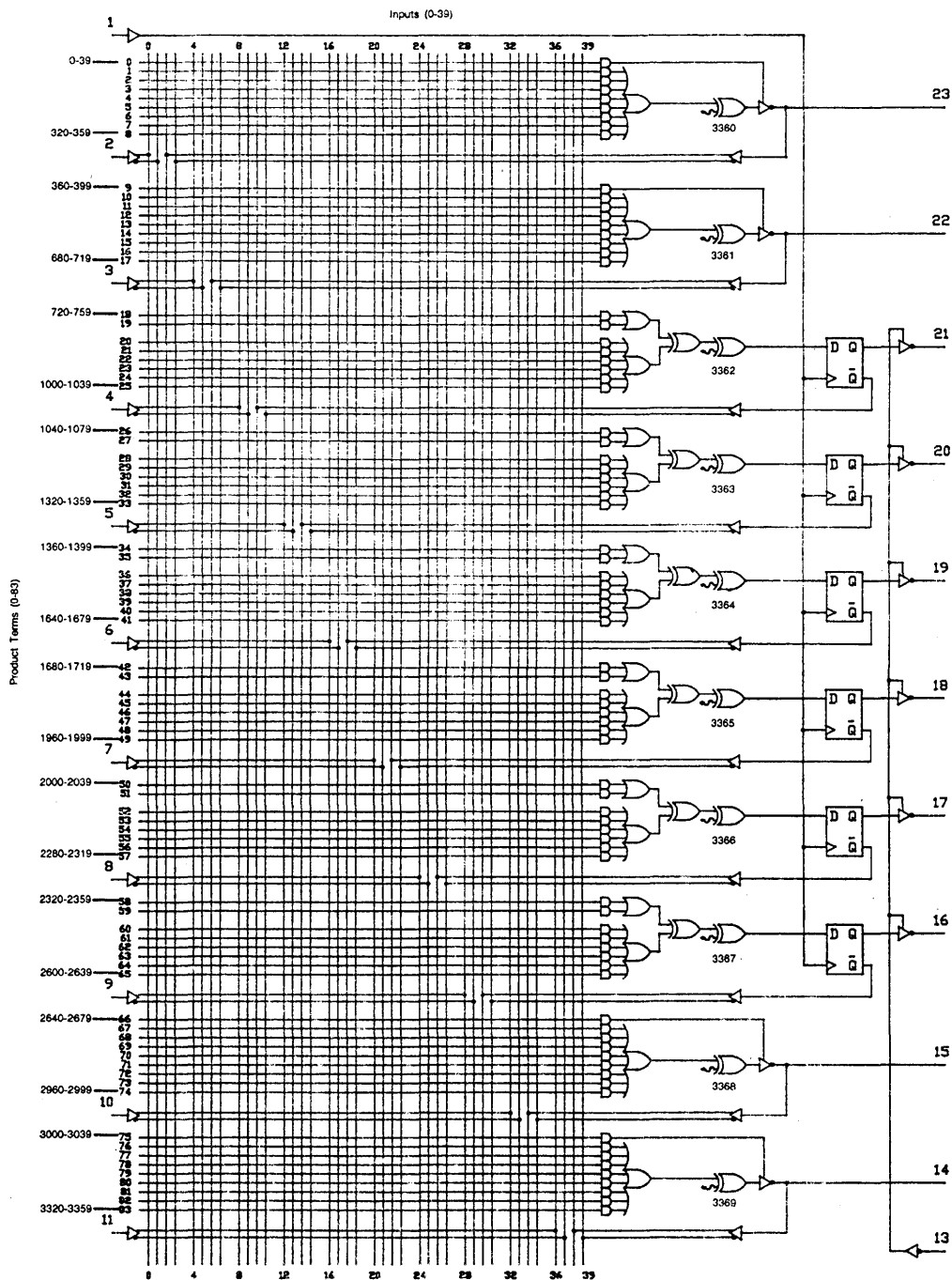
LD000870

Figure 1. AmPAL22XP10 Logic Diagram and JEDEC Fuse Numbering



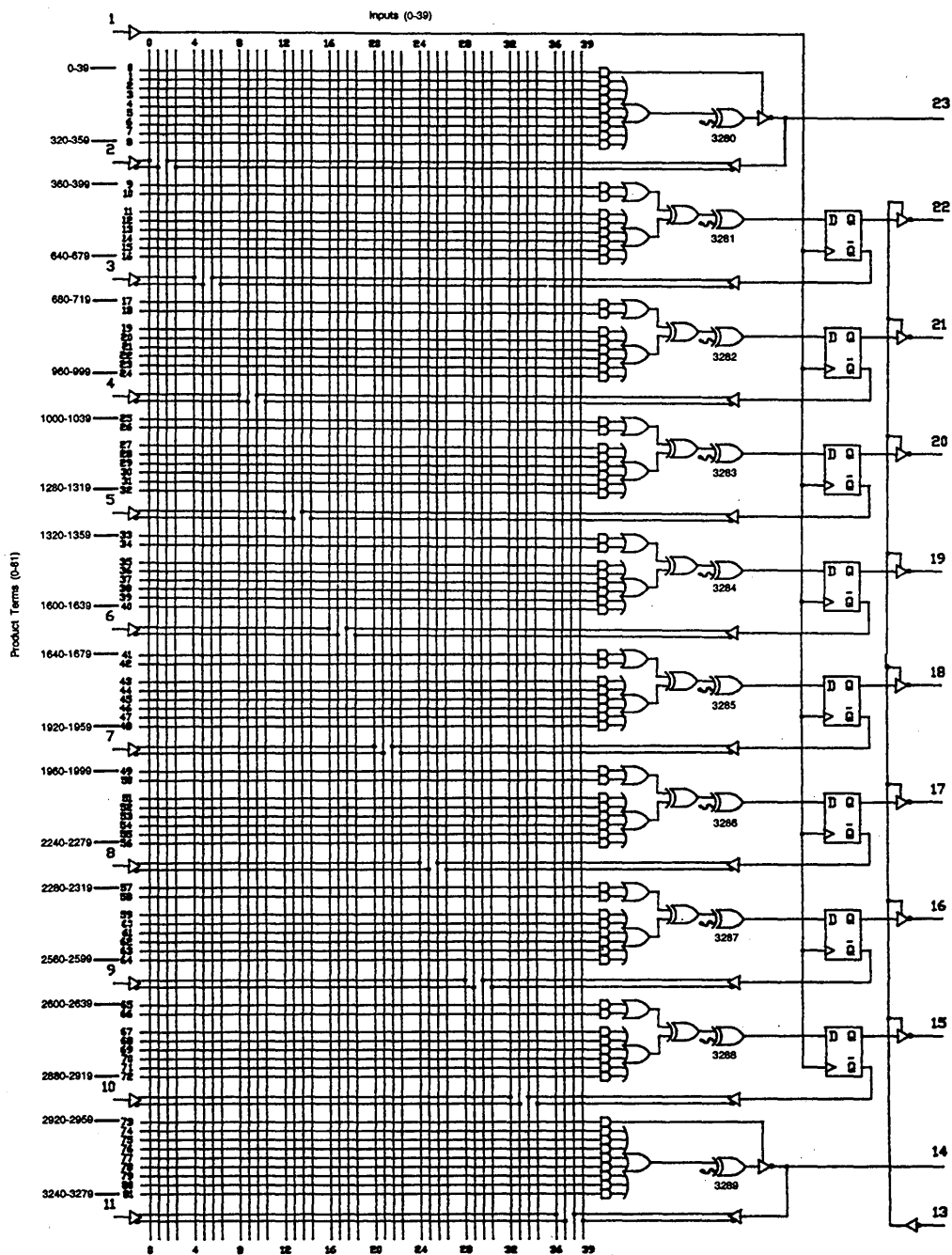
LD000860

Figure 2. AmpAL20XRP4 Logic Diagram and JEDEC Fuse Numbering



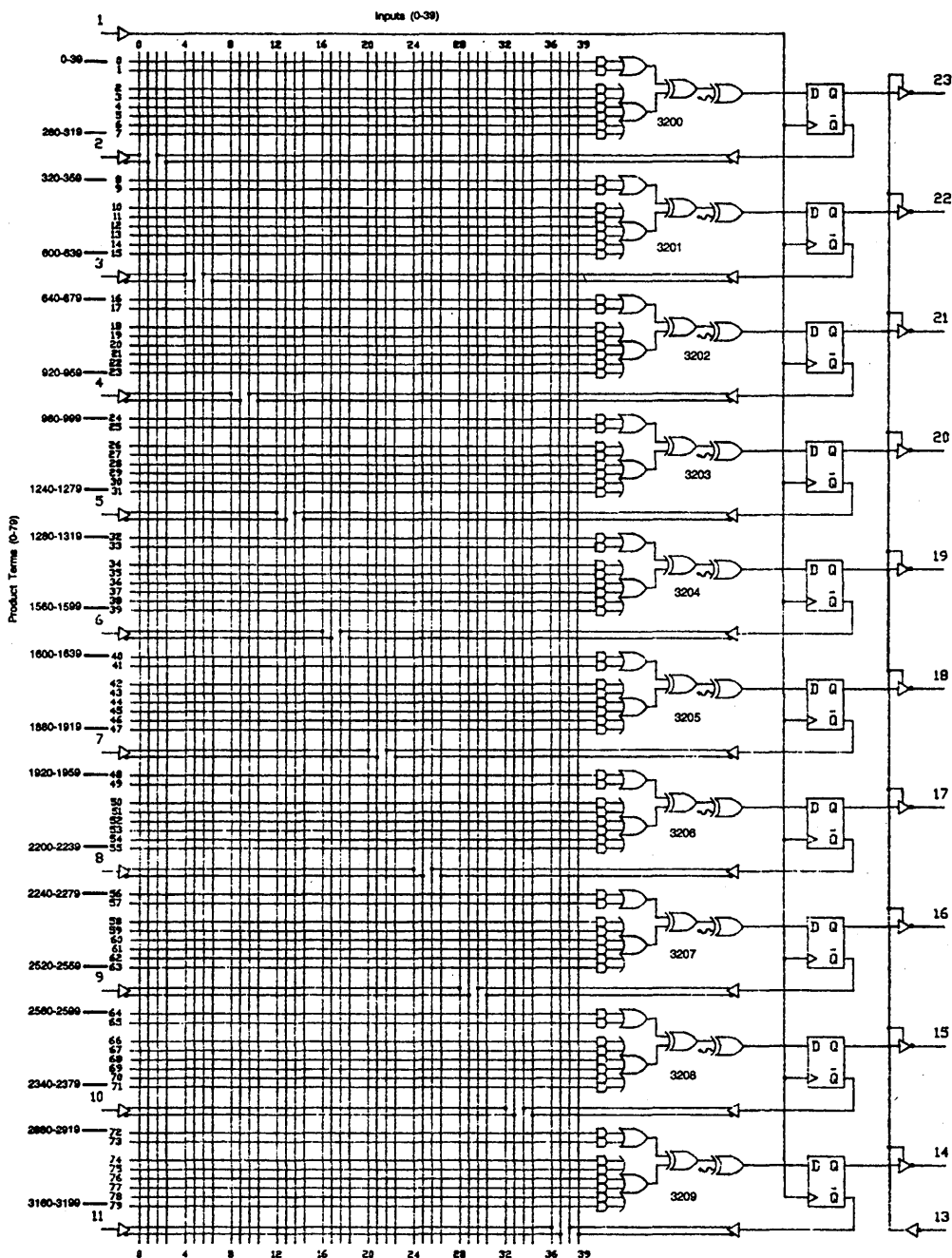
LD000890

Figure 3. AmpAL20XRP6 Logic Diagram and JEDEC Fuse Numbering



LD000850

Figure 4. AmPAL20XRP8 Logic Diagram and JEDEC Fuse Numbering



LD000880

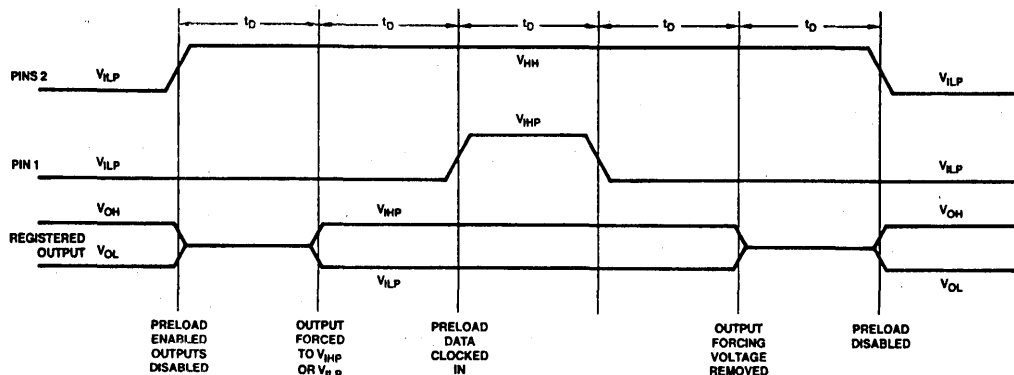
Figure 5. AmpPAL20XRP10 Logic Diagram and JEDEC Fuse Numbering

PRELOAD of Registered Outputs

The AMD 24-pin XOR PAL devices incorporate circuitry to allow loading each register synchronously to either a HIGH or

LOW state. This feature simplifies testing since any initial state for the registers can be set to optimize test sequencing.

The pin levels and timing necessary to perform the PRELOAD function are detailed below:



WF022294

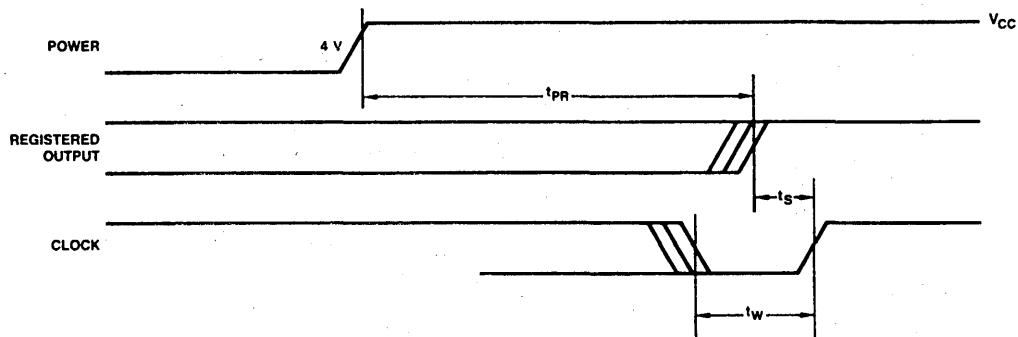
Level forced on registered output pin during PRELOAD cycle	Register Q output state after cycle
V_{IHP}	HIGH
V_{ILP}	LOW

Power-Up Reset

The registered devices in the AMD 24-Pin XOR PAL Family have been designed with the capability to reset during system power-up. Following power-up, all registers will be reset to LOW. The output state will be HIGH. This feature provides flexibility to the designer and is especially valuable in simplifying state-machine initialization. A timing diagram and parameter table are shown below. Due to the asynchronous operation

of the power-up RESET and the wide range of ways V_{CC} can rise to its steady state, two conditions are required to ensure a valid power-up RESET. These conditions are:

1. The V_{CC} rise must be monotonic.
2. Following reset, the clock input must not be driven from LOW to HIGH until all applicable input and feedback setup times are met.



WF022300

Parameters	Description	Min.	Typ.	Max.	Units
t_{PR}	Power-Up Reset Time		600	1000	ns
t_s	Input or Feedback Setup Time	See Switching Characteristics			
t_w	Clock Width				

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Supply Voltage to Ground Potential (Pin 24 to Pin 12) Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	16 V
Output Current Into Outputs During Programming (Max Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 to +5.5 V
DC Input Current	-30 to +5 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES**Commercial (C) Devices**

Temperature (T _A)	0 to +75°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V

Extended Commercial (E) Devices

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Military (M) Devices*

Temperature (T _A)	-55°C Min.
Temperature (T _C)	+125°C Max.
Supply Voltage (V _{CC})	+4.50 to +5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameter Symbol	Parameter Description	Test Conditions	Min.	Typ. (Note 1)	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL} I _{OH} = -3.2 mA COM'L I _{OH} = -2 mA MIL	2	3		V
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL} I _{OL} = -24 mA COM'L I _{OL} = -12 mA MIL			0.5	V
V _{IH} (Note 2)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs	2.0		5.5	V
V _{IL} (Note 2)	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs			0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max., I _{IN} = 0.40 V		-20	-100	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.7 V			25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V			1.0	mA
I _{SC}	Output Short-Circuit Current	V _{CC} = Max., V _{OUT} = 0 V (Note 3)	-30	-60	-90	mA
I _{CC}	Power Supply Current	V _{CC} = Max. I _{OH} = -3.2 mA COM'L I _{OH} = -2 mA MIL I _{OL} = -24 mA COM'L I _{OL} = -12 mA MIL I _{OH} = -30, -40 mA I _{OL} = -30L, -40L I _{OH} = -35L, -45L I _{OL} = -35L, -45L			210 180 90	mA
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA		-0.9	-1.2	V
I _{OZH}	Output Leakage Current (Note 4)	V _{CC} = Max., V _{IL} = 0.8 V V _{IH} = 2.0 V V _O = 2.7 V			100	μA
I _{OZL}		V _O = 0.4 V			-100	μA

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.

3. Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second.

V_{OUT} = 0.5V has been chosen to avoid test problems caused by tester ground degradation.

4. I/O pin leakage is the worst case of I_{OZH} or I_{IX} (where X = H or L).

CAPACITANCE*

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Units
C _{IN}	Input Capacitance	V _{IN} = 2.0 V @ f = 1 MHz Pins 1, 13 Others	11 6	pF
C _{OUT}	Output Capacitance	V _{OUT} = 2.0 V @ f = 1 MHz	9	

*These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified; included in Group A Subgroup 9, 10, 11 tests unless otherwise noted

COMMERCIAL RANGE

No.	Parameter Symbol	Parameter Description	-20 Version			-30 & -30L Version			-40 & -40L Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t _{PD}	Input or Feedback to Non-Registered Output 22XP10, 20XRP4, 20XRP6, 20XRP8			20			30			40	ns
2	t _{EA}	Input to Output Enable 22XP10, 20XRP4, 20XRP6, 20XRP8			20			30			40	ns
3	t _{ER}	Input to Output Disable 22XP10, 20XRP4, 20XRP6, 20XRP8			20			30			40	ns
4	t _{PZX}	Pin 13 to Output Enable 20XRP4, 20XRP6, 20XRP8, 20XRP10			15			20			35	ns
5	t _{PXZ}	Pin 13 to Output Disable 20XRP4, 20XRP6, 20XRP8, 20XRP10			15			20			35	ns
6	t _{CO}	Clock to Output 20XRP4, 20XRP6, 20XRP8, 20XRP10			13			15			30	ns
7	t _S	Input or Feedback Setup Time 20XRP4, 20XRP6, 20XRP8, 20XRP10		20			30				40	ns
8	t _H	Hold Time 20XRP4, 20XRP6, 20XRP8, 20XRP10		0			0				0	ns
9	t _P	Clock Period (t _S + t _{CO})		33			45				70	ns
10	t _W	Clock Width		10			15				25	ns
11	f _{MAX}	Maximum Frequency			30.3			22.2			14.3	MHz

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. t_{PD} is tested with switch S₁ closed and C_L = 50 pF.

3. For three-state outputs, output enable times are tested with C_L = 50 pF to the 1.5 V level; S₁ is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with C_L = 5 pF. HIGH to high impedance tests are made to an output voltage of V_{OH} - 0.5 V with S₁ open; LOW to high impedance tests are made to the V_{OL} + 0.5 V level with S₁ closed.

MILITARY RANGE

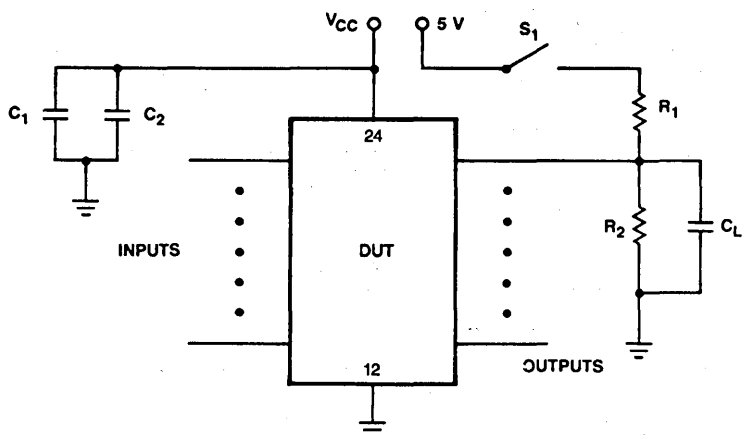
No.	Parameter Symbol	Parameter Description	-25 Version			-35 & -35L Version			-45 & -45L Versions			Units
			Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	Typ. (Note 1)	Min.	Max.	
1	t _{PD}	Input or Feedback to Non-Registered Output 22XP10, 20XRP4, 20XRP6, 20XRP8			25			35			45	ns
2	t _{EA}	Input to Output Enable 22XP10, 20XRP4, 20XRP6, 20XRP8			25			35			45	ns
3	t _{ER}	Input to Output Disable 22XP10, 20XRP4, 20XRP6, 20XRP8			25			35			45	ns
4	t _{PZX}	Pin 13 to Output Enable 20XRP4, 20XRP6, 20XRP8, 20XRP10			20			25			40	ns
5	t _{PXZ}	Pin 13 to Output Disable 20XRP4, 20XRP6, 20XRP8, 20XRP10			20			25			40	ns
6	t _{CO}	Clock to Output 20XRP4, 20XRP6, 20XRP8, 20XRP10			15			25			35	ns
7	t _S	Input or Feedback Setup Time 20XRP4, 20XRP6, 20XRP8, 20XRP10		25			35				45	ns
8	t _H	Hold Time 20XRP4, 20XRP6, 20XRP8, 20XRP10		0			0				0	ns
9	t _P	Clock Period (t _S + t _{CO})		40			60				80	ns
10	t _W	Clock Width		12			20				30	ns
11	f _{MAX}	Maximum Frequency			25			16.7			12.5	MHz

Notes: 1. Typical limits are at V_{CC} = 5.0 V and T_A = 25°C.

2. t_{PD} is tested with switch S₁ closed and C_L = 50 pF.

3. For three-state outputs, output enable times are tested with C_L = 50 pF to the 1.5 V level; S₁ is open for high impedance to HIGH tests and closed for high impedance to LOW tests. Output disable times are tested with C_L = 5 pF. HIGH to high impedance tests are made to an output voltage of V_{OH} - 0.5 V with S₁ open; LOW to high impedance tests are made to the V_{OL} + 0.5 V level with S₁ closed.

SWITCHING TEST CIRCUIT

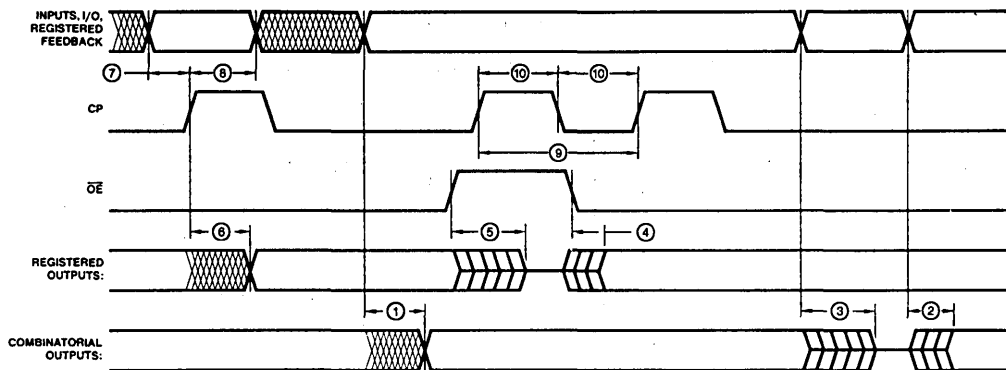


TC003051

Note: C_1 and C_2 are to bypass V_{CC} to ground.

TEST OUTPUT LOADS		
Pin Name	Commercial	Military
R_1	200 Ω	390 Ω
R_2	390 Ω	750 Ω
C_1	1 μF	1 μF
C_2	0.1 μF	0.1 μF
C_L	50 pF	50 pF

SWITCHING WAVEFORMS



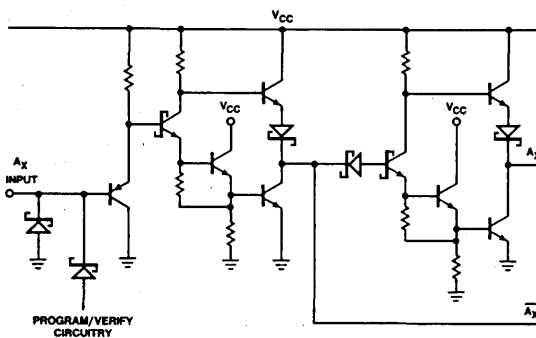
WF002571

KEY TO TIMING DIAGRAM

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

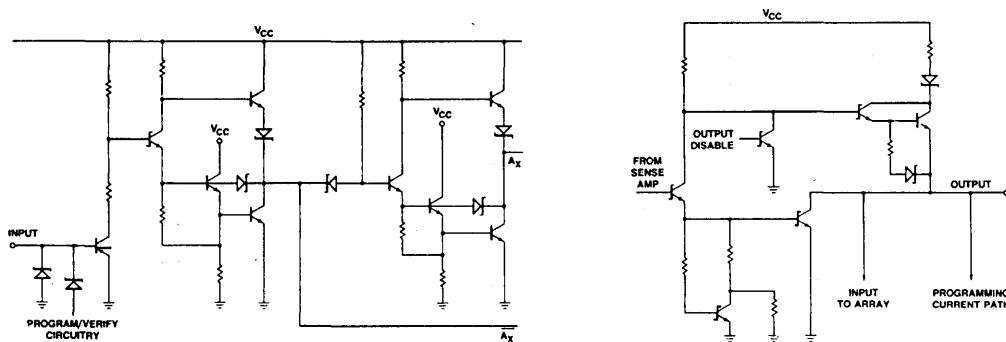
KS000010

INPUT CIRCUITRY



IC000720

OUTPUT CIRCUITRY



IC000801

Programming and Verification

AMD 24-Pin XOR PAL Family devices are programmed and verified using AMD's standard programmable logic algorithm. The fuse to be programmed is selected by input line number (array row), product term (array column), and by output (one at a time). The fuse is then programmed and verified by applying a simple sequence of voltages to two control pins (1 and 13).

Input line numbers are addressed using a full decode scheme via TTL levels on pins 6-11 where 6 is the LSB and 11 is the MSB. Even-numbered input lines represent the true version of a signal and odd-numbered lines represent the complement. Input line addressing is shown in Table 1. Note that input line 63 is utilized for selecting the fuses used for programming output polarity.

Product terms are addressed using a 1-of-16 addressing scheme on pins 2-5 where pin 2 is the LSB and 5 is the MSB. Product term addressing is shown in Table 2. Logical and architectural product terms are selected via TTL levels on the four addressing pins.

Fuse selection by output must be done one output at a time (following control pin 1 going to V_{HH}), as shown in the programming timing diagram.

Once fuses have been selected, the simple programming and verification sequence may be completed as shown in the programming timing diagram. AC and DC requirements for programming are shown in the programming parameter table.

Security Fuse Programming

A single fuse is provided on each device to prevent unauthorized copying of PAL fuse patterns. Once blown, the circuitry enabling fuse verification and registered output PRELOAD is permanently disabled.

Programming of the security fuse is the same as an array fuse. Verification of a blown security fuse is accomplished by verifying the whole fuse array as if every fuse is blown.

Programming Yield

AMD PAL devices have been designed to ensure extremely high programming yields (> 98%). To help ensure that a part was correctly programmed, once programming is completed, the entire fuse array should be verified at both LOW and HIGH V_{CC} . Reverification can be accomplished by reading all ten outputs in parallel rather than one at a time. This verification cycle checks that the array fuses have been blown and can be sensed by the outputs under varying conditions.

AMD PAL devices contain many internal test features, including circuitry and extra fuses which allow AMD to test the ability of each part to perform programming before shipping, to assure high programming yields and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment out of calibration, or improper usage of said equipment.

PROGRAMMING PARAMETERS $T_A = 25^\circ\text{C}$

Parameter Symbol	Parameter Description		Min.	Typ.	Max.	Units
V_{HH}	Control Pin Extra High Level	Pin 1 @ 5-10 mA	10	11	12	V
		Pin 13 @ 5-10 mA	10	11	12	
V_{OP}	Program Voltage Pins 14-23 @ 15-200 mA		14	15	16	V
V_{IHP}	Input HIGH Level During Programming and Verify		2.4	5	5.5	V
V_{ILP}	Input LOW Level During Programming and Verify		0.0	0.3	0.5	V
V_{CCP}	V_{CC} During Programming @ $I_{CC} = 50-275$ mA		5	5.2	5.5	V
V_{CCL}	V_{CC} During First Pass Verification @ $I_{CC} = 50-275$ mA		4.4	4.5	4.6	V
V_{CCH}	V_{CC} During Second Pass Verification @ $I_{CC} = 50-275$ mA		5.4	5.5	5.6	V
V_{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	V
V_{OP}/dt	Rate of Output Voltage Change		20		250	V/ μ s
dV_{13}/dt	Rate of Fusing Enable Voltage Change (Pin 13 Rising Edge)		100		1000	V/ μ s
t_p	Fusing Time First Attempt		40	50	100	μ s
	Subsequent Attempts		4	5	10	ms
t_D	Delays Between Various Level Changes		100	200	1000	ns
t_v	Period During which Output is Sensed for V_{Blown} Level				500	ns
V_{ONP}	Pull-Up Voltage On Outputs Not Being Programmed		$V_{CCP} - 0.3$	V_{CCP}	$V_{CCP} + 0.3$	V
R	Pull-Up Resistor On Outputs Not Being Programmed		1.9	2	2.1	k Ω

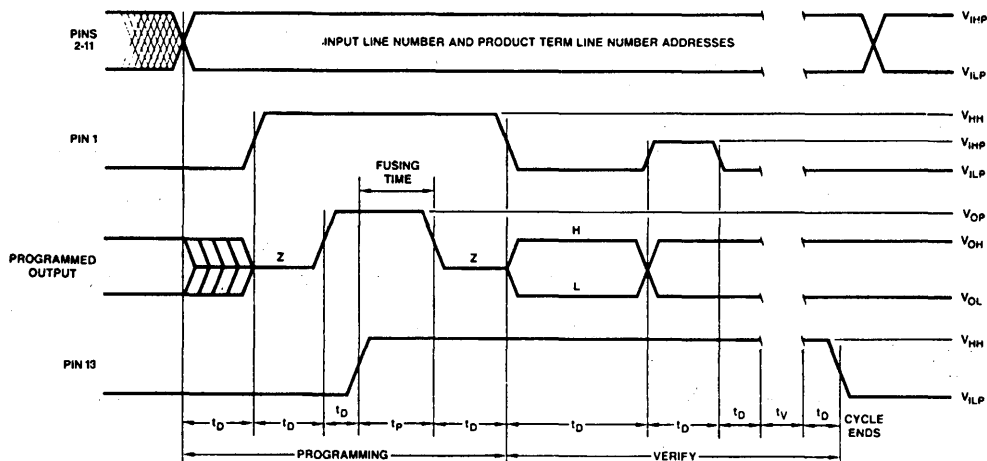
Design Aid Software for AMD 24-Pin XOR PAL Family

Name	Vendor	Versions	Notes
ABEL	Data I/O (206) 881-6444	IBM PC VAX/VMS VAX/UNIX	
CUPL	P-CAD Systems (408) 971-1300	IBM PC VAX/VMS VAX/UNIX CPM 80/86	
AmCUPL	Advanced Micro Devices (408) 732-2400	IBM PC	Supported by P-CAD Systems

AMD Qualified Programmers

Name	Programmer Model(s)	AMD PAL Personality Module	Socket Adapter
Data I/O 10525 Willow Road N.E. Redmond, WA 93052	Systems 19, 29	950-1942-0044	303A-011A
	60	N/A	Under Development
Stag Microsystems 528-5 Weddell Drive Sunnyvale, CA 94086	Model PPZ	Under Development	On Board
	ZL30	Under Development	
Valley Data Sciences 2426 Charleston Road Mountain View, CA 94043	160 Series	Under Development	On Board

PROGRAMMING TIMING DIAGRAM



PF001101

TABLE 1. INPUT ADDRESSING

Input Line Number	Input Line Number Address Pin States					
	6	7	8	9	10	11
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	L
3	L	L	L	L	H	H
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	H
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	H
12	L	L	H	H	L	L
13	L	L	H	H	L	H
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	H
18	L	H	L	L	H	L
19	L	H	L	L	H	H
20	L	H	L	H	L	L
21	L	H	L	H	L	H
22	L	H	L	H	H	L
23	L	H	L	H	H	H
24	L	H	H	L	L	L
25	L	H	H	L	L	H
26	L	H	H	L	H	L
27	L	H	H	L	H	H
28	L	H	H	H	L	L
29	L	H	H	H	L	H
30	L	H	H	H	H	L
31	L	H	H	H	H	H
32	H	L	L	L	L	L
33	H	L	L	L	L	H
34	H	L	L	L	H	L
35	H	L	L	L	H	H
36	H	L	L	H	L	L
37	H	L	L	H	L	H
38	H	L	L	H	H	L
39	H	L	L	H	H	H
40*	H	L	H	L	L	L
41*	H	L	H	L	L	H
42*	H	L	H	L	H	L
43*	H	L	H	L	H	H
63**	H	H	H	H	H	H

* Used for AmPAL22XP10 only

** Used for programming polarity

L = V_{ILP} H = V_{IHP}

TABLE 2-1. PRODUCT TERM ADDRESSING (AmPAL22XP10)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	27	36	45	54	63	72	81
L	L	L	H	1	10	19	28	37	46	55	64	73	82
L	L	H	L	2	11	20	29	38	47	56	65	74	83
L	L	H	H	3	12	21	30	39	48	57	66	75	84
L	H	L	L	4	13	22	31	40	49	58	67	76	85
L	H	L	H	5	14	23	32	41	50	59	68	77	86
L	H	H	L	6	15	24	33	42	51	60	69	78	87
L	H	H	H	7	16	25	34	43	52	61	70	79	88
H	L	L	L	8	17	26	35	44	53	62	71	80	89
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									
L = VILP H = VIHHP OE = Output Enable POL = Polarity SF = Security Fuse				Logical/Architectural Product Term Line Number									

TABLE 2-2. PRODUCT TERM ADDRESSING (AmPAL20XRP4)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	27	35	43	51	59	68	77
L	L	L	H	1	10	19	28	36	44	52	60	69	78
L	L	H	L	2	11	20	29	37	45	53	61	70	79
L	L	H	H	3	12	21	30	38	46	54	62	71	80
L	H	L	L	4	13	22	31	39	47	55	63	72	81
L	H	L	H	5	14	23	32	40	48	56	64	73	82
L	H	H	L	6	15	24	33	41	49	57	65	74	83
L	H	H	H	7	16	25	34	42	50	58	66	75	84
H	L	L	L	8	17	26	-	-	-	-	67	76	85
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									
L = VILP H = VIHHP OE = Output Enable POL = Polarity SF = Security Fuse				Logical/Architectural Product Term Line Number									

TABLE 2-3. PRODUCT TERM ADDRESSING (AmPAL20XRP6)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	18	26	34	42	50	58	66	75
L	L	L	H	1	10	19	27	35	43	51	59	67	76
L	L	H	L	2	11	20	28	36	44	52	60	68	77
L	L	H	H	3	12	21	29	37	45	53	61	69	78
L	H	L	L	4	13	22	30	38	46	54	62	70	79
L	H	L	H	5	14	23	31	39	47	55	63	71	80
L	H	H	L	6	15	24	32	40	48	56	64	72	81
L	H	H	H	7	16	25	33	41	49	57	65	73	82
H	L	L	L	8	17	-	-	-	-	-	-	74	83
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-4. PRODUCT TERM ADDRESSING (AmPAL20XRP8)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	9	17	25	33	41	49	57	65	73
L	L	L	H	1	10	18	26	34	42	50	58	66	74
L	L	H	L	2	11	19	27	35	43	51	59	67	75
L	L	H	H	3	12	20	28	36	44	52	60	68	76
L	H	L	L	4	13	21	29	37	45	53	61	69	77
L	H	L	H	5	14	22	30	38	46	54	62	70	78
L	H	H	L	6	15	23	31	39	47	55	63	71	79
L	H	H	H	7	16	24	32	40	48	56	64	72	80
H	L	L	L	8	-	-	-	-	-	-	-	-	81
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

TABLE 2-5. PRODUCT TERM ADDRESSING (AmPAL20XRP10)

Product Term Select Address Pin				Programming Access and Verify Pin									
5	4	3	2	23	22	21	20	19	18	17	16	15	14
L	L	L	L	0	8	16	24	32	40	48	56	64	72
L	L	L	H	1	9	17	25	33	41	49	57	65	73
L	L	H	L	2	10	18	26	34	42	50	58	66	74
L	L	H	H	3	11	19	27	35	43	51	59	67	75
L	H	L	L	4	12	20	28	36	44	52	60	68	76
L	H	L	H	5	13	21	29	37	45	53	61	69	77
L	H	H	L	6	14	22	30	38	46	54	62	70	78
L	H	H	H	7	15	23	31	39	47	55	63	71	79
H	L	L	L	-	-	-	-	-	-	-	-	-	-
H	L	L	H	POL	POL	POL	POL	POL	POL	POL	POL	POL	POL
H	H	H	H	SF									

Logical/Architectural Product Term Line Number

L = VILP
 H = VIHPL
 OE = Output Enable
 POL = Polarity
 SF = Security Fuse

AmPAL*HC29M16/AmPALHCT29M16

24-Pin E²-Based CMOS Programmable Array Logic

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- High-performance semi-custom logic replacement; Electrically Erasable (E²) technology allows reprogrammability
- 16 bidirectional user-programmable I/O logic macrocells for Combinatorial/Registered/Latched operation
- Output Enable controlled by a pin or product terms
- Variable product term distribution for increased design flexibility
- Programmable clock selection with two clocks/latch enables (LEs) and LOW/HIGH clock/LE polarity
- Register/Latch PRELOAD permits full logical verification
- Available in high-speed (t_{PD} = 35 ns, f_{MAX} = 20 MHz) and standard-speed (t_{PD} = 45 ns, f_{MAX} = 15.0 MHz) versions
- 100% post-programming functional yield (PPFY), fast programming and excellent reliability assured through proven E²PROM technology
- Full-function AC and DC testing at the factory
- CMOS and TTL-compatible versions
- 24-pin 300-mil DIP and 28-pin chip carrier packages

GENERAL DESCRIPTION

The AmPAL29M16 is a high-speed, E²-based CMOS Programmable Array Logic device designed for general logic replacement in TTL or CMOS digital systems. It offers high-speed, low-power consumption, high programming yield, fast programming and excellent reliability. Programmable logic devices (PLDs) combine the flexibility of custom logic with the off-the-shelf availability of standard products, providing major advantages over other semicustom solutions such as gate arrays and standard cells, including reduced development time and low up-front development cost.

The AmPAL29M16 uses the familiar sum-of-products (AND-OR) structure, allowing users to customize logic functions by programming the device for specific applications. It provides up to twenty-nine array inputs and sixteen outputs. It incorporates AMD's unique input/output logic macrocell which provides flexible input/output structure and polarity, flexible feedback selection, multiple Output Enable choices, and a programmable clocking scheme. The macrocells can be individually programmed as "Combinatorial", "Registered", or "Latched" with active-HIGH or active-LOW polarity. The flexibility of the logic macrocells permits the

system designer to tailor the device to particular application requirements.

Increased logic power has been built into the AmPAL29M16 by providing a variable number of logical product terms per output. Eight outputs have eight product terms each, four outputs have twelve product terms each, and the other four outputs have sixteen product terms each. This variable product-term distribution allows complex functions to be implemented in a single PAL device. Each output can be dynamically controlled by an Output Enable pin or Output Enable product terms. Each output can also be permanently enabled or disabled.

System operation has been enhanced by the addition of common asynchronous-PRESET and RESET product terms and a power-up RESET feature. The AmPAL29M16 also incorporates PRELOAD and Observability functions which permit full logical verification of the design.

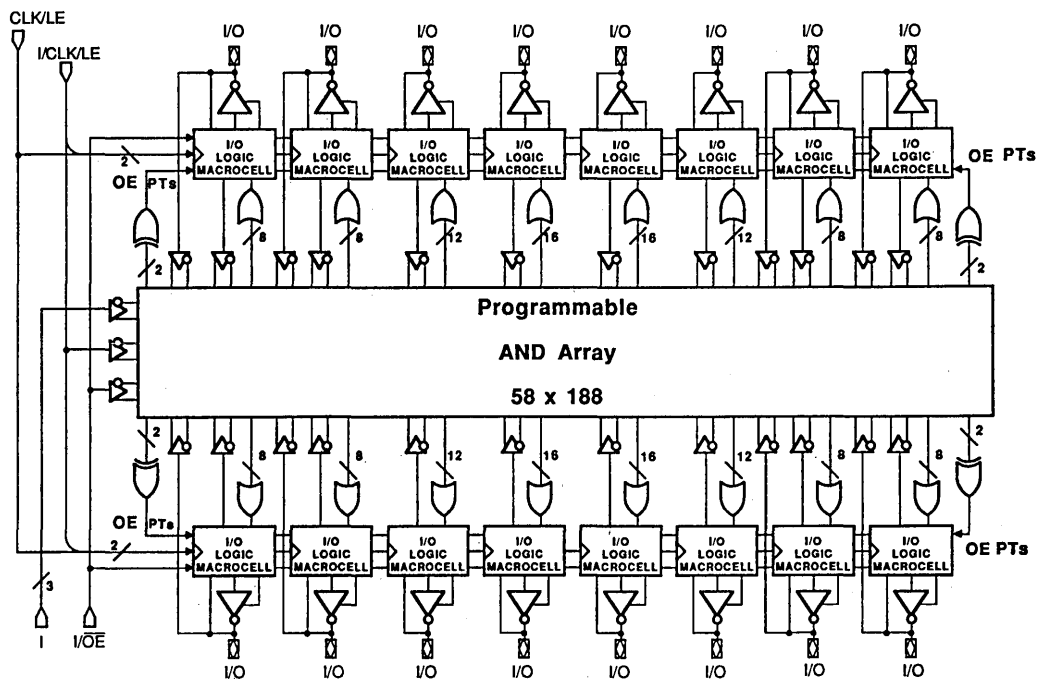
The AmPAL29M16 is compatible with HCT (High-performance CMOS & TTL) and HC (High-performance CMOS) logic levels and is offered in the space-saving 300-mil DIP package as well as chip carrier surface-mount packages.

AmPAL*HC29M16/AmPALHCT29M16

4

* PAL is a registered trademark of and is used under license from Monolithic Memories, Inc.

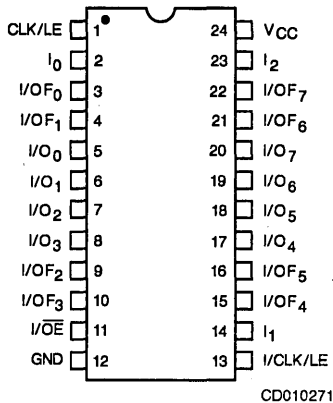
BLOCK DIAGRAM



BD006811

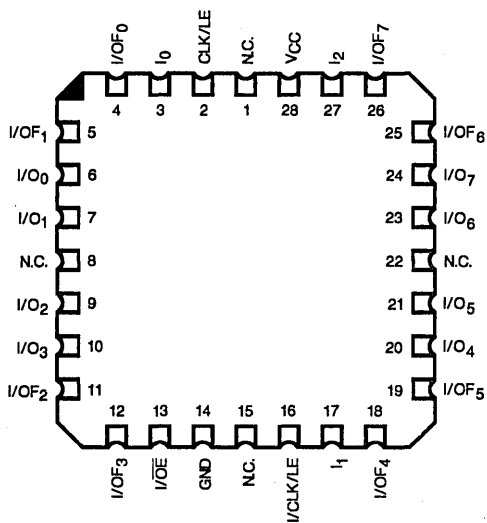
CONNECTION DIAGRAMS Top View

DIPs



CD010271

LCC*



CD010280

*Also available in PLCC. Pinouts identical to LCC.

Note: Pin 1 is marked for orientation.

PIN DESCRIPTION

The following describes the functionality of all the pins on the 24-pin DIP. The 28-pin chip carrier has the same functionality with NO CONNECTS on pins 1,8,15,22.

CLK/LE (PIN 1):

Used as dedicated clock/latch enable pin for all registers/latches on the device if so selected. (See I/O Logic Macrocell Configurations.) This pin is a clock pin for macrocells configured as registers and a latch enable pin for macrocells configured as latches.

I/CLK/LE PIN (PIN 13):

Used as dedicated input or as an alternate clock/latch enable pin for all the registers/latches if so selected. (See I/O Logic Macrocell Configurations.) This pin is a clock pin for macrocells configured as registers and a latch enable pin for macrocells configured as latches.

I/OE PIN (PIN 11):

Used as a dedicated input pin to the AND array or as the Output Enable control pin (Active LOW) for all macrocells with pin-controlled Output Enable selected.

I₀-I₂ (PINS 2,14,23):

Dedicated input pins.

I/O₀-I/O₇ (PINS 3,4,9,10,15,16,21,22):

Eight bidirectional I/O pins with two independent feedback paths to the AND array. The first feedback path is a dedicated I/O pin feedback to the AND array for combinatorial input. The second feedback path consists of direct register/latch feedback to the array (see Figure 1).

I/O₀-I/O₇ (PINS 5,6,7,8,17,18,19,20):

Eight bidirectional I/O pins with user-programmable register/latch or I/O pin feedback to the AND array (see Figure 1).

VCC (PIN 24):

Supply Voltage

GND (PIN 12):

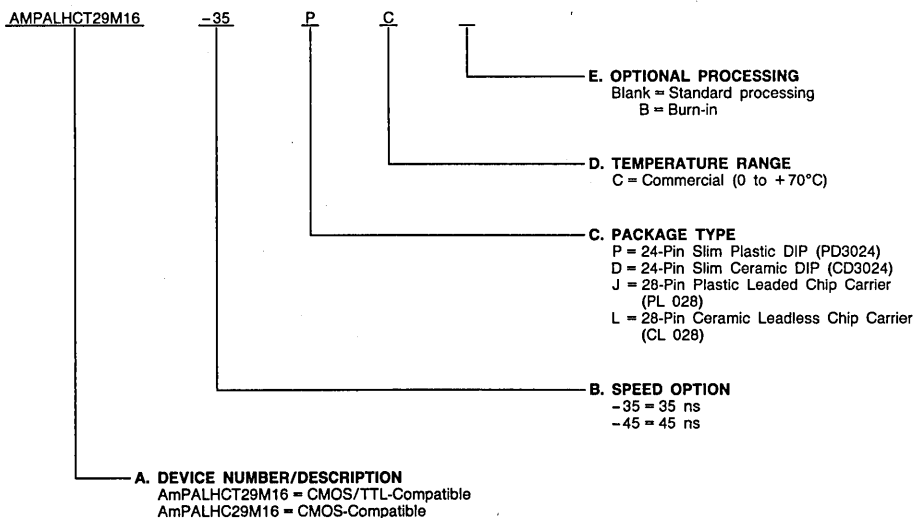
Circuit Ground

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

AmPALHCT29M16-35, -45	PC, DC, DCB,
AmPALHC29M16-35, -45	JC, LC, LCB

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, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

user-selectable allowing complete flexibility of the macrocell configuration.

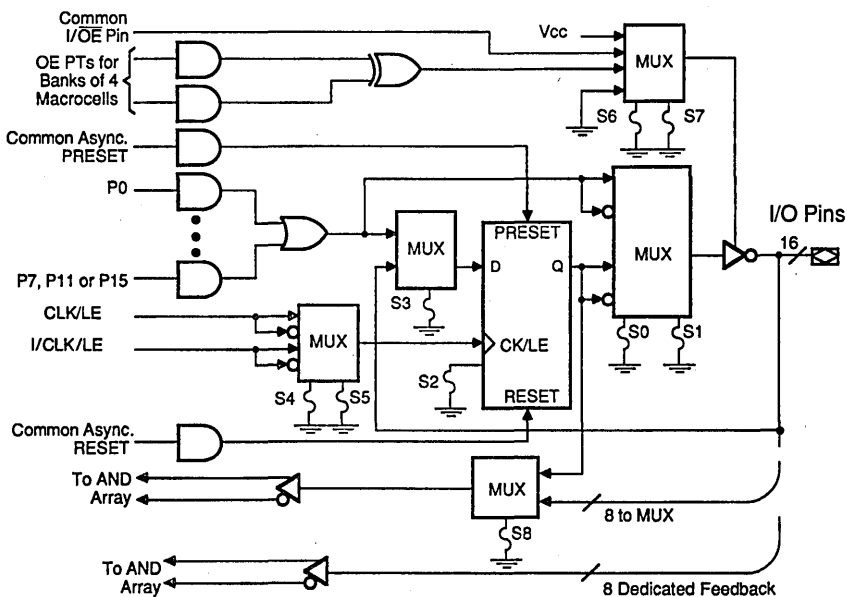
Eight of the macrocells (I/O_{F0}-I/O_{F7}) have two independent feedback paths to the AND array (see Figure 1). The first is a dedicated I/O pin feedback to the AND array for combinatorial input. The second path consists of a direct register/latch feedback to the array. If the pin is used as a dedicated input using the first feedback path, the register/latch feedback path is still available to the AND array. This path provides the capability of using the register/latch as a buried state register/latch. The other eight macrocells have a single feedback path to the AND array. This feedback is user-selectable as either an I/O pin or a register/latch feedback.

Each macrocell can provide true input/output capability. The user can select each macrocell register/latch to be driven by either the output generated by the AND-OR array or the I/O pin. When the I/O pin is selected as the input, the feedback path provides the register/latch input to the array. When used

as an input, each macrocell is also user-programmable for registered, latched, or combinatorial input.

The AmPAL29M16 has one dedicated CLK/LE pin and one I/CLK/LE pin. All macrocells have a programmable select to choose between these two pins as the clock or the latch enable signal. These pins are clock pins for macrocells configured as registers and latch enable pins for macrocells configured as latches. The polarity of these CLK/LE signals is also individually programmable. Thus different registers can be driven by multiple clocks and clock phases.

The Output-Enable mode of each of the macrocells can be selected by the user. The I/O pin can be configured as an output pin (permanently enabled) or as an input pin (permanently disabled). It can also be configured as a dynamic I/O controlled by the Output Enable pin or by two AND-XOR product terms which are available for each bank of four I/O logic macrocells.



DF006181

Figure 1. AmPALHC(HCT)29M16 I/O Macrocell

I/O Logic Macrocell Configuration

AMD's unique I/O macrocell offers major benefits through its versatile, programmable input/output cell structure, multiple clock choices, flexible Output Enable and feedback selection. Eight I/O macrocells with single feedback contain nine E²cells, while the other eight macrocells contain eight E²cells for programming the input/output functions (see Table 1, Figure 2).

E²cell S1 controls whether the macrocell will be combinatorial or registered/latched. S0 controls the output polarity (active-HIGH or active-LOW). S2 determines whether the output is a register or a latch. S3 allows the use of the macrocell as an input register/latch or as an output register/latch. It selects the direction of the data path through the register/latch. If

connected to the usual AND-OR array output, the register/latch is an output connected to the I/O pin. If connected to the I/O pin, the register/latch becomes an input register/latch to the AND array using the feedback data path.

Programmable E²cells S4 and S5 allow the user to select one of the four CLK/LE signals for each macrocell. S6 and S7 are used to control Output Enable as pin controlled, two product term controlled, permanently enabled or permanently disabled. S8 is a feedback multiplexer for the macrocells with a single feedback path only.

In the virgin erased state (charged, disconnected), an architectural cell is said to have a value of "1"; In the programmed state (discharged, connected), an architectural cell is said to have a value of "0".

Table 1. AmpPAL29M16 I/O Logic Macrocell Architecture Selections

S3	I/O Cell
1	Output Cell
0	Input Cell

S2	Storage Element
1	Register
0	Latch

S1	Output Type
1	Combinatorial
0	Register/Latch

S0	Output Polarity
1	Active LOW
0	Active HIGH

S8	Feedback*
1	Register/Latch
0	I/O

*Applies to macrocells with single feedback only.

TC003961

Table 1. AmpPAL29M16 I/O Logic Macrocell Clock Polarity & Output Enable Selections
(Cont'd.)

S4	S5	Clock Edge/Latch Enable Level	S6	S7	Output Buffer Control
1	1	CLK/LE pin positive-going edge, active-HIGH LE	1	1	Pin-Controlled 3-State Enable
1	0	CLK/LE pin negative-going edge, active-LOW LE	1	0	XOR PT-Controlled 3-State Enable
0	1	I/CLK/LE pin positive-going edge, active-HIGH LE	0	1	Permanently Enabled (Output only)
0	0	I/CLK/LE pin negative-going edge, active-LOW LE	0	0	Permanently Disabled (Input only)

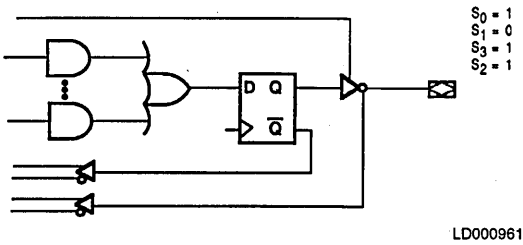
TC003971

1 = Erased State (Charged or disconnected)

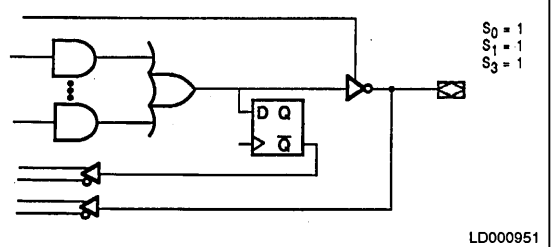
0 = Programmed State (Discharged or connected)

SOME POSSIBLE CONFIGURATIONS OF THE INPUT/OUTPUT LOGIC MACROCELL

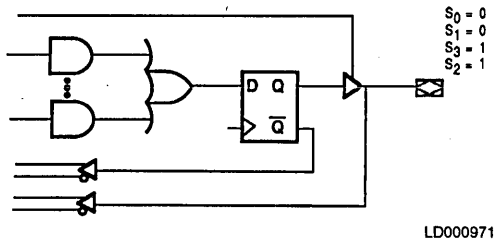
OUTPUT REGISTERED/ACTIVE LOW



OUTPUT COMBINATORIAL/ACTIVE LOW



OUTPUT REGISTERED/ACTIVE HIGH



OUTPUT COMBINATORIAL/ACTIVE HIGH

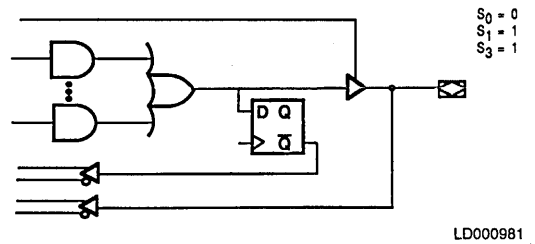
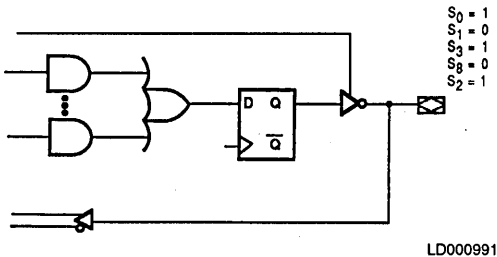
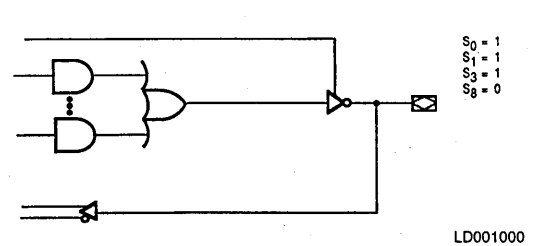


Figure 2A: Dual Feedback Macrocells

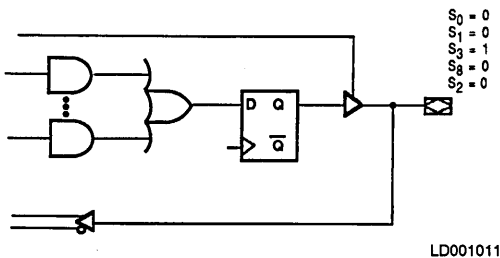
OUTPUT REGISTERED/ACTIVE LOW, I/O FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE LOW, I/O FEEDBACK



OUTPUT LATCHED/ACTIVE HIGH, I/O FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE HIGH, I/O FEEDBACK

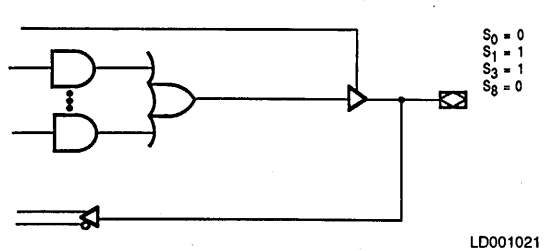
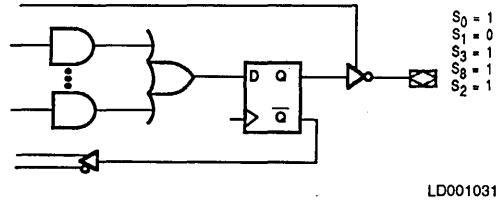
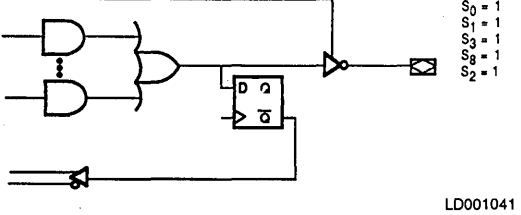


Figure 2B: Single Feedback Macrocells

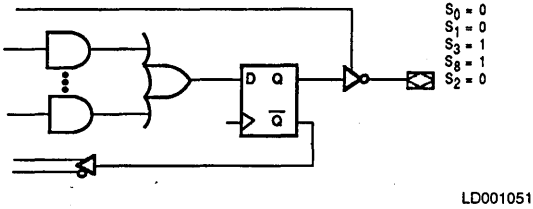
OUTPUT REGISTERED/ACTIVE LOW, REG. FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE LOW, REG. FEEDBACK



OUTPUT LATCHED/ACTIVE LOW, LATCHED FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE LOW, LATCH FEEDBACK

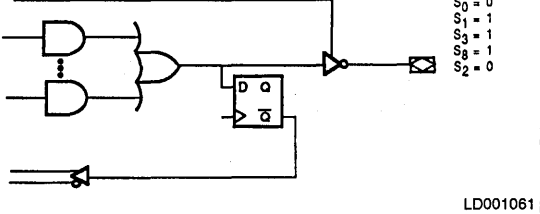


Figure 2B: Single Feedback Macrocells (Cont'd.)

INPUT REGISTERED/LATCHED

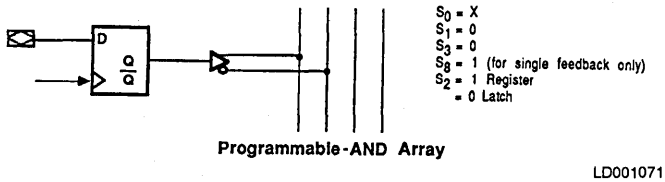
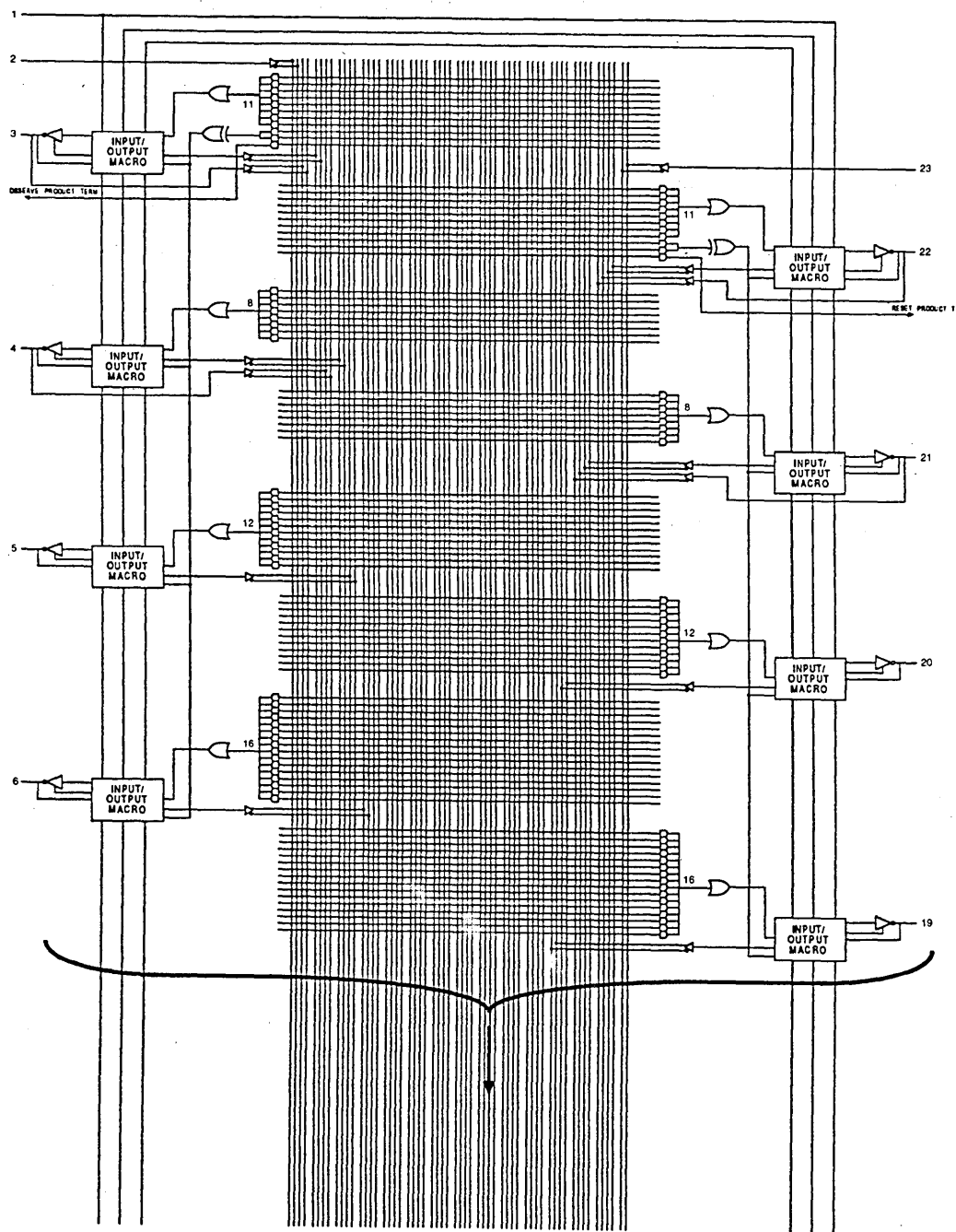
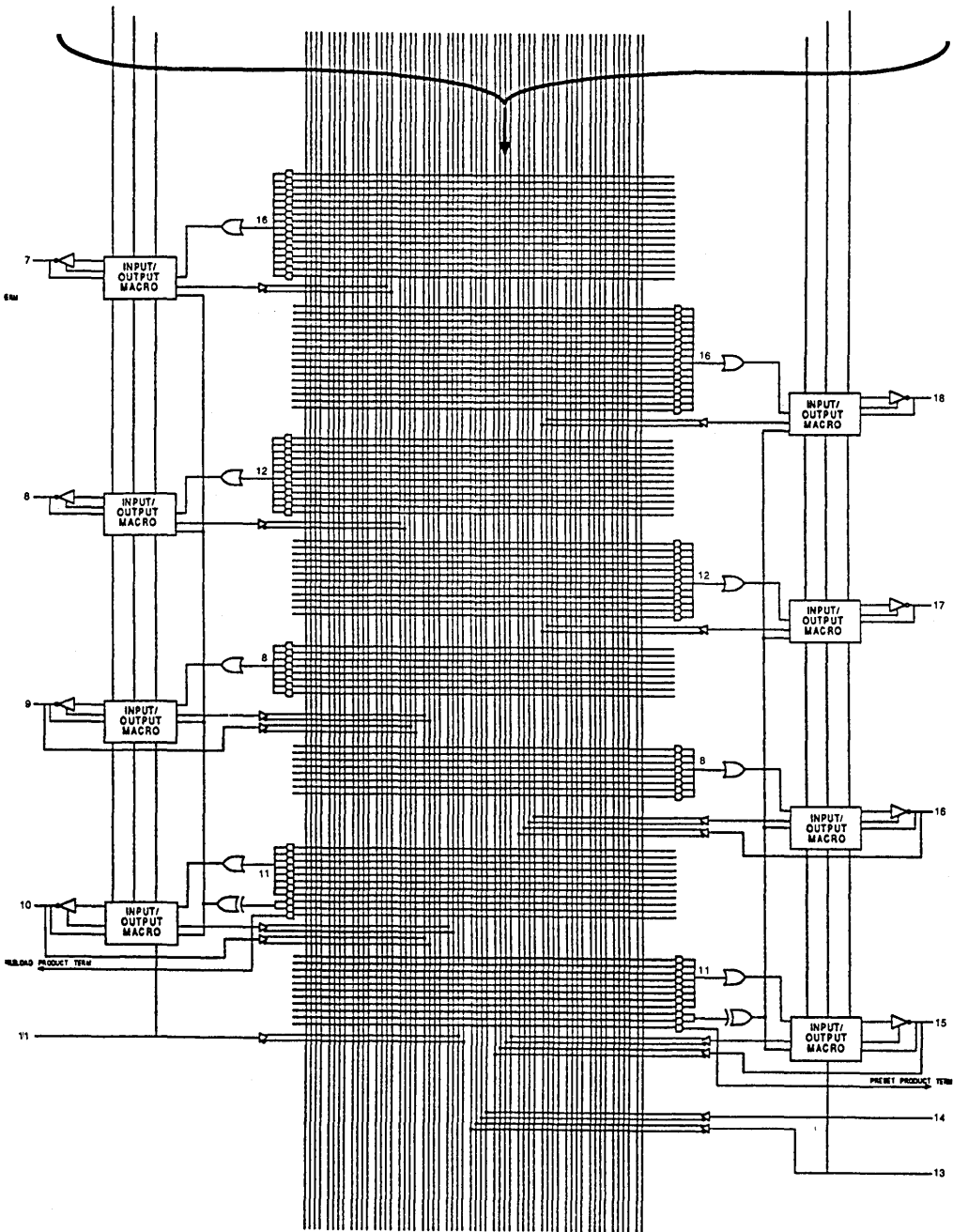


Figure 2C: All Macrocells



LD001350

Figure 3. AmpAL29M16 Logic Diagram



LD001340

Figure 3. AmPAL29M16 Logic Diagram

Designed in Testability and Debugging

PRELOAD

To simplify testing, the AmPAL29M16 is designed with PRELOAD circuitry that provides an easy method for testing logical functionality. Both TTL-level and supervoltage-enabled PRELOAD modes are available. This offers even more test capability than previously implemented in AMD's PAL devices. The TTL-level PRELOAD product term can be useful during debugging, where supervoltages may not be available.

PRELOAD allows any arbitrary state value to be loaded into the registers/latches of the device. A typical functional-test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the devices inputs into any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state, or "next state", which can be checked to validate the transition from the "present state". In this way any transition can be checked.

Since PRELOAD can provide the capability to go directly to any desired arbitrary state, test sequences may be greatly shortened. Also, all possible states can be tested, thus greatly reducing test time and development costs and guaranteeing proper in-system operation.

Observability

The output register/latch observability product term, when asserted, suppresses the combinatorial output data from appearing on the I/O pin and allows the observation of the contents of the register/latch on the output pin for each of the logic macrocells. This unique feature allows for easy debugging and tracing of the buried state machines. In addition, a capability of supervoltage observability is also provided.

Power-Up Reset

All the device registers/latches have been designed to reset during device power-up. Following the power-up, all registers/latches will be cleared, setting the outputs to a state determined by the output select multiplexer. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

Security Cell

A security cell is provided on each device to prevent unauthorized copying of the user's proprietary logic design. Once programmed, the security cell disables the programming, verification, PRELOAD, and the observability modes. The only way to erase the protection cell is by charging the entire array and architecture cells, in which case no proprietary design can be copied. (This cell should be programmed only after the rest of the device has been completely programmed and verified.)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Ambient Temperature under bias -55 to +125°C
 Supply Voltage with
 Respect to Ground -0.5 V to +7.0 V
 DC Output Voltage -0.5 V to $V_{CC} + 0.5$ V
 DC Input Voltage
 (Except Pin 1/OE) -0.5 V to $V_{CC} + 0.5$ V
 DC Input Voltage (Pin 1/OE) -0.6 V to +17 V
 DC Input Current -1 mA to +1 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices — HCT Devices

Temperature (T_A) Operating Free Air 0°C to +70°C

Supply Voltage (V_{CC}) +4.75 to +5.25 V

Commercial (C) Devices — HC Devices

Temperature (T_A) 0°C to +70°C

Supply Voltage (V_{CC}) +4.50 to +5.50 V

Military (M) Devices*

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Consult Factory for Military Specifications

DC CHARACTERISTICS over operating range unless otherwise specified**HCT Devices***

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH} \text{ or } V_{IL}$ $I_{OH} = -2 \text{ mA}$	2.4		V
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH} \text{ or } V_{IL}$ $I_{OL} = 6 \text{ mA}$		0.5	V
		$I_{OL} = 4 \text{ mA}$		0.33	
		$I_{OL} = 20 \mu\text{A}$		0.1	
V_{IH}	Input HIGH Voltage	Guaranteed Logic HIGH for all Inputs	2.0		V
V_{IL}	Input LOW Voltage	Guaranteed Logic LOW for all Inputs		0.8	V
I_I	Input Leakage Current	$V_{IN} = 0 \text{ to } 5.5 \text{ V}, V_{CC} = \text{Max.}$		10	μA
I_O	Output Leakage Current	$V_{IN} = 0 \text{ to } 5.5 \text{ V}, V_{CC} = \text{Max.}$		10	μA
I_{CCOP}	Operating Current Supply	$f = f_{MAX}$, Outputs Open ($I_O = 0$)		120	mA
I_{SC}	Output Short Circuit Current	$V_{CC} = \text{Max.}, V_O = 0 \text{ V}$	-30	-90	mA

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Units
C_{IN}	Input Capacitance	$V_{CC} = 5.00 \text{ V.}, T_A = 25^\circ\text{C}$ $V_{IN} = 0 \text{ V @ } f = 1 \text{ MHz}$	5	pF
C_{OUT}	Output Capacitance		8	

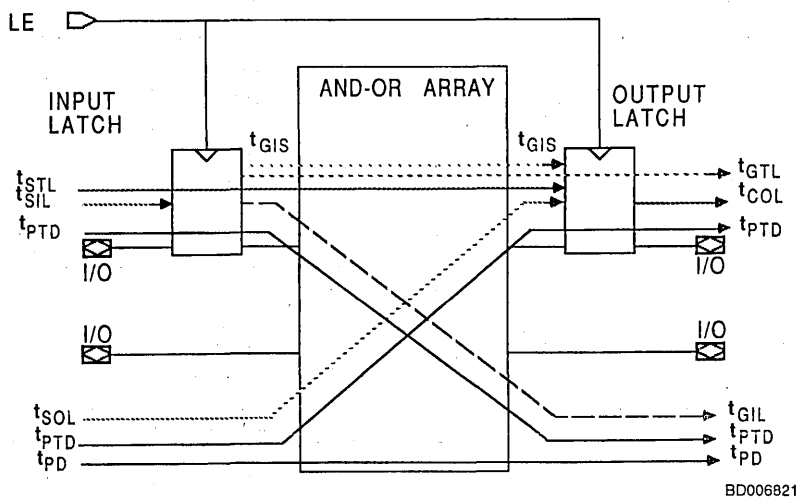
Note: These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

* Consult factory for DC specification on HC Devices.

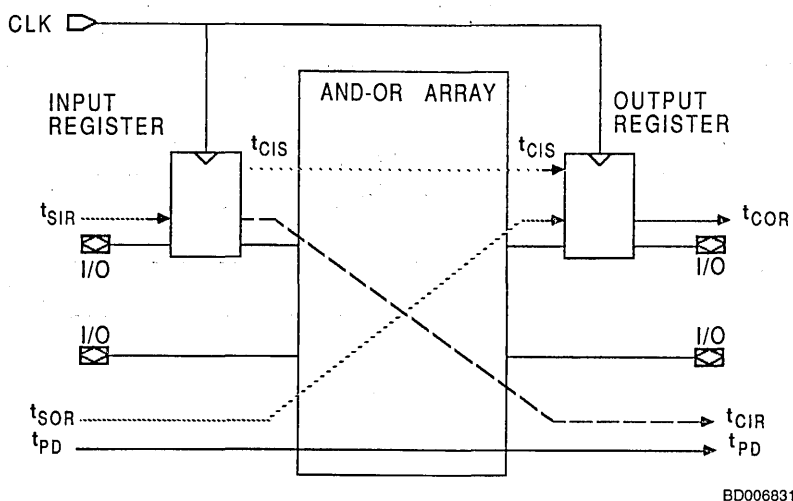
SWITCHING CHARACTERISTICS over operating range unless otherwise specified All values are determined under the loading of one TTL gate and a capacitance of 50 pF.
All Typical values are determined at $V_{CC} = 5\text{ V}$ and $T_A = 25\text{ C}$.

Parameter Number	Parameter Symbol	Parameter Description	-35		-45		Units
			Min.	Max.	Min.	Max.	
REGISTERED OPERATION (Numbers 1 through 12)							
1	t _{PD}	Input or I/O Pin to Combinatorial Output		35		45	ns
Output Register							
2	t _{SOR}	Input or I/O Pin to Output Register Setup	27		34		ns
3	t _{COR}	Output Register Clock to Output		23		32	ns
4	t _{HOR}	Data Hold Time for Output Register	0		0		ns
Input Register							
5	t _{SIR}	I/O Pin to Input Register Setup	6		8		ns
6	t _{CIR}	Register Feedback Clock to Combinatorial Output		45		58	ns
7	t _{HIR}	Data Hold Time for Input Register	3		4		ns
Clocking and Frequency							
8	t _{CIS}	Register Feedback to Output Register/Latch Setup	35		45		ns
9	f _{MAX}	Maximum Frequency 1/(t _{SOR} + t _{COR})		20		15	MHz
10	f _{MAXI}	Max Internal Frequency 1/t _{CIS}		28.5		22.5	MHz
11	t _{CWH}	Clock Width HIGH	12		15		
12	t _{CWL}	Clock Width LOW	12		15		
LATCH OPERATION (Numbers 13 through 24)							
13	t _{PD}	Input or I/O Pin to Combinatorial Output		35		45	ns
14	t _{PTD}	Input or I/O Pin to Output via One Transparent Latch		45		55	ns
Output Latch							
15	t _{SOL}	Input or I/O Pin to Output Latch Setup	27		34		ns
16	t _{GOL}	Latch Enable to Transparent Mode Output		23		32	ns
17	t _{HOL}	Data Hold Time for Output Latch	0		0		ns
18	t _{STL}	Input or I/O Pin to Output Latch Setup via Transparent Input Latch	35		45		ns

Parameter Number	Parameter Symbol	Parameter Description	-35		-45		Units
			Min.	Max.	Min.	Max.	
Input Latch							
19	t _{SIL}	I/O Pin to Input Latch Setup	6		8		ns
20	t _{GIL}	Latch Feedback, Latch Enable Transparent Mode to Combinatorial Output		45		58	ns
21	t _{HIL}	Data Hold Time for Input Latch	3		4		ns
Latch Enable							
22	t _{GIS}	Latch Feedback Transparent Mode to Output Register/Latch Setup	35		45		ns
23	t _{GWH}	Latch Enable Width HIGH	12		15		ns
24	t _{GWL}	Latch Enable Width LOW	12		15		ns
RESET/PRESET & OUTPUT ENABLE OPERATION (Numbers 25 through 32)							
25	t _{APO}	Input or I/O Pin to Output Register/Latch RESET/PRESET		40		55	ns
26	t _{AW}	Async. RESET/PRESET Pulse Width	35		45		ns
27	t _{ARO}	Async. RESET/PRESET to Input Register/Latch Recovery	30		40		ns
28	t _{ARI}	Async. RESET/PRESET to Input Register/Latch Recovery	20		30		
29	t _{PZX}	I/ $\overline{O}$ E Pin to Output Enable		30		40	ns
30	t _{PXZ}	I/ $\overline{O}$ E Pin to Output Disable		30		40	ns
31	t _{EA}	Input or I/O to Output Enable via PT		35		45	ns
32	t _{ER}	Input or I/O to Output Disable via PT		35		45	ns

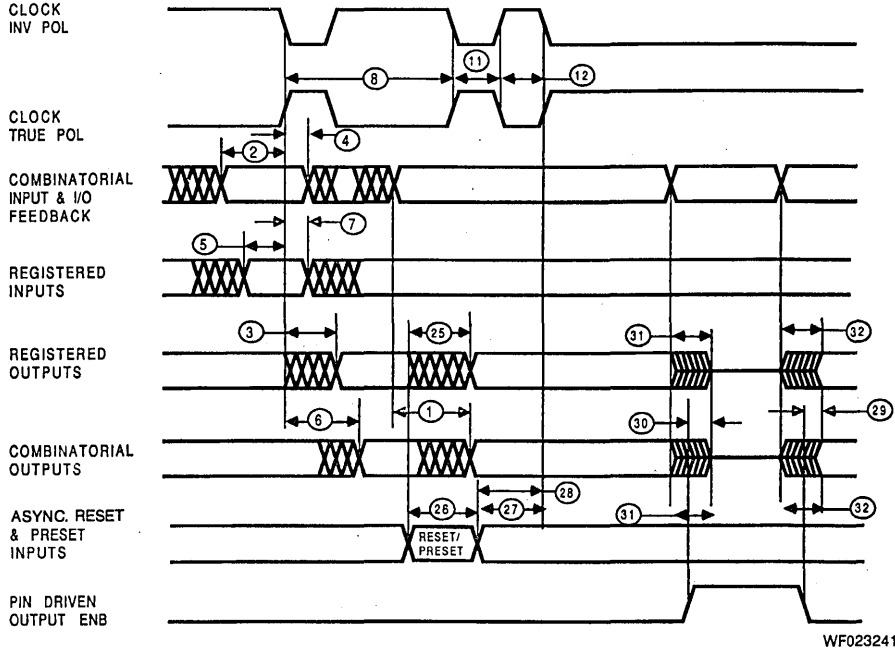


INPUT/OUTPUT SPECS (PIN LE REFERENCE)

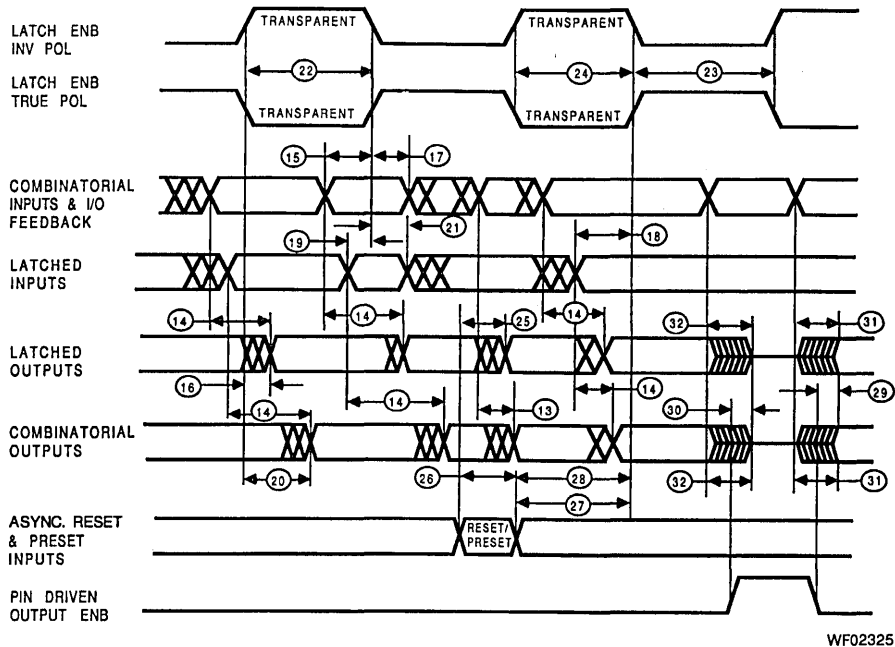


INPUT/OUTPUT REGISTER SPECS (PIN CLK REFERENCE)

SWITCHING WAVEFORMS

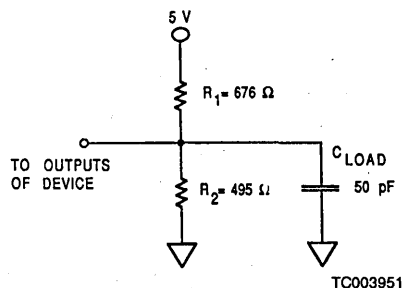


Register



Latch

SWITCHING TEST CIRCUIT



KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

AmPAL*HC29MA16/AmPALHCT29MA16

24-Pin E²-Based CMOS Programmable Array Logic

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- High-performance semi-custom logic replacement; Electrically Erasable (E²) technology allows reprogrammability
- 16 bidirectional user-programmable I/O logic macrocells for Combinatorial/Registered/Latched operation
- Output Enable controlled by a pin or product terms
- Variable product term distribution for increased design flexibility
- Programmable clock selection with common pin clock/latch enable (LE) or individual product term clock/LE with LOW/HIGH clock/LE polarity
- Register/Latch PRELOAD permits full logical verification
- Available in high-speed (t_{PD} = 35 ns, f_{MAX} = 20 MHz) and standard-speed (t_{PD} = 45 ns, f_{MAX} = 15.0 MHz) versions
- 100% post-programming functional yield (PPFY), fast programming and excellent reliability assured through proven E²PROM technology
- Full-function AC and DC testing at the factory
- CMOS and TTL-compatible versions
- 24-pin 300-mil DIP and 28-pin chip carrier packages

GENERAL DESCRIPTION

The AmPAL29MA16 is a high-speed, E²-based CMOS Programmable Array Logic device designed for general logic replacement in TTL or CMOS digital systems. It offers high-speed, low-power consumption, high programming yield, fast programming and excellent reliability. Programmable logic devices (PLDs) combine the flexibility of custom logic with the off-the-shelf availability of standard products, providing major advantages over other semicustom solutions such as gate arrays and standard cells, including reduced development time and low up-front development cost.

The AmPAL29MA16 uses the familiar sum-of-products (AND-OR) structure, allowing users to customize logic functions by programming the device for specific applications. It provides up to twenty-nine array inputs and sixteen outputs. It incorporates AMD's unique input/output logic macrocell which provides flexible input/output structure and polarity, flexible feedback selection, multiple Output Enable choices, and a programmable clocking scheme. The macrocells can be individually programmed as "Combinatorial", "Registered", or "Latched" with active-HIGH or active-LOW polarity. The flexibility of the logic macrocells

permits the system designer to tailor the device to particular application requirements.

Increased logic power has been built into the AmPAL29MA16 by providing a variable number of logical product terms per output. Eight outputs have four product terms each, four outputs have eight product terms each, and the other four outputs have twelve product terms each. This variable product-term distribution allows complex functions to be implemented in a single PAL device. Each output can be dynamically controlled by a common Output Enable pin or an individual Output Enable product terms. Each output can also be permanently enabled or disabled.

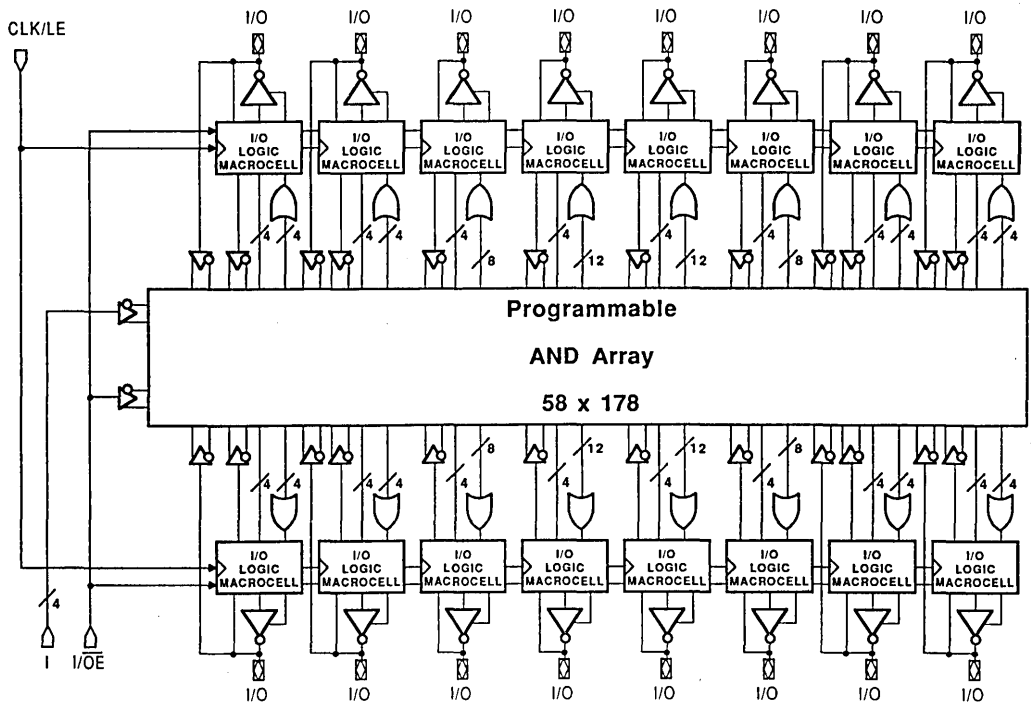
System operation has been enhanced by the addition of common asynchronous-PRESET and RESET product terms and a power-up RESET feature. The AmPAL29MA16 also incorporates PRELOAD and Observability functions which permit full logical verification of the design.

The AmPAL29MA16 is compatible with HCT (High-performance CMOS & TTL) and HC (High-performance CMOS) logic levels and is offered in the space-saving 300-mil DIP package as well as chip carrier surface-mount packages.

AmPAL*HC29MA16/AmPALHCT29MA16

4

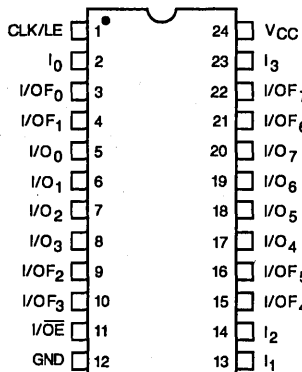
BLOCK DIAGRAM



BD006960

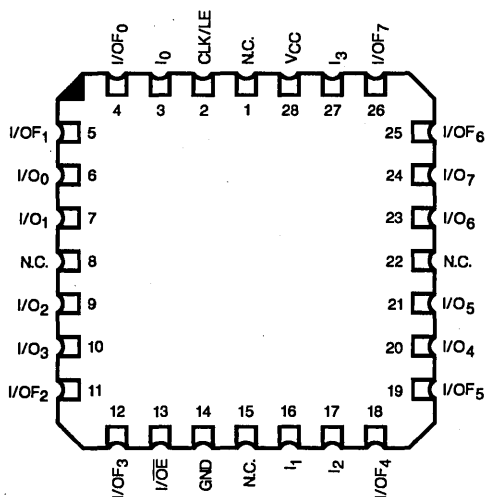
CONNECTION DIAGRAMS Top View

DIPs



CD010272

LCC*



CD010281

*Also available in PLCC. Pinouts identical to LCC.

Note: Pin 1 is marked for orientation.

PIN DESCRIPTION

The following describes the functionality of all the pins on the 24-pin DIP. The 28-pin chip carrier has the same functionality with NO CONNECTS on pins 1,8,15,22.

CLK/LE (PIN 1):

Used as dedicated clock/latch enable pin for all registers/latches on the device if so selected. (See I/O Logic Macrocell Configurations.) This pin is a clock pin for macrocells configured as registers and a latch enable pin for macrocells configured as latches.

I/OE PIN (PIN 11):

Used as a dedicated input pin to the AND array or as the Output Enable control pin (Active LOW) for all macrocells with pin-controlled Output Enable selected.

I₀-I₃ (PINS 2,13,14,23):

Dedicated input pins.

I/OF₀-I/OF₇ (PINS 3,4,9,10,15,16,21,22):

Eight bidirectional I/O pins with two independent feedback paths to the AND array. The first feedback path is a dedicated I/O pin feedback to the AND array for combinatorial input. The second feedback path consists of direct register/latch feedback to the array (see Figure 1).

I/O₀-I/O₇ (PINS 5,6,7,8,17,18,19,20):

Eight bidirectional I/O pins with user-programmable register/latch or I/O pin feedback to the AND array (see Figure 1).

VCC (PIN 24):

Supply Voltage

GND (PIN 12):

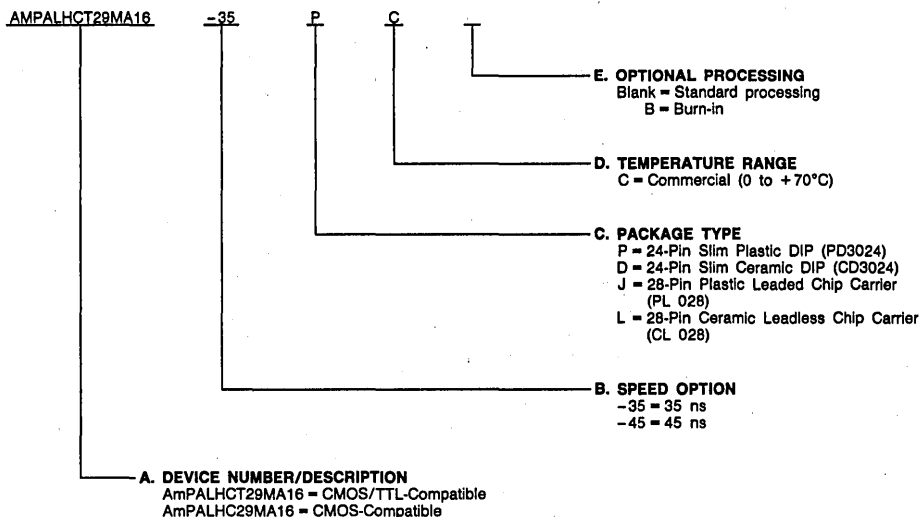
Circuit Ground

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

Valid Combinations	
AmPALHCT29MA16-35, -45	PC, DC, DCB, JC, LC, LCB
AmPALHC29MA16-35, -45	

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, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

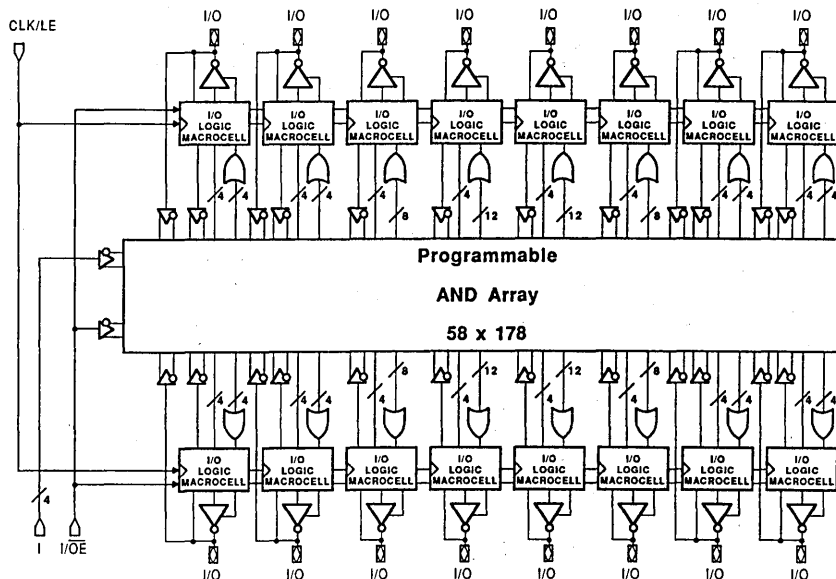
FUNCTIONAL DESCRIPTION

Inputs

The AmPAL29MA16 has 29 inputs to drive each product term (up to 58 inputs with both TRUE and complement versions available to the AND array) as shown in the block diagram below. Of these 29 inputs, 4 are dedicated inputs, 16 are from 8 I/O logic macrocells with 2 feedbacks, 8 are from other I/O logic macrocells with single feedback and 1 is for I/OE input.

Initially the AND-array gates are disconnected from all the inputs. This condition represents a logical TRUE to the AND array. By selectively programming the E² cells, the AND array may be connected to either the TRUE input or the complement input. When both the TRUE and complement inputs are connected, a logical FALSE results at the output of the AND gate.

BLOCK DIAGRAM



BD006860

Product Terms

The degree of programmability and complexity of a PAL device is determined by the number of connections that form the programmable-AND and OR gates. Each programmable-AND gate is called a product term. The AmPAL29MA16 has 178 product terms. 176 of these product terms provide logic capability and 2 are architectural product terms. Among the 2 control product terms, 1 is for Observability, and 1 is for PRELOAD. The Output Enable of each macrocell can be programmed to be controlled by a common Output Enable pin or an individual product term. It may be also permanently enabled or permanently disabled.

Each product term on the AmPAL29MA16 consists of a 58-input AND gate. The outputs of these AND gates are connected to a fixed-OR plane. Product terms are allocated to OR gates in a variable distribution across the device ranging from 4-to-12 wide, with an average of 7 logical product terms per output. Increased number of product terms per output allows more complex functions to be implemented in a single PAL device. This flexibility aids in implementing functions such as counters, exclusive-OR functions, or complex state machines, where different states require different numbers of product terms.

Individual asynchronous-PRESET and RESET product terms are connected to all Registered/Latched inputs/outputs.

When the asynchronous-PRESET product term is asserted (HIGH) all the registers/latches will immediately be loaded with a HIGH, independent of the clock. When the asynchronous-RESET product term is asserted (HIGH) all the registers/latches will be immediately loaded with a LOW, independent of the clock. The actual output state will depend on the macrocell polarity selection. The latches must be in latched mode (not transparent mode) for the RESET/PRESET, PRELOAD, and power-up RESET modes to be meaningful.

Input/Output Logic Macrocells

The I/O logic macrocell allows the user the flexibility of defining the architecture of each input or output on an individual basis. It also provides the capability of using the associated pin either as an input or an output.

The AmPAL29MA16 has 16 macrocells, one for each I/O pin. Each I/O macrocell can be programmed for combinatorial, registered or latched operation (see Figure 1). Combinatorial output is desired when the PAL device is used to replace combinatorial glue logic. Registers are used in synchronous logic applications while latches are used in asynchronous applications where speed is critical. The output polarity for each macrocell in each of the three modes of operation is user-selectable allowing complete flexibility of the macrocell configuration.

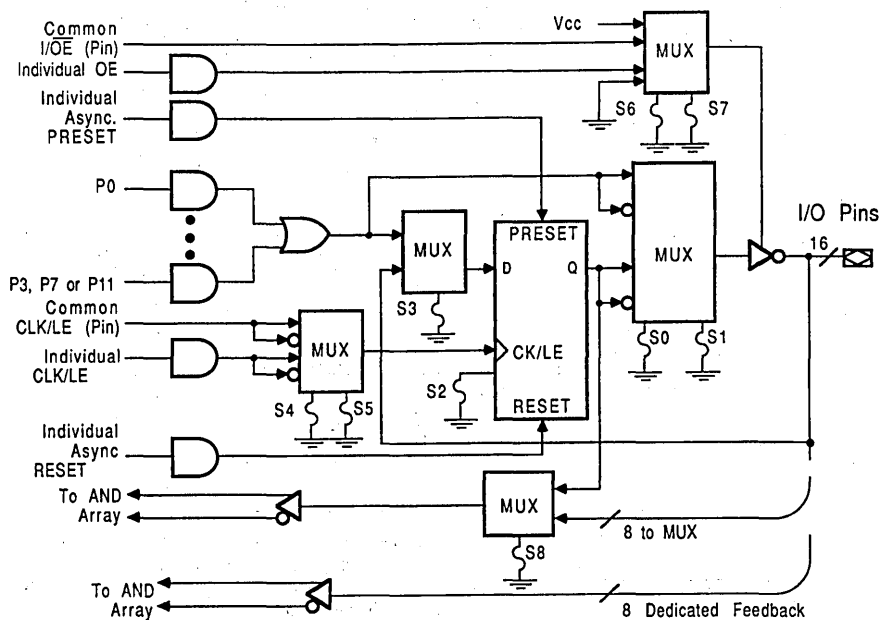
Eight of the macrocells (I/OF₀-I/OF₇) have two independent feedback paths to the AND array (see Figure 1). The first is a dedicated I/O pin feedback to the AND array for combinatorial input. The second path consists of a direct register/latch feedback to the array. If the pin is used as a dedicated input using the first feedback path, the register/latch feedback path is still available to the AND array. This path provides the capability of using the register/latch as a buried state register/latch. The other eight macrocells have a single feedback path to the AND array. This feedback is user-selectable as either an I/O pin or a register/latch feedback.

Each macrocell can provide true input/output capability. The user can select each macrocell register/latch to be driven by either the output generated by the AND/OR array or the I/O pin. When the I/O pin is selected as the input, the feedback path provides the register/latch input to the array. When used

as an input, each macrocell is also user-programmable for registered, latched, or combinatorial input.

The AmPAL29MA16 has one dedicated CLK/LE pin and an individual CLK/LE product term. All macrocells have a programmable select to choose between these two as the clock or the latch enable signal. These signals are clock signals for macrocells configured as registers and latch enable signal for macrocells configured as latches. The polarity of these CLK/LE signals is also individually programmable. Thus different registers can be driven by multiple clocks and clock phases.

The Output-Enable mode of each of the macrocells can be selected by the user. The I/O pin can be configured as an output pin (permanently enabled) or as an input pin (permanently disabled). It can also be configured as a dynamic I/O controlled by the Output Enable pin or by product term.



BD006870

Figure 1. AmPALHC(HCT)29MA16 I/O Macrocell

I/O Logic Macrocell Configuration

AMD's unique I/O macrocell offers major benefits through its versatile, programmable input/output cell structure, multiple clock choices, flexible Output Enable and feedback selection. Eight I/O macrocells with single feedback contain nine E²cells, while the other eight macrocells contain eight E²cells for programming the input/output functions (see Table 1, Figure 2).

E²cell S1 controls whether the macrocell will be combinatorial or registered/latched. S0 controls the output polarity (active-HIGH or active-LOW). S2 determines whether the input/output is a register or a latch. S3 allows the use of the macrocell as an input register/latch or as an output register/latch. It selects the direction of the data path through the register/latch. If

connected to the usual AND-OR array output, the register/latch is an output connected to the I/O pin. If connected to the I/O pin, the register/latch becomes an input register/latch to the AND array using the feedback data path.

Programmable E²cells S4 and S5 allow the user to select one of the four CLK/LE signals for each macrocell. S6 and S7 are used to control Output Enable as pin controlled, product term controlled, permanently enabled or permanently disabled. S8 is a feedback multiplexer for the macrocells with a single feedback path only.

In the virgin erased state (charged, disconnected), an architectural cell is said to have a value of "1"; In the programmed state (discharged, connected), an architectural cell is said to have a value of "0".

Table 1. AmpAL29MA16 I/O Logic Macrocell Architecture Selections

S3	I/O Cell
1	Output Cell
0	Input Cell

S2	Storage Element
1	Register
0	Latch

S1	Output Type
1	Combinatorial
0	Register/Latch

S0	Output Polarity
1	Active LOW
0	Active HIGH

S8	Feedback*
1	Register/Latch
0	I/O

*Applies to macrocells with single feedback only.

TC003961

**Table 1. AmpAL29MA16 I/O Logic Macrocell Clock Polarity & Output Enable Selections
(Cont'd.)**

S4	S5	Clock Edge/Latch Enable Level	S6	S7	Output Buffer Control
1	1	CLK/LE pin positive-going edge, active-HIGH LE	1	1	Pin-Controlled 3-State Enable
1	0	CLK/LE pin negative-going edge, active-LOW LE	1	0	PT-Controlled 3-State Enable
0	1	CLK/LE PT positive-going edge, active-HIGH LE	0	1	Permanently Enabled (Output only)
0	0	CLK/LE PT negative-going edge, active-LOW LE	0	0	Permanently Disabled (Input only)

TC003972

1 = Erased State (Charged or disconnected)

0 = Programmed State (Discharged or connected)

SOME POSSIBLE CONFIGURATIONS OF THE INPUT/OUTPUT LOGIC MACROCELL

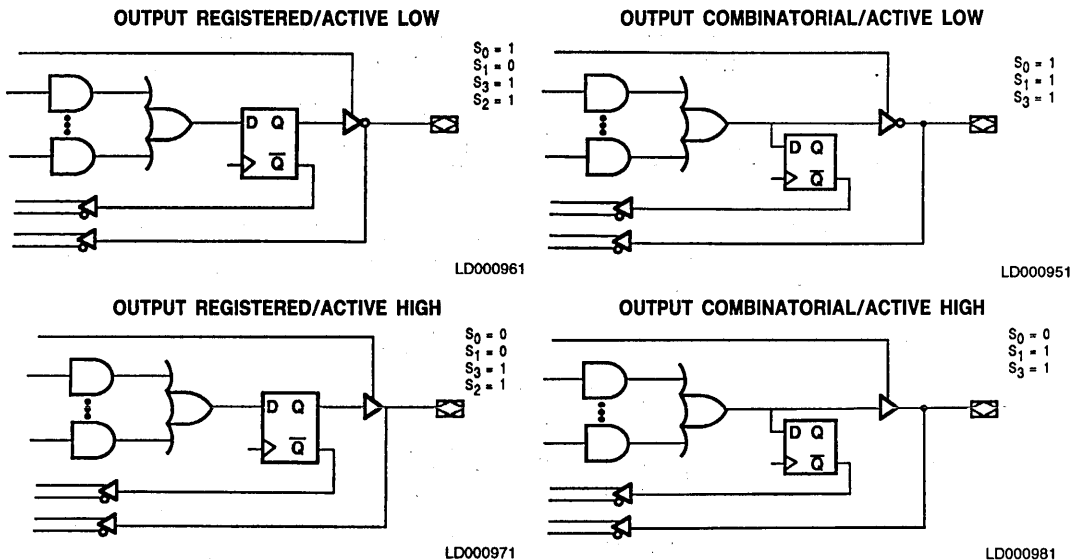


Figure 2A: Dual Feedback Macrocells

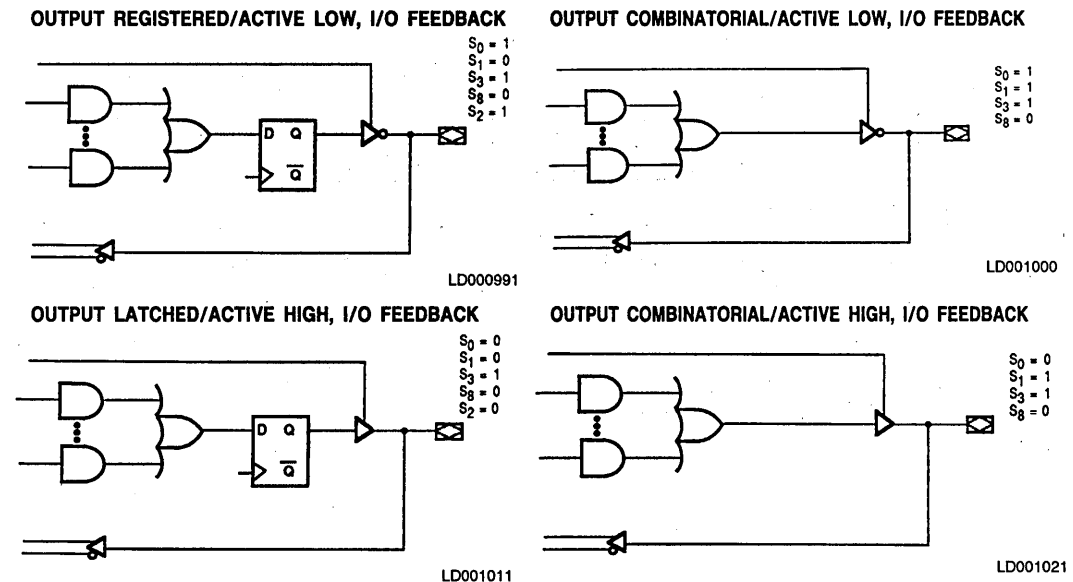
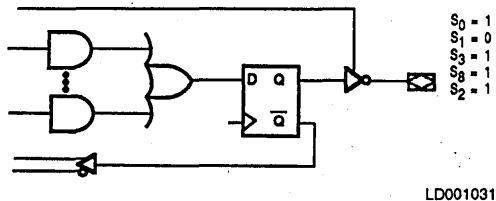
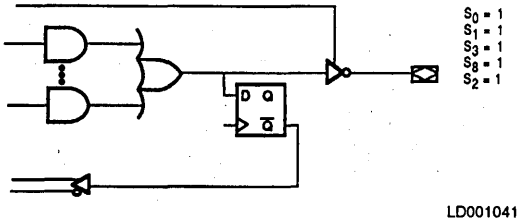


Figure 2B: Single Feedback Macrocells

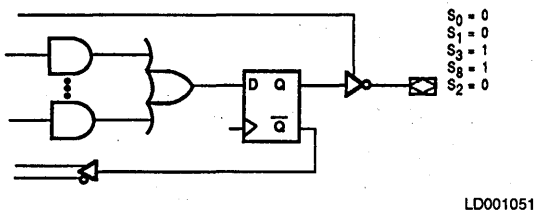
OUTPUT REGISTERED/ACTIVE LOW, REG. FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE LOW, REG. FEEDBACK



OUTPUT LATCHED/ACTIVE LOW, LATCHED FEEDBACK



OUTPUT COMBINATORIAL/ACTIVE LOW, LATCH FEEDBACK

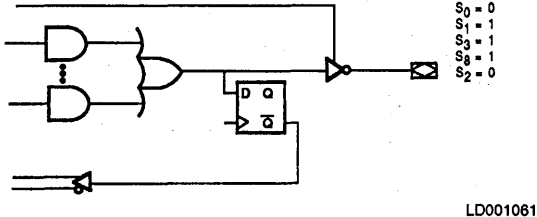


Figure 2B: Single Feedback Macrocells (Cont'd.)

INPUT REGISTERED/LATCHED

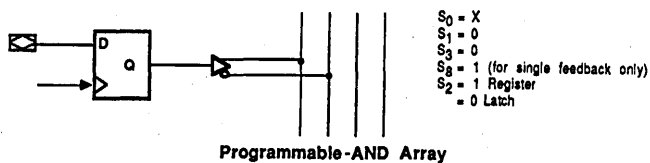
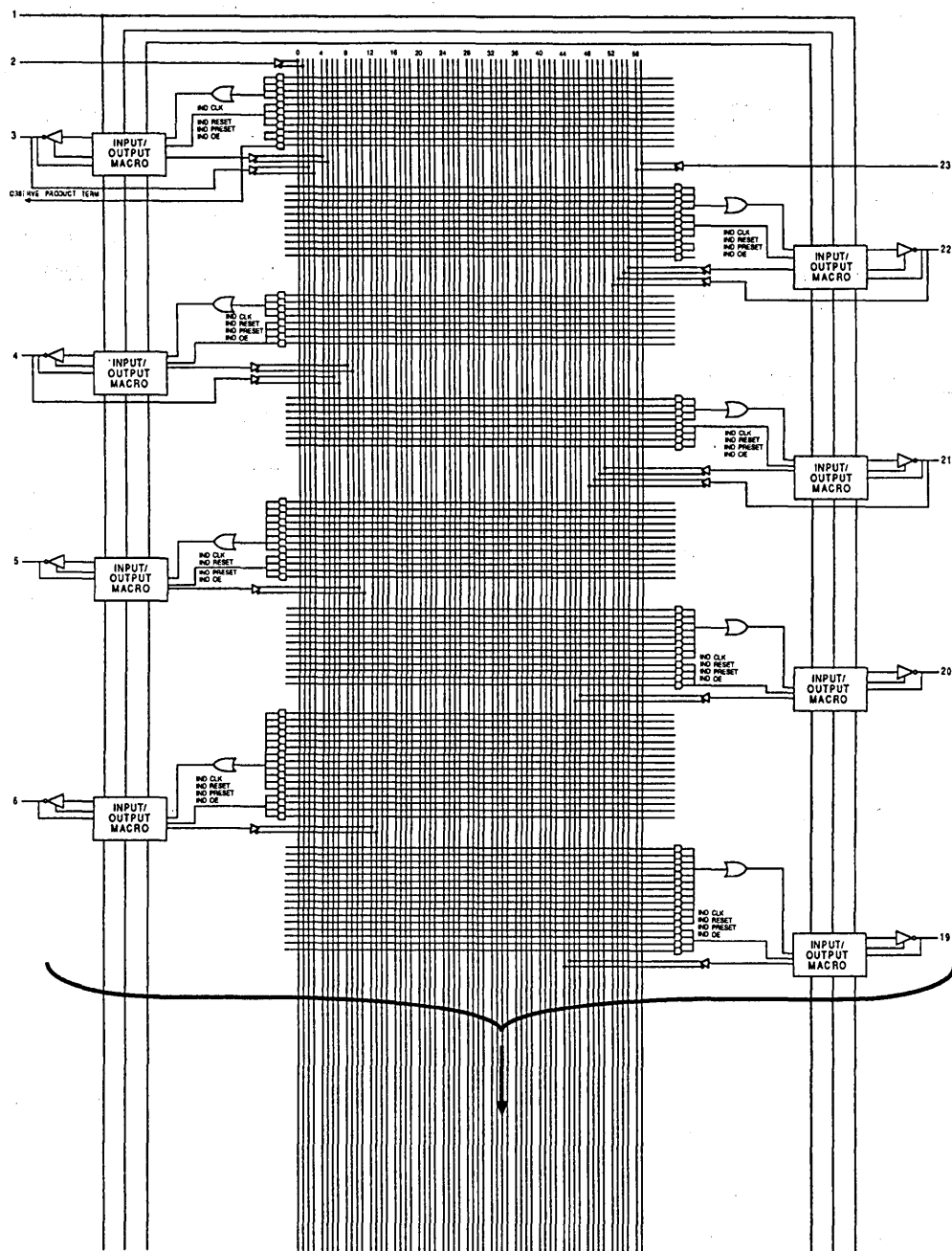
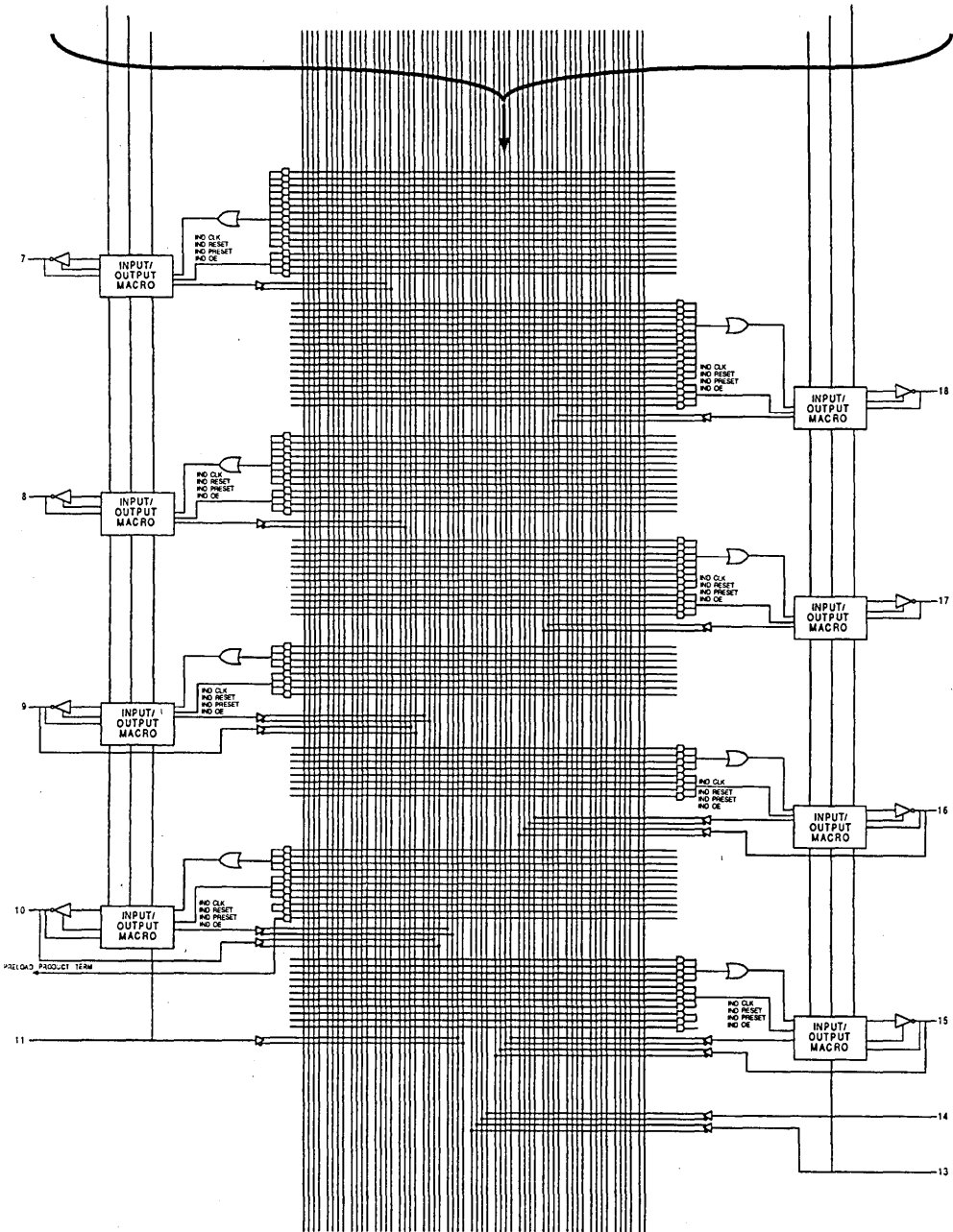


Figure 2C: All Macrocells



LD001320

Figure 3. AmPAL29MA16 Logic Diagram



LD001310

Figure 3. AmPAL29MA16 Logic Diagram

Designed in Testability and Debugging

PRELOAD

To simplify testing, the AmPAL29MA16 is designed with PRELOAD circuitry that provides an easy method for testing logical functionality. Both TTL-level and supervoltage-enabled PRELOAD modes are available. This offers even more test capability than previously implemented in AMD's PAL devices. The TTL-level PRELOAD product term can be useful during debugging, where supervoltages may not be available.

PRELOAD allows any arbitrary state value to be loaded into the registers/latches of the device. A typical functional-test sequence would be to verify all possible state transitions for the device being tested. This requires the ability to set the state registers into an arbitrary "present state" value and to set the devices inputs into any arbitrary "present input" value. Once this is done, the state machine is clocked into a new state, or "next state", which can be checked to validate the transition from the "present state". In this way any transition can be checked.

Since PRELOAD can provide the capability to go directly to any desired arbitrary state, test sequences may be greatly shortened. Also, all possible states can be tested, thus greatly reducing test time and development costs and guaranteeing proper in-system operation.

Observability

The output register/latch observability product term, when asserted, suppresses the combinatorial output data from appearing on the I/O pin and allows the observation of the contents of the register/latch on the output pin for each of the logic macrocells. This unique feature allows for easy debugging and tracing of the buried state machines. In addition, a capability of supervoltage observability is also provided.

Power-Up Reset

All the device registers/latches have been designed to reset during device power-up. Following the power-up, all registers/latches will be cleared, setting the outputs to a state determined by the output select multiplexer. This feature provides extra flexibility to the designer and is especially valuable in simplifying state machine initialization.

Security Cell

A security cell is provided on each device to prevent unauthorized copying of the user's proprietary logic design. Once programmed, the security cell disables the programming, verification, PRELOAD, and the observability modes. The only way to erase the protection cell is by charging the entire array and architecture cells, in which case no proprietary design can be copied. (This cell should be programmed only after the rest of the device has been completely programmed and verified.)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Ambient Temperature under bias -55 to +125°C
 Supply Voltage with
 Respect to Ground -0.5 V to +7.0 V
 DC Output Voltage -0.5 V to $V_{CC} + 0.5$ V
 DC Input Voltage
 (Except Pin 1/ $\overline{OE}$) -0.5 V to $V_{CC} + 0.5$ V
 DC Input Voltage (Pin 1/ $\overline{OE}$) -0.6 V to +17 V
 DC Input Current -1 mA to +1 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices — HCT Devices
 Temperature (T_A) Operating Free Air 0°C to +70°C
 Supply Voltage (V_{CC}) +4.75 to +5.25 V
 Commercial (C) Devices — HC Devices
 Temperature (T_A) 0°C to +70°C
 Supply Voltage (V_{CC}) +4.50 to +5.50 V
 Military (M) Devices*

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Consult Factory for Military Specifications

DC CHARACTERISTICS over operating range unless otherwise specified

HCT Devices**

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH} \text{ or } V_{IL}$ $I_{OH} = -2 \text{ mA}$	2.4		V
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH} \text{ or } V_{IL}$ $I_{OL} = 6 \text{ mA}$		0.5	V
		$I_{OL} = 4 \text{ mA}$		0.33	
		$I_{OL} = 20 \mu\text{A}$		0.1	
V_{IH}	Input HIGH Voltage	Guaranteed Logic HIGH for all Inputs	2.0		V
V_{IL}	Input LOW Voltage	Guaranteed Logic LOW for all Inputs		0.8	V
I_I	Input Leakage Current	$V_{IN} = 0 \text{ to } 5.5 \text{ V}, V_{CC} = \text{Max.}$		10	μA
I_O	Output Leakage Current	$V_{IN} = 0 \text{ to } 5.5 \text{ V}, V_{CC} = \text{Max.}$		10	μA
I_{CCOP}	Operating Current Supply	$f = f_{MAX}$, Outputs Open ($I_O = 0$)		120	mA
I_{SC}	Output Short Circuit Current	$V_{CC} = \text{Max.}, V_O = 0 \text{ V}$	-30	-90	mA

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	Typ.	Units
C_{IN}	Input Capacitance	$V_{CC} = 5.00 \text{ V}, T_A = 25^\circ\text{C}$ $V_{IN} = 0 \text{ V @ } f = 1 \text{ MHz}$	5	pF
C_{OUT}	Output Capacitance		8	

Note: These parameters are not 100% tested, but are evaluated at initial characterization and at any time the design is modified where capacitance may be affected.

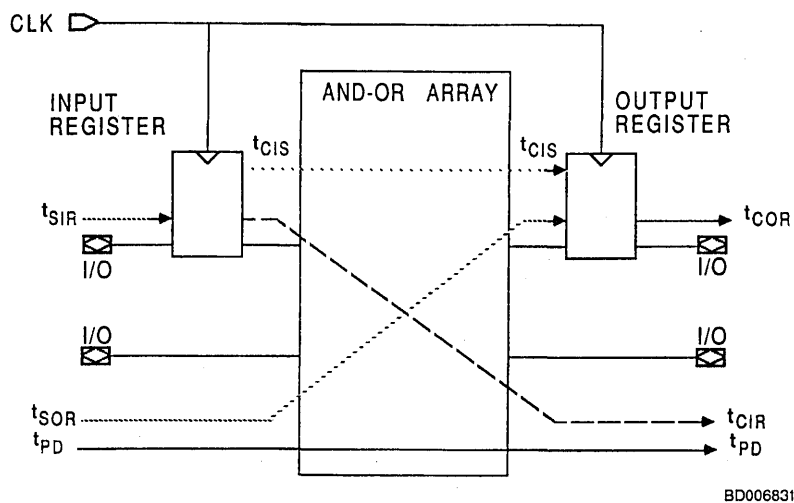
** Consult Factory for DC specifications for HC Devices.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified All values are determined under the loading of one TTL gate and a capacitance of 50 pF.
All Typical values are determined at $V_{CC} = 5\text{ V}$ and $T_A = 25\text{ }^{\circ}\text{C}$.

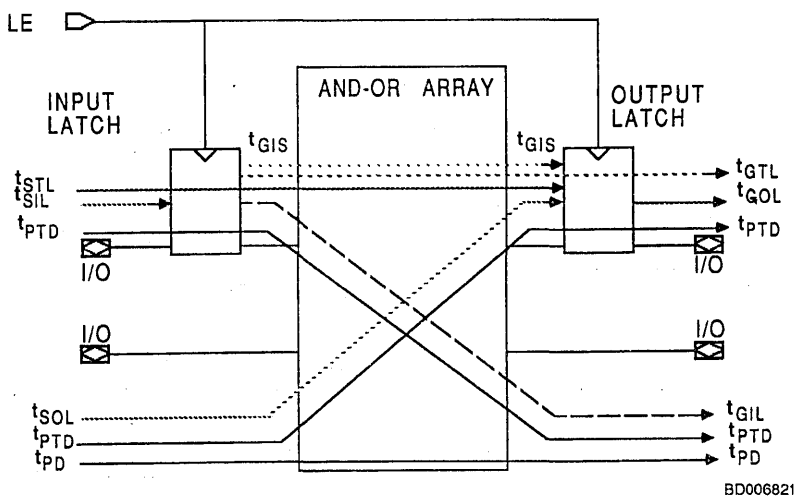
Parameter Number	Parameter Symbol	Parameter Description	-35		-45		Units
			Min.	Max.	Min.	Max.	
REGISTERED OPERATION (Numbers 1 through 20)							
1	t _{PD}	Input or I/O Pin to Combinatorial Output		35		45	ns
Output Register – Pin Clock							
2	t _{SOR}	Input or I/O Pin to Output Register Setup	27		34		ns
3	t _{COR}	Output Register Clock to Output		23		32	ns
4	t _{HOR}	Data Hold Time for Output Register	0		0		ns
Output Register – Product Term Clock							
5	t _{SORP}	I/O Pin or Input to Output Register Setup	20		24		ns
6	t _{CORP}	Output Register Clock to Output		45		56	ns
7	t _{HORP}	Data Hold Time for Output Register	15		20		ns
Input Register – Pin Clock							
8	t _{SIR}	I/O Pin to Input Register Setup	6		8		ns
9	t _{CIR}	Register Feedback Clock to Combinatorial Output		45		58	ns
10	t _{HIR}	Data Hold Time for Input Register	3		4		ns
Clock and Frequency							
11	t _{CIS}	Register Feedback (Pin Driven Clock) to Output Register/Latch (Pin Driven) Setup	35		45		ns
12	t _{CISPP}	Register Feedback (PT Driven Clock) to Output Register/Latch (PT Driven) Setup	45		60		ns
13	f _{MAX}	Maximum Frequency (Pin Driven) 1/(t _{SOR} + t _{COR})		20		15	MHz
14	f _{MAXI}	Maximum Internal Frequency (Pin Driven) 1/t _{CIS}		28.5		22.5	MHz
15	f _{MAXP}	Maximum Frequency (PT Driven) 1/(t _{SORP} + t _{CORP})		15.5		12.5	MHz
16	f _{MAXIPP}	Maximum Internal Frequency (PT Driven) 1/t _{CISPP}		22.5		16.5	MHz
17	t _{CWH}	Pin Clock Width HIGH	12		15		ns
18	t _{CWL}	Pin Clock Width LOW	12		15		ns
19	t _{CWHP}	PT Clock Width HIGH	15		20		ns
20	t _{CWLP}	PT Clock Width LOW	15		20		ns

Parameter Number	Parameter Symbol	Parameter Description	-35		-45		Units
			Min.	Max.	Min.	Max.	
LATCHED OPERATION (Numbers 21 through 39)							
21	t _{PD}	Input or I/O Pin to Combinatorial Output		35		45	ns
22	t _{PTD}	Input or I/O Pin to Output via Transparent Latch		45		55	ns
Output Latch – Pin LE							
23	t _{SOL}	Input or I/O Pin to Output Latch Setup	27		34		ns
24	t _{GOL}	Latch Enable to Transparent Mode Output		23		32	ns
25	t _{HOL}	Data Hold Time for Output Latch	0		0		ns
26	t _{STL}	Input or I/O Pin to Output Latch Setup via Transparent Input Latch	35		45		ns
Output Latch – PT LE							
27	t _{SOLP}	Input or I/O Pin to Output Latch Setup	20		24		ns
28	t _{GOLP}	Latch Enable to Transparent Mode Output		45		56	ns
29	t _{HOLP}	Data Hold Time for Output Latch	15		20		ns
30	t _{STLP}	Input or I/O Pin to Output Latch Setup via Transparent Input Latch	30		35		ns
Input Latch – Pin LE							
31	t _{SIL}	I/O Pin to Input Latch Setup	6		8		ns
32	t _{GIL}	Latch Feedback, Latch Enable Transparent Mode to Combinatorial Output		45		58	ns
33	t _{HIL}	Data Hold Time for Input Latch	3		4		ns
Latch Enable							
34	t _{GIS}	Latch Feedback (Pin Driven) to Output Register/Latch (Pin Driven) Setup	35		45		ns
35	t _{GISPP}	Latch Feedback (PT Driven) to Output Register/Latch (PT Driven) Setup	45		60		ns
36	t _{GWH}	Pin Enable Width HIGH	12		15		ns
37	t _{GWL}	Pin Enable Width LOW	12		15		ns
38	t _{GWHP}	PT Enable Width HIGH	15		20		ns
39	t _{GWLP}	PT Enable Width LOW	15		20		ns

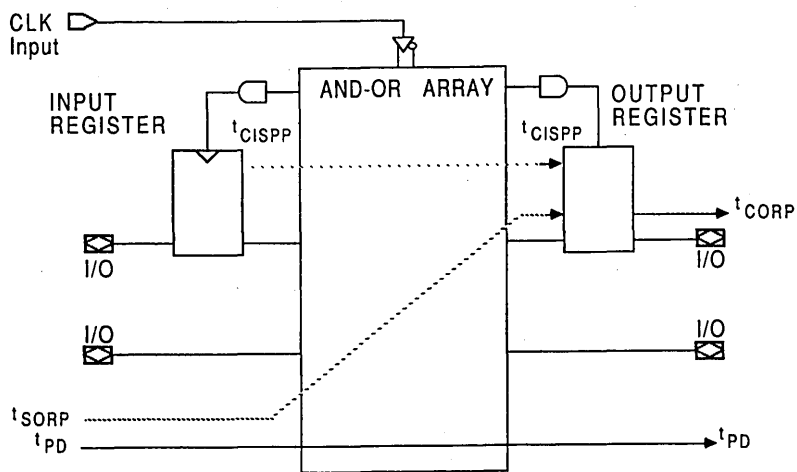
Parameter Number	Parameter Symbol	Parameter Description	-35		-45		Units
			Min.	Max.	Min.	Max.	
RESET/PRESET & OUTPUT ENABLE (Numbers 40 through 49)							
40	tAPO	Input or I/O Pin to Output Register/Latch RESET/PRESET		40		55	ns
41	tAW	Async. RESET/PRESET Pulse Width	35		45		ns
42	tARO	Async. RESET/PRESET to Output Register/Latch Recovery	30		40		ns
43	tARI	Async. RESET/PRESET to Input Register/Latch Recovery	20		30		
44	tARPO	Async. RESET/PRESET to Output Register/Latch Recovery PT Clock/LE	20		25		ns
45	tARPI	Async. RESET/PRESET to Input Register/Latch Recovery PT Clock/LE	15		20		ns
46	tpZX	I/ $\overline{O}$ E Pin to Output Enable		30		40	ns
47	tpXZ	I/ $\overline{O}$ E Pin to Output Disable		30		40	ns
48	tEA	Input or I/O to Output Enable via PT		35		45	ns
49	tER	Input or I/O to Output Disable via PT		35		45	ns



INPUT/OUTPUT REGISTER SPECS (PIN CLK REFERENCE)

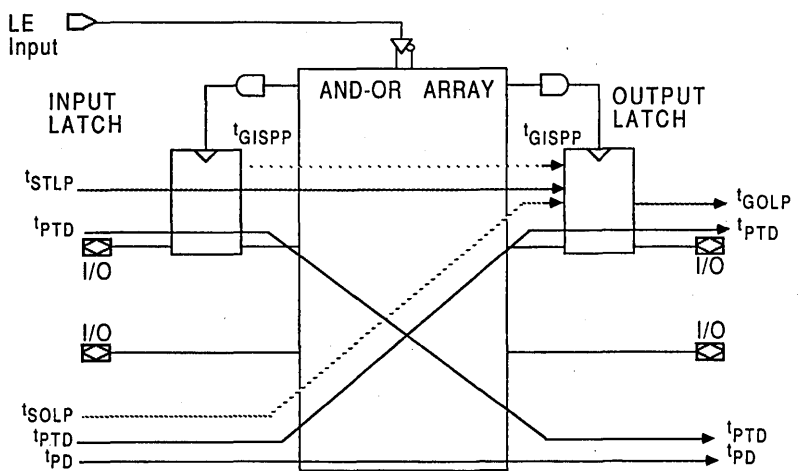


INPUT/OUTPUT LATCH SPECS (PIN LE REFERENCE)



BD006840

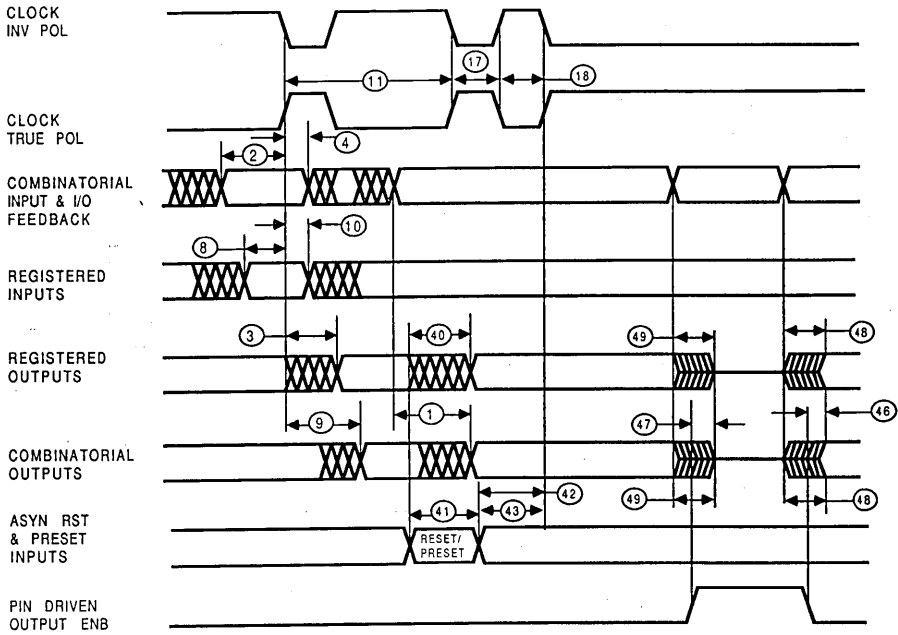
INPUT/OUTPUT REGISTER SPECS (PT CLK REFERENCE)



BD006850

INPUT/OUTPUT LATCH SPECS (PT LE REFERENCE)

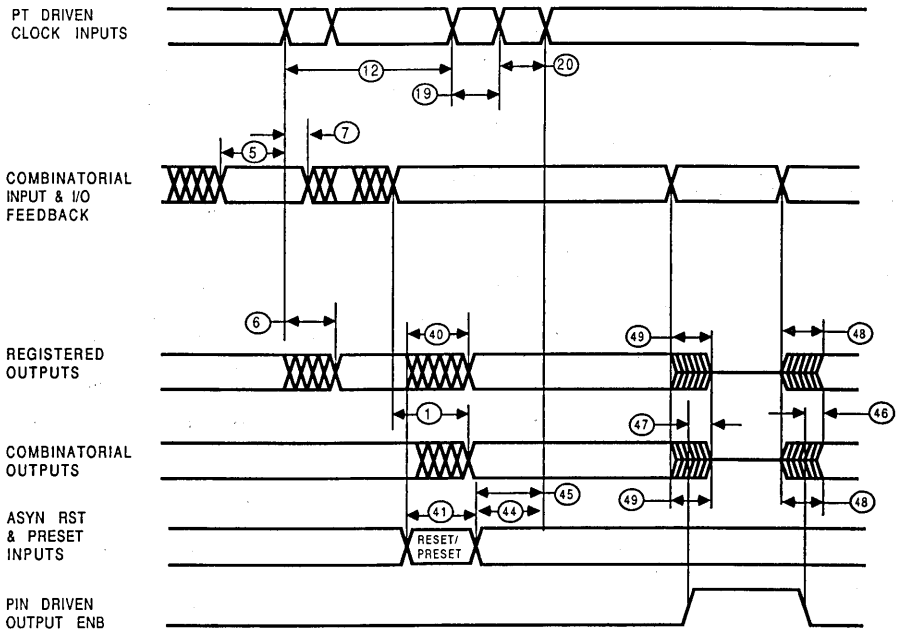
SWITCHING WAVEFORMS



WF023270

Register (Pin CLK Reference)

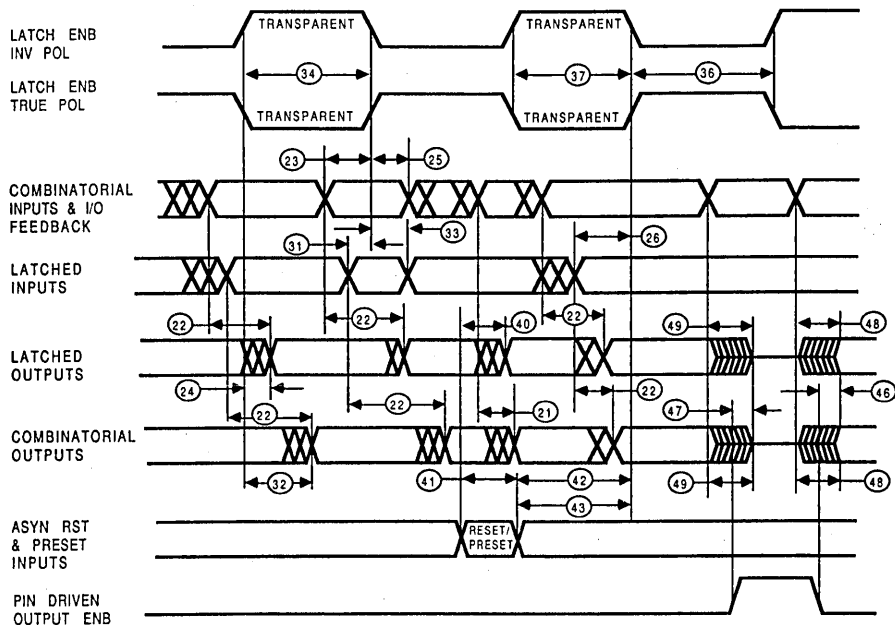
4



WF023280

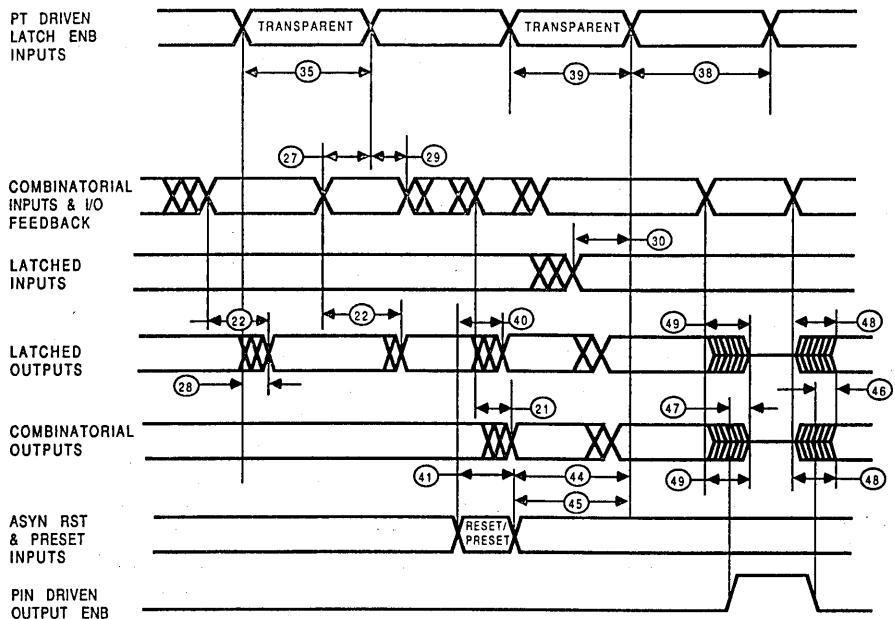
Register (PT CLK Reference)

SWITCHING WAVEFORMS (Cont'd.)



WF023290

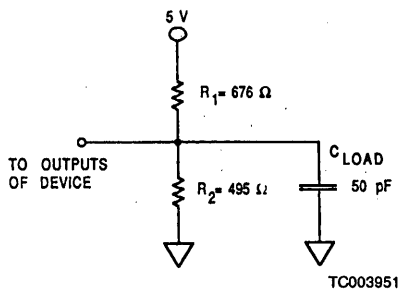
Latch (Pin LE Reference)



WF023300

Latch (PT LE Reference)

SWITCHING TEST CIRCUIT



KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

4.17 Bipolar PROMs as Programmable Logic Products

Bipolar PROMs as Programmable Logic Products

Selection Guide					
PART NUMBER	INPUTS	OUTPUTS	PRODUCT TERMS	REGISTERED OUTPUTS	t_{PD}/I_{CC} $t_{SU}/t_{CP-Q}/I_{CC}$ COM'L (MAX)
Am27S19A	5	8	32		25ns/115mA
Am27S19SA	5	8	32		15ns/115mA
Am27S21A	8	4	256		30ns/130mA
Am27S13A	9	4	512		30ns/130mA
Am27S29A	9	8	512		35ns/160mA
Am27S25A	9	8	512	8	30ns/20ns/185mA
Am27S25SA	9	8	512	8	25ns/12ns/185mA
Am27S33A	10	4	1024		35ns/140mA
Am27S65A*	10	4	1024	4	23ns/10ns/165mA
Am27S65*	10	4	1024	4	30ns/15ns/165mA
Am27S281A	10	8	1024		35ns/185mA
Am27S35A	10	8	1024	8	35ns/20ns/185mA
Am27S37A	10	8	1024	8	35ns/20ns/185mA
Am27S185A	11	4	2048		35ns/150mA
Am27S75*	11	4	2048	4	30ns/15ns/175mA
Am27S75A*	11	4	2048	4	25ns/12ns/175mA
Am27S291A	11	8	2048		35ns/185mA
Am27LS291	11	8	2048		30ns/90mA
Am27S291SA	11	8	2048		20ns/185mA
Am27S45A	11	8	2048	8	40ns/20ns/185mA
Am27S45SA	11	8	2048	8	25ns/10ns/185mA
Am27S47A	11	8	2048	8	40ns/20ns/185mA
Am27S47SA	11	8	2048	8	25ns/10ns/185mA
Am27S41A	12	4	4096		35ns/185mA
Am27S85*	12	4	4096	4	35ns/15ns/185mA
Am27S85A*	12	4	4096	4	27ns/12ns/185mA
Am27S55A	12	8	4096	8	20ns/10ns/185mA
Am10P14/100P14	10	4	1024		10ns/-200mA
Am10P44/100P44	12	4	4096		15ns/-200mA

Note 1: For ordering information see Bipolar/MOS Memories Databook (1986) or contact your local sales office

*These devices contain SSR™ on chip diagnostics

Am10P14/Am100P14/Am10KP14

4,096-Bit (1024 x 4) ECL Bipolar PROM

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- Fast Access time (8 ns typ.) — improves system cycle times
- Power dissipation decreases with increasing temperature
- Internally voltage compensated providing flat AC performance
- Open emitter outputs (50 Ω drive), wired-OR capability

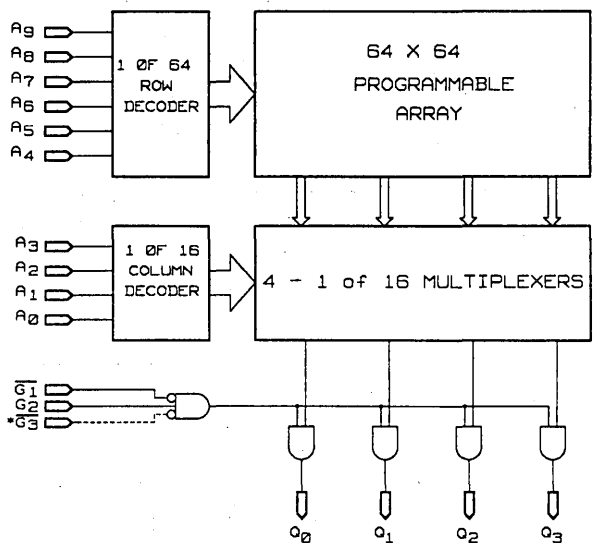
GENERAL DESCRIPTION

The Am10P14, Am10KP14, & Am100P14 (1024-words by 4-bits) are Schottky array, ECL Programmable Read-Only Memories (PROMs).

The 10K Versions are compatible with standard voltage-compensated 10K series ECL. The 100K Versions are compatible with standard temperature and voltage-com-

pensated 100K series ECL. Both are capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word-depth expansion is provided by both active LOW ($\overline{G_1}$ & $\overline{G_3}$) and active HIGH (G_2) output enables and an unterminated emitter follower output capable of wired-OR bus connection.

BLOCK DIAGRAM



BD006370

PRODUCT SELECTOR GUIDE

Part Number	Am10P14		Am10KP14		Am100P14
Address Access Time (ns)	10	15	10	15	10
Operating Range	C	M	C	M	C



Am10P44/Am100P44

16,384-Bit (4096 x 4) ECL Bipolar PROM

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- Fast access time (12 ns typical) improves system cycle times
- Power dissipation decreases with increasing temperature
- Internally voltage compensated providing flat AC performance
- Open emitter outputs (50-Ω drive), wired-OR capability

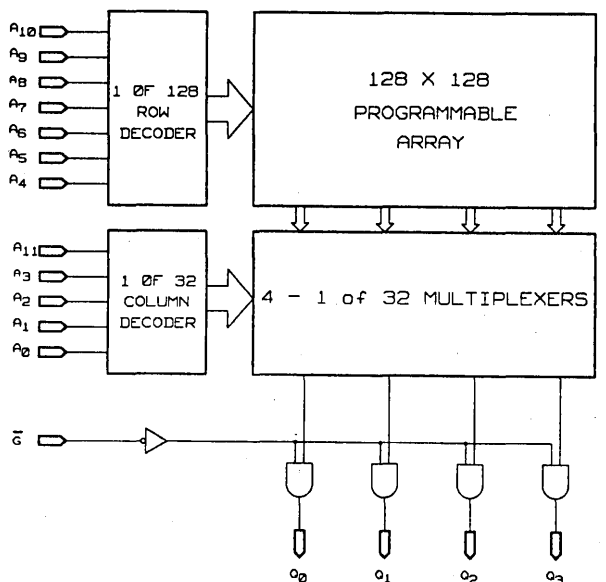
GENERAL DESCRIPTION

The Am10P44 and Am100P44 (4096 words by 4 bits) are Schottky array, ECL Programmable Read-Only Memories (PROMs).

The 10K Versions are compatible with standard voltage compensated 10K series ECL. The 100K versions are compatible with standard temperature and voltage com-

pensated 100K series ECL. Both are capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word-depth expansion is provided by an active LOW output enable ($\bar{G}$) and an unterminated emitter-follower output capable of wired-OR bus connection.

BLOCK DIAGRAM



BD005680

PRODUCT SELECTOR GUIDE

Part Number	10P44	10P44	100P44
Address Access Time (ns)	15	20	15
Operating Range	C	M	C

Am27S12/13

2,048-Bit (512 x 4) Bipolar PROM



Am27S12/13

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- High speed
- Highly reliable, ultra-fast programming Platinum-Silicide fuses
- High programming yield
- Low-current PNP inputs
- High-current open-collector and three-state outputs
- Fast chip select

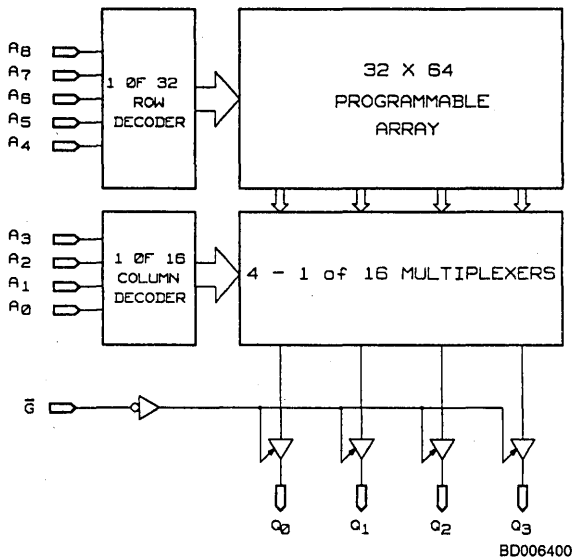
GENERAL DESCRIPTION

The Am27S12/13 (512 words by 4 bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S12) and three-state (Am27S13) output versions. These outputs

are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word-depth expansion is facilitated by an active LOW output enable ($\bar{G}$).

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Open-Collector Part Number	Am27S12A		Am27S12	
Three-State Part Number	Am27S13A		Am27S13	
Address Access Time	30 ns	40 ns	50 ns	60 ns
Operating Range	C	M	C	M

Publication # 03208 Rev. C Amendment /0
Issue Date: May 1986

Am27S18/19

256-Bit (32 x 8) Bipolar PROM

DISTINCTIVE CHARACTERISTICS

- Ultra high speed
- Highly reliable, ultra-fast programming Platinum-Silicide fuses
- High-programming yield
- Low-current PNP inputs
- High-current open collector and three-state outputs
- Fast chip select

GENERAL DESCRIPTION

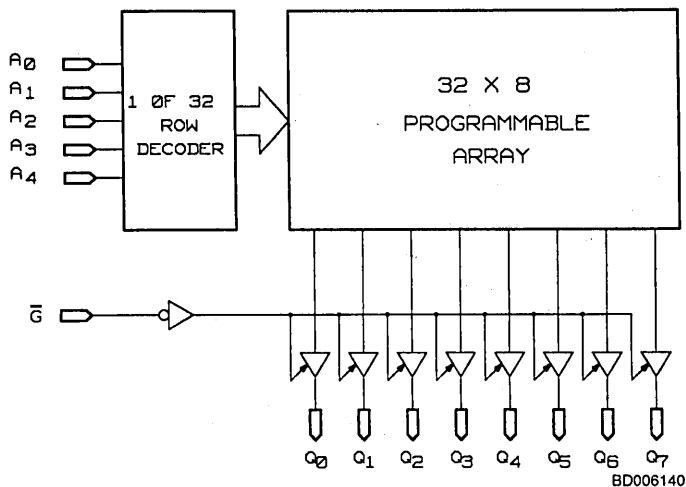
The Am27S18/19 (32-words by 8-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S18) and three-state (Am27S19) output versions. These outputs are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of

microprogrammable controls, mapping functions, code conversions, or logic replacements. Easy word depth expansion is facilitated by an active LOW output enable ($\bar{G}$).

This device is also available in a low-power version Am27LS18/19.

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Open-Collector Part Number	27S19SA		27S18A		27S18		27LS18	
Three-State Part Number	27S19SA		27S19A		27S19		27LS19	
Address Access Time	15 ns	20 ns	25 ns	35 ns	40 ns	50 ns	55 ns	70 ns
Operating Range	C	M	C	M	C	M	C	M

Am27S180/27S181/PS181 Am27S280/27S281/PS281



8,192-Bit (1024 x 8) Bipolar PROM

DISTINCTIVE CHARACTERISTICS

- Fast access time allows high system speed
- 50% power savings on deselected parts — enhances reliability through total system heat reduction
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ > 98%)
- Rapid recovery from power-down state provides minimum delay

GENERAL DESCRIPTION

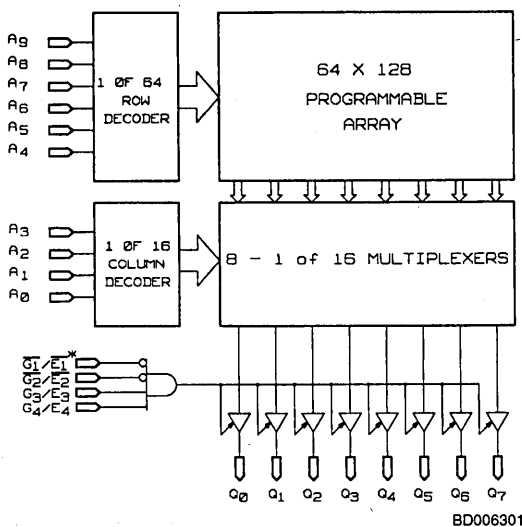
The Am27S180/27S181 (1024 words by 8 bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S180) and three-state (Am27S181) output versions. These outputs are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code con-

version, or logic replacement. Easy word-depth expansion is facilitated by both active LOW ($\overline{G_1}$ and $\overline{G_2}$) and active HIGH (G_3 and G_4) output enables.

This device is also available in a 300-mil. lateral-center DIP (Am27S280/27S281), as well as a power-switched three-state version (Am27PS181/27PS281).

BLOCK DIAGRAM



*E nomenclature applies to the power-switched versions only (Am27PSXX).

PRODUCT SELECTOR GUIDE

Open-Collector Part Number	Am27S180A, Am27S280A		Am27S180, Am27S280		-		-	
Three-State Part Number	Am27S181A, Am27S281A		Am27S181, Am27S281		Am27PS181A, Am27PS281A		Am27PS181, Am27PS281	
Address Access Time	35 ns	50 ns	60 ns	80 ns	50 ns	65 ns	65 ns	75 ns
Operating Range	C	M	C	M	C	M	C	M

Publication # 03182 Rev. C Amendment /0
Issue Date: May 1986

Am27S184/185/PS185

8,192-Bit (2048 x 4) Bipolar PROM

Am27S184/185/PS185

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- Ultra-fast access time "A" version (35 ns Max.) — Fast access time Standard version (50 ns Max.) — allow tremendous system speed improvements
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ > 98%)
- AC performance is factory tested utilizing programmed test words and columns
- Voltage and temperature compensated providing extremely flat AC performance over military range
- Member of generic PROM series utilizing standard programming algorithm

GENERAL DESCRIPTION

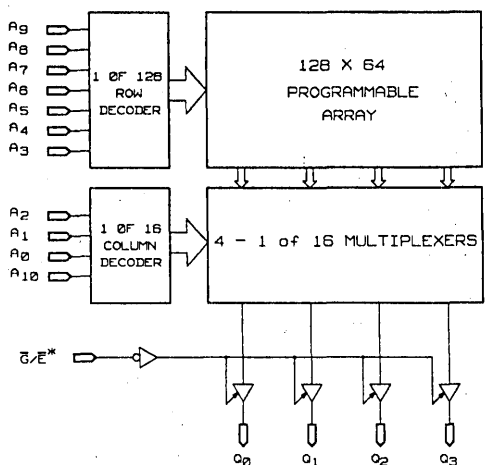
The Am27S184/185 (2048-words by 4-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S184) and three-state (Am27S185) output versions. These outputs are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of

microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy-word depth expansion is facilitated by an active LOW ($\bar{G}$) output enable.

This device is also offered in a low-power, three-state version, the Am27LS185, as well as a power-switched three-state version.

BLOCK DIAGRAM



BD006320

PRODUCT SELECTOR GUIDE

Open-Collector Part Number	27S184A		27S184					
Three-State Part Number	27S185A		27S185		27LS185		27PS185	
Address Access Time	35 ns	45 ns	50 ns	55 ns	60 ns	65 ns	50 ns	55 ns
Operating Range	C	M	C	M	C	M	C	M

Publication # 03192 Rev. C Amendment /0
Issue Date: May 1986

Am27S190/27S191/PS191/LS191 Am27S290/27S291/PS291/LS291



16,384-Bit (2048 x 8) Bipolar PROM

DISTINCTIVE CHARACTERISTICS

- Fast access time allows high system speed
- 50% power savings on deselected parts — enhances reliability through total system heat reduction (27PS devices)
- Plug in replacement for industry standard product — no board changes required
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ > 98%)
- Voltage and temperature compensated providing extremely flat AC performance over military range
- Rapid recovery from power-down state provides minimum delay (27PS devices)

GENERAL DESCRIPTION

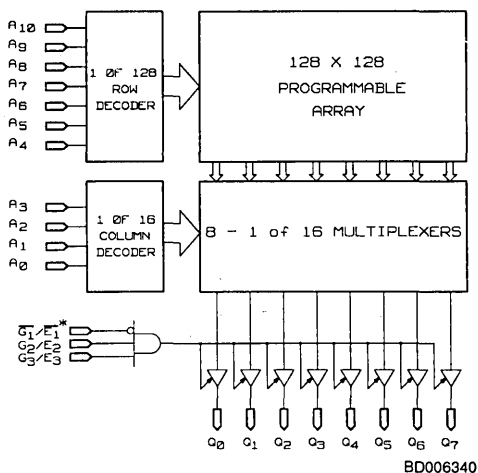
The Am27S190/191 (2048-words by 8-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S190) and three-state (Am27S191) output versions. These outputs are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word-depth expansion

is facilitated by both active LOW ($\overline{G_1}$) and active HIGH (G_2 and G_3) output enables.

This device is also available in 300-mil, lateral center DIP (Am27S290/27S291). Additionally, this device is offered in a low-power, three-state version, the Am27LS191 and Am27LS291, as well as a power-switched, three-state version, the Am27PS191 and Am27PS291.

BLOCK DIAGRAM



*E nomenclature applies to the power-switched versions only (Am27PSXXX).

PRODUCT SELECTOR GUIDE

Open-Collector Part Number	-		Am27S190A, Am27S290A		Am27S190, Am27S290		-		-		-	
Three-State Part Number	Am27S191SA, Am27S291SA		Am27S191A, Am27S291A		Am27S191, Am27S291		Am27LS191*, Am27LS291*		Am27PS191A, Am27PS291A		Am27PS191, Am27PS291	
Address Access Time (ns)	20	30	35	50	50	65	35	45	50	65	65	75
Operating Range	C	M	C	M	C	M	C	M	C	M	C	M

*Advance Information applies only to "SA" version.

Am27S190/27S191/PS191/LS191 Am27S290/27S291/PS291/LS291 Advanced Micro Devices

Am27S20/21

1,024-Bit (256 x 4) Bipolar PROM

DISTINCTIVE CHARACTERISTICS

- High speed
- Highly reliable, ultra-fast programming Platinum-Silicide fuses
- High programming yield
- Low-current PNP inputs
- High-current open-collector and three-state outputs
- Fast chip select

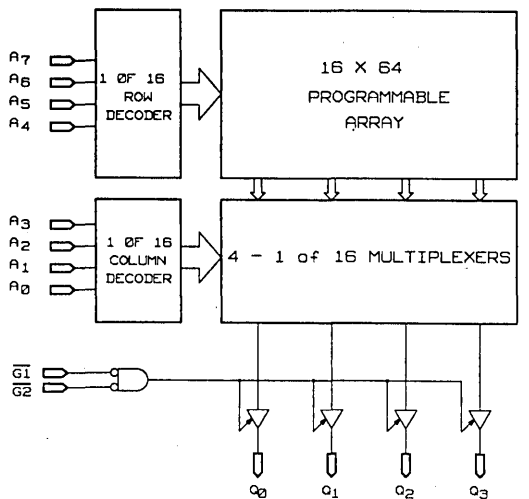
GENERAL DESCRIPTION

The Am27S20/21 (256 words by 4-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S20) and three-state (Am27S21) output versions. These outputs

are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code version, or logic replacement. Easy word-depth expansion is facilitated by active LOW ($\overline{G}_1$ and $\overline{G}_2$) output enables.

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Open-Collector Part Number	27S20A		27S20	
Three-State Part Number	27S21A		27S21	
Address Access Time	30 ns	40 ns	45 ns	60 ns
Operating Range	C	M	C	M



Am27S25

4096-Bit (512 x 8) Bipolar Registered PROM
With Preset and Clear Inputs

Am27S25

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- 'SA' version offers ultrafast AC performance (25 ns set-up and 12 ns clock-to-output)
- On-chip edge-triggered registers — ideal for pipelined microprogrammed systems
- Versatile synchronous and asynchronous enables for simplified word expansion
- Buffered common Preset ($\overline{PS}$) and Clear ($\overline{CR}$) inputs
- Slim, 24-pin, 300-mil lateral center package occupies approximately 1/3 the board space required by standard discrete PROM and register
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability
- Platinum-Silicide fuses guarantee high reliability, fast programming, and exceptionally high programming yields (typ > 98%)

GENERAL DESCRIPTION

The Am27S25 (512 words by 8 bits) is a fully decoded, Schottky array, TTL Programmable Read-Only Memory (PROM), incorporating D-type master-slave data registers on chip. This device has three-state outputs compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls and state machines.

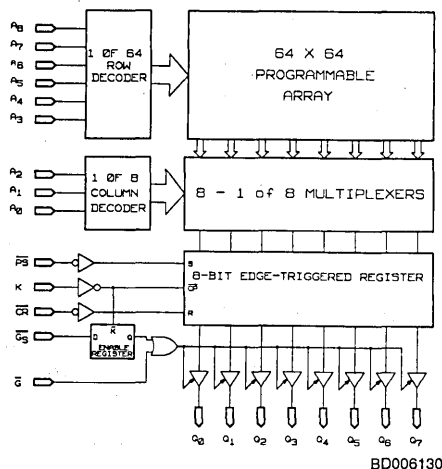
This device contains an 8-bit parallel data register in the array-to-output path which allows PROM data to be stored while other data is being addressed. This meets the

requirements for pipelined microprogrammable control stores where instruction execute and instruction fetch are performed in parallel.

To offer the system designer maximum flexibility, this device contains both asynchronous and synchronous output enables as well as common asynchronous preset and clear register controls.

Upon power-up the outputs ($Q_0 - Q_7$) will be in a floating or high-impedance state.

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Part Number	Am27S25SA		Am27S25A		Am27S25	
Address Set-up Time (ns)	25	30	30	35	50	55
Clock-to-Output Delay (ns)	12	15	20	25	27	30
Operating Range	C	M	C	M	C	M

Publication # 03300
Rev. D
Issue Date: May 1986
Amendment /0

Am27S28/27S29

4,096-Bit (512 x 8) Bipolar PROM

Am27S28/27S29

ADVANCED MICRO DEVICES

DISTINCTIVE CHARACTERISTICS

- High Speed
- Highly reliable, ultra-fast programming Platinum-Silicide fuses
- High programming yield
- Low-current PNP inputs
- High-current open-collector and three-state outputs
- Fast chip select

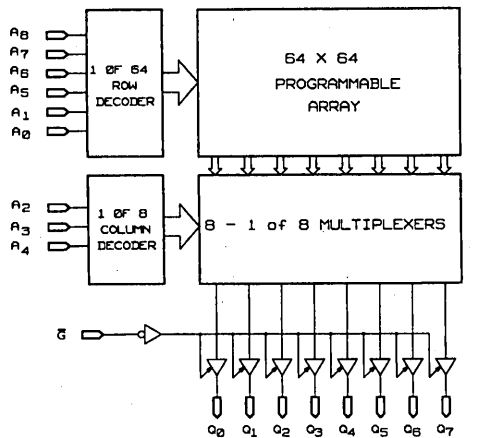
GENERAL DESCRIPTION

The Am27S28/29 (512-words by 8-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open collector (Am27S28) and three-state (Am27S29) output versions. These outputs

are compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word depth expansion is facilitated by an active LOW ($\bar{G}$) output enable.

FUNCTIONAL BLOCK DIAGRAM



BD006182

PRODUCT SELECTOR GUIDE

Open Collector Part Number	Am27S28A		Am27S28	
Three-State Part Number	Am27S29A		Am27S29	
Address Access Time	35 ns	45 ns	55 ns	70 ns
Operating Range	C	M	C	M

Am27S32/27S33

4,096-Bit (1024 x 4) Bipolar PROM



Am27S32/27S33

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- High speed
- Highly reliable, ultra-fast programming Platinum-Silicide fuses
- High programming yield
- Low-current PNP inputs
- High-current open-collector and three-state outputs
- Fast chip select

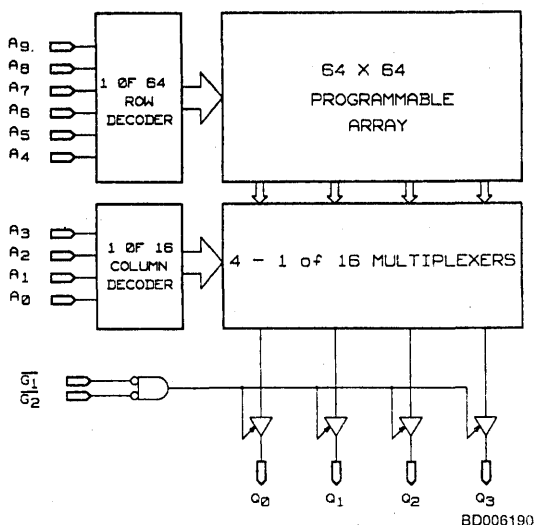
GENERAL DESCRIPTION

The Am27S32/27S33 (1024-words by 4-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device is available in both open-collector (Am27S32) and three-state (Am27S33) output versions. These outputs are compatible with low-power Schottky bus standards

capable of satisfying the requirements of a variety of microprogrammable controls, mapping functions, code conversion, or logic replacement. Easy word-depth expansion is facilitated by active LOW ($\overline{G}_1$ & $\overline{G}_2$) output enables.

BLOCK DIAGRAM



PRODUCT SELECTOR GUIDE

Open-Collector Part Number	Am27S32A		Am27S32	
Three-State Part Number	Am27S33A		Am27S33	
Address Access Time	35 ns	45 ns	55 ns	70 ns
Operating Range	C	M	C	M

Am27S35/Am27S37

8,192-Bit (1024 x 8) Bipolar Registered PROM
with Programmable INITIALIZE Input

DISTINCTIVE CHARACTERISTICS

- Slim, 24-pin, 300-mil lateral center package occupies approximately 1/3 the board space required by standard discrete PROM and register
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability
- Versatile programmable asynchronous or synchronous enable for simplified word expansion
- Buffered common INITIALIZE input either asynchronous (Am27S35) or synchronous (Am27S37)
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ. > 98%)

GENERAL DESCRIPTION

The Am27S35 and the Am27S37 (1024-words by 8-bits) are fully decoded, Schottky array, TTL Programmable Read-Only Memories (PROMs), incorporating D-type master-slave data registers on chip. These devices have three-state outputs compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls and state machines.

These devices contain an 8-bit parallel data register in the array-to-output path which allows PROM data to be stored while other data is being addressed. This meets the requirements for pipelined microprogrammable control

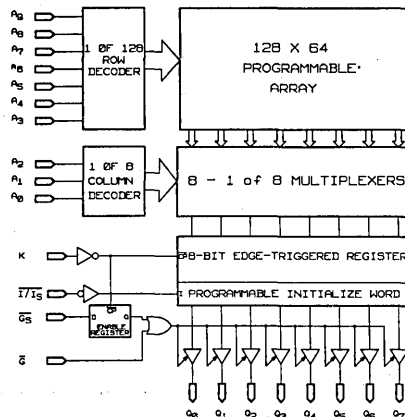
stores where instruction execute and instruction.fetch are performed in parallel.

To offer the system designer maximum flexibility, these devices contain both asynchronous ($\bar{G}$) and synchronous ($\bar{G}_S$) output enables.

These devices contain a single pin initialize function capable of loading any arbitrary microinstruction for system interrupt or initialization. On the Am27S35 this function operates asynchronously, independent of clock. The Am27S37 provides synchronous operation of this function.

Upon power-up the outputs ($Q_0 - Q_7$) will be in a floating or high-impedance state.

BLOCK DIAGRAM



BD006351

PRODUCT SELECTOR GUIDE

Part Number Asynchronous Initialize	Am27S35A		Am27S35	
Part Number Synchronous Initialize	Am27S37A		Am27S37	
Address Setup Time	35 ns	40 ns	40 ns	45 ns
Clock-to-Output Delay	20 ns	20 ns	25 ns	30 ns
Operating Range	C	M	C	M

Am27S41/27PS41

16,384-Bit (4,096 x 4) Bipolar PROM



Am27S41/27PS41

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- Ultra-fast access time "A" version (35 ns Max.) — Fast access time Standard version (50 ns Max.) — allow
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ > 98%)
- AC performance is factory tested utilizing programmed test words and columns
- Voltage and temperature compensated providing extremely flat AC performance over military range
- Member of generic PROM series utilizing standard programming algorithm

GENERAL DESCRIPTION

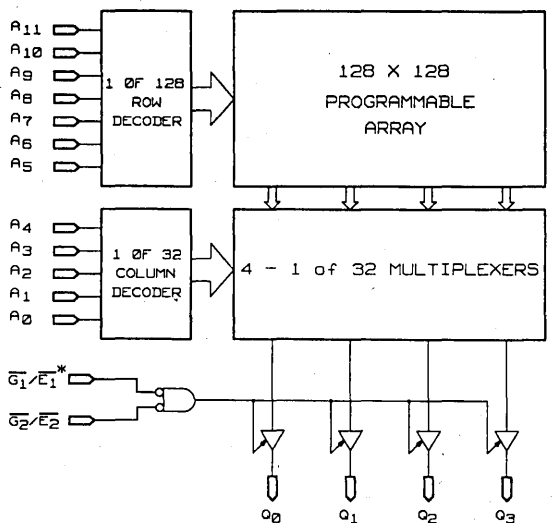
The Am27S41 (4,096-words by 4-bits) is a Schottky TTL Programmable Read-Only Memory (PROM).

This device has three-state outputs compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls,

mapping functions, code conversion, or logic replacement. Easy-word depth expansion is facilitated by active LOW ($\overline{G}_1$ & $\overline{G}_2$) output enables.

This device is also offered in a power-switched version, the Am27PS41.

BLOCK DIAGRAM



*E nomenclature applies only to Am27PS power-switched versions.

PRODUCT SELECTOR GUIDE

Part Number	27S41A		27S41		27PS41	
Address Access Time	35 ns	50 ns	50 ns	65 ns	50 ns	65 ns
Operating Range	C	M	C	M	C	M

Publication # 02122 Rev. C Amendment /0
Issue Date: May 1986

Am27S45/Am27S47

16,384-Bit (2048 x 8) Bipolar Registered PROM
with Programmable INITIALIZE Input

DISTINCTIVE CHARACTERISTICS

- "SA" version offers superior performance with 25 ns setup time and 10 ns clock-to-output delay*
- Slim, 24-pin, 300-mil lateral center package occupies approximately 1/3 the board space required by standard discrete PROM and register
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability
- Versatile programmable asynchronous or synchronous enable for simplified word expansion
- Buffered common INITIALIZE input either asynchronous (Am27S45) or synchronous (Am27S47)
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ. > 98%)

GENERAL DESCRIPTION

The Am27S45 and the Am27S47 (2048-words by 8-bits) are fully decoded, Schottky array, TTL Programmable Read-Only Memories (PROMs), incorporating D-type master-slave data registers on chip. These devices have three-state outputs compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls and state machines.

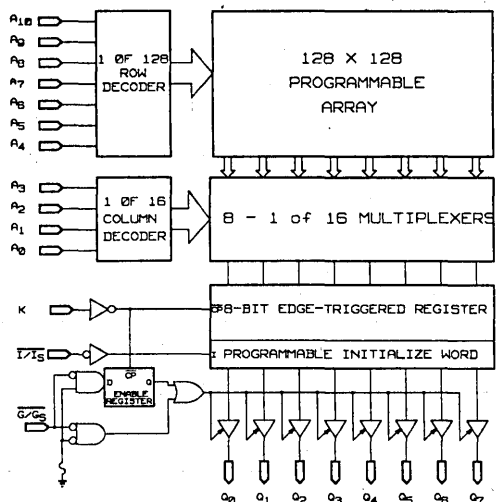
These devices contain an 8-bit parallel data register in the array-to-output path which allows PROM data to be stored while other data is being addressed. This meets the requirements for pipelined microprogrammable control stores where instruction execute and instruction fetch are performed in parallel.

To Offer the system designer maximum flexibility, these devices contain a user programmable asynchronous or synchronous output enable. The unprogrammed state of the enable pin operates as an Asynchronous Enable ($\bar{G}$) input. An architecture word permits the programming of the functionality of this pin to Synchronous Enable ($\bar{G}_S$).

These devices contain a single pin initialize function capable of loading any arbitrary microinstruction for system interrupt or initialization. On the Am27S45 this function operates asynchronously, independent of clock. The Am27S47 provides synchronous operation of this function.

If the architecture has been programmed to synchronous enable, upon power-up the outputs ($Q_0 - Q_7$) will be in a floating or high-impedance state.

BLOCK DIAGRAM



BD006381

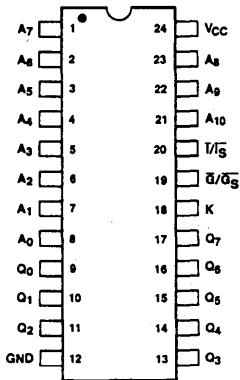
Publication # 03186 Rev. C Amendment /0
Issue Date: May 1986

PRODUCT SELECTOR GUIDE

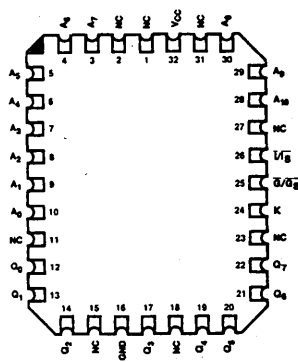
Part Number Asynchronous Initialize	27S45SA*		27S45A		27S45	
Part Number Synchronous Initialize	27S47SA*		27S47A		27S47	
Address Setup Time (ns)	25	28	40	45	45	50
Clock-to-Output Delay (ns)	10	12	20	25	25	30
Operating Range	C	M	C	M	C	M

CONNECTION DIAGRAMS

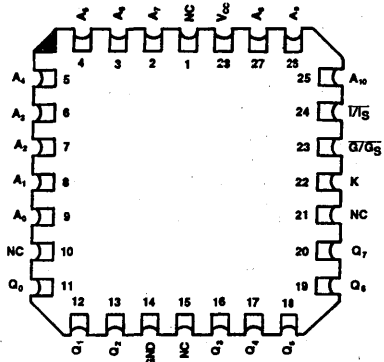
Top View



CD000461



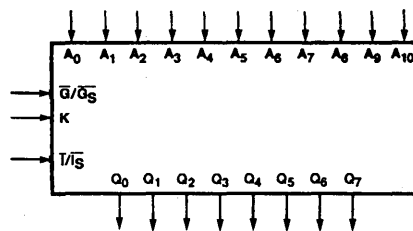
CD000471



CD009630

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



LS000051

Am27S55

32,768-Bit (4096 x 8) Bipolar Registered PROM

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- "A" version offers superior performance with 20 ns set-up time and 10 ns clock-to-output delay
- Slim, 24-pin, 300-mil lateral center package occupies approximately $\frac{1}{3}$ the board space required by standard discrete PROM and register
- Consumes approximately $\frac{1}{2}$ the power of separate PROM/register combination for improved system reliability
- User-programmable for Asynchronous Enable, Synchronous Enable, Asynchronous Initialize, or Synchronous Initialize
- Platinum-Silicide fuses guarantee high reliability, fast programming, and exceptionally high programming yields (Typ. > 98%)

GENERAL DESCRIPTION

The Am27S55 (4096 words by 8 bits) is a fully decoded Schottky Array TTL Programmable Read-Only Memory (PROM) incorporating D-type master-slave data registers on-chip. This device has three-state outputs compatible with low-power Schottky bus standards capable of satisfying the requirements of a variety of microprogrammable controls and state machines.

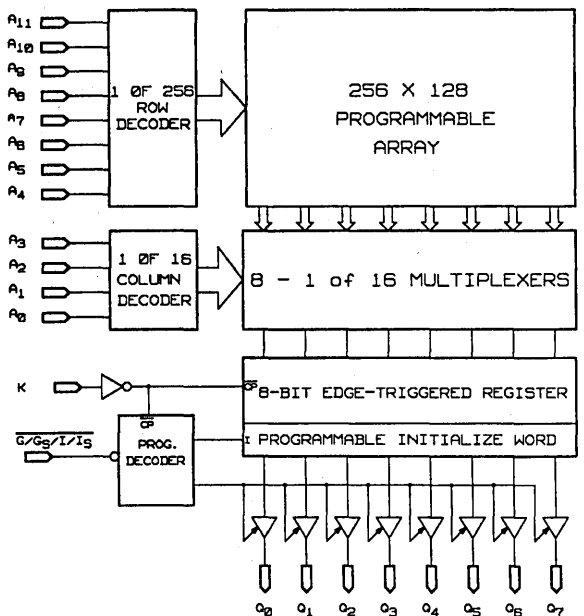
This device contains an 8-bit parallel data register in the array-to-output path which allows PROM data to be stored while other data is being addressed. This meets the requirements for pipelined microprogrammable control

stores where instruction execute and instruction fetch are performed in parallel.

To offer the system designer maximum flexibility, this device contains a single programmable multi-functional input ($\bar{G}/\bar{G}_S/\bar{I}/\bar{I}_S$). The unprogrammed state of this pin operates as an Asynchronous Enable ($\bar{G}$) input. An architecture word permits the programming of the functionality of this pin to Synchronous Enable ($\bar{G}_S$), Asynchronous Initialize ($\bar{I}$), or Synchronous Initialize ($\bar{I}_S$).

If the architecture has been programmed to synchronous enable, upon power-up the outputs ($Q_0 - Q_7$) will be in a floating or high-impedance state.

BLOCK DIAGRAM

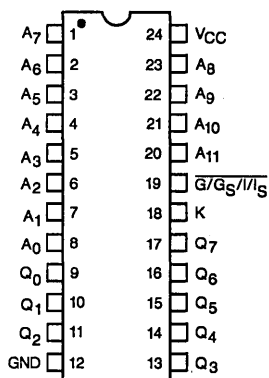


BD006440

PRODUCT SELECTOR GUIDE

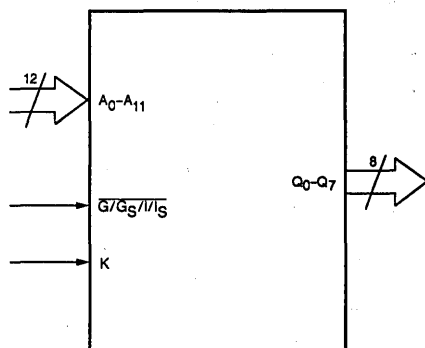
Part Number	Am27S55A		Am27S55	
Address Setup Time (ns)	20	25	25	30
Clock-to-Output Delay (ns)	10	13	13	16
Operating Range	C	M	C	M

CONNECTION DIAGRAM Top View



CD009590

LOGIC SYMBOL



LS002450

Am27S65

4096-Bit (1024 x 4) Bipolar Registered PROM
with SSR™ Diagnostics Capability

DISTINCTIVE CHARACTERISTICS

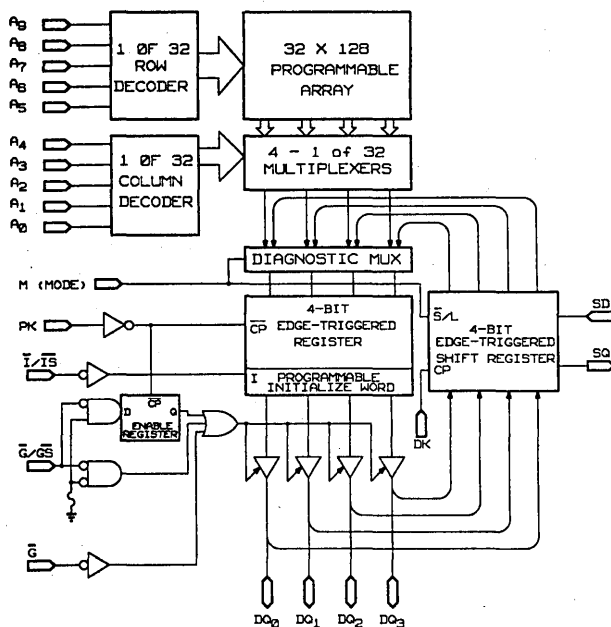
- On-chip diagnostic shift register for serial observability and controllability of the output register
- User-programmable Enable Pin for Asynchronous or Synchronous Enable operation
- User-programmable Initialization Pin for Asynchronous or Synchronous Initialize operation
- Slim, 24-pin, 300-mil lateral center package permits a reduction in board space over standard discrete PROM and registers
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ. > 98%)
- Increased drive capability, 24 mA I_{OL}

GENERAL DESCRIPTION

This device contains a 4-bit parallel data register in the array-to-output path intended for normal registered data operations. In parallel with the output data registers is another 4-bit register with shifting capability, called a shadow register. As the name implies, the shadow register is intended to operate in the background of the normal output data register. This shadow register can be used in a systematic way to control and observe the output data register to exercise desired system functions during a diagnostic test mode.

To offer the system designer maximum flexibility, this device contains user-programmable architecture for Enable and Initialize. The unprogrammed state of these pins operates as Asynchronous inputs ($\bar{G}$) and ($\bar{I}$), respectively. An architecture word permits the programming of the functionality of these pins to Synchronous Enable ($\bar{G}$ S) and Synchronous Initialize ($\bar{I}$ S). A non-programmable Asynchronous Enable ($\bar{G}$) is also provided.

BLOCK DIAGRAM



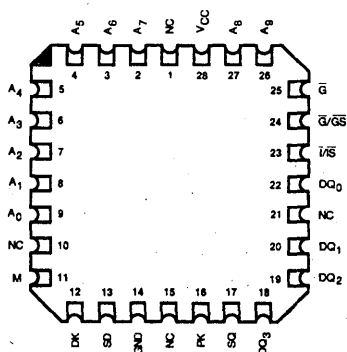
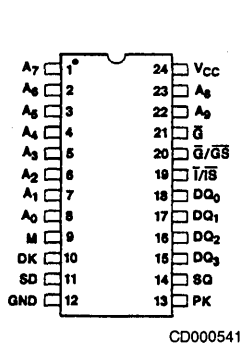
BD005830

PRODUCT SELECTOR GUIDE

Part Number	27S65A	27S65	27S65A	27S65
Address Set-up Time	23 ns	30 ns	27 ns	35 ns
Clock-to-Output Delay	10 ns	15 ns	13 ns	20 ns
Operating Range	C	C	M	M

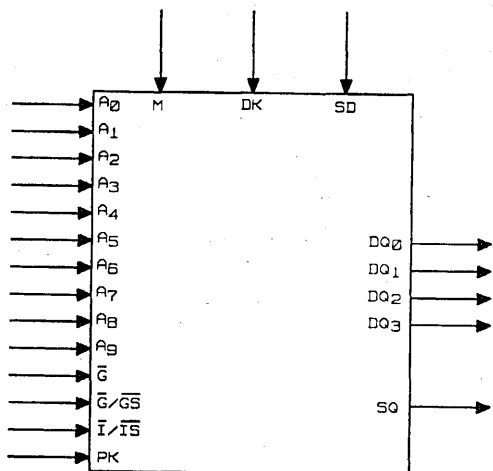
CONNECTION DIAGRAMS

Top View



Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



Am27S75

8192-Bit (2048 x 4) Bipolar Registered PROM
with SSR™ Diagnostics Capability



Am27S75

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

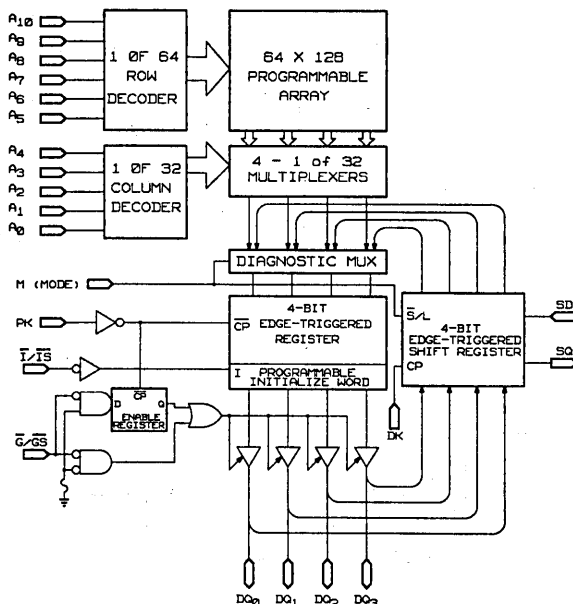
- On-chip diagnostic shift register for serial observability and controllability of the output register
- User-programmable synchronous and asynchronous Enables
- User-programmable for synchronous or asynchronous Initialize
- Slim, 24-pin, 300-mil lateral center package permits a reduction in board space over standard discrete PROM and registers.
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability.
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ. > 98%).
- Increased drive capability, 24 mA I_{OL}

GENERAL DESCRIPTION

This device contains a 4-bit parallel data register in the array-to-output path intended for normal registered data operations. In parallel with the output data registers is another 4-bit register with shifting capability, called a shadow register. As the name implies, the shadow register is intended to operate in the background of the normal output data register. This shadow register can be used in a systematic way to control and observe the output data register to exercise desired system functions during a diagnostic test mode.

To offer the system designer maximum flexibility, this device contains user programmable architecture for Enable and Initialize. The unprogrammed state of these pins operates as Asynchronous inputs ($\bar{G}$) and ($\bar{I}$) respectively. An architecture word permits the programming of the functionality of these pins to Synchronous Enable ($\bar{G}$ S) and Synchronous Initialize ($\bar{I}$ S).

BLOCK DIAGRAM



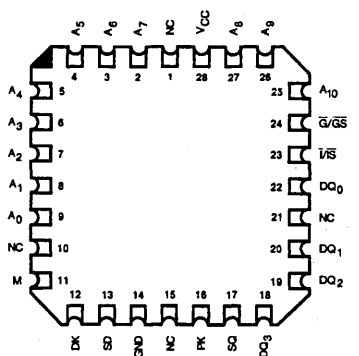
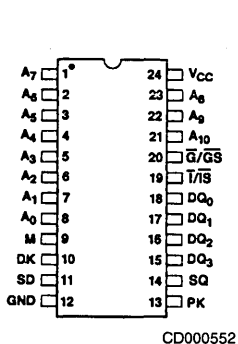
BD005840

PRODUCT SELECTOR GUIDE

Part Number	27S75A	27S75	27S75A	27S75
Address Set up Time	25 ns	30 ns	30 ns	35 ns
Clock-to-Output Delay	12 ns	15 ns	17 ns	20 ns
Operating Range	C	C	M	M

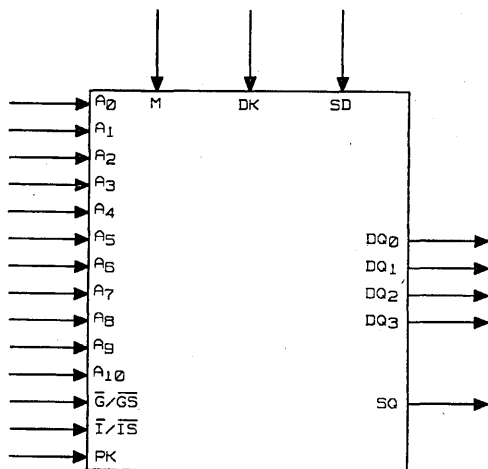
CONNECTION DIAGRAMS

Top View



Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



LS002110

Am27S85

16,384-Bit (4096 x 4) Registered PROM
with SSR™ Diagnostics Capability

DISTINCTIVE CHARACTERISTICS

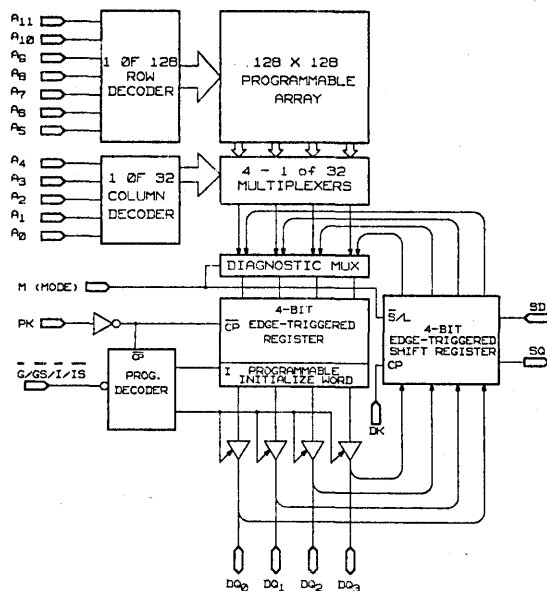
- On-chip diagnostic shift register for serial observability and controllability of the output register
- User-programmable for Asynchronous Enable, Synchronous Enable, Asynchronous Initialize, or Synchronous Initialize
- Slim, 24-pin, 300-mil lateral center package occupies approximately 1/3 the board space required by standard discrete PROM and register.
- Consumes approximately 1/2 the power of separate PROM/register combination for improved system reliability.
- Platinum-Silicide fuses guarantee high reliability, fast programming and exceptionally high programming yields (typ. > 98%).
- Increased drive capability, 24 mA I_{OL}

GENERAL DESCRIPTION

This device contains a 4-bit parallel data register in the array-to-output path intended for normal registered data operations. In parallel with the output data registers is another 4-bit register with shifting capability, called a shadow register. As the name implies, the shadow register is intended to operate in the background of the normal output data register. This shadow register can be used in a systematic way to control and observe the output data register to exercise desired system functions during a diagnostic test mode.

To offer the system designer maximum flexibility, this device contains a single programmable multi-functional input ($\bar{G}/\bar{GS}/I/\bar{IS}$). The unprogrammed state of this pin operates an Asynchronous Enable ($\bar{G}$) input. An architecture word permits the programming of the functionality of this pin to Synchronous Enable ($\bar{GS}$), Asynchronous Initialize (I), or Synchronous Initialize ($\bar{IS}$).

BLOCK DIAGRAM



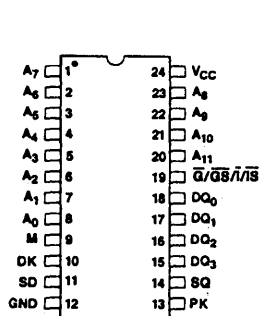
BD005850

PRODUCT SELECTOR GUIDE

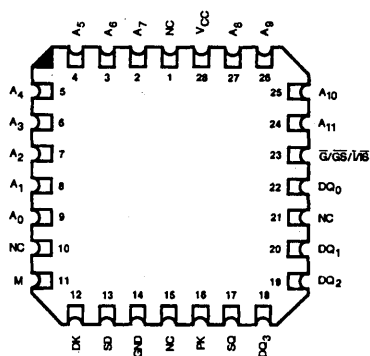
Part Number	27S85A	27S85	27S85A	27S85
Address Set-up Time	27 ns	35 ns	30 ns	40 ns
Clock-to-Output Delay	12 ns	15 ns	17 ns	20 ns
Operating Range	C	C	M	M

CONNECTION DIAGRAMS

Top View



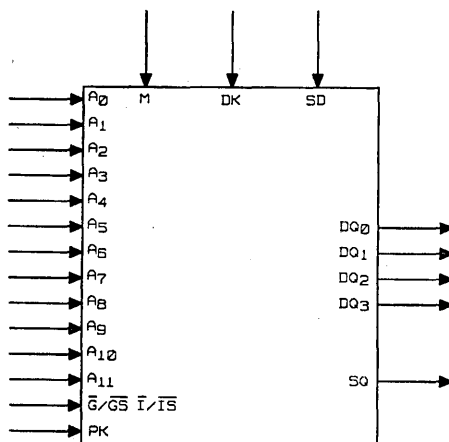
CD000561



CD004902

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



LS002111

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 (SSRTM) 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.

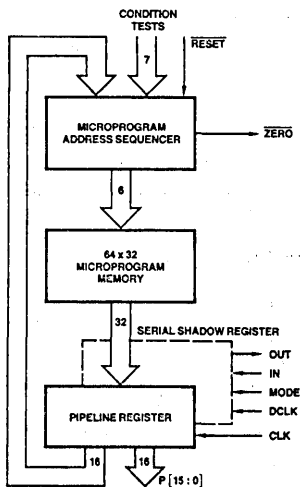
The Am29PL141 FPC also allows distribution of intelligent control throughout the system. It off-loads the central controller by distributing FPCs as the control for various

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

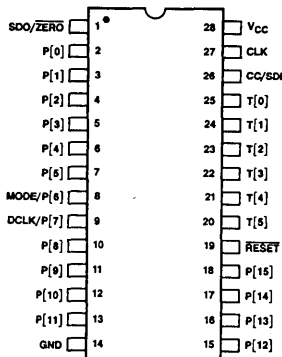


BDR02340

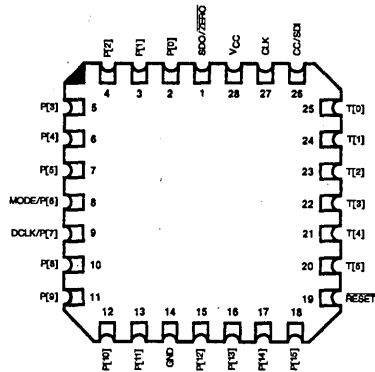
RELATED PRODUCTS

Part No.	Description
Am2914	Vectored Priority Interrupt Controller
Am29100	Controller Family Products

CONNECTION DIAGRAMS Top View



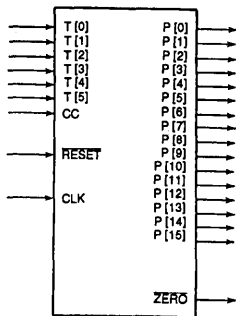
CDR04480



CD009110

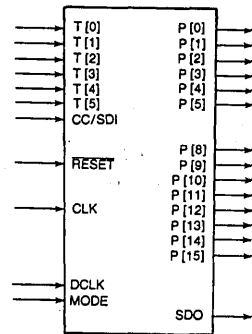
Note: Pin 1 is marked for orientation.

LOGIC SYMBOLS



LS002131

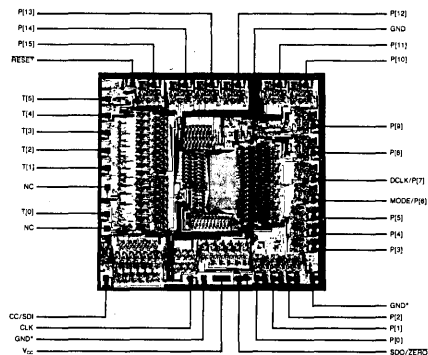
Normal Configuration



LS002140

SSR™ Diagnostics Configuration

METALLIZATION AND PAD LAYOUT



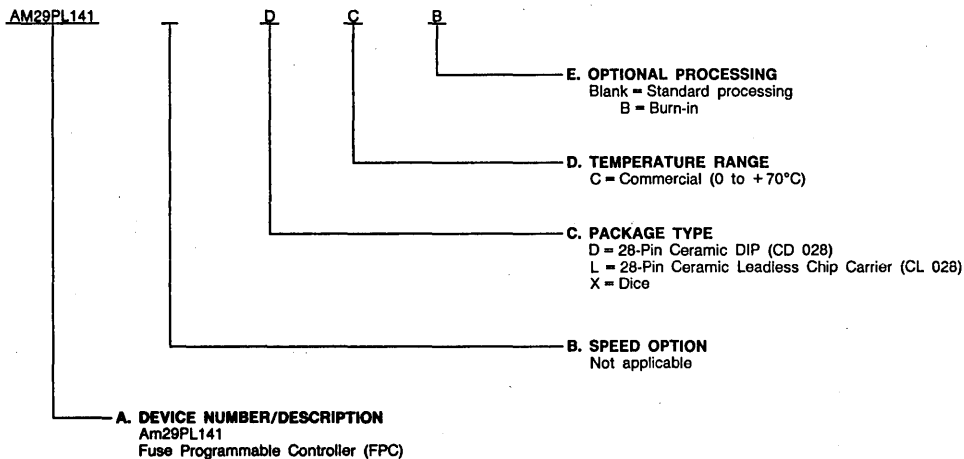
Die Size: 0.211" x 0.202"
Gate Count: 600 Equivalent Gates and 2K of PROM

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

Valid Combinations	
AM29PL141	DC, DCB, LC, XC

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, 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

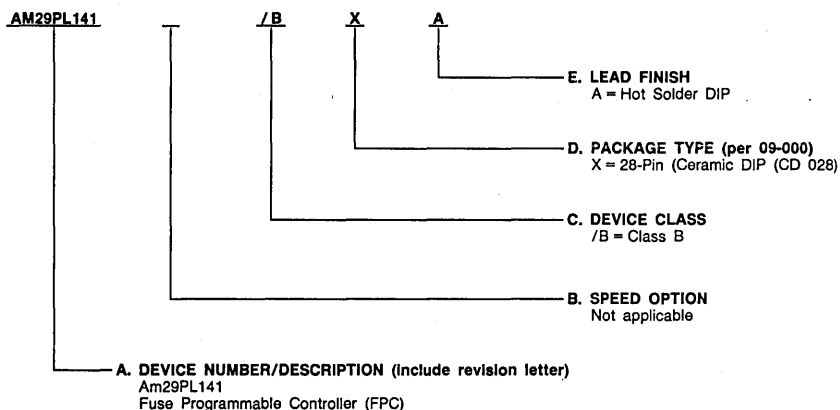
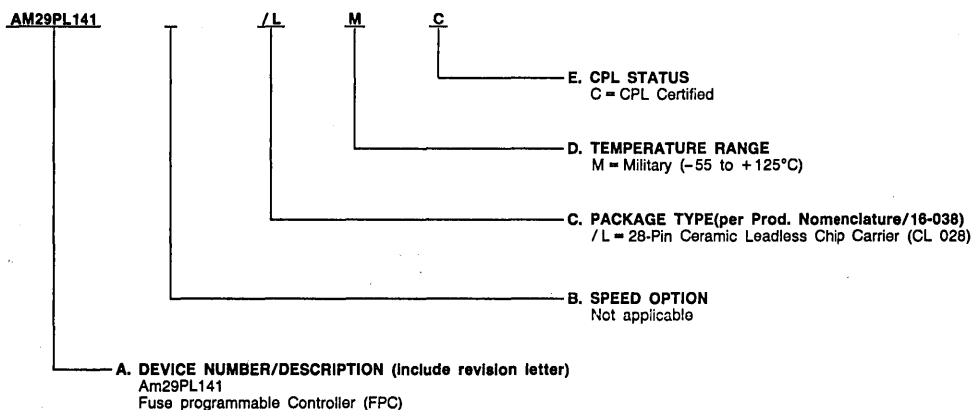
CPL Products: A. Device Number

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.)

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.

ZERO [SDO]

Zero output. A Low state indicates that the CREG value is zero. (Note: In the SSR diagnostic configuration, **ZERO** becomes the Serial Data output SDO. This change is only on the output pin; internally, the zero detect functions is unchanged.)

FUNCTIONAL DESCRIPTION

Figure 1, the block diagram of the Am29PL141 FPC, shows logic blocks and interconnecting buses. These allow parallel performance of different operations in a single microinstruction. The FPC consists of four main logic blocks: the microprogram memory, microaddress control logic, condition code selection logic, and microinstruction decode. A fifth optional block is the Serial Shadow Register (SSR).

The microprogram memory contains the user-defined instruction flow and output sequence. The microaddress control logic addresses the microprogram memory. This control logic supports high-level microinstruction functions including conditional branches, subroutine calls and returns, loops, and multiway branches. The condition code selection logic selects the condition code input to be tested when a conditional microinstruction is executed. The polarity of the selected condition code input is controlled by the POL bit in the microword. The microinstruction decode generates the control signals necessary to perform the microinstruction specified by

the microinstruction part ($P[31 : 16]$) of the microword. The SSR enables in-system testing that allows isolation of problems down to the IC level.

MICROPROGRAM MEMORY

The FPC microprogram memory is a 64-word by 32-bit PROM with a 32-bit pipeline register at its output. The upper 16 bits ($P[31 : 16]$) of the pipeline register stay internal to the FPC and form the microinstruction to control address sequencing. The format for microinstructions is: a one-bit synchronous Output Enable OE, a five-bit OPCODE, a one-bit test polarity select POL, a three-bit TEST condition select field, and a six-bit immediate DATA field. The DATA field is used to provide branch addresses, test input masks, and counter values.

The lower 16 bits ($P[15 : 0]$) of the pipeline register are brought out as user-defined, general purpose control outputs. The upper eight control outputs ($P[15 : 8]$) are three-stated when OE is programmed as a LOW. The lower eight control bits ($P[7 : 0]$) are always enabled.

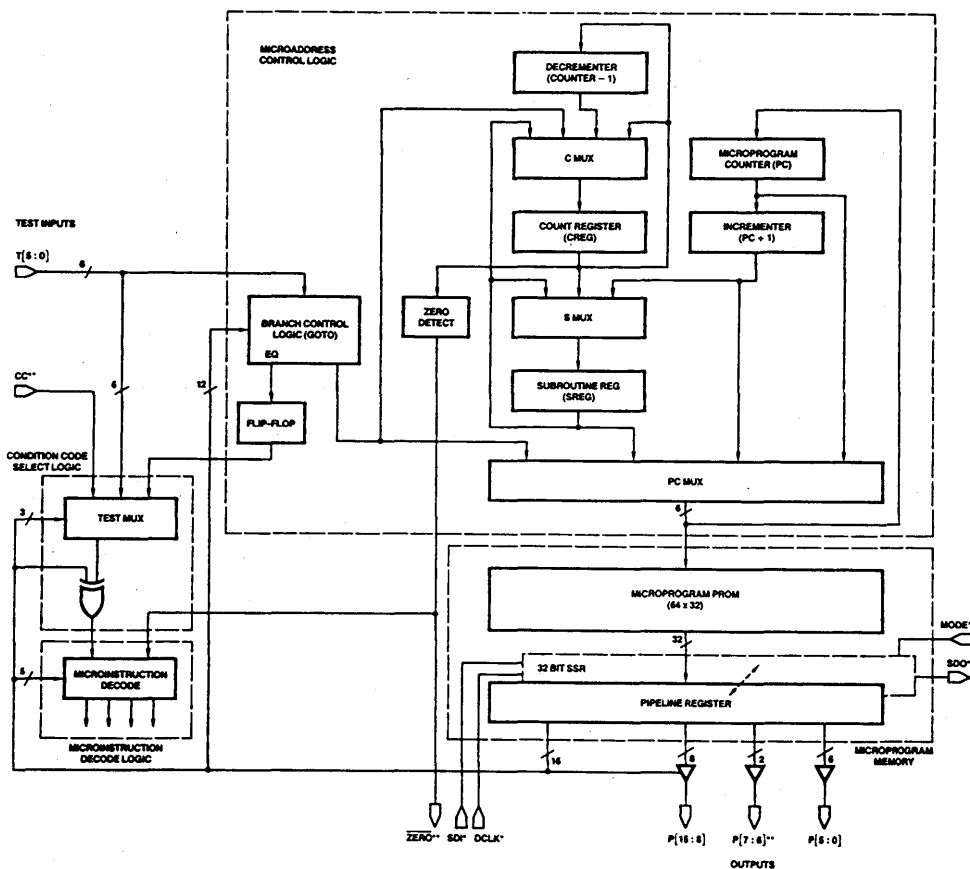


Figure 1. Am29PL141 Block Diagram

*Note: These pins available only in SSR mode.

**Note: These pins available only in normal mode.

BDR02330

MICROADDRESS CONTROL LOGIC

The microaddress control logic consists of five smaller logic blocks. These are:

- PC MUX - The microprogram counter multiplexer
- P CNTR - Microprogram counter (PC) and incrementer (PC + 1)
- SUBREG - Subroutine register (SREG) with subroutine mux (S MUX)
- CNTR - Count register (CREG) with counter mux (C MUX), decremter (COUNTER-1) and zero detect
- GOTO - Specialized branch control logic

The PC MUX is a six-bit, four-to-one multiplexer. It selects either the PC, PC+1, SREG, or GOTO output as the next microaddress input to the microprogram memory and to the PC. The PC thus always contains the address of the microinstruction in the pipeline register. During a Reset, the PC MUX output is forced to all ones, selecting location 63 of the microprogram memory.

The P CNTR block consists of a six-bit register (PC) driving a six-bit combinatorial incrementer (PC+1). Either the present or the incremented values of PC can address the microprogram PROM. The incremented value of PC can be saved as a subroutine return address. The present PC value can address the microprogram PROM when waiting for a condition to become valid. PC+1 addresses the microprogram PROM for sequential microprogram flow, for unconditional microinstructions, and as a default for conditional microinstructions.

The SUBREG block consists of a six-bit, three-to-one multiplexer (S MUX) driving a six-bit register (SREG). The three possible SREG inputs are PC+1, CREG, and SREG. SREG normally operates as a one-deep stack to save subroutine return addresses. PC+1 is the input source when performing subroutine calls and PC MUX is the output destination when performing return from subroutine.

The CNTR block consists of a six-bit, four-to-one multiplexer (C MUX); driving a six-bit register (CREG); a six-bit, combinatorial decremter (COUNTER-1); and a zero detection circuit. The CNTR logic block is typically used for timing functions and iterative loop counting.

The SUBREG and CNTR can be considered as one logic block because of their unique interaction. To explain this interaction, notice that both have an additional input source and output destination not used in typical operation—each other. This allows the CREG to be an additional stack location when not used for counting, and the SREG to be a nested count location when not used as a stack location. Thus, the SREG and CREG can operate in three different modes:

1. As a separate one-deep stack and counter.
2. As a two-deep stack.
3. As a two-deep nested counter.

The GOTO logic block serves three functions:

1. It provides a six-bit count value from the DATA Field in the pipeline register (P[21 : 16]) or from the TEST inputs (T[5 : 0]) masked by the DATA Field (P[21 : 16]). (This is represented by T*M.)
2. It provides a branch address from the DATA Field in the pipeline register (P[21 : 16]) or from the TEST inputs (T[5 : 0]) masked by the DATA Field (P[21 : 16]). (This is represented by T*M.)

3. It compares the TEST inputs

(T[5 : 0]) masked by the DATA Field (P[21 : 16]), called T*M, to the CONSTANT Field from the pipeline register (P[27 : 22]). If a match occurs, the EQ Flip-flop is set. EQ remains unchanged if there is no match. Constant field bits that correspond to masked test bits must be ZERO.

The EQ flag can be tested by the condition code selection logic. Multiple tests of any group of T inputs in a manner analogous to Sum-of-Products can be performed since a no match comparison does not reset the EQ flag. Any conditional branch on EQ will reset the EQ flag. Conditional returns on EQ will not change the EQ flag. RESET input LOW will reset the EQ flag.

NOTE: A zero in the DATA Field blocks the corresponding bit in the TEST Field; a one activates the corresponding bit.

The constant field bits that correspond to masked test field bits must be zero. A zero is substituted for masked test field bits. The 'POL' bit is a "don't care" when using test inputs to load registers.

CONDITION CODE SELECTION LOGIC

The condition code selection logic consists of an eight-to-one multiplexer. The eight test condition inputs are the device inputs (CC and T[5 : 0]) and the EQ flag. The TEST field P[24 : 22] selects one of the eight conditions to test.

The polarity bit POL in the microinstructions allows the user to test for either a true or false condition. Refer to Table 2 for details.

MICROINSTRUCTION DECODE

The microinstruction decoder is a PLA that generates the control for 29 different microinstructions. The decoder's inputs include the OPCODE Field (P[30 : 26]), the zero detection output from the CNTR, and the selected test condition code from the conditional code selection logic.

Am29PL141 SSR DIAGNOSTICS OPTION

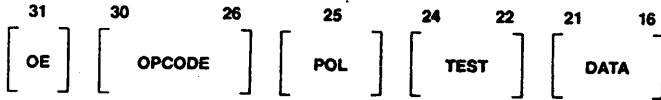
As a programmable option, the Am29PL141 FPC may be configured to contain Serial Shadow Register (SSR) diagnostics capability. SSR diagnostics is a simple, straightforward method of in-system testing that allows isolation of problems down to the IC level.

The SSR diagnostics configuration activates a 32-bit-wide, D-type register, called a "shadow" register, on the pipeline register inputs. The shadow register can be serially loaded from the SDI pin, parallel loaded from the pipeline register, or held. The pipeline register can be loaded from the microprogram memory in normal operation or from the shadow register during diagnostics. A redefinition of four device pins is required to control the different diagnostics functions. CC also functions as the Serial Data Input (SDI), ZERO becomes the Serial Data Output (SDO), P[7] becomes the diagnostic clock (DCLK), and P[6] becomes the diagnostic mode control (MODE). The various diagnostic and normal modes are shown in Table 1.

Serially loading a test microinstruction into the shadow register and parallel loading the shadow register contents into the pipeline register forces execution of the test microinstruction. The result of the test microinstruction can then be clocked into the pipeline register, as in normal operation mode, parallel loaded into the shadow register, and serially shifted out for system diagnostics.

The general microinstruction format is shown below:

Am29PL141 General Microinstruction Format



DFR00730

WHERE:

- OE = Synchronous Output Enable for P[15:8].
- OPCODE = A five-bit opcode field for selecting one of the twenty-eight single data field microinstructions.
- POL = A one-bit test condition polarity select.
0 = Test for true (HIGH) condition.
1 = Test for false (LOW) condition.
- TEST = A three-bit test condition select.

TEST[2:0]

000
001
010
011
100
101
110
111

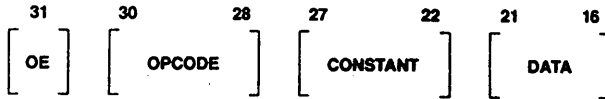
UNDER TEST

T[0]
T[1]
T[2]
T[3]
T[4]
T[5]
CC
EQ

- DATA = A six-bit conditional branch microaddress, test input mask, or counter value field designated as PI in microinstruction mnemonics.

The special two data field comparison microinstruction format is shown below:

Am29PL141 Comparison Microinstruction Format



DFR00740

WHERE:

- OE = Synchronous Output Enable for P[15:8].
- OPCODE = Compare microinstruction (binary 100).
- CONSTANT = A six-bit constant for equal to comparison with T*M.
- DATA = A six-bit mask field for masking the incoming T[5:0] inputs.

TABLE 1.

Inputs				Outputs			Operation
SDI	MODE	DCLK	CLK	SDO	Shadow Register	Pipeline Register	
D	L	↑	H,L,↑	S ₀	S _{i-1} ← S _i S ₃₁ ← D	Hold	Serial Right Shift Register
X	L	H,L,↓	↑	S ₀	Hold	P _i ← PROM _i	Normal Load Pipeline Register from PROM
L	H	↑	H,L,↓	L	S _i ← P _i	Hold	Load Shadow Register from Pipeline* Register
X	H	H,L,↓	↑	SDI	Hold	P _i ← S _i	Load Pipeline Register from Shadow Register
H	H	↑	H,L,↓	H	Hold	Hold	Hold Shadow Register

*S7, S6 are undefined. S₁₅–S₈ load from the source driving pins P[15]–P[8]. If P[31] in the microword is a ONE, S₁₅–S₈ are loaded from the pipeline register. If P[31] in the microword is a ZERO, S₁₅–S₈ are loaded from an external source.

FUNCTION TABLE DEFINITIONS

INPUTS

H = HIGH X = Don't Care
L = LOW ↑ = LOW-to-HIGH transition
 ↓ = High-to-Low transition

TABLE 2.

Input Condition Being Tested	POL	Condition
0	0	Fail
0	1	Pass
1	0	Pass
1	1	Fail

Am29PL141 MICROINSTRUCTION SET DEFINITION

● = Other instruction
 ○ = Instruction being described
 ε = Register in part

P = Test Pass
 F = Test Fail
 M,N are arbitrary values in the CREG or SREG

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
19	GOTOPL	If (cond) Then Go To Pipeline Conditional branch to the address in the PL (DATA field). The EQ flag will be reset if the test field selects it and the condition passes.		If (cond = true) Then PC = PL(data) Else PC = PC + 1
0B	GOTOPLZ	If (CREG = 0) Then Go To Pipeline Conditional branch, when the CREG is equal to zero, to the address in the PL (DATA field). This instruction does not depend on the pass/fail condition. The EQ flag will be reset if the test field selects it and the CREG is equal to zero.		If (CREG = 0) Then PC = PL(data) Else PC = PC + 1
0F	GOTOTM	If (cond) Then Go To TM Conditional branch to the address defined by the T*M (T[5:0] under bitwise mask from the DATA field). This microinstruction is intended for multiway branches. The EQ flag will be reset if the test field selects it and the condition passes.		If (cond = true) Then PC = T*M Else PC = PC + 1
18	FORK	If (cond) Then Go To Pipeline Else Go To (SREG) Conditional branch to the address in the PL (DATA field) or the SREG. A branch to PL is taken if the condition is true and a branch to SREG if false. The EQ flag will be reset if the test field selects it and the condition passes.		If (cond = true) Then PC = PL(data) Else PC = SREG

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
1C	CALPL	If (cond) Then Call Pipeline Conditional jump to subroutine at the address in the PL (DATA field). The PC + 1 is pushed into the SREG as the return address. The EQ flag will be reset if the test field selects it and the condition passes.		<pre> If (cond = true) Then SREG = PC + 1 PC = PL(data) Else PC = PC + 1 </pre>
1D	CALPLN	If (cond) Then Call Pipeline, Nested Conditional jump to subroutine at the address in the PL (DATA field) nested. The SREG and CREG are treated as a two-deep stack, the PC + 1 is pushed into the SREG as the return address and the previous SREG value is transferred into the CREG as a nested return address. The EQ flag will be reset if the test field selects it and the condition passes.		<pre> If (cond = true) Then CREG = SREG SREG = PC + 1 PC = PL(data) Else PC = PC + 1 </pre>
1E	CALTM	If (cond) Then Call TM Conditional jump to subroutine at the address specified by the T*M (T[5:0] under bitwise mask from the DATA field). The PC + 1 is pushed into the SREG as the return address. The EQ flag will be reset if the test field selects it and the condition passes.		<pre> If (cond = true) Then SREG = PC + 1 PC = T*M Else PC = PC + 1 </pre>
1F	CALTMN	If (cond) Then Call TM, Nested Conditional jump to subroutine at the address specified by the T*M (T[5:0] under bitwise mask from the DATA field) nested. The PC + 1 is pushed into the SREG as the return address and the previous SREG value is transferred into the CREG as a nested return address. The EQ flag will be reset if the test field selects it and the condition passes.		<pre> If (cond = true) Then CREG = SREG SREG = PC + 1 PC = T*M Else PC = PC + 1 </pre>

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
04	LDPL	If (cond) Then Load Pipeline Conditional Load the CREG from the PL (DATA field).		If (cond = true) Then CREG = PL(data) PC = PC + 1 Else PC = PC + 1
			PF001510	
05	LDPLN	If (cond) Then Load Pipeline, Nested Conditional load the CREG from the PL (DATA field) nested. The CREG and SREG are treated as a two-deep nested count register, the previous CREG value is pushed into the SREG as a nested count, and the CREG is loaded from PL.		If (cond = true) Then SREG = CREG CREG = PL(data) PC = PC + 1 Else PC = PC + 1
			PF001500	
06	LDTM	If (cond) Then Load TM Conditional load the CREG from the T*M (T[5:0] inputs under bitwise mask from the DATA field).		If (cond = true) Then CREG = T*M PC = PC + 1 Else PC = PC + 1
			PF001520	
07	LDTMN	If (cond) Then Load TM, Nested Conditional load the CREG from the T*M (T[5:0] inputs under bitwise mask from the DATA field) nested. The SREG and CREG are treated as a two-deep nested count register, the previous CREG value is transferred into the SREG and the CREG is loaded from T*M.		If (cond = true) Then SREG = CREG CREG = T*M PC = PC + 1 Else PC = PC + 1
			PF001530	

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
15	PSH	If (cond) Then Push Conditional push the PC + 1 into the SREG.		If (cond = true) Then SREG = PC + 1 PC = PC + 1 Else PC = PC + 1
17	PSHN	If (cond) Then Push, Nested Conditional push the PC + 1 into the SREG nested. This microinstruction treats the SREG and CREG as a two-deep stack, PC + 1 is pushed into SREG and the previous value in SREG is transferred into the CREG.		If (cond = true) Then CREG = SREG SREG = PC + 1 PC = PC + 1 Else PC = PC + 1
14	PSHPL	If (cond) Then Push, Load Pipeline Conditional push the PC + 1 into the SREG and load the CREG from the PL (DATA) field.		If (cond = true) Then CREG = PL(data) SREG = PC + 1 PC = PC + 1 Else PC = PC + 1
16	PSHTM	If (cond) Then Push, Load TM Conditional push the PC + 1 into the SREG and load the CREG from the T*M (T[5:0] under bitwise mask from the DATA field).		If (cond = true) Then CREG = T*M SREG = PC + 1 PC = PC + 1 Else PC = PC + 1

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
02	RET	If (cond) Then Return Conditional return from subroutine. The SREG provides the return from subroutine address.	Diagram for RET instruction: A stack of registers 30-33. SREG (32) points to PC+1. The stack contains values 40, 41, 42, 43. A flag F is set.	If (cond = true) Then PC = SREG Else PC = PC + 1
03	RETN	If (cond) Then Return Nested Conditional return from nested subroutine. This microinstruction treats the SREG and CREG as a two-deep stack providing the SREG value as a return address and the CREG value as a nested return address that is transferred into the SREG.	Diagram for RETN instruction: A stack of registers 30-33. SREG (32) points to PC+1. CREG (32) points to PC+1. The stack contains values 40, 41, 42, 43. A flag F is set.	If (cond = true) Then PC = SREG SREG = CREG Else PC = PC + 1
00	RETPL	If (cond) Then Return, Load Pipeline Conditional return from subroutine and load the CREG from the PL (DATA field). The SREG provides the return from subroutine address.	Diagram for RETPL instruction: A stack of registers 30-33. SREG (32) points to PC+1. CREG (32) points to PL (DATA). The stack contains values 40, 41, 42, 43. A flag F is set.	If (cond = true) Then CREG = PL(data) PC = SREG Else PC = PC + 1
01	RETPLN	If (cond) Then Return Nested, Load Pipeline Conditional return from nested subroutine and load the CREG from the PL (DATA field). This microinstruction treats the SREG and CREG as a two-deep stack providing the SREG value as a return address and the CREG value as a nested return address that is transferred into the SREG.	Diagram for RETPLN instruction: A stack of registers 30-33. SREG (32) points to PC+1. CREG (32) points to PC+1. The stack contains values 40, 41, 42, 43. A flag F is set.	If (cond = true) Then PC = SREG SREG = CREG CREG = PL(data) Else PC = PC + 1

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
09	DEC	If (cond) Then Decrement Conditional decrement of the CREG.		If (cond = true) Then CREG = CREG - 1 PC = PC + 1 Else PC = PC + 1
0C	DECPL	While (CREG ≠ 0) Wait Else Load Pipeline Conditional Hold until the counter is equal to zero, then load CREG from the PL (DATA field). This microinstruction is intended for timing waveform generation. If the CREG is not equal to zero, the same microinstruction is refetched while CREG is decremented. Timing is complete when the CREG is equal to zero, causing the next microinstruction to be fetched and the CREG to be reloaded from PL. This instruction does not depend on the pass/fail condition.		While (CREG < > 0) CREG = CREG - 1 PC = PC End While CREG = PL(data) PC = PC + 1
0E	DECTM	While (CREG ≠ 0) Wait Else Load TM Conditional Hold until the counter is equal to zero, then load CREG from the T*M (T[5:0] under bitwise mask from the DATA field). This microinstruction is intended for timing waveform generation. If the CREG is not equal to zero, the same microinstruction is refetched while the CREG is decremented. Timing is complete when the CREG is equal to zero, causing the next microinstruction to be fetched and the CREG to be reloaded from T*M. This instruction does not depend on the pass/fail condition.		While (CREG < > 0) CREG = CREG - 1 PC = PC End While CREG = T*M PC = PC + 1
1B	DECGOPL	If (cond) Then Go To Pipeline Else While (CREG ≠ 0) Wait Conditional Hold/Count. The current microinstruction will be refetched and the CREG decremented until the condition under test becomes true or the counter is equal to zero. If the condition becomes true, a branch to the address in the PL (DATA field) is executed. If the counter becomes zero without the condition becoming true, a CONTINUE is executed. The EQ flag will be reset if the test field selects it and the condition passes.		While (cond = false) If (CREG < > 0) CREG = CREG - 1 PC = PC Else PC = PC + 1 End While PC = PL(data)

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
1A	WAIT	If (cond) Then Go To Pipeline Else Wait Conditional Hold. The current microinstruction will be refetched and executed until the condition under test becomes true. When true, a branch to the address in the PL (DATA field) is executed. The EQ flag will be reset if the test field selects it and the condition passes.	PF001660	If (cond = true) Then PC = PL(data) Else PC = PC
08	LPPL	While (CREG ≠ 0) Loop to Pipeline Conditional loop to the address in the PL (DATA field). This microinstruction is intended to be placed at the bottom of an iterative loop. If the CREG is not equal to zero, it is decremented (signifying completion of an iteration), and a branch to the PL address (top of the loop) is executed. If the CREG is equal to zero, looping is complete and the next sequential microinstruction is executed. This instruction does not depend on the pass/fail condition. The EQ flag will be reset if the test field selects it and CREG is not equal to zero.	PF001670	While (CREG < > 0) CREG = CREG - 1 PC = PL (data) End While PC = PC + 1
0A	LPPLN	While (CREG ≠ 0) Loop to Pipeline Else Nest Conditional loop to the address in the PL (DATA field) nested. The SREG and CREG are treated as a two-deep nested count register, and the microinstruction is intended to be placed at the bottom of an "inner-nested" iterative loop. If the CREG is not equal to zero, the CREG is decremented (signifying completion of an iteration), and a branch to the PL address (top of the loop) is executed. If the CREG is equal to zero, the inner loop is complete, and the count value for the outer loop is transferred from the SREG into the CREG. This instruction does not depend on the pass/fail condition. The EQ flag will be reset if the test field selects it and CREG is not equal to zero.	PF001680	While (CREG < > 0) CREG = CREG - 1 PC = PL(data) End While CREG = SREG PC = PC + 1

Opcode	Mnemonics	Description	Execution Example	Register Transfer Description
0D	CONT	Continue The next sequential microinstruction is fetched unconditionally.		PC ← PC + 1
10 - 13 (100XX binary)	CMP	Compare TM to Pipeline (DATA) This microinstruction performs bitwise exclusive-or of T*M (T[5:0] under bitwise mask from the DATA field) with CONSTANT (P[27:22]). If T*M equals CONSTANT, the EQ flag is set to one which may be branched on in a following microinstruction. If not equal, the EQ flag is unaffected. This allows sequences of compares, in a manner analogous to sum-of-products, to be performed which can be followed by a single conditional branch if one or more of the comparisons were true. Note: The EQ flag is set to zero on reset or when EQ is selected as the condition in a branch. Conditional returns on EQ leave the flag unchanged. Constant field bits that correspond to masked test field bits must be zero. This instruction does not depend on the pass/fail condition.		Compare T*M and PL(data) $EQ = ((T[5:0] \text{ .AND. DATA}) \text{ .XNOR. CONSTANT}) \text{ .OR. EQ}$

MICROINSTRUCTION SET TABLE

Code	Mnemonics	Definition	CREG Content	Pass				Fall			
				PC MUX	SREG	CREG	EQ	PC MUX	SREG	CREG	EQ
00	RETPL	Return: Load Pipeline	X	SREG	Hold	Data	NC	PC + 1	Hold	Hold	NC
01	RETPLN	Return Nested: Load Pipeline	X	SREG	CREG	Data	NC	PC + 1	Hold	Hold	NC
02	RET	Return	X	SREG	Hold	Hold	NC	PC + 1	Hold	Hold	NC
03	RETN	Return Nested	X	SREG	CREG	Hold	NC	PC + 1	Hold	Hold	NC
04	LDPL	Load Pipeline	X	PC + 1	Hold	Data	NC	PC + 1	Hold	Hold	NC
05	LDPLN	Load Pipeline Nested	X	PC + 1	CREG	Data	NC	PC + 1	Hold	Hold	NC
06	LDTM	Load T*M	X	PC + 1	Hold	T*M	NC	PC + 1	Hold	Hold	NC
07	LDTMN	Load T*M Nested	X	PC + 1	CREG	T*M	NC	PC + 1	Hold	Hold	NC
08	LPPL	Loop Pipeline	≠ 0	Data	Hold	DCRMT	Reset				
			= 0	PC + 1	Hold	Hold	NC				
09	DEC	Decrement	X	PC + 1	Hold	DCRMT	NC	PC + 1	Hold	Hold	NC
0A	LPPLN	Loop Pipeline Nested	≠ 0	Data	Hold	DCRMT	Reset				
			= 0	PC + 1	Hold	SREG	NC				
0B	GOTOPLZ	Go to Pipeline Zero	≠ 0	PC + 1	Hold	Hold	NC				
			= 0	Data	Hold	Hold	Reset				
0C	DECPL	Count/Load Pipeline	≠ 0	PC	Hold	DCRMT	NC				
			= 0	PC + 1	Hold	Data	NC				
0D	CONT	Continue	X	PC + 1	Hold	Hold	NC	PC + 1	Hold	Hold	NC
0E	DECTM	Count/Load T*M	≠ 0	PC	Hold	DCRMT	NC				
			= 0	PC + 1	Hold	T*M	NC				
0F	GOTOTM	Go to T*M	X	T*M	Hold	Hold	Reset	PC + 1	Hold	Hold	NC
10 - 13 (100XX Binary)	CMP	Compare*	X	PC + 1	Hold	Hold	Set	PC + 1	Hold	Hold	NC
14	PSHPL	Push: Load Pipeline	X	PC + 1	PC + 1	Data	NC	PC + 1	Hold	Hold	NC
15	PSH	Push	X	PC + 1	PC + 1	Hold	NC	PC + 1	Hold	Hold	NC
16	PSHTM	Push: Load T*M	X	PC + 1	PC + 1	T*M	NC	PC + 1	Hold	Hold	NC
17	PSHN	Push Nested	X	PC + 1	PC + 1	SREG	NC	PC + 1	Hold	Hold	NC
18	FORK	Fork	X	Data	Hold	Hold	Reset	SREG	Hold	Hold	NC
19	GOTOPL	Go to Pipeline	X	Data	Hold	Hold	Reset	PC + 1	Hold	Hold	NC
1A	WAIT	Hold Pipeline	X	Data	Hold	Hold	Reset	PC	Hold	Hold	NC
1B	DECGOPL	Count: Hold Pipeline	≠ 0	Data	Hold	Hold	Reset	PC	Hold	DCRMT	NC
			= 0	Data	Hold	Hold	Reset	PC + 1	Hold	Hold	NC
1C	CALPL	Call Pipeline	X	Data	PC + 1	Hold	Reset	PC + 1	Hold	Hold	NC
1D	CALPLN	Call Pipeline Nested	X	Data	PC + 1	SREG	Reset	PC + 1	Hold	Hold	NC
1E	CALTM	Call T*M	X	T*M	PC + 1	Hold	Reset	PC + 1	Hold	Hold	NC
1F	CALTMN	Call T*M Nested	X	T*M	PC + 1	SREG	Reset	PC + 1	Hold	Hold	NC

EQ = (T[5:0], AND, DATA), XNOR, CONSTANT), OR, EQ

CONSTANT field bits that correspond to masked test field bits must be zero.

NC = No Change

Notes:

1. (/) Signifies two different operations may occur, depending on the condition.
2. (:) Signifies two parallel operations on the same condition.
3. The EQ flag will be affected only if the test field selects it, with the exception of instructions 10 - 13.

PROGRAMMING

The Am29PL141 FPC is programmed and verified using a simple algorithm that is almost identical to that used for AMD's Programmable Array Logic family. The internal programmable array of the Am29PL141 is organized as a 64 word by 32 bit (column) PROM. The fuse to be programmed is selected by its address (1 of 64), the byte at that address (1 of 4), and the bit in the byte (1 of 8). Control of programming and verifying is accomplished by applying a simple sequence of voltages on two control pins (CLK and CC).

The fuse address is selected using a full decode of the T[5 : 0] inputs, where T[5] is the MSB and T[0] the LSB. The one of four byte addressing is done on the P[7] (MSB) and P[6] (LSB) outputs. The bit selection is done one output at a time by applying the programming voltage (V_{OP}) to the output pin. The output pins that accept V_{OP} are P[15 : 8]. A graphical representation of the fuse array organization for programming, with fuse numbering compatible to the JEDEC standard programmable logic transfer format, is shown in Figure 2.

The complete program and verify cycle timing is shown in the programming waveform. A programming sequence is initiated by raising the CLK pin to V_{HH} . This places the device in the program mode and disables the output pins so that they may be used as fuse addressing inputs. The next step is to address the fuse to be blown as previously stated. Note that bit selection, with V_{OP} , should follow address and byte selection. Raising the CC pin to V_{HH} initiates programming and lowering V_{OP} terminates programming. Lowering the CLK pin to a TTL LOW level places the device in the fuse verification mode by enabling the programming outputs, P[15 : 8]. Following a clock pulse the fuse may be verified on the same output as bit

selection was performed. This scheme allows fuses to be verified in parallel as a byte if desired. The verification mode is terminated by lowering the CC pin back to a normal TTL level.

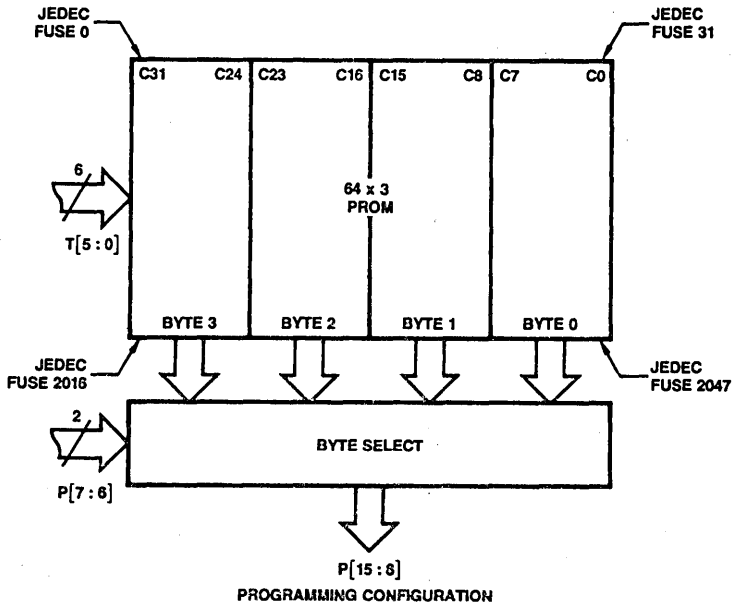
SSR DIAGNOSTICS CONFIGURATION PROGRAMMING

One additional fuse (#2048) is used to alter the configuration of the Am29PL141 to include on-chip SSR Diagnostics. This fuse is addressed by applying V_{HH} to the RESET and T[5], followed by V_{OP} on pin P[15]. To verify the diagnostic fuse, P[7] and P[6] must select byte #3, i.e., P[7] must be low and P[6] must be high.

PROGRAMMING YIELD

AMD programmable logic devices have been designed to insure extremely high programming yields (> 98%). To help insure that a part was correctly programmed, once the programming sequence is completed, the entire fuse array should be reverified at both low and high V_{CC} (V_{CCL} and V_{CCH}). Reverification can be accomplished in a verification only mode (CC at V_{HH}) by reading the outputs in parallel. This verification cycle checks that the array fuses have been blown correctly and can be sensed under varying conditions by the outputs.

AMD programmable logic devices contain many internal test features, including circuitry and extra fuses which allow AMD to test the ability of each part to perform programming before shipping, to assure high programming yields, and correct logical operation for a correctly programmed part. Programming yield losses are most likely due to poor programming socket contact, programming equipment that is out of calibration, or improper usage of said equipment.



PFR00970

$$\text{JEDEC FUSE NUMBER} = 32 (\text{FUSE ADDRESS}) + 8(3 - \text{BYTE}) + (7 - \text{BIT})$$

Figure 2. Programming Configuration

BYTE SELECT		
BYTE	P[7]	P[6]
0	H	L
1	H	H
2	L	L
3	L	H

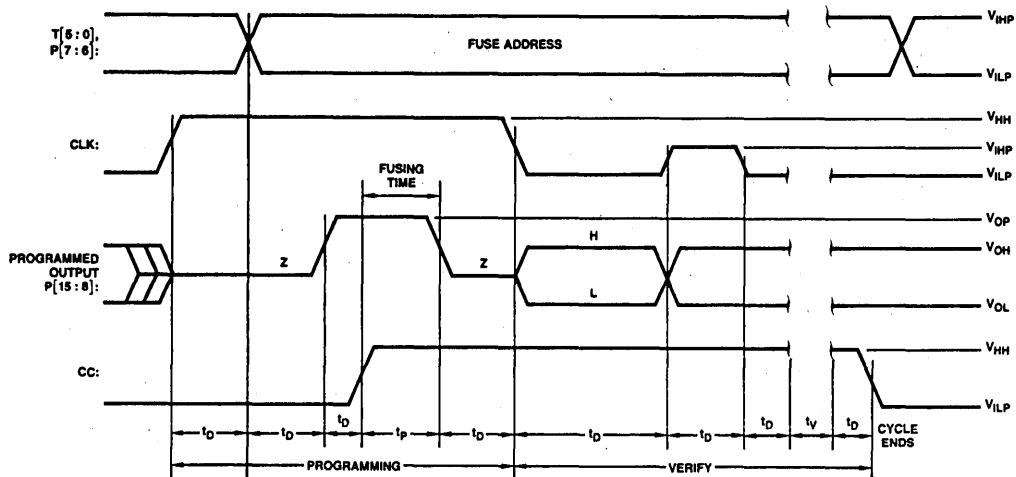
BIT SELECT								
BIT	P[15]	P[14]	P[13]	P[12]	P[11]	P[10]	P[9]	P[8]
0	L	L	L	L	L	L	L	H
1	L	L	L	L	L	L	L	L
2	L	L	L	L	L	H	L	L
3	L	L	L	L	H	L	L	L
4	L	L	L	H	L	L	L	L
5	L	L	H	L	L	L	L	L
6	L	H	L	L	L	L	L	L
7	H	L	L	L	L	L	L	L

COLUMN DECODE				
0	C0	C8	C16	C24
1	C1	C9	C17	C25
2	C2	C10	C18	C26
3	C3	C11	C19	C27
4	C4	C12	C20	C28
5	C5	C13	C21	C29
6	C6	C14	C22	C30
7	C7	C15	C23	C31

FUSE ADDRESS DECODE						
FUSE ADDRESS	T[5]	T[4]	T[3]	T[2]	T[1]	T[0]
0	L	L	L	L	L	L
1	L	L	L	L	L	H
2	L	L	L	L	H	L
3	L	L	L	L	H	H
4	L	L	L	H	L	L
5	L	L	L	H	L	H
6	L	L	L	H	H	L
7	L	L	L	H	H	L
8	L	L	H	L	L	L
9	L	L	H	L	L	H
10	L	L	H	L	H	L
11	L	L	H	L	H	L
12	L	L	H	H	L	L
13	L	L	H	H	L	L
14	L	L	H	H	H	L
15	L	L	H	H	H	H
16	L	H	L	L	L	L
17	L	H	L	L	L	L
18	L	H	L	L	H	L
19	L	H	L	L	H	L
20	L	H	L	H	L	L
21	L	H	L	H	L	H
22	L	H	L	H	H	L
23	L	H	L	H	H	L
24	L	H	H	L	L	L
25	L	H	H	L	L	L
26	L	H	H	L	H	L
27	L	H	H	L	H	L
28	L	H	H	H	L	L
29	L	H	H	H	L	L
30	L	H	H	H	H	L
31	L	H	H	H	H	L
32	H	L	L	L	L	L
33	H	L	L	L	L	L
34	H	L	L	L	H	L
35	H	L	L	L	H	L
36	H	L	L	H	L	L
37	H	L	L	H	L	L
38	H	L	L	H	H	L
39	H	L	L	H	H	L
40	H	L	H	L	L	L
41	H	L	H	L	L	L
42	H	L	H	L	H	L
43	H	L	H	L	H	L
44	H	L	H	H	L	L
45	H	L	H	H	L	L
46	H	L	H	H	H	L
47	H	L	H	H	H	L
48	H	H	L	L	L	L
49	H	H	L	L	L	L
50	H	H	L	L	H	L
51	H	H	L	L	H	L
52	H	H	L	H	L	L
53	H	H	L	H	L	L
54	H	H	L	H	H	L
55	H	H	L	H	H	L
56	H	H	H	L	L	L
57	H	H	H	L	L	L
58	H	H	H	L	H	L
59	H	H	H	L	H	L
60	H	H	H	H	L	L
61	H	H	H	H	L	L
62	H	H	H	H	H	L
63	H	H	H	H	H	H

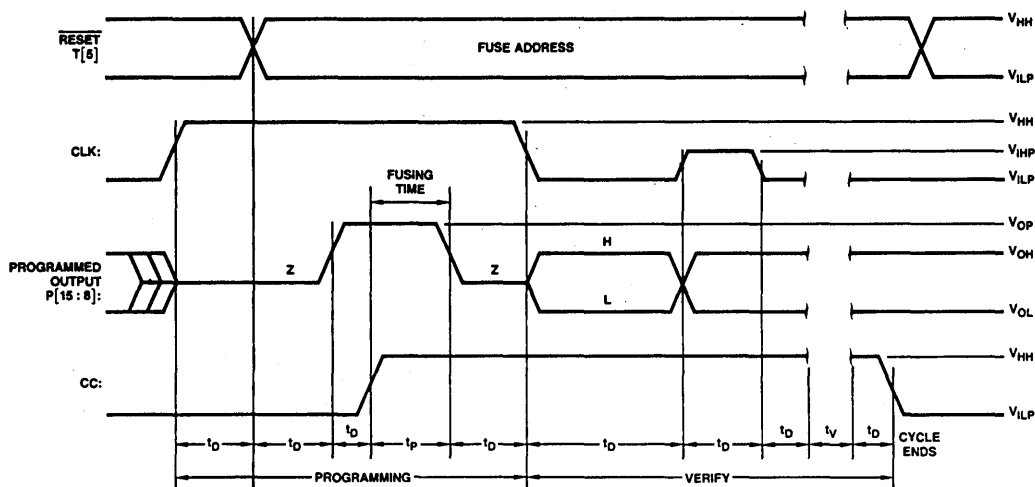
PROGRAMMING PARAMETERS $T_A = 25^\circ\text{C}$

Parameters	Description		Min.	Typ.	Max	Units
V _{HH}	Control Pin Extra High Level	CC @ 5 – 10 mA	19.5	20	20.5	Volts
		CLK@ 5 – 10 mA	19.5	20	20.5	
V _{OP}	Program Voltage, P [15:8] @ 15 – 200 mA		19.5	20	20.5	Volts
V _{IHP}	Input High Level During Programming and Verify		2.4	5	5.5	Volts
V _{ILP}	Input Low Level During Programming and Verify		0.0	0.3	0.5	Volts
V _{CCP}	V _{CC} During Programming @ I _{CC} = 425 mA		5	5.2	5.5	Volts
V _{CCL}	V _{CC} During First Pass Verification @ I _{CC} = 425 mA		4.1	4.3	4.5	Volts
V _{CCH}	V _{CC} During Second Pass Verification @ I _{CC} = 485 mA		5.4	5.7	6.0	Volts
V _{Blown}	Successful Blown Fuse Sense Level @ Output			0.3	0.5	Volts
dV _{OP} /dt	Rate of Output Voltage Change		20		250	V/ μ sec
dV _{FE} /dt	Rate of Fusing Enable Voltage Change (CC Rising Edge)		100		1000	V/ μ sec
t _p	Fusing Time First Attempt		40	50	100	μ sec
	Subsequent Attempts		4	5	10	msec
t _D	Delays Between Various Level Changes		100	200		ns
t _v	Period During which Output is Sensed for V _{Blown} Level		500			ns
V _{ONP}	Pull-Up Voltage on Outputs Not Being Programmed		V _{CCP} – 0.3	V _{CCP}	V _{CCP} + 0.3	Volts
R	Pull-Up Resistor on Outputs Not Being Programmed		1.9	2	2.1	K Ω



WF020830

Programming Waveforms



WF020840

SSR Diagnostics Configuration Programming Waveforms

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
(Ambient) Temperature Under Bias	-55 to +125°C
Supply Voltage to Ground Potential (Pin 28 to Pin 14) Continuous	-0.5 V to +7.0 V
DC Voltage Applied to Outputs (Except During Programming)	-0.5 V to +V _{CC} Max.
DC Voltage Applied to Outputs During Programming	21 V
DC Output Current, Into Outputs During Programming (Max Duration of 1 sec)	200 mA
DC Input Voltage	-0.5 V to +5.5 V
DC Input Current	-30 mA to +5.0 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature	0 to +70°C
Supply Voltage	+4.75 V to +5.25 V
Military (M) Devices	
Temperature	-55 to +125°C
Supply Voltage	+4.5 V to +5.5 V

Operating ranges define those limits over which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range unless otherwise specified; included in Group A, Subgroup 1, 2, 3 tests unless otherwise noted

Parameters	Description	Test Conditions			Min	Max	Units	
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}		I _{OH} = -3.0 mA I _{OH} = -1.0 mA	COM'L MIL	2.4		Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}		I _{OL} = 16 mA I _{OL} = 12 mA	COM'L MIL		0.50	Volts
V _{IH} (Note 1)	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs				2.0		Volts
V _{IL} (Note 1)	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs					0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.5 V	CLK P [15:6] All other Inputs				-1.5 -0.55 -0.50	mA
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = 2.4 V	CLK P [15:6] All other Inputs				150 100 25	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V					1.0	mA
I _{SC}	Output Short Circuit Current	V _{CC} = Max., V _{OUT} = 0.5 V (Note 2)				-20	-80	mA
I _{CC}	Power Supply Current	V _{CC} = Max.	COM'L	T _A = 0 to 70°C			450	mA
				T _A = 70°C			400	
			MIL	T _C = -55 to 125°C			490	
				T _C = 125°C			420	
V _I	Input Clamp Voltage	V _{CC} = Min, I _{IN} = -18 mA					-1.2	Volts
I _{OZH}	Output Leakage Current (Note 3)	V _{CC} = MAX, V _{IL} = 0.8 V V _{IH} = 2.0 V	V _O = 2.4 V			100	μA	
I _{OZL}			V _O = 0.5 V			-550		

Notes:

- These are absolute values with respect to device ground and all overshoots due to system or tester noise are included.
- Not more than one output should be tested at a time. Duration of the short circuit should not be more than one second. V_{OUT} = 0.5 V has been chosen to avoid test problems caused by tester ground degradation.
- I/O pin leakage is the worst case of I_{OZX} or I_{IX} (where X = H or L).

SWITCHING CHARACTERISTICS

over operating range unless otherwise specified; included in Group A Subgroup 9, 10, 11 tests unless otherwise noted. (APL and CPL products only.)

Parameters		Description	Test Conditions	COMMERCIAL		MILITARY		Units
				Min.	Max.	Min.	Max.	
t _{PD}	1	CLK to P[15:0]	See Test Output Load Conditions		15		20	ns
	2	CLK to ZERO			20		25	ns
	3	DCLK to SDO			30		35	ns
	4	Mode to SDO			30		35	ns
	5	SDI to SDO			30		35	ns
t _S	6	T[5:0] to CLK (Note 1)		40		45 †		ns
	7	CC to CLK (Note 1)		40		45 †		ns
	8	RESET to CLK		30		35		ns
	9	Mode to CLK		30		35		ns
	10	Mode to DCLK		30		35		ns
	11	SDI to DCLK		30		35		ns
	12	P[15:8] to DCLK		30		35		ns
t _H	13	T[5:0] to CLK		3		3		ns
	14	CC to CLK		3		3		ns
	15	RESET to CLK		3		3		ns
	16	Mode to CLK		3		3		ns
	17	Mode to DCLK		3		3		ns
	18	SDI to DCLK		3		3		ns
	19	P[15:8] to DCLK		3		3		ns
t _{PZX}	20	CLK to P[15:8] Enable			30		35	ns
t _{PXZ}	21	CLK to P[15:8] Disable			30		35	ns
t _{PW}	22	CLK Pulse Width (HIGH and LOW)		20		25		ns
	23	DCLK Pulse Width (HIGH and LOW)		30		35		ns
t _P	24	CLK and DCLK Period (Note 1)		45		50 †		ns

Notes:

1. These parameters cannot be measured directly on unprogrammed devices. They are determined as follows:

- Measure delay from input (CC, T[5:0], or CLK) to PROM address out in test mode. This will measure the delay through the sequence logic.
- Measure setup time from T[5:0] input through PROM test columns to pipeline register in verify test column mode. This will measure the delay through the PROM and register setup.
- Measure delay from T[5:0] input to PROM address out in verify test column mode. This will measure the delay through the logic and P[15:0] outputs.

To calculate the desired parameter measurement the following formula is used:

Measurement (a) + Measurement (b) - Measurement (c)

CLK PERIOD:

CLK (a) + (b) - (c) = CLK PERIOD

CC to CLK Set-up time:

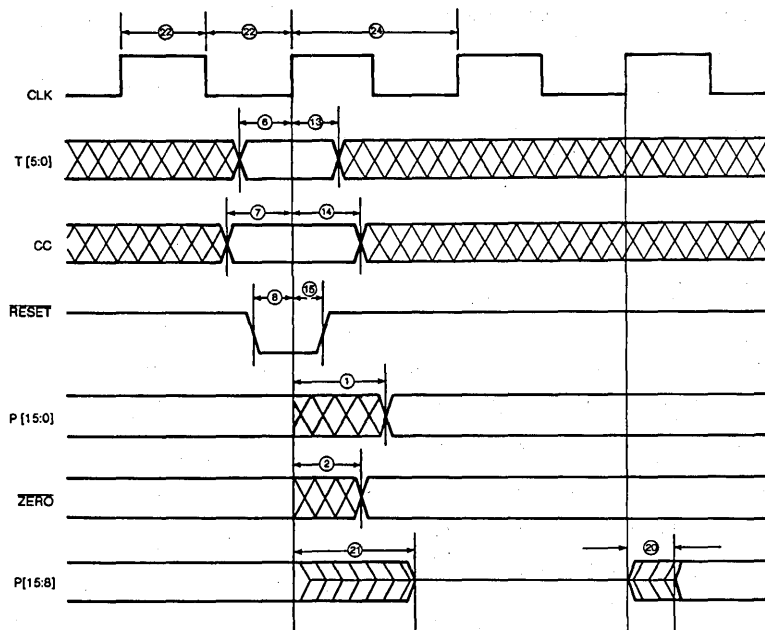
CC (a) + (b) - (c) = CC to CLK Set-up time

T[5:0] to CLK Set-up time:

T[5:0] (a) + (b) - (c) = T[5:0] to CLK Set-up time

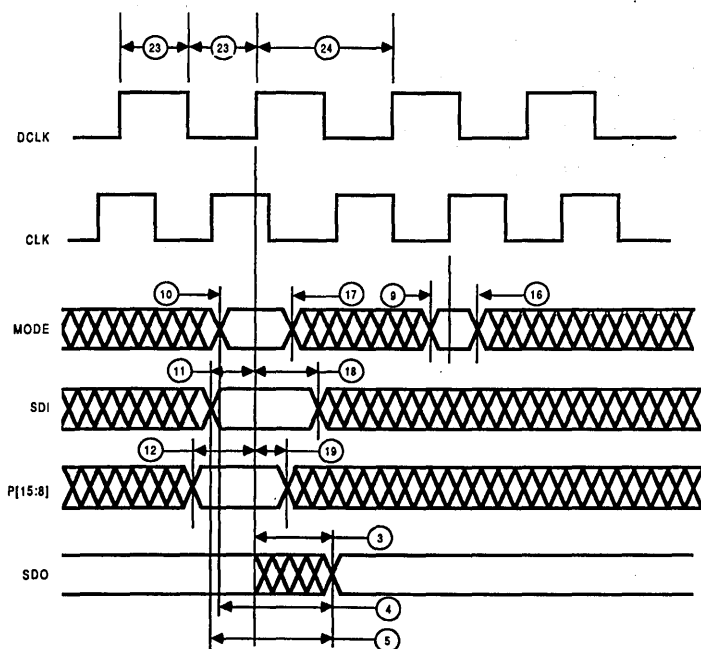
† = Not included in Group A tests

SWITCHING WAVEFORMS



WF020852

Normal Configuration



WF023120

SSR™ Configuration

Test Philosophy and Methods

The following points give the general philosophy that we apply to tests that must be properly engineered if they are to be implemented in an automatic testing environment. The specifics of what philosophies are applied to which test are shown in the data sheet and the data-sheet reconciliation that follow.

Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance that varies from one type of tester to another, but is generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters that call for smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays" that measure the propagation delays in to and out of the high-impedance state and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF) and engineering correlations based on data taken with a bench setup are used to determine the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impractical to make measurements at both capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is determined from engineering correlations based on data taken with a bench setup and the knowledge that certain DC tests are performed in order to facilitate this correlation.

AC loads specified in the data sheet are used for bench testing. Automatic tester loads, which simulate the data-sheet loads, may be used during production testing.

Threshold Testing

The noise associated with automatic testing, the long inductive cables, and the high gain of bipolar devices frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels.

AC Testing

AC parameters are specified that cannot be measured accurately on automatic testers because of tester limitations. Data-input hold times fall into this category. In these cases, the parameter in question is tested by correlating the tester to bench data or oscilloscope measurements made on the tester by engineering (supporting data on file).

Certain AC tests are redundant since they can be shown to be predicted by other tests that have already been performed. In these cases, the redundant tests are not performed.

Output Short-Circuit Current Testing

When performing I_{OS} tests on devices containing RAM or registers, great care must be taken that undershoot caused by grounding the high-state output does not trigger parasitic elements which in turn cause the device to change state. In order to avoid this effect, it is common to make the measurement at a voltage (V_{output}) that is slightly above ground. The V_{CC} is raised by the same amount so that the result (as confirmed by Ohm's law and precise bench testing) is identical to the V_{OUT} = 0, V_{CC} = Max. case.

APPLICATION

Cycle Time Calculation

The Am29PL141 has a 40 ns set-up time requirement on the T[5:0] test inputs.

If this set-up time is violated, the part may become metastable. It is therefore necessary to synchronize the test inputs with the CLK when the test inputs are asynchronous.

By selecting the appropriate speed register, 50 ns cycle time can be achieved.

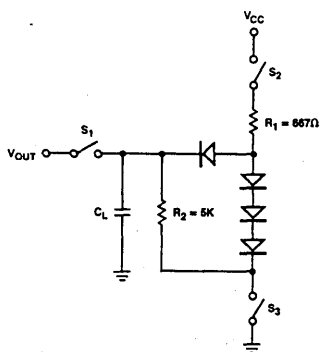
Register	CLK - Q	10 ns
Am29PL141	T[5:0] - CLK Set-up	<u>40 ns</u>
	Cycle Time	50 ns

KEY TO SWITCHING WAVEFORMS

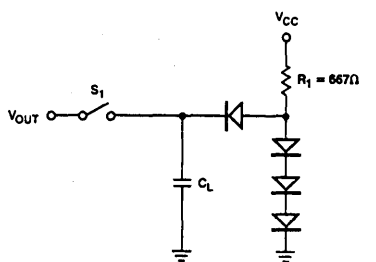
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING TEST CIRCUITS



TCR01330



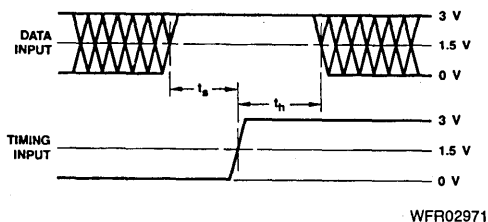
TCR01340

A. Three State Outputs

B. Normal Outputs

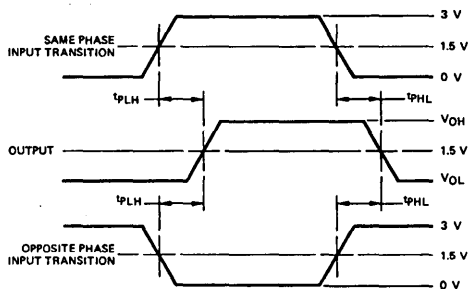
- Notes:
1. $C_L = 50$ pF includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1 , S_2 , and S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 4. $C_L = 5.0$ pF for output disable tests.

SWITCHING TEST WAVEFORMS



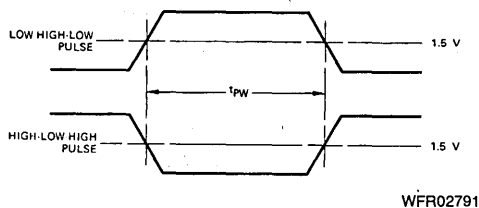
Set-up, Hold, and Release Times

- Notes: 1. Diagram shown for HIGH data only. Output transition may be opposite sense.
2. Cross hatched area is don't care condition.



Propagation Delay

Pulse Width



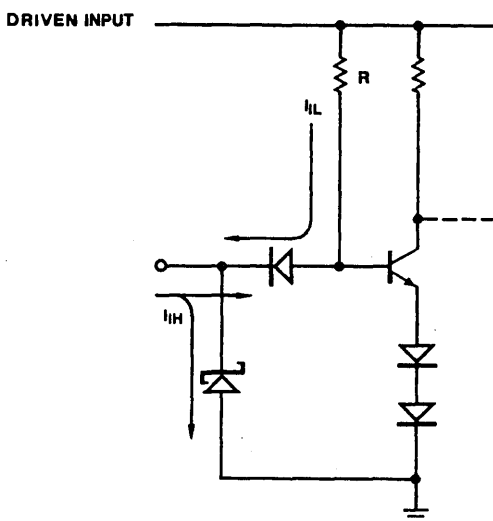
Enable and Disable Times

Test	V _X	Output Waveform - Measurement Level
All t _{PD} s	5.0V	V _{OH} 1.5V V _{OL}
t _{PHZ}	0.0V	V _{OH} 0.6V 0.0V
t _{PLZ}	5.0V	V _{OL} 0.6V 3V
t _{PZH}	0.0V	0.0V 1.5V V _{OH}
t _{PZL}	5.0V	3V 1.5V V _{OL}

- Notes: 1. Diagram shown for input Control Enable-LOW and input Control Disable-HIGH.
2. S₁, S₂, and S₃ of Load Circuit are closed except where shown.

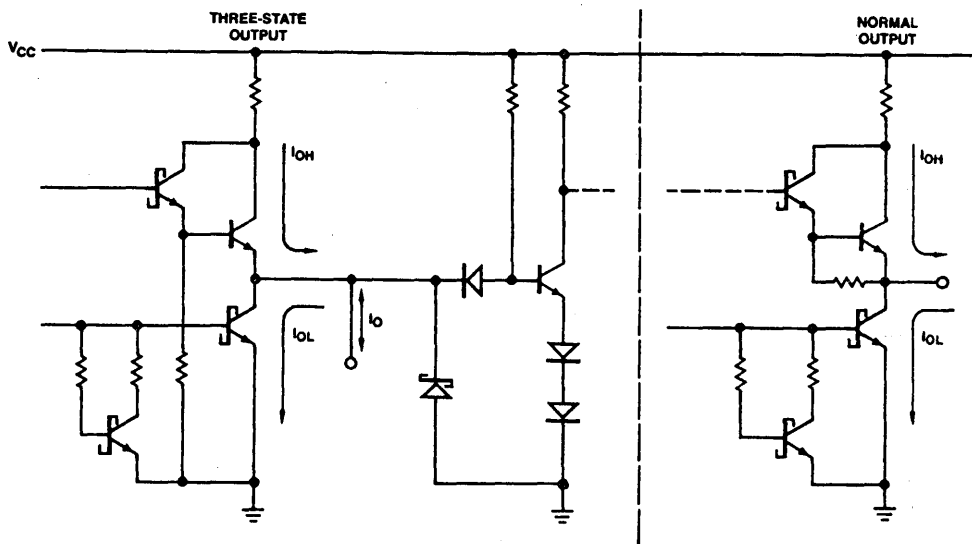
NOTE: Pulse generator for all pulses: Rate ≤ 1.0 MHz; Z₀ = 50 Ω; t_r ≤ 2.5 ns.

INPUT/OUTPUT CURRENT INTERFACE CONDITIONS



ALL
INPUTS
 $R = 16K\Omega$

ICR00533

 $C_0 \cong 5.0$ pF, all inputs


ICR00524

 $C_0 \cong 5.0$ pF, all outputs

NOTE: Actual current flow direction shown.

Am2971

Programmable Event Generator (PEG)

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- Twelve programmable, registered output waveforms
- Programmable event intervals down to 10 ns
- Multiplying phase-locked-loop (PLL) oscillator for improved timing accuracy
- Programmable clock output for system reference at 1/5 or 1/10 of internal phase-locked loop frequency
- Programmable control of timing sequence start and stop functions
- Can be clocked from either an external source or an on-board crystal oscillator

GENERAL DESCRIPTION

The Am2971 is a very versatile timing device. It can be used as a digital substitute for analog delay lines or as a general purpose user-programmable timing/waveform generator.

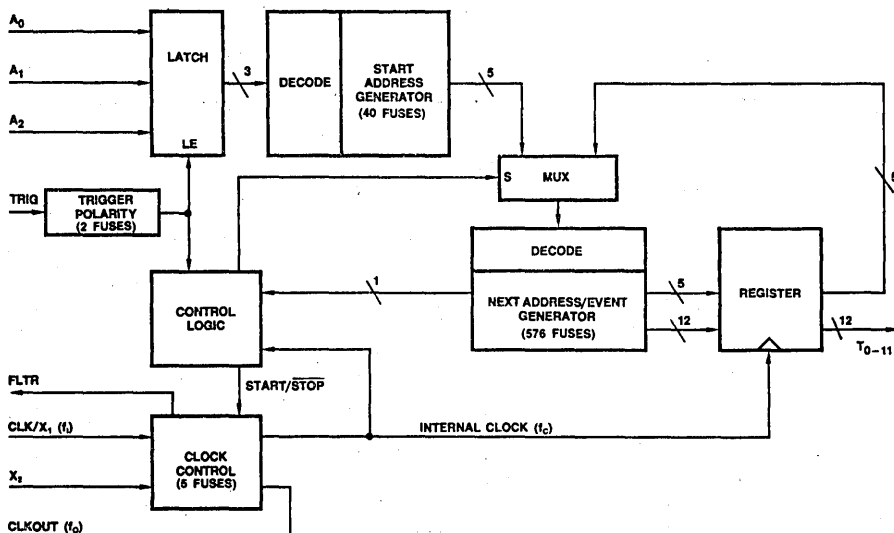
The Programmable Event Generator (PEG) has twelve independent, programmable, timing output waveforms. The resolution of the output waveforms is programmable down to 10 ns and the sequence start address is real-time user-definable. One out of eight start addresses may be selected to begin the timing sequence. The PEG can be clocked either by its 10 – 100 MHz on-chip PLL crystal oscillator

(with a clock output of 1/5 or 1/10 the PLL frequency) or by the existing system clock. The chip can be clocked up to 100 MHz if the PLL is bypassed.

The frequency of the internal clock (f_c) can be programmed to equal 1x, 5x, 10x, 5/2x, 5/4x, 10/2x or 10/4x the PEG's input clock frequency (f_i). Programming the PEG is done in the same way as a PROM.

The Am2971 complements the Am2960 Dynamic Memory Support Family by allowing the user to optimize his system design for highest performance (see Figure 3).

BLOCK DIAGRAM

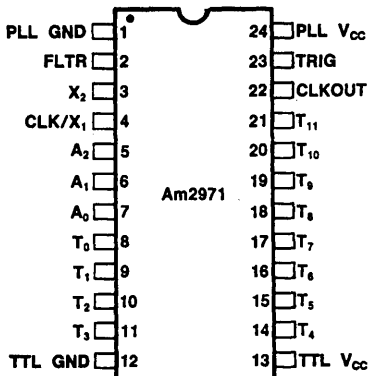


BD002473

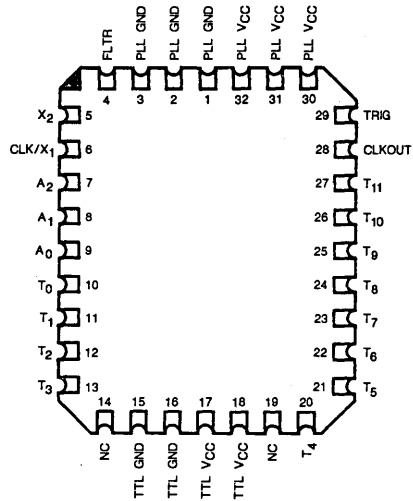
RELATED PRODUCTS

Part No.	Description
Am2960A	Error Detection and Correction Unit
Am2968A	Dynamic Memory Controller
Am2969	Memory Timing Controller with EDC Control
Am2970	Memory Timing Controller

CONNECTION DIAGRAM Top View



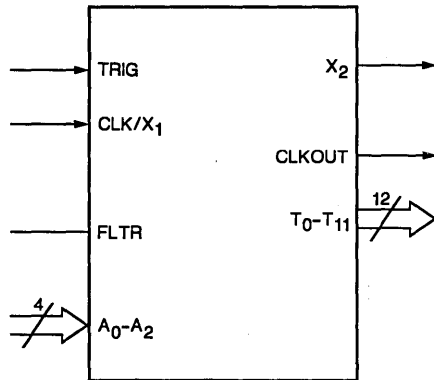
CD005852



CD009981

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



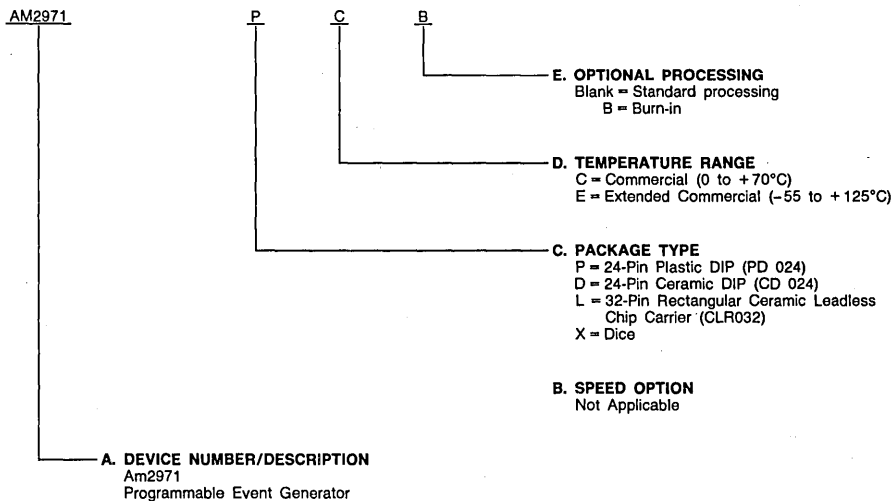
LS002711

ORDERING INFORMATION (Cont'd.)

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

Valid Combinations	
AM2971	PC, PCB, DC, DCB, DE, DEB, LC, XC

Valid Combinations

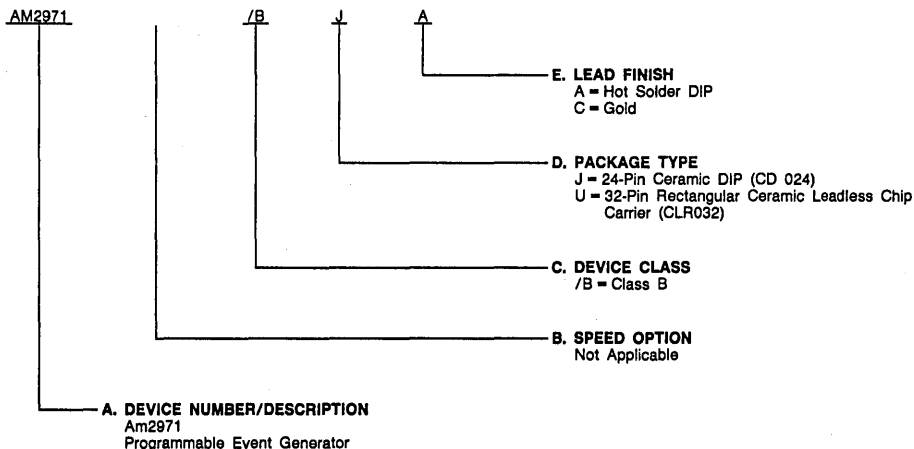
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION

APL 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) for APL products is formed by a combination of:

- A. Device Number**
- B. Speed Option** (if applicable)
- C. Device Class**
- D. Package Type**
- E. Lead Finish**



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.

Valid Combinations	
AM2971	/BJA, /BUC

PIN DESCRIPTION

TRIG Trigger (Input)

The timing cycle of the PEG can be started on either the positive or negative edge of the start (TRIG) pulse. The polarity is defined by the user as a fuse option (fuse 621) in the TRIGGER POLARITY block. The trailing edge of the start (TRIG) pulse may be used to stop the timing sequence. The STOP TRIG fuse (fuse 622) must not be blown if this option is desired. If the fuse is blown, the trailing edge of the start (TRIG) pulse will be disabled as a means of stopping the timing sequence. Instead, the PEG will search for a blown Stop Bit fuse in the Next Address/Event Generator. In the Program Mode, a high voltage level (V_{OP}) is applied to the TRIG input (see Figure 5).

A₀ - A₂ Addresses (Inputs (3))

These three bits define eight locations in the Start Address Generator. The Start Address Generator contains the user-programmed start locations. One of eight addresses can be selected for each cycle initiation. In the Program Mode, these inputs are unused and may be allowed to float.

CLK/X₁ and X₂ Clock/Crystal (Input/Output)

These two pins serve as crystal inputs f_1 (see Figure 1). It is recommended that an AT Cut Parallel Resonant Crystal be used. An external clock may also be applied to the CLK/X₁ input with the X₂ output left floating. Fuses 612 through 619 (see Table 8) in the CLOCK CONTROL logic block enable the user to set the internal clock frequency as a function of the crystal or input clock frequency.

FLTR Filter

This pin is used to connect a 0.47- μ F filter capacitor between the phase-locked-loop and ground.

CLKOUT Output Clock (Output)

CLKOUT, f_O , is a clock output pin which may be used for system reference. The output frequency for CLKOUT is either 1/5 or 1/10 the PLL clock frequency. It is fuse-programmable with the fuse located in the CLOCK CONTROL logic block. In the Program Mode, a high voltage level (V_{IH}) is applied to CLKOUT for a period of time (t_{PF}). Applying a steady-state V_{IH} voltage to this pin for a period in excess of 400 μ s is not recommended.

T₀ - T₁₁ Timing Outputs (Outputs, Active HIGH)

These are twelve timing outputs from the Next Address/Event Generator. These outputs follow a user-programmed timing pattern. Next Address/Event Generator outputs are registered allowing for glitch-free operation. In the Program Mode, T₀ through T₁₀ function as address inputs to access each individual fuse. T₁₀ through T₆ serve as Column Address inputs and T₅ through T₀ serve as Row Address inputs (see Table 8). T₁₁ functions as a Data Input/Output in the Program Mode, providing access to the addressed fuse for programming and verification.

Power, Ground TTL/PLL Power Pair

Two power and two ground pins are required by the PEG chip. One power pair is used by the PLL (phase-locked-loop) and the internal ECL circuitry. The other power pair is used by the remainder of the chip (TTL).

Note: A complete listing of all fuse numbers appears in Table 8 in the Programming Section of this specification. These numbers have been assigned for reference only.

FUNCTIONAL DESCRIPTION

The Programmable Event Generator (PEG) block diagram may be divided into four blocks: Clock Control, Start Address Generator, Next Address/Event Generator and Control Logic.

Internal to the CLOCK CONTROL logic are five user-programmable fuses. One of these (fuse number 620) is used to generate the desired output frequency (f_O) on the CLKOUT option. The remaining four fuses (numbers 616 through 619) are used to generate the desired internal reference clock frequency (f_C). As shown in Figure 2, there are a variety of internal reference frequency and external CLKOUT frequency options available as a function of input frequency (f_I). The input frequency may be supplied either by an external source or by the internal PLL oscillator (with a crystal connected as shown in Figure 3). The reader is directed to Tables 6 and 7 for an explanation of the possible internal frequency and output frequency options in regards to user programming.

A timing sequence is initiated by a transition at the TRIG input. Two user-programmable fuses are located in the TRIGGER POLARITY block. One of the two fuses (fuse number 621) is used to define the polarity of the TRIG input. When the fuse is left unprogrammed, a timing sequence is initiated during a negative transition of the TRIG input. The second fuse in this block (fuse number 622) is used to define the end of a timing sequence. If this fuse is left unprogrammed, the timing sequence is stopped on the trailing edge of the TRIG pulse. If the fuse is programmed, the end of the timing sequence is defined by the Stop Bits as programmed into the Next Address/Event Generator functional block.

A proper transition at the TRIG input initiates a timing sequence by latching the START ADDRESS GENERATOR inputs, A₀ - A₂. These latched inputs are used to select one of

eight different start addresses. The user defines these 5-bit start addresses by programming fuses 576 - 615 as required. Each 5-bit start address defines a starting point for the timing sequence from thirty-two possible selections in the user-defined Next Address/Event Generator block.

After the start address, subsequent addresses are generated from the information programmed into the NEXT ADDRESS/ EVENT GENERATOR. As defined in Tables 3 and 8, each of the thirty-two user-defined 18-bit data strings contain three pieces of information. The first 5 bits (fuses) are used to define the next address of the desired timing sequence (out of 32 possible addresses within the Next Address/Event Generator). The next 12 bits (fuses) are used to generate output waveforms to the twelve Timing Outputs (T₀ - T₁₁). These 12 bits define the current logic level at each of the twelve timing outputs. As the timing sequence progresses from address to address, these Timing Outputs will produce 12 independent waveforms, as defined by the user. Since both the address sequences and waveform logic levels are user-programmable, a wide variety of output patterns may be generated. One last bit (fuse), the Stop Bit, is used to define the end of a timing sequence.

When the Trigger Polarity Stop fuse (fuse number 622) has been programmed, the timing sequence will no longer stop on the trailing edge of the TRIG pulse. The end of a timing sequence is instead defined by the Stop Bit in each of the thirty-two Next Address/Event Generator data strings. Each Stop Bit is activated by programming. When a Timing Sequence addresses one of the thirty-two 18-bit data strings with a "1" programmed into the Stop Bit, the sequence is halted. Whenever a sequence is halted, the Timing Outputs (T₀ - T₁₁) will remain at the last value, as defined by the data string containing the active Stop Bit.

Each of the twelve timing waveform outputs ($T_0 - T_{11}$) from the Next Address/Event Generator has a minimum usable cycle of 40 ns (or 20 ns per change). When the PEG is operating at its maximum internal clock frequency (100 MHz), the outputs must be programmed to remain unchanged for at least two clock periods for each change of logic level. That is, although an output can only change every 20 ns minimum, the timing resolution between events may be as low as 10 ns. When the PEG's internal reference clock frequency (f_C) is programmed to operate at 40 MHz or slower, the timing waveform can be programmed to change on every clock.

Oscillator

The Am2971 contains an inverting, linear amplifier which is intended to form the basis of a crystal oscillator. In designing this oscillator it is necessary to consider several factors related to the application.

The first consideration is the desired frequency accuracy. This may be subdivided into several areas. An oscillator is considered stable if it is insensitive to variations in temperature and supply voltage, and if it is unaffected by individual component changes and aging. The design of the Am2971 is such that the degree to which these goals are met is determined primarily by the choice of external components. Various types of crystals are available and the manufacturers' literature should be consulted to determine the appropriate type. For good temperature stability, zero temperature coefficient capacitors should be used (Type NPO). For extreme temperature stability, an oven must be used or some other form of temperature compensation applied.

Absolute frequency accuracy must also be considered. The resonant frequency varies with load capacitance. It is therefore important to match the load specified by the crystal manufacturer for a standard crystal (usually 32 pF), or to specify the load when ordering a special crystal. It should then be possible to determine from the crystal characteristics the load tolerance to maintain a given accuracy. If the "set-on" error due to load tolerance is unacceptable, a trimmer capacitor should be incorporated for fine adjustment.

The mechanism by which a crystal resonates is electromechanical. This resonance occurs at a fundamental frequency (1st harmonic) and at all odd harmonics of this frequency (even harmonic resonance is not mechanically possible). Unless otherwise constrained crystal oscillators operate at their fundamental frequency. However, crystals are not generally available with fundamental frequencies above 20-25 MHz. At higher frequencies, an overtone oscillator must be used. In this case, the crystal is designed to oscillate efficiently at one of its odd harmonic frequencies and additional components are included in the oscillator circuit to prevent it oscillating at lower harmonics.

Where a high degree of accuracy or stability is not required, the amplifier may be configured as an L-C oscillator. It may also be driven from an external clock source if operation is required in synchronous with that source.

1st Harmonic (Fundamental) Oscillator

The circuit of a typical 1st harmonic oscillator is shown in Figure 3. The crystal load is comprised of the two 68 pF

capacitors in series. This 34 pF approximates the standard 32 pF crystal load. If a closer match is required then one of the capacitors should be replaced with a parallel combination of a fixed capacitor and a trimmer. The nominal value of the combination should be 60 pF to provide proper crystal loading.

A typical crystal specification for use in this circuit is:

Frequency Range: 2 – 20 MHz

Resonance: At Parallel Mode

Load: 32 pF

Stability: 0.1% or to match systems requirements

Case: H-17 — for smaller size

Temp Range: -30 to +70°C

Note: Frequency will change over temperature.

It is a good practice to ground the case of the crystal to eliminate stray pick-up and keep all connections as short as possible.

Note: At fundamental frequencies below 6 MHz it is possible for the oscillator to operate at the 3rd harmonic. To prevent this a resistor should be added in series with the X_2 pin as shown in the circuit diagram.

The resistor value should match the impedance of C:

$$R = X_C = \frac{1}{2\pi f C}$$

Design Considerations (reference Figure 1)

1. Oscillator external connections should be less than 1" long — wirewrap is not recommended.
2. V_{CC} and GND connections should be less than 1/2" long to power plane.
3. Supply decoupling includes both high frequency and bulk storage elements.
4. The same considerations apply for 3rd overtone configurations.

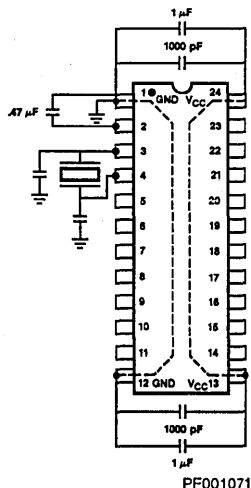


Figure 1. Typical External Connections

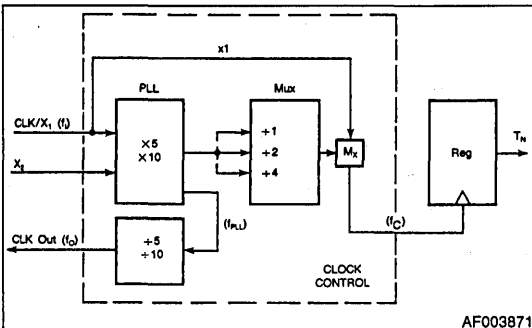
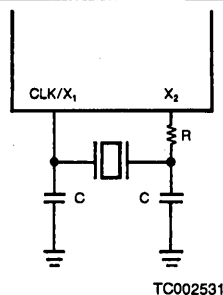


Figure 2. Clock Options

AF003871



$$R = X_C = \frac{1}{2\pi f_1 C} \text{ for 2-6 MHz}$$

$$R = 0 \text{ for 6-20 MHz}$$

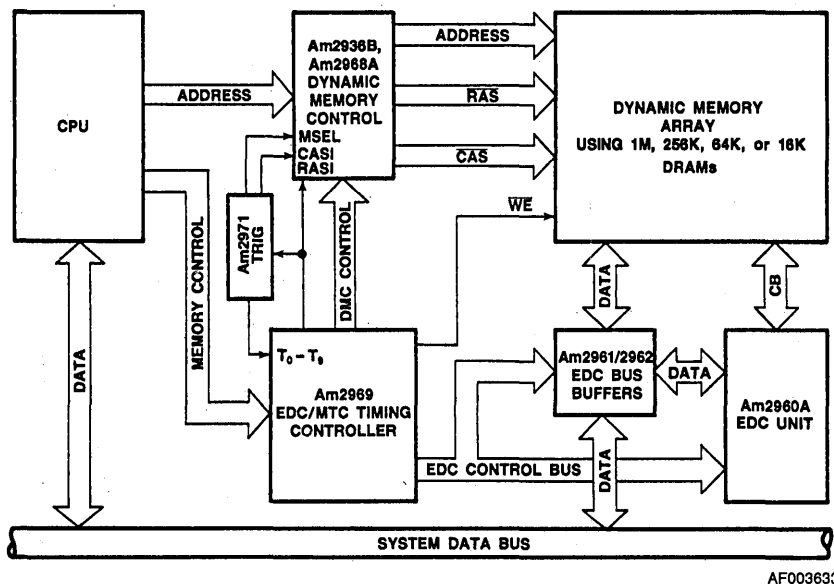
Figure 3. Fundamental Oscillator

APPLICATIONS

The Am2971 Programmable Event Generator (PEG) is a universal, programmable digital delay line and timing/wave-form generator which provides the designer an alternative to the expensive and difficult to use analog delay line. Because of the device's programmability and large number of outputs, it can replace several different delay lines at one time. The user is no longer restricted to the fixed, single event delay line. As one suggested application, this device can easily be used to

generate timing for the Am2968 Dynamic Memory Controller and the Am2969 Memory Timing Controller.

AMD's Dynamic RAM Memory Support system solution is shown in Figure 4. Typically, the Timing Controller generates the Row Address Strobe Input (RAS) signal whenever a refresh or a memory cycle is requested. This signal provides a leading edge to the TRIG input of the PEG chip, thus initiating the user-defined timing sequence. Custom-tailored waveforms are generated at the Timing Outputs ($T_0 - T_{11}$), providing proper and precise sequencing between the Am2968 and Am2969 for optimum system performance.



AF003633

Figure 4. 16-Bit High-Performance Computer Memory System Application

PROGRAMMING

Programming and Verification Overview

The fuse links in the PEG are manufactured using AMD's Platinum-Silicide fuse technology. Before the device is pro-

grammed, it can be verified to be unprogrammed by raising TRIG to V_{OH} , applying appropriate voltages (V_{IHP} and V_{ILP}) to the Timing Outputs (Programming Address Inputs), $T_0 - T_{10}$, lowering CLKOUT to V_{ILP} , and observing the output voltage on T_{11} (Programming Data Input/Output). In the Programming

Verify Mode, Timing Outputs, T_0 – T_{10} , serve as Row and Column Address Inputs for each individual fuse. Timing Output, T_{11} , reflects the condition of the addressed fuse. Initially, T_{11} should be at V_{OLP} as each fuse is addressed, indicating an unprogrammed state.

The fuse to be programmed is selected by applying the appropriate voltage levels (V_{IHP} and V_{ILP}) to the Timing Outputs, T_0 to T_{10} (with T_0 to T_5 acting as Row Address Inputs and T_6 to T_{10} acting as Column Address Inputs). The fuse is programmed with an input voltage, V_{IHP} , on Timing Output, T_{11} , and an input voltage, V_{IHH} , on CLKOUT (see Figure 5 and Table 1).

Most links will open within the programming time as specified in the Programming Parameters Table. Occasionally a link will be stronger and will require additional programming cycles. Successive links are programmed in the same manner until all desired bit locations have been programmed.

When all programming has been completed, the voltage on CLKOUT is lowered to V_{ILP} and all voltage source removed from T_{11} . The entire array should be verified according to the steps in Table 2. The fuse status of each bit location may be observed at T_{11} . An unprogrammed fuse will indicate V_{OLP} and a programmed fuse will indicate V_{OHP} .

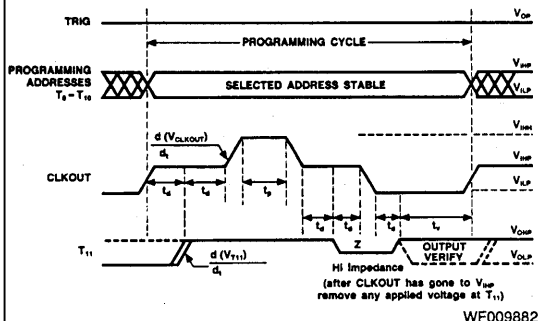


Figure 5. Timing Diagram for Programming and Verify Mode

PROGRAMMING EQUIPMENT INFORMATION

Source and Location	Data I/O
	10525 Willows Rd., N.E. Redmond, WA 98052
Programmer Model(s)	29 or 19
AMD PEG Personality Module	Logicpak 950-1942-001
Socket Adapter	TBA
Family and Device Code	97

The machines noted above have been qualified by AMD to insure high programming yields. Check with the factory to determine the current status of vendors noted TBA or other available models.

TABLE 1. PROGRAMMING PROCEDURE

Step	Item	Description
1	Determine location of fuses which are to be blown.	
2	Set TRIG = V_{OP}	This disables T_0 – T_{11} as outputs and prepares the chip for programming.
3	Set CLKOUT = V_{IHP}	This will three-state T_{11} and cause it to float up to V_{IHP} .
4	Set T_{11} = V_{IHP} after t_D	This prepares T_{11} to accept the programming current which will be gated through CLKOUT.
5	Set CLKOUT = V_{IHH} after t_D for a programming time, t_{PF}	This gates the programming current to T_{11} .
6	Set CLKOUT = V_{IHP}	This removes the programming current from CLKOUT.
7	Remove applied voltage from T_{11}	This sets up T_{11} as output for Program Verification.

TABLE 2. PROGRAM VERIFICATION PROCEDURE

Step	Item	Description
1	Set TRIG = V_{OP} after t_d	TRIG remains at V_{OP} during the entire programming/verify cycle.
2	Set CLKOUT = V_{ILP} after t_d	This enables T_{11} as an output.
3a	Verify T_{11} = V_{OHP}	This condition occurs if programming has been successful.
3b	Verify T_{11} = V_{OLP}	This condition occurs if programming has been unsuccessful.

Notes: 1. If verify indicates programming has been successful, proceed to the next fuse and program using the programming steps of the previous table.
2. If verify indicates programming has been unsuccessful, return to the same fuse and re-attempt programming using the programming time, t_{ps} .

TABLE 3. PROGRAMMING NEXT ADDRESS/EVENT GENERATOR FUSES

PROGRAM FUNCTION	WORD	BIT	TIMING OUTPUTS (PROGRAMMING ADDRESS INPUTS)										PROGRAM FUSE	
			ROW ADDRESS					COLUMN ADDRESS						
			T ₄	T ₃	T ₂	T ₁	T ₀	T ₁₀	T ₉	T ₈	T ₇	T ₆		
NEXT ADDRESS	0	0	0	0	0	0	0	0	0	0	0	0	0	000
	0	1	0	0	0	0	0	0	0	0	0	0	1	001
	0	2	0	0	0	0	0	0	0	0	0	1	0	002
	0	3	0	0	0	0	0	0	0	0	0	1	1	003
	0	4	0	0	0	0	0	0	0	0	1	0	0	004
NEXT EVENT	0	0	0	0	0	0	0	0	0	0	1	0	1	005
	0	1	0	0	0	0	0	0	0	0	1	1	0	006
	0	2	0	0	0	0	0	0	0	0	1	1	1	007
	0	3	0	0	0	0	0	0	0	1	0	0	0	008
	0	4	0	0	0	0	0	0	0	1	0	0	1	009
	0	5	0	0	0	0	0	0	0	1	0	1	0	010
	0	6	0	0	0	0	0	0	0	1	0	1	1	011
	0	7	0	0	0	0	0	0	0	1	1	0	0	012
	0	8	0	0	0	0	0	0	0	1	1	0	1	013
	0	9	0	0	0	0	0	0	0	1	1	1	0	014
	0	10	0	0	0	0	0	0	0	1	1	1	1	015
STOP	0	11	0	0	0	0	0	0	1	0	0	0	0	016
	0	0	0	0	0	0	0	0	1	0	0	0	1	017
NEXT ADDRESS	1	0	0	0	0	0	1	0	0	0	0	0	0	018
	1	0	0	0	0	0	1	0	0	0	0	0	1	019
	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.
NEXT EVENT	30	10	1	1	1	1	0	0	1	1	1	1	1	555
	30	11	1	1	1	1	0	1	0	0	0	0	0	556
STOP	30	0	1	1	1	1	0	1	0	0	0	0	1	557
NEXT ADDRESS	31	0	1	1	1	1	1	0	0	0	0	0	0	558
	31	1	1	1	1	1	1	0	0	0	0	0	1	559
	31	2	1	1	1	1	1	0	0	0	1	0	0	560
	31	3	1	1	1	1	1	0	0	0	0	1	1	561
	31	4	1	1	1	1	1	0	0	1	0	0	0	562
NEXT EVENT	31	0	1	1	1	1	1	0	0	1	0	0	1	563
	31	1	1	1	1	1	1	0	0	1	1	0	0	564
	31	2	1	1	1	1	1	0	0	1	1	1	1	565
	31	3	1	1	1	1	1	0	1	0	0	0	0	566
	31	4	1	1	1	1	1	0	1	0	0	0	1	567
	31	5	1	1	1	1	1	0	1	0	1	0	0	568
	31	6	1	1	1	1	1	0	1	0	1	1	1	569
	31	7	1	1	1	1	1	0	1	1	0	0	0	570
	31	8	1	1	1	1	1	0	1	1	0	0	1	571
	31	9	1	1	1	1	1	0	1	1	1	0	0	572
	31	10	1	1	1	1	1	0	1	1	1	1	1	573
STOP	31	11	1	1	1	1	1	1	0	0	0	0	0	574
	31	0	1	1	1	1	1	1	0	0	0	0	1	575

Notes: 1. During programming of Next Address/Event Generator Fuses, T₅ = V_{ILP}
 2. Refer to Table 8 for additional information.

TABLE 4. PROGRAMMING START ADDRESS GENERATOR FUSES

START ADDRESS WORD	START ADDRESS BIT	TIMING OUTPUTS (PROGRAMMING ADDRESS INPUTS)						PROGRAM FUSE (NOTE 2)
		ROW ADDRESS			COLUMN ADDRESS			
		T ₂	T ₁	T ₀	T ₈	T ₇	T ₆	
0	0	0	0	0	0	1	0	576
0	1	0	0	0	0	1	1	577
0	2	0	0	0	1	0	0	578
0	3	0	0	0	1	0	1	579
0	4	0	0	0	1	1	0	580
1	0	0	0	1	0	1	0	581
1	1	0	0	1	0	1	1	582
1	2	0	0	1	1	0	0	583
1	3	0	0	1	1	0	1	584
1	4	0	0	1	1	1	0	585
2	0	0	1	0	0	1	0	586
2	1	0	1	0	0	1	1	587
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
6	3	1	1	0	1	0	1	609
6	4	1	1	0	1	1	0	610
7	0	1	1	1	0	1	0	611
7	1	1	1	1	0	1	1	612
7	2	1	1	1	1	0	0	613
7	3	1	1	1	1	0	1	614
7	4	1	1	1	1	1	0	615

Notes: 1. T₅ = T₁₀ = V_{IHP}, T₃ = T₄ = T₉ = V_{ILP}
 2. Refer to Table 8 for additional information.

TABLE 5. PROGRAMMING TIMING AND CONTROL FUSES

PROGRAM FUNCTIONS	BIT	TIMING OUTPUTS (PROGRAMMING ADDRESS INPUTS)				PROGRAM FUSE (NOTE 2)	REMARKS
		COLUMN ADDRESS					
		T ₉	T ₈	T ₇	T ₆		
Internal Clock Frequency	0	0	1	1	1	616	See Table 6
	1	1	0	0	0	617	See Table 6
	2	1	0	0	1	618	See Table 6
	3	1	0	1	0	619	See Table 6
External Clock Frequency	0	1	0	1	1	620	See Table 7
TRIG Input Polarity	0	1	1	0	0	621	See Table 7
Stop Function	0	1	1	0	1	622	See Table 7

Notes: 1. T₁₀ = T₅ = T₃ = V_{IHP}, T₄ = T₂ = T₁ = T₀ = V_{ILP}
 2. Refer to Table 8 for additional information.

TABLE 6. INTERNAL CLOCK FREQUENCY TRUTH TABLE

INTERNAL CLOCK FREQUENCY	FUSE STATE				
	COLUMN (NOTE 4)	26	25	24	23
	BIT	3	2	1	0
f _C = f _I x 10		0	0	0	0
f _C = f _I x 10 x 1/2		0	0	0	1
f _C = f _I x 10 x 1/4		0	0	1	0
f _C = f _I x 5		1	0	0	0
f _C = f _I x 5 x 1/2		1	0	0	1
f _C = f _I x 5 x 1/4		1	0	1	0
f _C = f _I		X	1	0	0

Notes: 1. 1 = Programmed
 0 = Unprogrammed
 X = Don't Care
 2. f_I = Input Frequency
 f_C = Internal Clock Frequency
 3. Refer to Table 5 for additional information.
 4. Column Numerals in this table are decimal representations of the binary column address signals applied to Timing Inputs, T₁₀–T₆, to program or verify individual fuses (see Table 8).

TABLE 7. CLKOUT/TRIG POLARITY/STOP BIT TRUTH TABLE

FUNCTION	FUSE STATE				REMARKS
	COLUMN (NOTE 5)	27	28	29	
Output Clock Frequency		0 1	X X	X X	$f_O = f_C/10$ $f_O = f_{PLL}/5$
TRIG Polarity		X X	0 1	X X	Positive Edge Trigger Negative Edge Trigger
Stop (Note 4)		X X	X X	0 1	Stop On TRIG Trailing Edge Stop When Stop Bit = 1

Notes: 1. 1 = Programmed
 0 = Unprogrammed
 X = Don't Care
 2. f_O = Output Clock Frequency (CLKOUT)
 f_{PLL} = PLL Internal Clock Frequency
 3. Refer to Table 8 for additional information.
 4. If the Stop Bit is unprogrammed, the sequencer will be free-running and will not halt until a new TRIG pulse is issued.
 5. Column numerals in this table are decimal representations of the binary column address signals applied to Timing Inputs, T₁₀–T₆, to program or verify individual fuses (see Table 8).

TABLE 8. JEDEC FUSE NUMBERS*

FUSE																													BLOCK			
ROW	COLUMN																															
	L S B	NEXT ADDRESS	M S B	L S B	CURRENT EVENT T ₀ -T ₁₁												M S B	S B I O P	L S B	M S B												
					0	1	2	3	4	5	6	7	8	9	10	11					12	13	14	15	16	17	18	19		20	21	22
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	—	—	—	—	—	—	—	—	—	—	—	—	—	
1	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	—	—	—	—	—	—	—	—	—	—	—	—		
2	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	—	—	—	—	—	—	—	—	—	—	—	—		
3	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	—	—	—	—	—	—	—	—	—	—	—	—		
4	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	—	—	—	—	—	—	—	—	—	—	—	—		
5	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	—	—	—	—	—	—	—	—	—	—	—	—		
6	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	—	—	—	—	—	—	—	—	—	—	—	—		
7	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	—	—	—	—	—	—	—	—	—	—	—	—		
8	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	—	—	—	—	—	—	—	—	—	—	—	—		
9	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	—	—	—	—	—	—	—	—	—	—	—	—		
10	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	—	—	—	—	—	—	—	—	—	—	—	—		
11	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	—	—	—	—	—	—	—	—	—	—	—	—		
12	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	—	—	—	—	—	—	—	—	—	—	—	—		
13	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	—	—	—	—	—	—	—	—	—	—	—	—		
14	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	—	—	—	—	—	—	—	—	—	—	—	—		
15	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	—	—	—	—	—	—	—	—	—	—	—	—		
16	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	—	—	—	—	—	—	—	—	—	—	—	—		
17	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	—	—	—	—	—	—	—	—	—	—	—	—		
18	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	—	—	—	—	—	—	—	—	—	—	—	—		
19	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	—	—	—	—	—	—	—	—	—	—	—	—		
20	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	—	—	—	—	—	—	—	—	—	—	—	—		
21	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	—	—	—	—	—	—	—	—	—	—	—	—		
22	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	—	—	—	—	—	—	—	—	—	—	—	—		
23	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	—	—	—	—	—	—	—	—	—	—	—	—		
24	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	—	—	—	—	—	—	—	—	—	—	—	—		
25	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	—	—	—	—	—	—	—	—	—	—	—	—		
26	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	—	—	—	—	—	—	—	—	—	—	—	—		
27	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	—	—	—	—	—	—	—	—	—	—	—	—		
28	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	—	—	—	—	—	—	—	—	—	—	—	—		
29	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	—	—	—	—	—	—	—	—	—	—	—	—		
30	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	—	—	—	—	—	—	—	—	—	—	—	—		
31	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	—	—	—	—	—	—	—	—	—	—	—	—		
32	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	576	577	578	579	580	—	—	—	—	—	—	—		
33	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	581	582	583	584	585	—	—	—	—	—	—	—		
34	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	586	587	588	589	590	—	—	—	—	—	—	—		
35	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	591	592	593	594	595	—	—	—	—	—	—	—		
36	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	596	597	598	599	600	—	—	—	—	—	—	—		
37	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	601	602	603	604	605	—	—	—	—	—	—	—		
38	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	606	607	608	609	610	—	—	—	—	—	—	—		
39	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	611	612	613	614	615	—	—	—	—	—	—	—		
40	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	616	617	618	619	620	621	622	
CLOCK CONTROL & TRIGGER POLARITY FUSES**																																

NEXT ADDRESS/
EVENT
GENERATOR FUSESSTART ADDRESS
FUSESCLOCK CONTROL &
TRIGGER POLARITY FUSES**

*Row and Column numerals in this table are decimal number representations of the Binary Row and Column address signals applied to T₅-T₀ and T₁₀-T₆ respectively to program or verify each fuse individually. In this table, the reader should not confuse the use of T₁₀-T₀ as address inputs to program or verify fuses and the use of T₁₁-T₀ as Timing Outputs providing 12 programmable, registered output waveforms during normal operation (the output, T₁₁, provides the fuse condition during program or verify).

**Includes: Internal CLK, CLKOUT, TRIG Polarity and Stop Bit.

PROGRAMMING PARAMETERS ($T_A = 25^\circ\text{C}$) (Note 1)

Parameters	Description	Min.	Typ.	Max.	Units
V_{IH}	Control Pin Extra High Level	11.0	12.0	13.0	V
V_{OP}	Program Voltage at 15–200 mA	14.0	15.0	16.0	V
V_{IHP}	Input HIGH Level During Programming and Verify	2.4	5.0	5.5	V
V_{ILP}	Input LOW Level During Programming and Verify	0.0	0.3	0.5	V
V_{CCP}	V _{CC} During Programming @ $I_{CC} = 250\text{ mA}$	Com'I	5.0	5.2	V
		Mil	5.0	5.2	
dV_{T11}/dt	Rate of Output Voltage Change (T_{11} Rising Edge)	20		250	V/ μs
dV_{CLKOUT}/dt	Rate of Fuse Enable Voltage Change (CLKOUT Rising Edge)	100		1000	V/ μs
t_p	Programming Time First Attempt, t_{pf}	40.0	50.0	100.0	μs
	Programming Time Subsequent Attempts, t_{ps}	4.0	5.0	10.0	ms
t_D	Delays Between Various Level Changes	100.0	200.0	1000.0	ns
t_V	Period During which Timing Output, T_n is Valid for Program Verification			500	ns
V_{OHP}	Output HIGH Level During Programming and Verify	2.5	5.0	5.5	V
V_{OLP}	Output LOW Level During Programming and Verify	0.0	0.3	0.4	V

Notes: 1. Parameters are guaranteed by design — not tested.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65°C to +150°C
 Ambient Temperature with
 Power Applied -55°C to +125°C
 Supply Voltage to Ground Potential
 Continuous (TTL V_{CC} and PLL V_{CC}) 0 V to +7.0 V
 DC Voltage Applied to Outputs For
 High Output State 0 V to +V_{CC} max.
 DC Input Voltage -0.5 V to +5.5 V
 DC Input Current -18 mA to +5.0 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES**Commercial (C) Devices**

Temperature (T_A) 0 to +70°C
 TTL V_{CC} and PLL V_{CC} 5.0 V ±10%
 Min. 4.50 V
 Max. 5.50 V

Extended Commercial (E) or Military* (M) Devices

Temperature (T_C) -55 to +125°C
 TTL V_{CC} and PLL V_{CC} 5.0 V ±10%
 Min. 4.50 V
 Max. 5.50 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military Product 100% tested at T_C = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions		Min.	Max.	Units
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = -1 mA	COM'L. 2.7		V
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = 8 mA		0.4	V
V _{IH} (Note 1)	Input HIGH Voltage	Guaranteed Input HIGH Voltage for All Inputs		2.0		V
V _{IL} (Note 1)	Input LOW Voltage	Guaranteed Input LOW Voltage for All Inputs			0.8	V
V _{IHC}	Input HIGH Voltage to CLK/X ₁	Guaranteed Input HIGH Voltage for All Inputs		3.0		V
V _{ILC}	Input LOW Voltage to CLK/X ₁	Guaranteed Input LOW Voltage for All Inputs			0.8	V
V _I (Note 1)	Input Clamp	V _{CC} = Min.	I _{IN} = -18 mA		-1.2	V
I _{IH}	Input HIGH Current	V _{CC} = Min., V _{IN} = 3.0 V	CLK/X ₁ , X ₂		700	μA
		V _{CC} = Max., V _{IN} = 2.7 V	A ₀ - A ₂ and TRIG		20	
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.5 V	CLK/X ₁ , X ₂		-500	μA
		V _{CC} = Max., V _{IN} = 0.5 V	A ₀ - A ₂ and TRIG		-250	
I _I	Input Current	V _{CC} = Min., V _{IN} = 4.0 V	CLK/X ₁ , X ₂		1	mA
		V _{CC} = Max., V _{IN} = 5.5 V	A ₀ - A ₂		100	
		V _{CC} = Max., V _{IN} = V _{CC} - 0.5	TRIG		100	
I _{SC} (Note 2)	Output Short Circuit Current	V _{CC} = Max., V _{IN} = 0 V	V _O = 0 V	-15	-100	mA
I _{CC} (Note 3)	Power Supply Current	V _{CC} = Max.	T = -55°C, 0°C, +25°C		310	mA
			T = +70°C, +125°C		240	

Notes: 1. Does not apply to CLK/X₁ and X₂.

2. No more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.

3. I_{CC} varies with temperature and oscillation frequency.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified

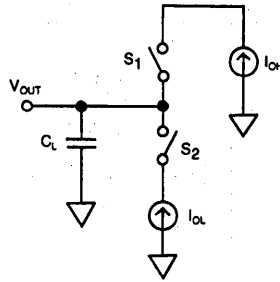
No.	Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Units
1	FREQ IN @ CLK/X ₁	CLK/X ₁ Input Clock Frequency (Note 2)	a) PLL Frequency Multiplication Mode (TTL Input @ CLK/X ₁)	10	70	MHz
			b) PLL Frequency Multiplication Mode (Crystal @ CLK/X ₁ , X ₂)			
			c) Flow-through Mode (PLL bypassed) (TTL Input @ CLK/X ₁)	0	85	
			d) Flow-through Mode (PLL bypassed) (Crystal @ CLK/X ₁ , X ₂)			
2	t _{RISE} @ CLKOUT	CLKOUT Rise Time (Note 2)			14	ns
3	t _{FALL} @ CLKOUT	CLKOUT Fall Time (Note 2)			10	ns
4	t _{RISE} @ T ₀ -T ₁₁	T ₀ -T ₁₁ Rise Time (Note 2)			10	ns
5	t _{FALL} @ T ₀ -T ₁₁	T ₀ -T ₁₁ Fall Time (Note 2)			9	ns
6	t _{SKEW} @ T ₀ -T ₁₁	Skews between the T ₀ -T ₁₁ Outputs (LOW-to-HIGH Transition)	C _L = 50 pF		3.5	ns
7	t _{SKEW} @ T ₀ -T ₁₁	Skews between the T ₀ -T ₁₁ Outputs (HIGH-to-LOW Transition)	C _L = 50 pF		5.0	ns
8	t _{SKEW} @ T ₀ -T ₁₁	Skews between the T ₀ -T ₁₁ Outputs (Mixed Transition)	C _L = 50 pF		8.5	ns
9	t _{SET} TRIG to CLK/X ₁	TRIG  to CLK/X ₁ Setup Time	a) PLL Frequency Multiplication Mode (Note 2) b) Flow-through Mode (PLL bypassed)		11	ns
10	t _{SET} TRIG to CLK/X ₁	TRIG  to CLK/X ₁ Setup Time	a) PLL Frequency Multiplication Mode (Note 2) b) Flow-through Mode (PLL bypassed)		6	ns
11	t _{PD} CLK/X ₁ to T ₀ -T ₁₁	Propagation Delay from CLK/X ₁  to the T ₀ -T ₁₁ Outputs	a) PLL Frequency Multiplication Mode (Note 2) b) Flow-through Mode (PLL bypassed)		25	ns
12	t _{PD} CLK/X ₁ to CLKOUT	Propagation Delay from CLK/X ₁ to the CLKOUT Outputs	PLL Frequency Multiplication Mode (Note 2)		17.5	ns
13	t _{SET} A ₀ -A ₂ to TRIG	A ₀ -A ₂ Inputs to TRIG  (Note 3)	PLL Frequency Multiplication Mode/Flow-through Mode (PLL bypassed)		1.0	ns
14	t _{HOLD} A ₀ -A ₂ to TRIG	A ₀ -A ₂ Inputs to TRIG  Hold Time (Note 3)	PLL Frequency Multiplication Mode/Flow-through Mode (PLL bypassed)		11.0	ns

Calculated Switching Characteristics

15	t _{PERIOD} Resolution @ T ₀ -T ₁₁	Timing Resolution between the T ₀ -T ₁₁ Outputs		1/f _I (1a or 1c)
16	tp _{WH} @ TRIG (Calculated)	TRIG Input Pulse Width (HIGH State) (Note 1)	a) PLL Frequency Multiplication Mode	t _{SET} (9a) + 1/f _C + 5
			b) Flow-through Mode (PLL bypassed)	t _{SET} (9b) + 1/f _I + 5
17	tp _{WL} @ TRIG (Calculated)	TRIG Input Pulse Width (LOW State) (Note 1)	a) PLL Frequency Multiplication Mode	1/f _C
			b) Flow-through Mode (PLL bypassed)	t _{SET} (10b)
18	t _{PD} TRIG T ₀ -T ₁₁ (Calculated)	Delay from TRIG  to T ₀ -T ₁₁ Active	a) PLL Frequency Multiplication Mode	t _{SET} (9a) + 1/f _C + t _{PD} (11a)
			b) Flow-through Mode (PLL bypassed)	t _{SET} (9b) + 1/f _I + t _{PD} (11b)
19	t _{PD} TRIG T ₀ -T ₁₁ (Calculated)	Delay from TRIG  to T ₀ -T ₁₁ Inactive	a) PLL Frequency Multiplication Mode	t _{SET} (10a) + 1/f _C + t _{PD} (11a)
			b) Flow-through Mode (PLL bypassed)	t _{SET} (10b) + 1/f _I + t _{PD} (11b)

Notes: 1. Not Tested; calculated using other parameters.
 2. Not Tested; correlated.
 3. Only A₂ is tested.

SWITCHING TEST CIRCUIT

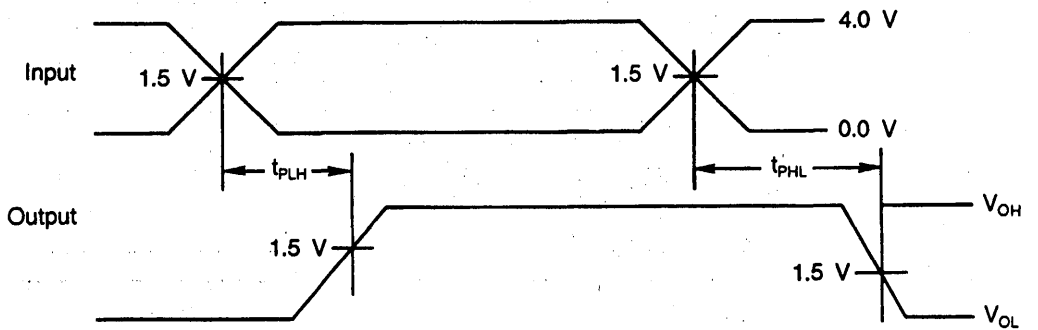


TC003132

A. Outputs

- Notes:
1. $C_L = 50$ pF, the load capacitance includes scope probe, wiring, and stray capacitance without the device in the test fixture.
 2. S_1 and S_2 are open during all DC and functional testing
 3. During AC testing, switches are set as follows:
 - 1) For $V_{OUT} > 1.5$ V, S_1 is closed and S_2 open
 - 2) For $V_{OUT} < 1.5$ V, S_1 is open and S_2 closed

SWITCHING TEST WAVEFORM



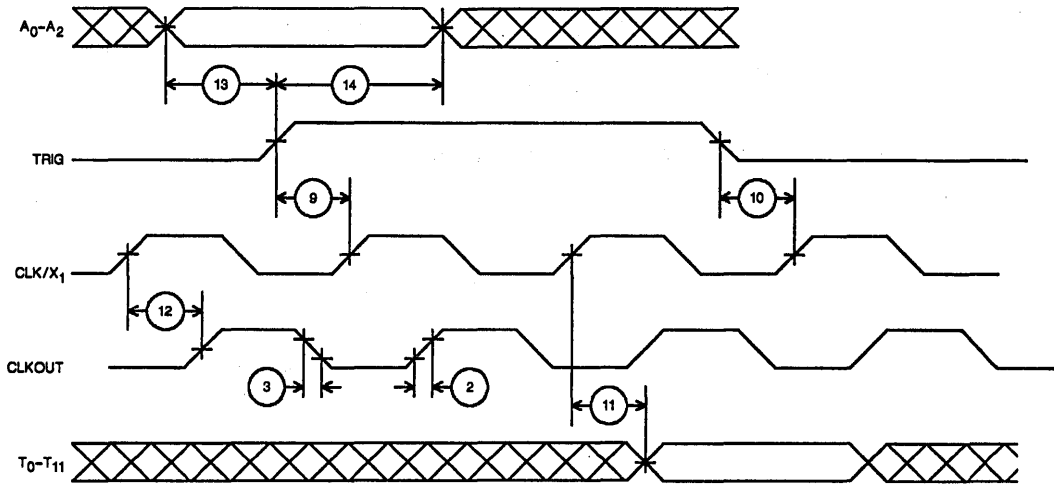
WF021341

KEY TO SWITCHING WAVEFORMS

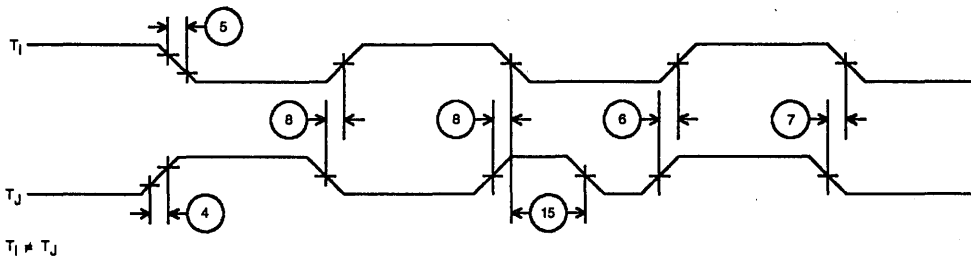
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS



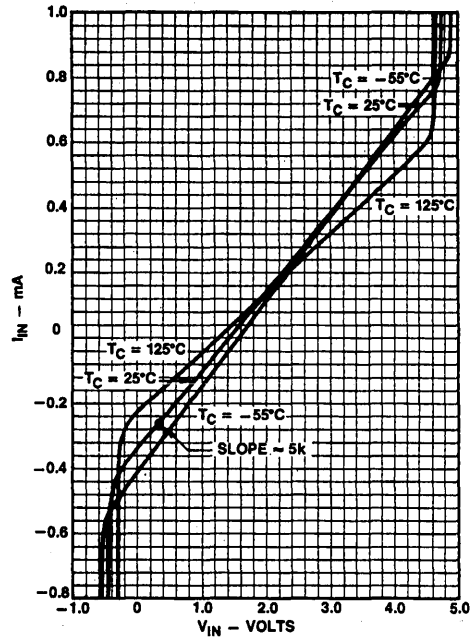
WF022521



WF022530

Rise Time/Fall Time/Skews

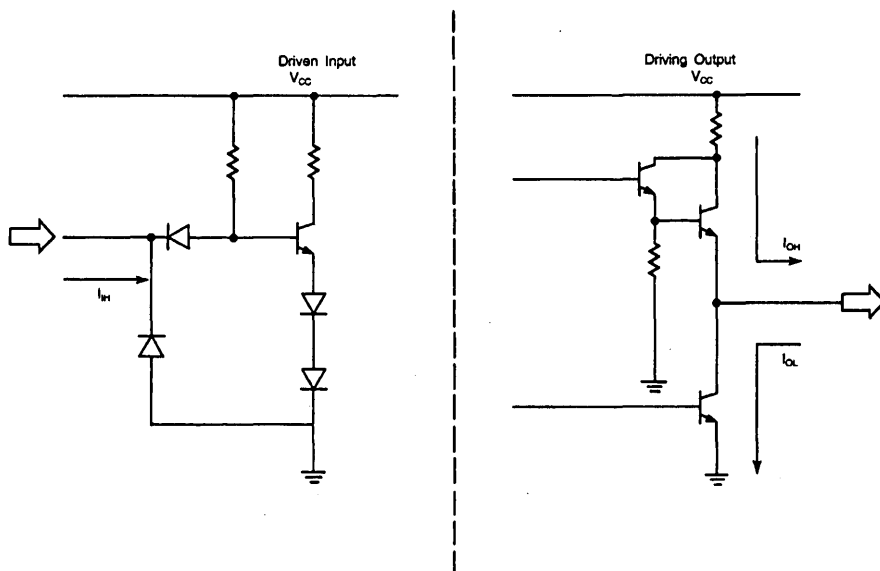
TYPICAL PERFORMANCE CURVE



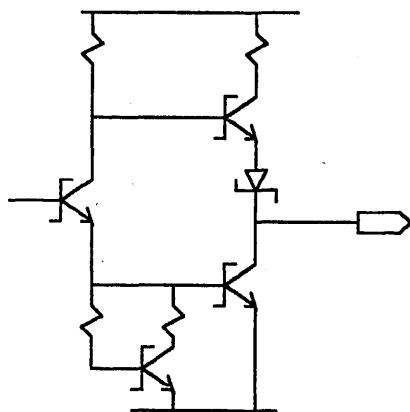
PF001081

CLK/X₁ Crystal Input Characteristics

INPUT/OUTPUT CIRCUIT DIAGRAMS



IC000881



IC000810

Output Configuration for CLKOUT

SECTION 5 — GENERAL INFORMATION



- 5.1 Cross Reference Guide**
- 5.2 Package Outlines**
- 5.3 Technical Report — Package Thermal Characteristics**
- 5.4 AMD Sales Offices**

5.1 CROSS REFERENCE GUIDE

AmPAL18P8 (Generic)

Competitors Part Numbers

MMI	National	T.I.	Other	AMD
PAL10H8 PAL10L8 PAL12H6 PAL12L6 PAL14H4 PAL14L4 PAL16H2 PAL16L2 PAL16P8	DMPAL10H8 DMPAL10L8 DMPAL12H6 DMPAL12L6 DMPAL14H4 DMPAL14L4 DMPAL16H2 DMPAL16L2 DMPAL16P8			AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8 AmPAL18P8
			PLHS18P8*	

Competitors Speed/Power Version

MMI			National			T.I.			Other			AMD		
COML MIL			COML MIL			COML MIL			COML MIL			COML MIL		
TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}
-2	60/45	80/45	A-2	35/45	50/45							Q	35/45	40/55
	35/90	45/90		35/90	45/90							L	35/90	40/90

20-PIN PAL DEVICES (Generic)

Competitors Part Numbers

MMI	National	T.I.	Other	AMD
PAL16L8 PAL16R4 PAL16R6 PAL16R8	DMPAL16L8 DMPAL16R4 DMPAL16R6 DMPAL16R8	(TIB)PAL16L8 (TIB)PAL16R4 (TIB)PAL16R6 (TIB)PAL16R8		AmPAL16L8 AmPAL16R4 AmPAL16R6 AmPAL16R8

Competitors Speed/Power Version

MMI			National			T.I.			Other			AMD		
COML MIL			COML MIL			COML MIL			COML MIL			COML MIL		
TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE ¹	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}
A-2	35/90	50/90	A-2	50/90	40/90	A-2	35/90	40/95				L ²	35/90	40/90
A-4	55/50	75/50										Q	35/45	40/45
A	25/180	30/180	A	25/180	30/180	A	25/180	30/185				A ³	25/180	30/180
B-2	25/90	30/90	B-2	25/90	30/90	-25/-30	25/100	30/105				AL	25/90	30/90
B-4	35/55	50/55										Q	35/45	40/45
B	15/180	20/180	B	15/180	20/180	-15/-20	15/180	20/190				B	15/180	20/180
	35/180	45/180		35/180	45/180							Note 3	35/180	40/180

*Signetics

**Cypress

Notes: 1. Types with two numbers follow this nomenclature: Commercial/Military.

2. I_{CC} = 80 mA for AmPAL16L8

3. I_{CC} = 155 mA for AmPAL16L8

4. Extended Commercial temperature range.

AmPAL20EV8						
Competitors Part Numbers						
MMI	National	T.I.	Other	AMD		
				AmPAL10H20EV8 AVAIL. Q486 AmPAL10020EV8 AVAIL. Q486 AmPAL10H20EG8 AVAIL. Q486 AmPAL10020EG8 AVAIL. Q486		
Competitors Speed/Power Version						
		AMD	MIL			
		COML				
TYPE		t_{PD}/I_{EE}	t_{PD}/I_{EE}			
-6		6/220				
-8		8/220 ⁴	8/220			
AmPAL22V10						
Competitors Part Numbers						
MMI	National	T.I.	Other	AMD		
		PAL22V10	CYPALC22V10**	AmPAL22V10		
Competitors Speed/Power Version						
T.I.			CYPRESS		AMD	
	COML	MIL	COML	MIL	COML	MIL
TYPE	t_{PD}/I_{CC}	t_{PD}/I_{CC}	TYPE ¹	t_{PD}/I_{CC}	t_{PD}/I_{CC}	t_{PD}/I_{CC}
A	25/180	30/180	-25/-30	25/80	30/100	A 25/180 30/180
			-35/-40	35/80	40/100	BLANK 35/180 40/180
AmPAL23S8						
Competitors Part Numbers						
MMI	National	T.I.	Other	AMD		
				AmPAL23S8 AVAIL. Q486		
Competitors Speed/Power Version						
			AMD			
			COML		MIL	
TYPE			t_{PD}/I_{CC}		t_{PD}/I_{CC}	
-20			20/200			
-25			25/200			
-27			27/200 ⁴		27/200	
-30			30/200 ⁴		27/200	
ENHANCED 24-PIN XOR PAL DEVICES						
Competitors Part Numbers						
MMI	National	T.I.	Other	AMD		
PAL20X4	DMPAL20X4	TIBPAL20X4		AmPAL20XRP4 AVAIL. Q486		
PAL20X8	DMPAL20X8	TIBPAL20X8		AmPAL20XRP6 AVAIL. Q486		
PAL20X10	DMPAL20X10	TIBPAL20X10		AmPAL20XRP8 AVAIL. Q486		
				AmPAL20XRP10 AVAIL. Q486		
				AmPAL22XP10 AVAIL. Q486		

See notes on page 5-1.

Competitors Speed/Power Versions														
MMI			National			T.I.			Other			AMD		
	COML	MIL		COML	MIL		COML	MIL		COML	MIL		COML	MIL
TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE ¹	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE ¹	t _{PD} /I _{CC}	t _{PD} /I _{CC}
						20/25	20/180	25/180				-20/-25	20/210	25/210
A	30/180	35/180										-30/-35	30/180	35/180
												-30L/-35L	30/90	35/90
	50/180	60/180		50/180	60/180							-40/-45	40/180	45/180
												-40L/-45L	40/90	45/90

STANDARD 24-PIN PAL DEVICES

Competitors Part Numbers

MMI		National		T.I.		Other		AMD	
PAL20L8		DMPAL20L8		(TIB)PAL20L8				AmPAL20L8 AVAIL. Q486	
PAL20R4		DMPAL20R4		(TIB)PAL20R4				AmPAL20R4 AVAIL. Q486	
PAL20R6		DMPAL20R6		(TIB)PAL20R6				AmPAL20R6 AVAIL. Q486	
PAL20R8		DMPAL20R8		(TIB)PAL20R8				AmPAL20R8 AVAIL. Q486	
PAL20L10		DMPAL20L10		(TIB)PAL20L10				AmPAL20L10 AVAIL. Q486	

Competitors Speed/Power Versions

MMI			National			T.I.			Other			AMD		
	COML	MIL		COML	MIL		COML	MIL		COML	MIL		COML	MIL
TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}
A-2	35/105	50/105				A-2	35/100	40/100				AL	25/105	30/105
A	25/210	30/210	A	25/210	30/210	A	25/210	30/210				A	25/210	30/210
B	15/210	20/210	B	15/210	20/210	B	15/210	20/210				B	15/210	20/210

ENHANCED 24-PIN PAL DEVICES

Competitors Part Numbers

MMI		National		T.I.		Other		AMD	
		DMPAL20RP4						AmPAL20RP4 AVAIL. Q486	
		DMPAL20RP6						AmPAL20RP6 AVAIL. Q486	
		DMPAL20RP8						AmPAL20RP8 AVAIL. Q486	
								AmPAL20RP10 AVAIL. Q486	
								AmPAL22P10 AVAIL. Q486	

Competitors Speed/Power Versions

MMI			National			T.I.			Other			AMD		
	COML	MIL		COML	MIL		COML	MIL		COML	MIL		COML	MIL
TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}	TYPE	t _{PD} /I _{CC}	t _{PD} /I _{CC}
A-2	35/105	50/105				A-2	35/100	40/100				AL	25/105	30/105
A	25/210	30/210	A	25/210	30/210	A	25/210	30/210				A	25/210	30/210
B	15/210	20/210	B	15/210	20/210	B	15/210	20/210				B	15/210	20/210

See notes on page 5-1.

CROSS REFERENCE GUIDE

Competitors Package/Temperature Guide

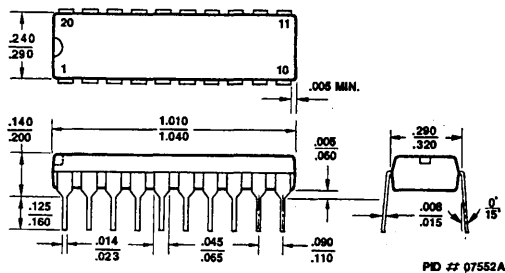
MMI	National	T.I.	AMD	Package Type
CJ	JC	CJ	DC	Ceramic DIP-Commercial
CL			LC	Ceramic Leadless Chip Carrier – Commercial
CN	NC	CN	PC	Plastic DIP – Commercial
CNL		CFN	JC	Plastic Leaded Chip Carrier – Commercial
MJ	JM	MJ	DE	Ceramic DIP – Extended Commercial
ML			LE	Ceramic Leadless – -Extended Commercial
MJ883B		MJB	/BRA	20-Pin Ceramic DIP – Military
MJ883B		MJB	/BLA	24-Pin Ceramic DIP – Military
ML883B		MFKB	/B2C	20-Pin Ceramic Leadless Chip Carrier – Military
ML883B		MFKB	/B3C	28-Pin Ceramic Leadless Chip Carrier – Military
MW883B			/BSA	20-Pin Ceramic Flatpack – Military
MW882B			/BKA	24-Pin Ceramic Flatpack – Military

See notes on page 5-1.

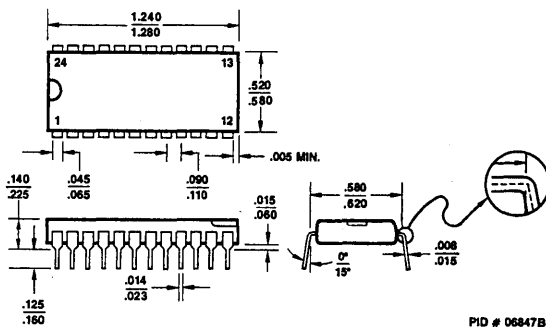
5.2 PACKAGE OUTLINES

Plastic Dual-In-Line Packages (PD)

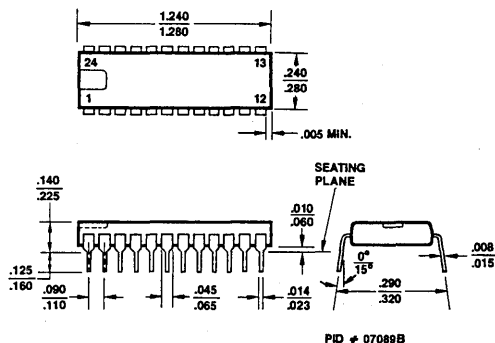
PD 020



PD 024

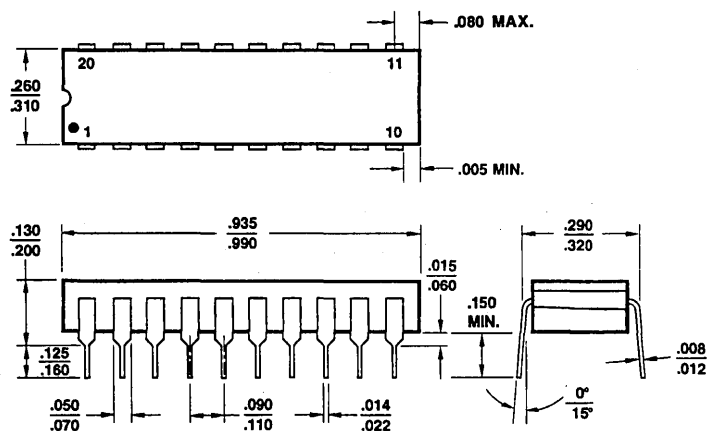


PD3024

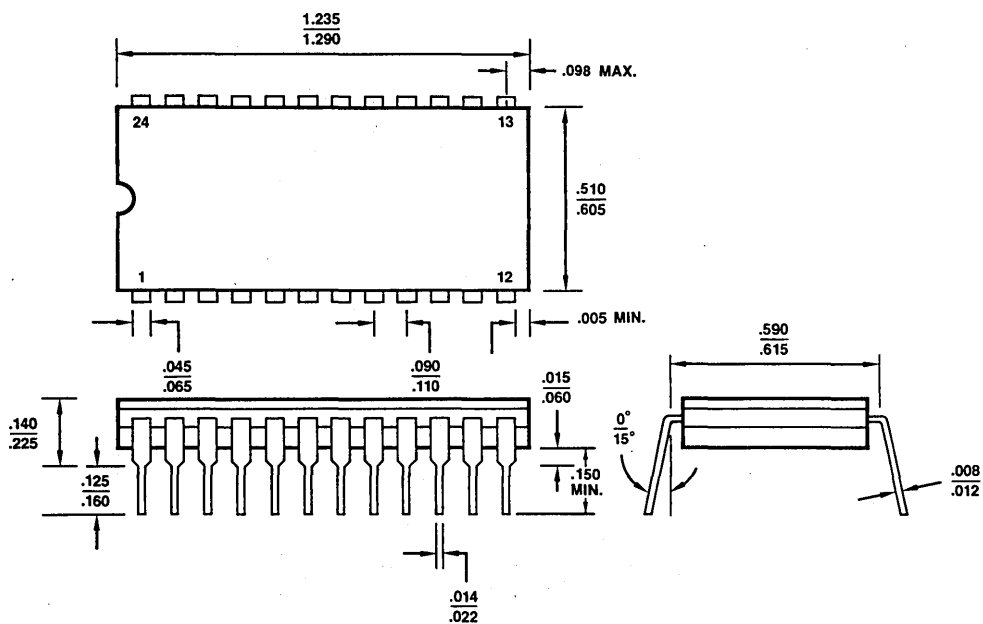


Ceramic Hermetic Dual-In-Line Packages (CD)

CD 020

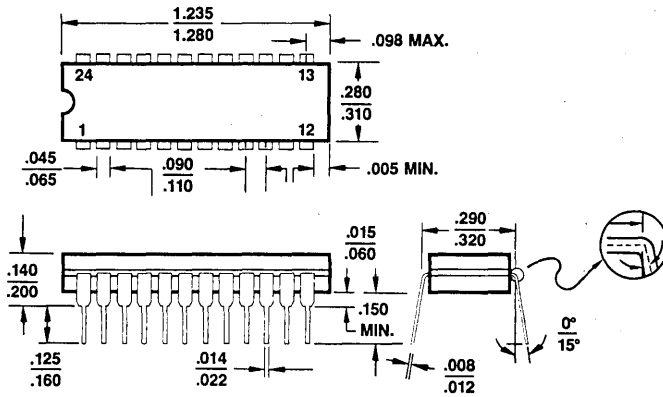


CD 024



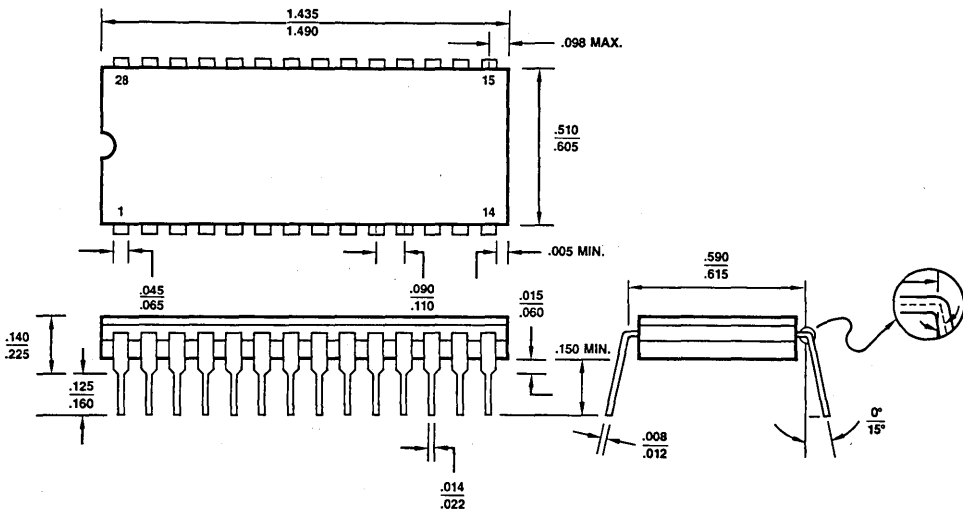
Ceramic Hermetic Dual-In-Line Packages (CD) (Cont'd.)

CD3024

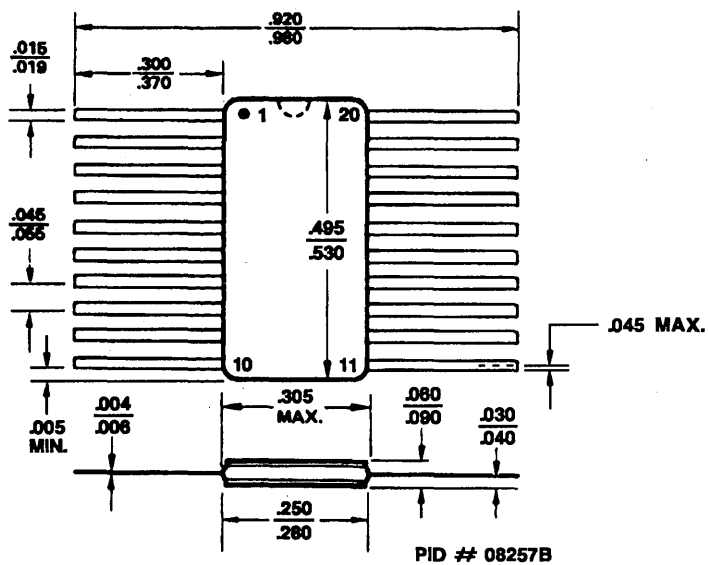
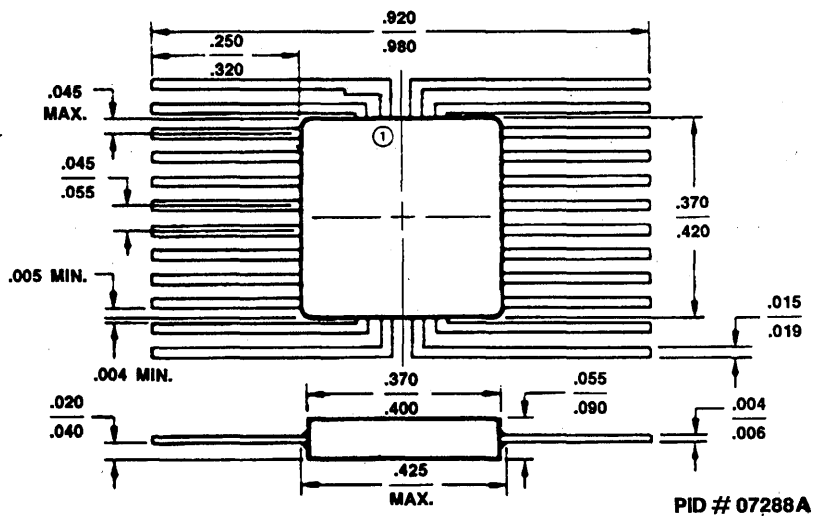


PID # 06850B

CD 028

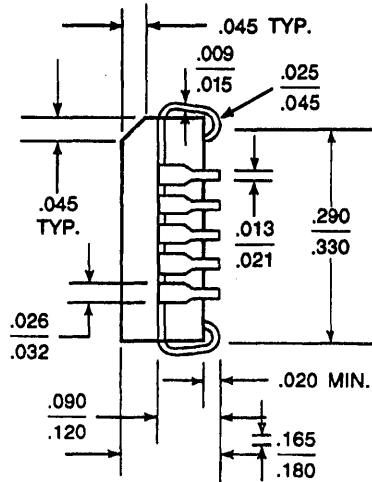
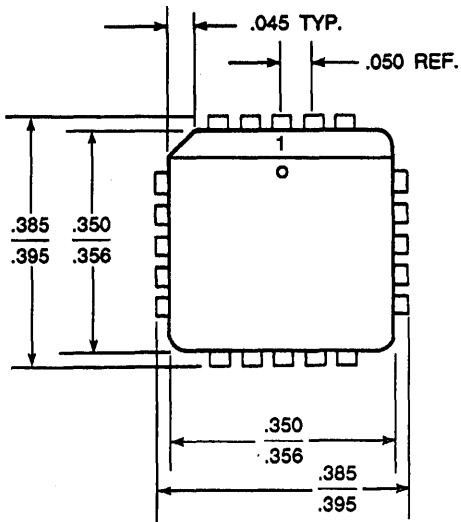


PID # 06837A

CF 020**CF 024**

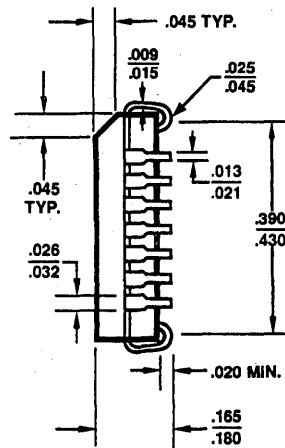
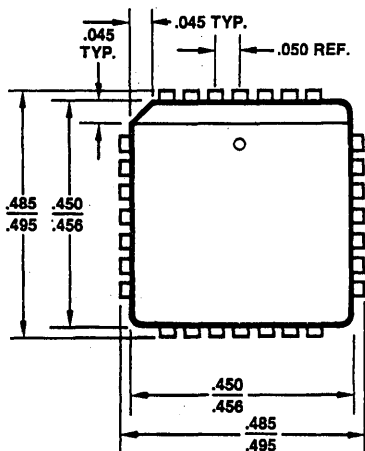
Plastic Leaded Chip Carriers (PL)

PL 020



PID # 06970C

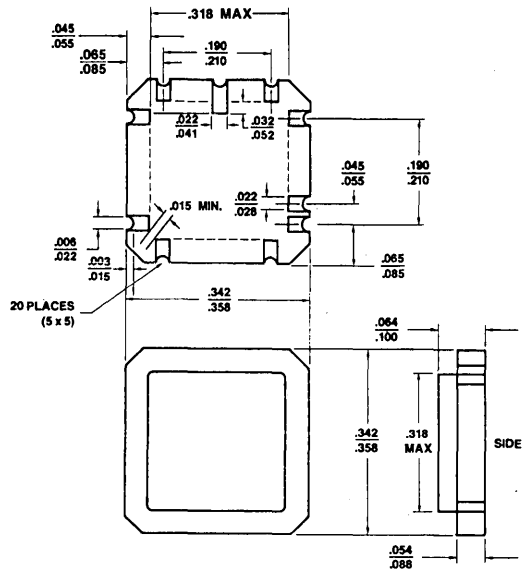
PL 028



PID # 06751D

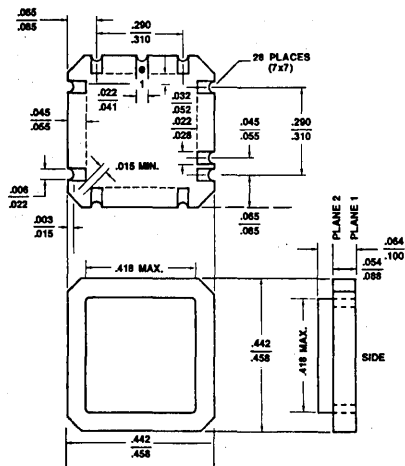
Ceramic Leadless Chip Carriers (CL & CLR)

CL 020



PID # 07318B

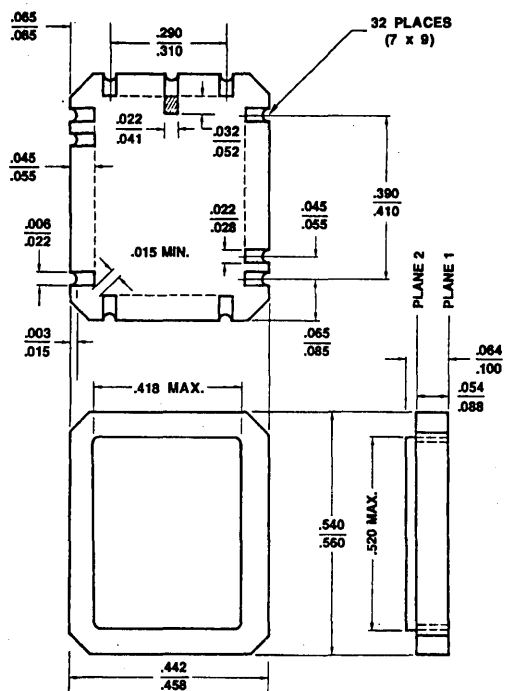
CL 028



PID # 06595D

Ceramic Leadless Chip Carriers (CL & CLR) (Cont'd.)

CLR032



PID # 06841C

5.3 TECHNICAL REPORT — PACKAGE THERMAL CHARACTERISTICS

**THERMAL CHARACTERIZATION
OF
PACKAGE DEVICES
BY James. D. Hayward**

©1986 Advanced Micro Devices

All information herein is company private and the property of Advanced Micro Devices, Inc. ("AMD") and shall not be reproduced or copied or used in whole or in part as the basis of manufacture or sale of product(s) except as expressly permitted or directed by AMD.

Publication #	Rev.	Amendment
08765	A	/0
Issue Date: October 1986		

ABSTRACT

Determination of the Thermal Resistance of Packaged Devices is of concern to the designer of new devices and to AMD customers. The Advanced Package and Material Development group has undertaken the task of characterizing current AMD products and quantifying package-related influences on Thermal Resistance. This report describes some of these effects and the technique used to measure Thermal Resistance.

1.0 DEFINITION OF THERMAL RESISTANCE

The reliability of an integrated circuit is largely dependent on the maximum temperature which the device will attain during operation. Because the stability of a semiconductor junction declines with increasing temperature, knowledge of the thermal properties of the packaged device becomes an important factor during device design. In order to increase the operating lifetime of a given device, the junction temperatures must be minimized. This demands knowledge of the thermal resistance of the completed assembly and specification of the conditions in which the device will function properly. As devices become both smaller and more complex and the requirement for high speed operation becomes more important, heat dissipation will become an ever more critical parameter.

Thermal resistance is defined as the temperature rise per unit power dissipation above some referenced condition. The unit of measure is typically °C/watt. The relationship between junction temperature and thermal resistance is given by:

$$T_j = T_x + P_d \theta_{jx} \quad (1)$$

where: T_j = junction temperature
 T_x = reference temperature
 P_d = power dissipation
 θ_{jx} = thermal resistance
 X = some defined test condition

In general, one of three conditions is defined for measurement of thermal resistance:

- θ_{jc} - thermal resistance measured with reference to the temperature at some specified point on the package surface.
- θ_{ja} (still air) - thermal resistance measured with respect to the temperature of a specified volume of still air.
- θ_{ja} (moving air) - thermal resistance measured with respect to the temperature of air moving at a specified velocity.

The relationship between θ_{jc} and θ_{ja} is

$$\theta_{ja} = \theta_{jc} + \theta_{ca}$$

where θ_{ca} is a measure of the heat dissipation due to natural convection (still air) or forced convection (moving air) and the effect of heat radiation and mounting techniques. θ_{jc} is dependent solely on material properties and package geometry; θ_{ja} includes the influence of the surface area of the package and environmental conditions. Each of these definitions of thermal resistance is an attempt to simulate some manner in which the package device may be used.

The thermal resistance of a packaged device, however measured, is a summation of the thermal resistances of the individual components of the assembly. These in turn are functions of the thermal conductivity of the component materials and the geometry of the heat flow paths. Like other

material properties, thermal conductivity is usually temperature dependent. For alumina and silicon, two common package materials, this dependence can amount to a 30% variation in thermal conductivity over the operating temperature range of the device. The thermal resistance of a component is given by

$$\theta = \frac{L}{K(T)A} \quad (2)$$

where: L = length of the heat flow path
 A = cross sectional area of the heat flow path
 $K(T)$ = thermal conductivity as a function of temperature

and the overall thermal resistance of the assembly (discounting convective effects) will be:

$$\theta = \sum \theta_n = \sum \frac{L_n}{K_n A_n}$$

But since the heat flow path through a component is influenced by the materials surrounding it, determination of L and A is not always straightforward.

A second factor that affects the thermal resistance of a packaged device is the power dissipation level and, more particularly, the relationship between power level and die geometry, i.e., power distribution and power density. By rearrangement of equation 1 to

$$P_d = \frac{1}{\theta_{jx}} (T_j - T_x) = \frac{1}{\sum \theta_n} (T_j - T_x) \quad (3)$$

the relationship between P_d and T_j can be more clearly seen. Thus, to dissipate a greater quantity of heat for a given geometry, T_j must increase and, since the individual θ_n will also increase with temperature, the increase in T_j will not be a linear function of increasing power levels.

A third factor of concern is the quality of the material interfaces. In terms of package construction, this relates specifically to the die attach bond, and for those packages having a heatsink, the heatsink attach bond. The quality of the die attach bond will most severely influence the package thermal resistance as this is the area which first impedes the transfer of heat out of the silicon die. Indeed, it seems likely that the initial thermal response of a powered device can be directly related to the quality of the die attach bond.

2.0 EXPERIMENTAL METHOD

The technique for measurement of thermal resistance involves the identification of a temperature-sensitive parameter on the device and monitoring this parameter while the device is powered. For bipolar integrated circuits the forward voltage of the substrate isolation diode provides a convenient parameter to measure and has the advantage of a linear dependence on temperature. MOS devices which do not have an accessible substrate diode present greater measurement difficulties and may require simulation through use of a specially designed thermal test die. Choice of the parameter to be measured must be made with some care to insure that the results of the measurement are truly representative of the thermal state of the device being investigated. Thus measurement of the substrate isolation diode which is generally diffused across the area of the die yields a weighted average of the condition of

the individual junctions across the die surface. Measurement of a more local source would yield a less generalized result.

For MOS devices, simulation is accomplished using the thermal test die. The basis for this test die is a 25 mil square cell containing an isolated diode and a $1K\Omega$ resistor. The resistors are interconnected from cell to cell on the wafer before it is cut into multiple arrays of the basic unit cell. In use the device is powered via the resistors with voltage or current adjusted for the proper level and the voltage drop of the individual diodes is monitored as in the case of actual devices.

Prior to the thermal resistance test, the diode voltage/temperature calibration must be determined. This is done by measuring the forward voltage at 1mA current level at two different temperatures. The diode calibration factor is then:

$$K_I = \frac{T_2 - T_1}{V_2 - V_1} = \frac{\Delta T}{\Delta V} \quad (4)$$

in units of $^{\circ}C/mV$. For most diodes used for this test the voltage/temperature relationship is linear and these two measurement points are sufficient to determine the calibration.

The actual thermal resistance measurement has two alternating phases: measurement and power on. The device under test is pulse powered with an ON duty cycle of 99% and a repetition rate of $< 100Hz$. During the brief OFF states the device is reverse-biased with a 1mA current and the voltage

drop is measured. The series of voltage readings are averaged over short periods and compared to the voltage reading obtained before the device was first powered ON. The thermal resistance is then computed as:

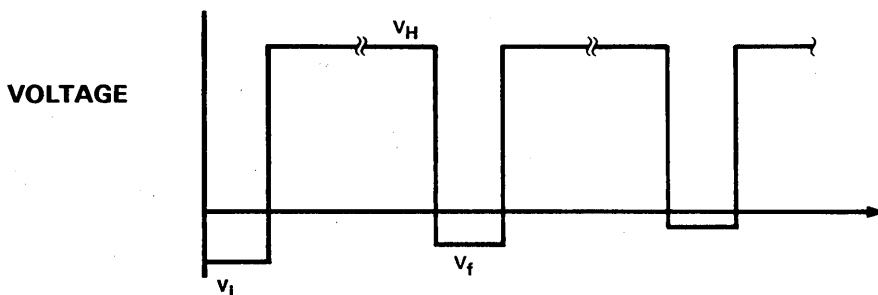
$$\theta_{jx} = \frac{K_I(V_I - V_I)}{V_H I_H} = \frac{K_I \Delta V}{P_d} \quad (5)$$

where: K_I = calibration factor
 V_I = initial forward voltage value
 V_I = current forward voltage value
 V_H = heating voltage
 I_H = heating current

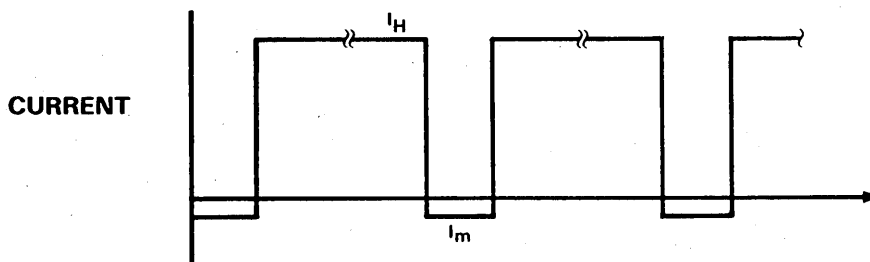
The pulsing measurement is continued until the device has reached thermal equilibrium and the final value measured is the equilibrium thermal resistance of the device under test.

When the end result desired is θ_{ja} (still air), the device and the test fixture (typically a standard burn-in socket) are enclosed in a box containing approximately 1 cubic foot of air. For θ_{jc} measurements the device is attached to a large metal heatsink. This insures that the reference point on the device surface is maintained at a constant temperature. The requirements for measurement of θ_{ja} (moving air) are rather more complex and involve the use of a small wind tunnel with capability for monitoring air pressure, temperature and velocity in the area immediately surrounding the device tested. Standardization of this last test requires much careful attention.

WAVEFORMS FOR PULSED THERMAL RESISTANCE TEST



WF009091



WF009080

3.0 Experimental Results

The thermal resistance data included in Table 1 is representative of the output of tests on representative samples of AMD products. This data has resulted from an on-going program

undertaken by members of the Advanced Package and Material Development group.

The data represents what can be expected from a device in the specified package. Specific device types may differ; these numbers are for example only.

TABLE 1. THERMAL RESISTANCE OF AMD PRODUCTS
(Notes 1, 2, & 3)

Pin Count	Package Type (Note 3)	θ_{JA}	θ_{JC}
20	Ceramic DIP	60	11
	Plastic DIP	61	30
	Ceramic Flatpack	CR	CR
	Ceramic LCC	61	CR
	Plastic LCC*	CR	CR
24	Ceramic DIP	57	15
	Plastic DIP	60	CR
	Ceramic Flatpack	CR	CR
28	Ceramic LCC	CR	CR
	Plastic LCC*	58	CR

- Notes: 1. Representative values for each package type — for information only.
2. Any given device may differ from these values. Consult local AMD sales office for specific-device information.
3. CR = Consult local AMD Representative.
4. DIP = Dual-In-Line Package
LCC = Leadless Chip Carrier
LCC* = Leaded Chip Carrier

ADVANCED MICRO DEVICES DOMESTIC SALES OFFICES

ALABAMA	(205) 882-9122	MARYLAND	(301) 796-9310
ARIZONA,		MASSACHUSETTS	(617) 273-3970
Tempe	(602) 242-4400	MINNESOTA	(612) 938-0001
Tucson	(602) 792-1200	NEW JERSEY	(201) 299-0002
CALIFORNIA,		NEW YORK,	
El Segundo	(213) 640-3210	Liverpool	(315) 457-5400
Newport Beach	(714) 752-6262	Poughkeepsie	(914) 471-8180
San Diego	(619) 560-7030	Woodbury	(516) 364-8020
Sunnyvale	(408) 720-8811	NORTH CAROLINA	(919) 847-8471
Woodland Hills	(818) 992-4155	OREGON	(503) 245-0080
COLORADO	(303) 741-2900	OHIO,	
CONNECTICUT		Columbus	(614) 891-6455
	(203) 264-7800	PENNSYLVANIA,	
FLORIDA,		Allentown	(215) 398-8006
Altamonte Springs	(305) 339-5022	Willow Grove	(215) 657-3101
Clearwater	(813) 530-9971	PUERTO RICO	(809) 764-4524
Ft. Lauderdale	(305) 484-8600	TEXAS,	
Melbourne	(305) 729-0496	Austin	(512) 346-7830
GEORGIA	(404) 449-7920	Dallas	(214) 934-9099
ILLINOIS	(312) 773-4422	Houston	(713) 785-9001
INDIANA	(317) 244-7207	WASHINGTON	(206) 455-3600
KANSAS	(913) 451-3115	WISCONSIN	(414) 782-7748

INTERNATIONAL SALES OFFICES

BELGIUM,		HONG KONG,	
Bruxelles	TEL: (02) 771 99 93	Kowloon	TEL: 3-695377
	FAX: (02) 762-3716		FAX: 1234276
	TLX: 61028		TLX: 50426
CANADA, Ontario,		ITALY, Milano	TEL: (02) 3390541
Kanata	TEL: (613) 592-0090		FAX: (02) 3498000
Willowdale	TEL: (416) 224-5193		TLX: 315286
	FAX: (416) 224-0056	JAPAN, Tokyo	TEL: (03) 345-8241
FRANCE,			FAX: 3425196
Paris	TEL: (01) 45 60 00 55		TLX: J24064 AMDTKOJ
	FAX: (01) 46 86 21 85	LATIN AMERICA,	
	TLX: 202053F	Ft. Lauderdale	TEL: (305) 484-8600
GERMANY,			FAX: (305) 485-9736
Hannover area	TEL: (05143) 50 55		TLX: 5109554261 AMDFTL
	FAX: (05143) 55 53	SWEDEN, Stockholm	TEL: (08) 733 03 50
	TLX: 925287		FAX: (08) 733 22 85
München	TEL: (089) 41 14-0		TLX: 11602
	FAX: (089) 406490	UNITED KINGDOM,	
	TLX: 523883	Manchester area	TEL: (0925) 828008
Stuttgart	TEL: (0711) 62 33 77		FAX: (0925) 827693
	FAX: (0711) 625187		TLX: 628524
	TLX: 721882	London area	TEL: (04862) 22121
			FAX: (04862) 22179
			TLX: 859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA		NEW MEXICO	
I ² INC	OEM (408) 988-3400	THORSON DESERT STATES	(505) 293-8555
	DIST (408) 496-6868	NEW YORK	
IDAHO		NYCOM, INC	(315) 437-8343
INTERMOUNTAIN TECH MKGT	(208) 888-6071	OHIO	
INDIANA		Dayton	
SAI MARKETING CORP	(317) 241-9276	DOLFUSS ROOT & CO	(513) 433-6776
IOWA		Strongsville	
LORENZ SALES	(319) 377-4666	DOLFUSS ROOT & CO	(216) 238-0300
MICHIGAN		PENNSYLVANIA	
SAI MARKETING CORP	(313) 227-1786	DOLFUSS ROOT & CO	(412) 221-4420
NEBRASKA		UTAH	
LORENZ SALES	(402) 475-4660	R ² MARKETING	(801) 595-0631

SECTION 6 — INDEX



A			<i>Documentation</i>	2-11
<i>ABEL</i>	2-3, 2-8		<i>DRAM Register</i>	3-218
<i>Address Decode</i>	3-25		<i>Dynamic Memory Contr.</i>	3-221, 3-252
<i>AmCUPL</i>	2-35			3-281
<i>Analog Subsystem</i>	3-362			
<i>Applications</i>	3-1		E	
<i>Arbiter</i>	3-201, 3-212		<i>EE-Cell</i>	2-60
	3-231, 3-243		<i>EEPROM Technology</i>	2-60
	3-252, 3-315		<i>Encoder</i>	3-9
	3-345		<i>Encoder/Decoder</i>	3-401
			<i>Equivalent Gate Count</i>	2-57
			<i>Error Detection</i>	2-41
B				
<i>Barrel Shifter</i>	3-51, 3-324		F	
<i>Bidirectional Output</i>	1-17		<i>Fingerprint</i>	2-45
<i>Boolean Assembler</i>	2-1		<i>Fuse Characteristics</i>	2-52
<i>Boolean Equation</i>	1-25, 1-27		<i>Fuse Reliability</i>	2-52
<i>Buried State Register</i>	1-22, 1-23		<i>Fuse</i>	1-9, 2-46
<i>Burn-in</i>	2-55			
<i>Bus Interface</i>	3-187		G	
<i>Byte/Word Converter</i>	3-141		<i>Gate Array</i>	1-3
			<i>Gate Count</i>	2-57
C			<i>GCR-Coder</i>	3-11
<i>CEP Evaluation Board</i>	3-159		<i>Glitches</i>	3-25
<i>Checksum</i>	1-32		<i>Gray Code Converter</i>	3-43
<i>Chip Select Decoder</i>	3-25		<i>Gray Counter</i>	3-43
<i>Clocking Scheme</i>	1-21			
<i>CMOS PAL Family</i>	2-61		H	
<i>Combinatorial Design</i>	1-33		<i>Hazard Conditions</i>	3-26
<i>Combinatorial Logic</i>	3-3		<i>Hold Time</i>	3-55
<i>Comparator</i>	3-22			
<i>Compiler</i>	2-1		I	
<i>Compression/Expansion</i>	3-117, 3-159		<i>I/O Architecture</i>	1-16
<i>Control Path</i>	3-1		<i>iAPX Interface</i>	3-60, 3-137
<i>Counter</i>	1-41, 3-29		<i>IMOX</i>	2-46, 2-58
	3-325		<i>Interrupt</i>	3-214
<i>CRT Controller</i>	3-146		<i>Inverted Expression</i>	1-28
<i>CUPL</i>	2-3, 2-12,			
	2-27, 2-28		J	
<i>Customer Logic</i>	1-2		<i>JEDEC File</i>	2-42
			<i>JEDEC Fuse Map</i>	2-38
D			<i>JEDEC</i>	1-32
<i>Data Ciphering Proc.</i>	3-71, 3-76		<i>Johnson Counter</i>	3-38
	3-87, 3-146			
	3-171		L	
<i>Data Funnel</i>	3-142		<i>LANCE Interface</i>	3-64, 3-185
<i>Data Path</i>	3-1		<i>LANCE</i>	3-61, 3-84,
<i>Decoder</i>	3-11			3-185
<i>Dedicated Output</i>	1-18		<i>Life-Test</i>	2-55
<i>Delay Line Substitute</i>	3-398		<i>Load File</i>	2-11
<i>Demultiplexer</i>	3-3		<i>Logic Compiler</i>	2-1
<i>Device Map</i>	1-32		<i>Logic Equation</i>	1-25
<i>DMA Controller</i>	3-65, 3-92,		<i>Logic Simulation</i>	2-2
	3-125, 3-155,		<i>LOGIC</i>	2-3
	3-171, 3-187			
	3-357			

M		S	
Master Device	2-41	Sequencer	1-22
Metastable Operation	3-57	Sequential Design	1-41
Microproc. Interface	3-55	Sequential Logic	3-29
Min Term	1-12	Serial Comm. Contr.	3-76, 3-47, 3-327
Minimum Delay	2-51	Setup Time	3-55
Modulo 360 Counter	3-34	Shifter	3-43, 3-47, 3-327
Multibus Arbiter	3-201, 3-206	Simulation	1-28, 2-2, 2-9
MULTIBUS	3-59, 3-187, 3-201, 3-315	Software	2-1, 2-3, 3-366
Multiplexers	1-34, 3-3	Special Test Input	2-50
N		Standard Products	1-2
Notation	1-8	State Machine	2-27, 3-342
O		Supermini	3-320
Observability	1-23	Syntax	2-14
Output Cell	1-16	T	
Output Enable	1-16	Test Methods	3-382
Output Logic Macrocell	1-19, 1-20	Test Pads	2-47
Output Register	1-20	Test Vector	2-45
P		Testability	2-43, 2-46
PALASM	2-3	Testing	1-32, 2-2, 2-43
PEG	3-396	Thermal Resistance	2-57
PERFECT	2-3	Third-Party Software	2-3
PLA	2-52	Timing Generator	3-229
Platinum-Silicide	2-52	Transmission Protocol	1-33
PLPL	2-3, 2-35, 3-68	U	
Polarity	1-21, 215	Universal Compiler	2-2
Power-up Reset	1-23	V	
Post-Prog. Funct. Yield	2-45	Vectored Interrupt	3-334
Preload	1-23	VME Bus	3-211, 3-357
Preset	1-23	W	
Priority Resolution	3-202, 3-214	Word Rotation	3-326
Product Term	1-12	Z	
Programmer	2-40	Z-Bus	3-191
Programming Language	3-366	Z80, Z8000	3-171, 3-245
Programming Yield	2-44	68000, 68020	3-81, 3-237
Programming	1-28	80186	3-60, 3-301
PROM	1-8, 1-12, 1-15	80286	3-71
R		8087	3-301
Refresh Timer	2-235, 3-244, 3-267	8088, 80188	3-137, 3-191, 3-221
Reliability	2-52	82284/82288	3-290
Reset	1-23	8500	3-97
RLL-Codes	3-401		

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES



**ADVANCED
MICRO
DEVICES, INC.**

901 Thompson Place
P.O. Box 453

Sunnyvale,
California 94086

(408) 732-2400

TWX: 910-339-9280

TELEX: 34-6306

TOLL FREE

(800) 538-8450