Programmable Logic Handbook

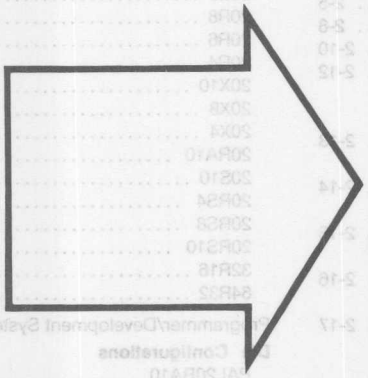
FOURTH EDITION

BROMILOW
Reg No 64 1688 07
P O BOX 1194 RANDBURG 2125
M F KENT HOUSE DOVER STREET RANDBURG TVL 2194
TEL (011) 789 1400/2 TELEX 4-20452 FAX (011) 7870263

Monolithic Memories

PAL®/PLE™ Programmable Logic Handbook

FOURTH EDITION



* PAL®, HAL®, PALASM® and SKINNYDIP® are registered trademarks of Monolithic Memories.
PLE™, MegaPAL™, ZHAL™ and SMAC™ are trademarks of Monolithic Memories.

© Copyright 1978, 1981, 1983, 1985 Monolithic Memories Inc. • 2175 Mission College Blvd. • Santa Clara, CA 95054-1592 • (408) 970-9700 • (910) 339-9220

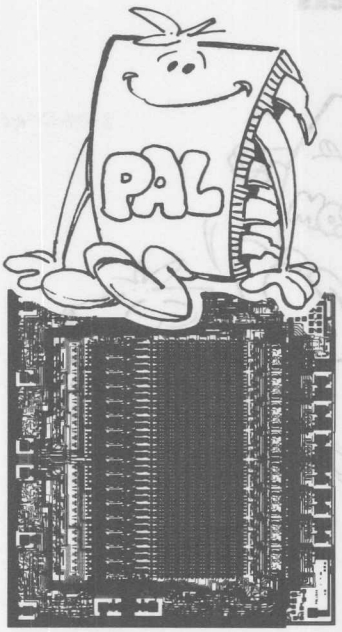
Monolithic Memories MMI

Table of Contents

Table of Contents (cont.)

1

Table of Contents (cont.)



The PAL® Concept

Monolithic Memories' family of PAL devices gives designers a powerful tool with unique capabilities for use in new and existing logic designs. The PAL circuit saves time and money by solving many of the system partitioning and interface problems brought about by increases in semiconductor device technology.

Rapid advances in large-scale integration technology have led to larger and larger standard logic functions; single I.C.s now perform functions that formerly required complete circuit cards. While LSI offers many advantages, advances have been made at the expense of device flexibility. Most LSI devices still require large numbers of SSI/MSI devices for interfacing with user systems. Designers are still forced to turn to random logic for many applications.

The designer is confronted with another problem when a product is designed. Often the function is well defined and could derive significant benefits from fabrication as an integrated circuit. However, the design cycle for a custom circuit is long and the costs can be very high. This makes the risk significant enough to deter most users. The technology to support maximum flexibility combined with fast turnaround on custom logic has simply not been available. Monolithic Memories offers the programmable solution.

The PAL device family offers a fresh approach to using fuse programmable logic. PAL circuits are a conceptually unified group of devices which combine programmable flexibility with high speed and an extensive selection of interface options. PAL devices can lower inventory, cut design cycles and provide high complexity with maximum flexibility. These features, combined with lower package count and high reliability, truly make the PAL circuit a designer's best friend.

* PAL®, HAL® and SKINNYDIP® are registered trademarks of Monolithic Memories.

The PAL Circuit — Teaching Old PROMs New Tricks

Monolithic Memories developed the modern PROM and introduced many of the architectures and techniques now regarded as industry standards. As a major PROM manufacturer, Monolithic Memories has the proven technology and high volume production capability required to manufacture and support the PAL device.

The PAL circuit is an extension of the fusible link technology pioneered by Monolithic Memories for use in bipolar PROMs. The fusible link PROM first gave the digital systems designer the power to "write on silicon." In a few seconds he was able to transform a blank PROM from a general purpose device into one containing a custom algorithm, microprogram, or Boolean transfer function. This opened up new horizons for the use of PROMs in computer control stores, character generators, data storage tables and many other applications. The wide acceptance of this technology is clearly demonstrated by today's multi-million dollar PROM market.

The key to the PROM's success is that it allows the designer to quickly and easily customize the chip to fit his unique requirements. The PAL circuit extends this programmable flexibility by utilizing proven link technology to implement logic functions. Using PAL circuits the designer can quickly and effectively implement custom logic varying in complexity from random gates to complex arithmetic functions.

ANDs and ORs

The PAL device implements the familiar sum of products logic by using a programmable AND array whose output terms feed a fixed OR array. Since the sum of products form can express any Boolean transfer function, the PAL circuit uses are only limited by the number of terms available in the AND - OR arrays. PAL devices come in different sizes to allow for effective logic optimization.

Figure 1 shows the basic PAL circuit structure for a two-input, one-output logic segment. The general logic equation for this segment is:

$$\text{Output} = (I_1+\overline{f_1})(\overline{I_1}+\overline{f_2})(I_2+\overline{f_3})(\overline{I_2}+\overline{f_4}) +$$
$$(I_1+\overline{f_5})(\overline{I_1}+\overline{f_6})(I_2+\overline{f_7})\ (\overline{I_2}+\overline{f_8})$$

where the "f" terms represent the state of the fusible links in the PAL AND array. An unblown link represents a logic 1. Thus,

fuse blown, f = 0

fuse intact, f = 1

An unprogrammed PAL circuit has all fuses intact.

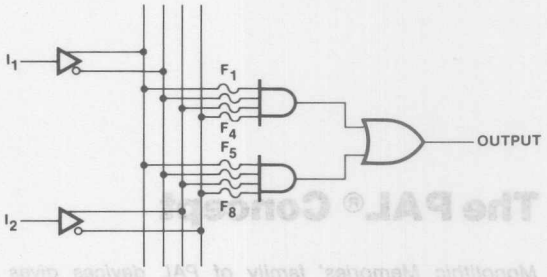


Figure 1

PAL Circuit Notation

Logic equations, while convenient for small functions, rapidly become cumbersome in large systems. To reduce possible confusion, complex logic networks are generally defined by logic diagrams and truth tables. Figure 2 shows the logic convention adopted to keep PAL logic easy to understand and use. In the figure, an "x" represents an intact fuse used to perform the logic AND function. (Note: the input terms on the common line with the x's are not connected together.) The logic symbology shown in Figure 2 has been informally adopted by integrated circuit manufacturers because it clearly establishes a one-to-one correspondence between the chip layout and the logic diagram. It also allows the logic diagram and truth table to be combined into a compact and easy to read form, thereby serving as a convenient shorthand for PAL circuits. The two input - one output example from Figure 1 redrawn using the new logic convention is shown in Figure 3.

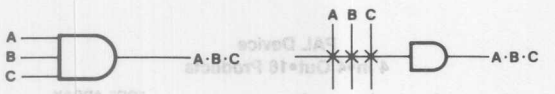


Figure 2

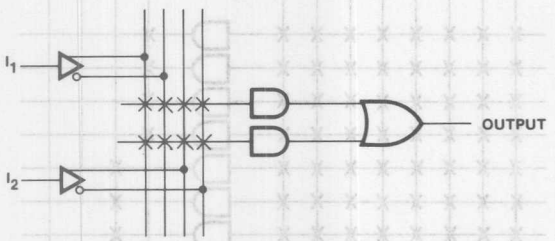


Figure 3

As a simple PAL logic example, consider the implementation of the transfer function:

$$\text{Output} = I_1\overline{I_2} + \overline{I_1}I_2$$

The normal combinatorial logic diagram for this function is shown in figure 4, with the PAL logic equivalent shown in figure 5.

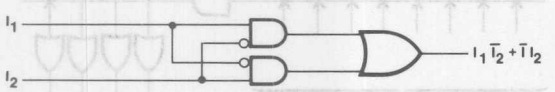


Figure 4

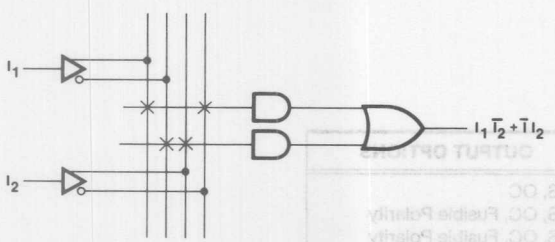


Figure 5

Using this logic convention it is now possible to compare the PAL structure to the structure of the more familiar PROM and PLA. The basic logic structure of a PROM consists of a fixed AND array whose outputs feed a programmable OR array (figure 6). PROMs are low-cost, easy to program, and available in a variety of sizes and organizations. They are most commonly used to store computer programs and data. In these applications the fixed input is a computer memory address; the output is the contents of that memory location.

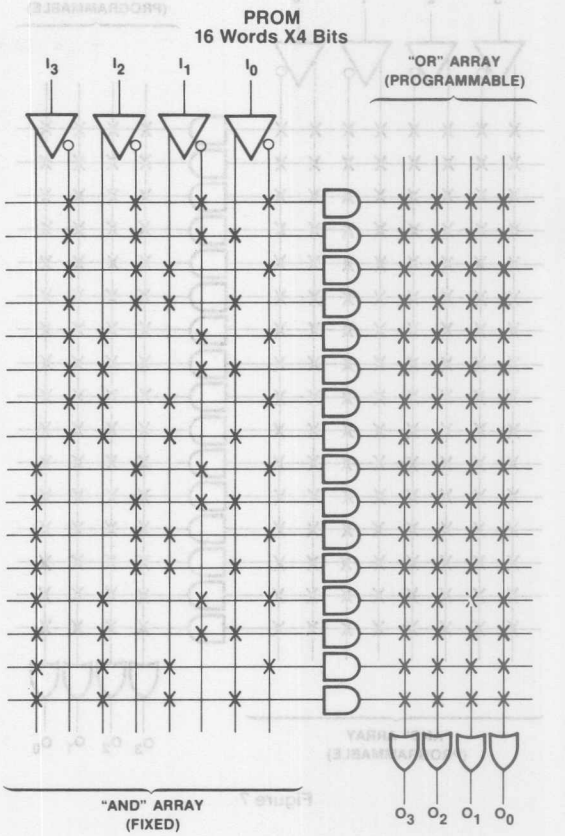


Figure 6

The basic logic structure of the PLA consists of a programmable AND array whose outputs feed a programmable OR array (Figure 7). Since the designer has complete control over all inputs and outputs, the PLA provides the ultimate flexibility for implementing logic functions. They are used in a wide variety of applications. However, this generality makes PLAs expensive, quite formidable to understand, and costly to program (they require special programmers).

The basic logic structure of the PAL circuit, as mentioned earlier, consists of a programmable AND array whose outputs feed a fixed OR array (Figure 8). The PAL circuit combines much of the flexibility of the PLA with the low cost and easy programmability of the PROM. Table 1 summarizes the characteristics of the PROM, PLA and PAL logic families.

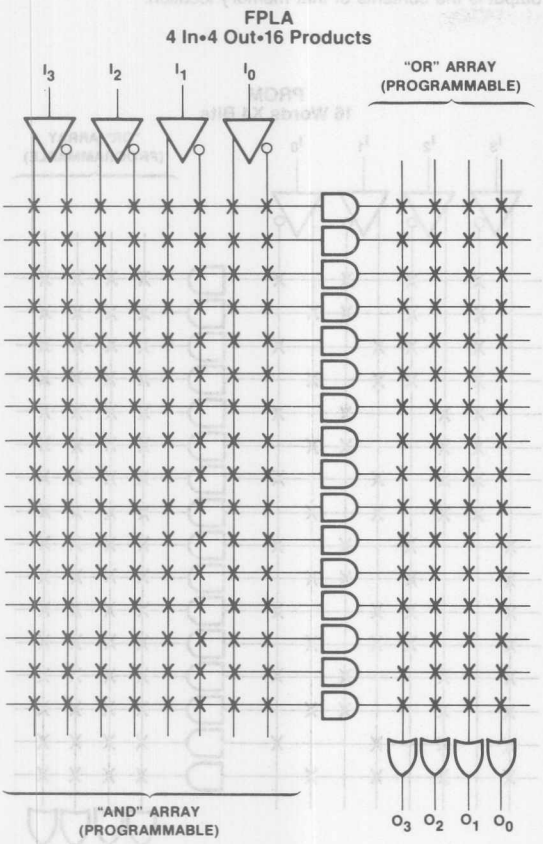


Figure 7

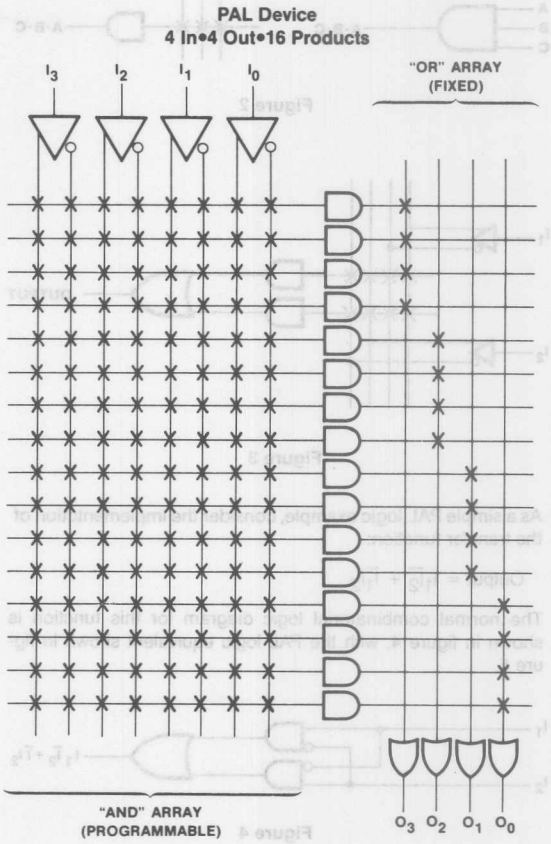


Figure 8

	AND	OR	OUTPUT OPTIONS
PROM	Fixed	Prog	TS, OC
FPLA	Prog	Prog	TS, OC, Fusible Polarity
FPGA	Prog	None	TS, OC, Fusible Polarity
FPLS	Prog	Prog	TS, Registered Feedback, I/O
PAL Circuit	Prog	Fixed	TS, Registered Feedback, I/O

Table 1

PAL Circuit Input/Output/Function/Performance Chart

PART NO.	INPUT	OUTPUT	PROG. I/O'S	OUTPUT POLARITY	FEEDBACK REGISTER	FUNCTIONS	PERFORMANCE				
							STD	A	-2	A-2	A-4
10H8	10	8			AND-OR	AND-OR Gate Array	X		X		
12H6	12	6			AND-OR	AND-OR Gate Array	X		X		
14H4	14	4			AND-OR	AND-OR Gate Array	X		X		
16H2	16	2			AND-OR	AND-OR Gate Array	X		X		
16C1	16	2			BOTH[1]	AND-OR/NOR Gate Array	X		X		
10L8	10	8			AND-NOR	AND-OR Invert Gate Array	X		X		
12L6	12	6			AND-NOR	AND-OR Invert Gate Array	X		X		
14L4	14	4			AND-NOR	AND-OR Invert Gate Array	X		X		
16L2	16	2			AND-NOR	AND-OR Invert Gate Array	X		X		
12L10	12	10			AND-NOR	AND-OR Invert Gate Array	X				
14L8	14	8			AND-NOR	AND-OR Invert Gate Array	X				
16L6	16	6			AND-NOR	AND-OR Invert Gate Array	X				
18L4	18	4			AND-NOR	AND-OR Invert Gate Array	X				
20L2	20	2			AND-NOR	AND-OR Invert Gate Array	X				
20C1	20	2			BOTH[1]	AND-OR/NOR Gate Array	X				
16L8	10	2	6		AND-NOR	AND-OR Invert Gate Array		X		X	X
20L8	14	2	6		AND-NOR	AND-OR Invert Gate Array		X			
20L10	12	2	8		AND-NOR	AND-OR Invert Gate Array	X				
16R8	8	8		8	AND-NOR	AND-OR Invert Gate Array w/Regs		X		X	X
16R6	8	6	2	6	AND-NOR	AND-OR Invert Gate Array w/Regs		X		X	X
16R4	8	4	4	4	AND-NOR	AND-OR Invert Gate Array w/Regs		X		X	X
20R8	12	8		8	AND-NOR	AND-OR Invert w/Regs		X			
20R6	12	6	2	6	AND-NOR	AND-OR Invert w/Regs		X			
20R4	12	4	4	4	AND-NOR	AND-OR Invert w/Regs		X			
20X10	10	10		10	AND-NOR	AND-OR-XOR Invert w/Regs	X				
20X8	10	8	2	8	AND-NOR	AND-OR-XOR Invert w/Regs	X				
20X4	10	4	6	4	AND-NOR	AND-OR-XOR Invert w/Regs	X				
16X4	8	4	4	4	AND-NOR	AND-OR-XOR Invert w/Regs	X				
16A4	8	4	4	4	AND-NOR	AND-CARRY-OR-XOR Invert w/Regs	X				
16P8	10	2	6		PROG[2]	AND-OR Gate Array		X			
16RP8	8	8		8	PROG[2]	AND-OR Gate Array w/Regs		X			
16RP6	8	6	2	6	PROG[2]	AND-OR Gate Array w/Regs		X			
16RP4	8	4	4	4	PROG[2]	AND-OR Gate Array w/Regs		X			
20RA10	10		10[3]	10[3]	PROG[2]	Asynchronous Gate Array		X			
20RS10	10			10	PROG[2]	AND-OR Gate Array w/Regs		X			
20RS8	10		2	8	PROG[2]	AND-OR Gate Array w/Regs		X			
20RS4	10		6	4	PROG[2]	AND-OR Gate Array w/Regs		X			
20S10	10		10		PROG[2]	AND-OR Gate Array		X			
32R16	16	16[3]		16[3]	PROG[2]	AND-OR Gate Array w/Regs		X			
64R32	32	32[3]		32[3]	PROG[2]	AND-OR Gate Array w/Regs		X			

Table 2

[1] Simultaneous AND-OR and AND-NOR outputs

[2] Programmable active high or active low. i.e. AND-OR or AND-NOR

[3] Output can be registered or non-registered

PAL Circuits for Every Task

The members of the PAL family and their characteristics are summarized in Table 2. They are designed to cover the spectrum of logic functions at reduced cost and lower package count. This allows the designer to select the PAL circuit that best fits his application. PAL units come in the following basic configurations:

Logic Arrays

PAL logic arrays are available in sizes from 12x10 (12 input terms, 10 output terms) to 20x2, with both active high and active low output configurations available (Figure 9). This wide variety of input/output formats allows the PAL device to replace many different sized blocks of combinatorial logic with single packages.

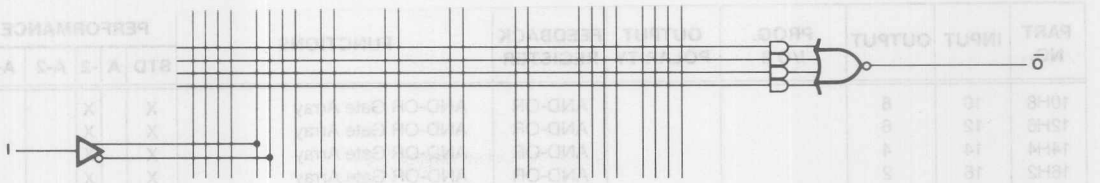


Figure 9

Programmable I/O

A feature of the high-end members of the PAL family is programmable input/output. This allows the produt terms to directly control the outputs of the PAL circuit (Figure 10). One product term is used to enable the three-state buffer, which in turn gates the summation term to the output pin. The output is also fed back into the PAL array as an input. Thus the PAL circuit drives the I/O pin when the three-state gate is enabled; the I/O pin is an input to the PAL array when the three-state gate is disabled. This feature can be used to allocate available pins for I/O functions or to provide bidirectional output pins for operations such as shifting and rotating serial data.

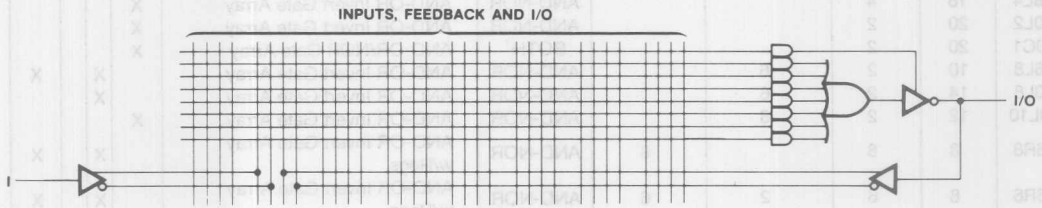


Figure 10

Registered Outputs with Feedback

Another feature of the high end members of the PAL family is registered data outputs with registered feedback. Each product term is stored into a D-type output flip-flop on the rising edge of the system clock (Figure 11). The Q output of the flip-flop can then be gated to the output pin by enabling the active low three-state buffer.

In addition to being available for transmission, the Q output is fed back into the PAL array as an input term. This feedback allows the PAL circuit to "remember" the previous state, and it can alter its function based upon that state. This allows the designer to configure the PAL circuit as a state sequencer which can be programmed to execute such elementary functions as count up, count down, skip, shift, and branch. These functions can be executed by the registered PAL device at rates of up to 25 MHz.

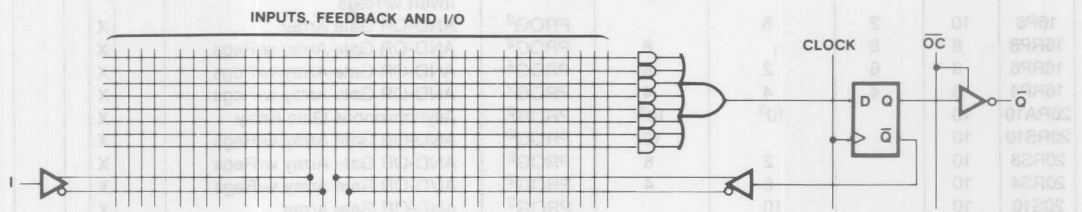


Figure 11

XOR PAL Circuits

These PAL devices feature an exclusive OR function. The sum of products is segmented into two sums which are then exclusive ORed (XOR) at the input of the D-type flip-flop (Figure 12). All of the features of the Registered PAL circuits are included in the XOR PAL unit. The XOR function provides an easy implementation of the HOLD operation used in counters and other state sequencers.

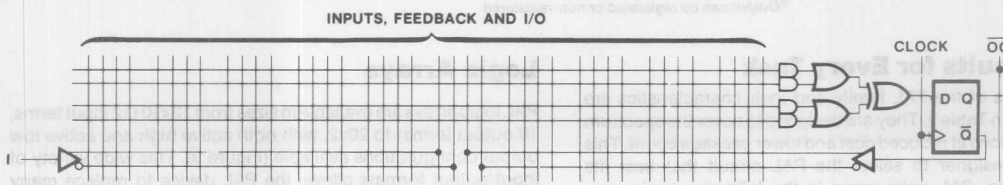


Figure 12

Programmable Output Polarity

The outputs can be programmed either active-low or active-high. This is represented by the exclusive-or gates shown in Figure 13, PAL20RA10 Logic Diagram. When the output polarity fuse is blown, the lower input to the exclusive-or gate is high, so the output is active-high. Similarly, when the output polarity fuse is intact, the output is active-low. The programmable output polarity feature allows the user a higher degree of flexibility when writing equations.

Programmable Clock

One of the product lines in each group is connected to the clock. This provides the user with the additional flexibility of a programmable clock, so each output can be clocked independently of all the others. (See Figure 13.)

Programmable Set and Reset

Two product lines are dedicated to asynchronous set and reset. If the set product line is high, the register output becomes a logic 1. If the reset product line is high, the register output becomes a logic 0. The operation of the programmable set and reset overrides the clock. (See Figure 13.)

Individually Programmable Register Bypass

If both the set and reset product lines are high, the sum-of-products bypasses the register and appears immediately at the output, thus making the output combinatorial. This allows each output to be configured in the registered or combinatorial mode. (See Figure 13.)

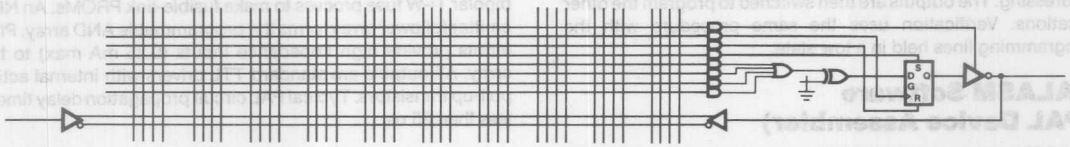


Figure 13

Product Term Sharing

The basic configuration is sixteen product terms shared between two output cells. For a typical output pair, each product term can be used by either output; but, since product term sharing is exclusive, a product term can be used by only one output, not both. If equations call for an output pair to use the same product term, two product terms are generated, one for each output. This should be taken into account when writing equations. PAL assemblers configure product terms automatically.

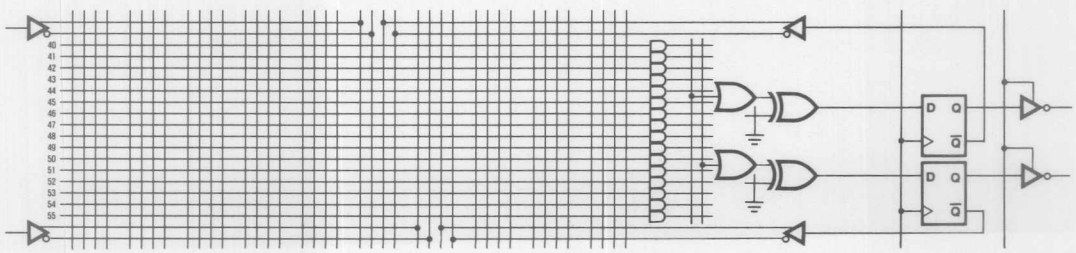


Figure 14

Advanced PAL Circuit Features

For 1985, a number of new features have been incorporated into the PAL family, including:

- Programmable output polarity for active high or active low operation
- Register preload which allows complete functional testing
- Product term sharing*, a feature making the number of product terms per output user-determinable
- Register bypass facilitating registered or combinatorial outputs
- Asynchronous clocks, sets, resets and output enables

A full description of each function is given on page 5-17.

PAL Device Programming

PAL devices can be programmed in most standard PROM programmers with the addition of a PAL personality card. The PAL circuit appears to the programmer as a PROM. During programming half of the PAL device outputs are selected for programming while the other outputs and the inputs are used for addressing. The outputs are then switched to program the other locations. Verification uses the same procedure with the programming lines held in a low state.

PALASM Software (PAL Device Assembler)

PALASM software is used to define, simulate, build and test PAL device units. PALASM software accepts the PAL circuit Design Specification as an input file. It verifies the design against an optional function table and generates the fuse plot which is used to program the PAL devices. Presently, PALASM software is being replaced by its successor: PALASM2 software. PALASM2 software has added features that simplify the task of defining and simulating PAL Design Specifications.

HAL® Device (Hard Array Logic)

The HAL family is the mask-programmed version of a PAL circuit. The HAL circuit is to a PAL circuit just as ROM is to a PROM. A standard wafer is fabricated to the 6 mask. Then a custom metal mask is used to fabricate aluminum links for a HAL circuit instead of the programmable Ti-W fuse array used in a PAL circuit.

The HAL device is a cost-effective solution for large quantities and is unique in that it is a gate array with a programmable prototype.

ZHAL Device (Zero Power Hard Array Logic)

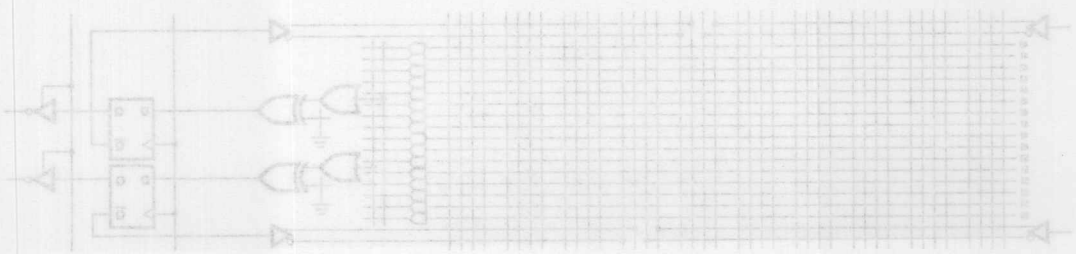
ZHAL devices are functionally identical to regular HAL devices but with the added feature of consuming zero standby power. This is highly desirable in portable digital equipment and lap-top computers.

PAL Circuit Technology

PAL circuits are manufactured using the proven TTL Schottky bipolar Ti-W fuse process to make fusible-link PROMs. An NPN emitter follower array forms the programmable AND array. PNP inputs provide high impedance inputs (0.25 mA max) to the array. All outputs are standard TTL drivers with internal active pull-up transistors. Typical PAL circuit propagation delay time is less than 25 ns.

PAL Device Data Security

The circuitry used for programming and logic verification can be used at any time to determine the logic pattern stored in the PAL array. For security, the PAL array has a "last fuse" which can be blown to disable the verification logic. This provides a significant deterrent to potential copiers, and it can be used to effectively protect proprietary designs.



* Patent pending

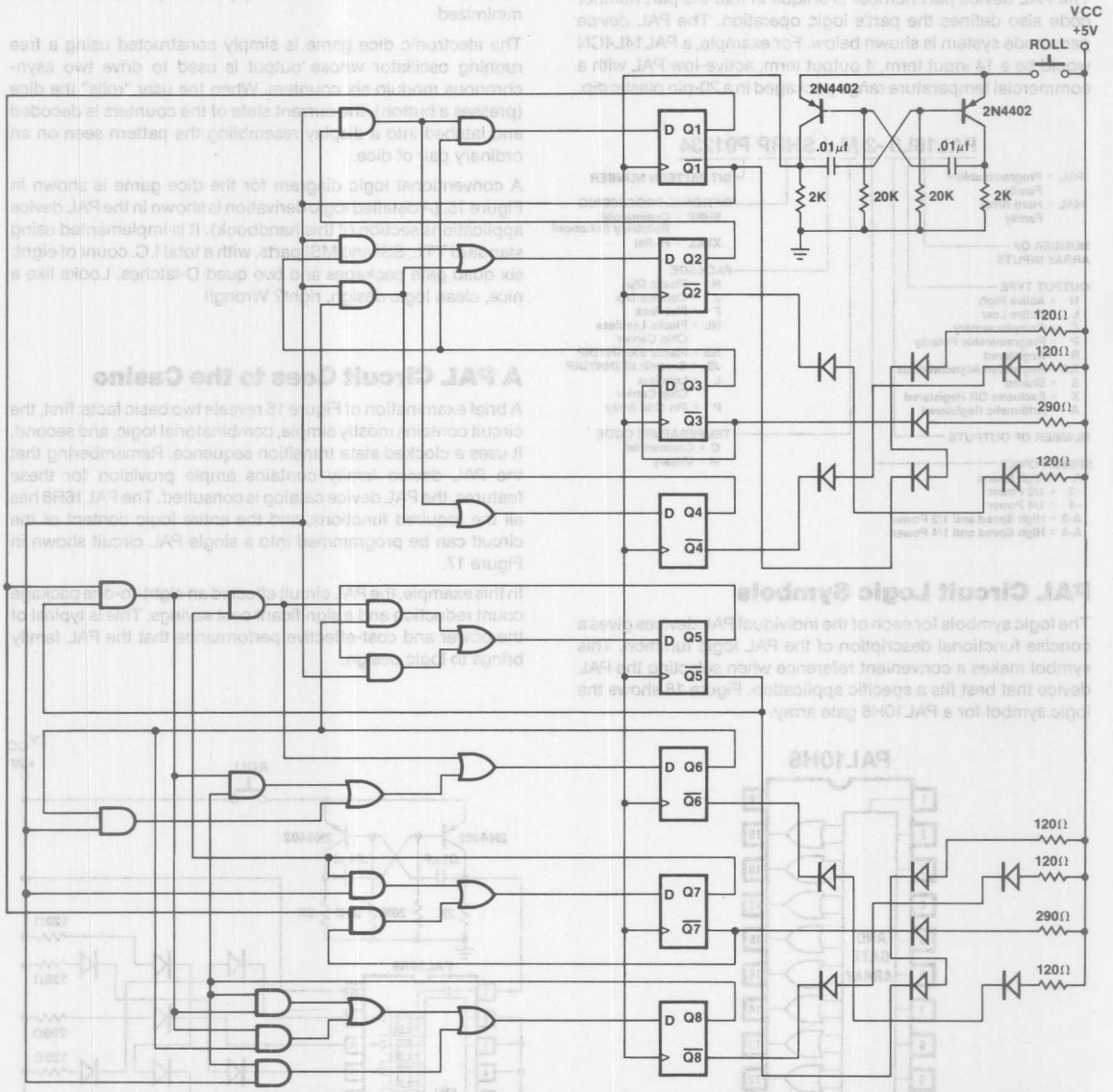
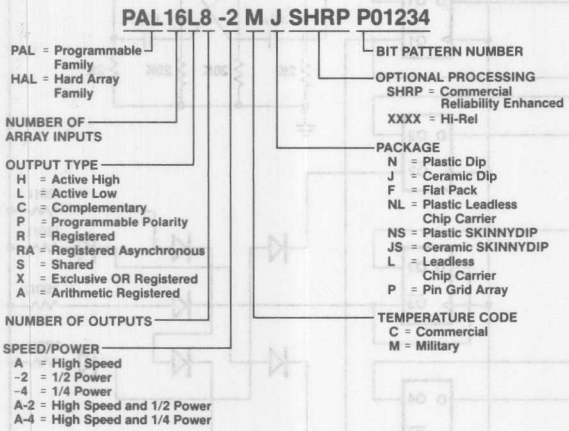

VCC
+5V
ROLL
2N4402
2N4402
.01μf
.01μf
2K
20K
20K
2K
D Q1
Q1
D Q2
Q2
D Q3
Q3
D Q4
Q4
D Q5
Q5
D Q6
Q6
D Q7
Q7
D Q8
Q8
120Ω
120Ω
290Ω
120Ω
120Ω
120Ω
290Ω
120Ω

Figure 15

The PAL device part number is unique in that the part number code also defines the part's logic operation. The PAL device parts code system is shown below. For example, a PAL14L4CN would be a 14 input term, 4 output term, active-low PAL with a commercial temperature range packaged in a 20-pin plastic dip.



PAL Circuit Logic Symbols

The logic symbols for each of the individual PAL devices gives a concise functional description of the PAL logic function. This symbol makes a convenient reference when selecting the PAL device that best fits a specific application. Figure 18 shows the logic symbol for a PAL10H8 gate array.

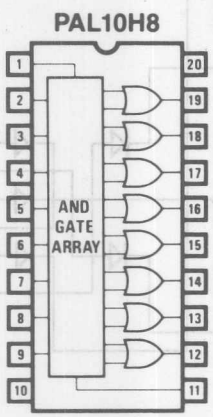


Figure 16

A PAL Circuit Example

As an example of how the PAL device enables the designer to reduce costs and simplify logic design, consider the design of a simple, high-volume consumer product: an electronic dice game. This type of product will be produced in extremely high volume, so it is essential that every possible production cost be minimized

The electronic dice game is simply constructed using a free running oscillator whose output is used to drive two asynchronous modulo six counters. When the user "rolls" the dice (presses a button), the current state of the counters is decoded and latched into a display resembling the pattern seen on an ordinary pair of dice.

A conventional logic diagram for the dice game is shown in Figure 15. (A detailed logic derivation is shown in the PAL device applications section of this handbook). It is implemented using standard TTL, SSI and MSI parts, with a total I.C. count of eight: six quad gate packages and two quad D-latches. Looks like a nice, clean logic design, right? Wrong!!

A PAL Circuit Goes to the Casino

A brief examination of Figure 16 reveals two basic facts: first, the circuit contains mostly simple, combinatorial logic, and second, it uses a clocked state transition sequence. Remembering that the PAL device family contains ample provision for these features, the PAL device catalog is consulted. The PAL16R8 has all the required functions, and the entire logic content of the circuit can be programmed into a single PAL circuit shown in Figure 17.

In this example, the PAL circuit effected an eight-to-one package count reduction and a significant cost savings. This is typical of the power and cost-effective performance that the PAL family brings to logic design.

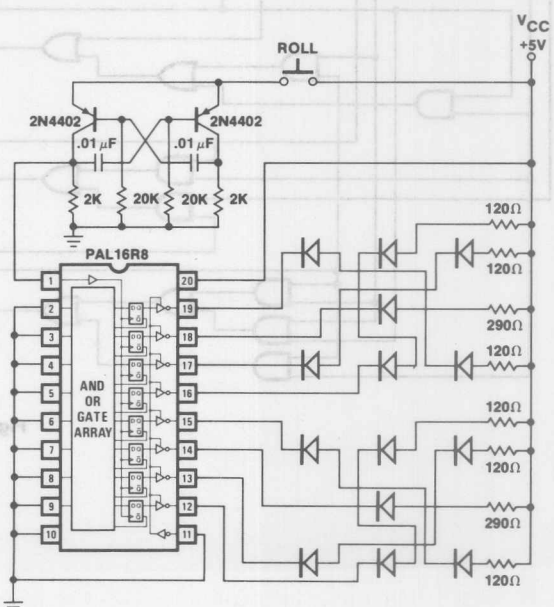
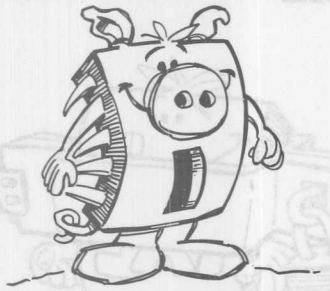


Figure 17

Advantages of Using PAL Circuits



The PAL device has a unique place in the world of logic design. Not only does it offer many advantages over conventional logic, it also provides many features not found anywhere else. The PAL family:

- Programmable replacement for conventional TTL logic.
- Reduces IC inventories substantially and simplifies their control.
- Reduces chip count by at least 4 to 1.
- Expedites and simplifies prototyping and board layout.
- Saves space with 20-pin and 24-pin SKINNYDIP® packages.
- High speed: 15ns typical propagation delay.
- Programmed on standard PROM programmers.
- Programmable three-state outputs.
- Special feature eliminates possibility of copying by competitors.

All of these features combine together to lower product development costs and increase product cost effectiveness. The bottom line is that PAL units save money.

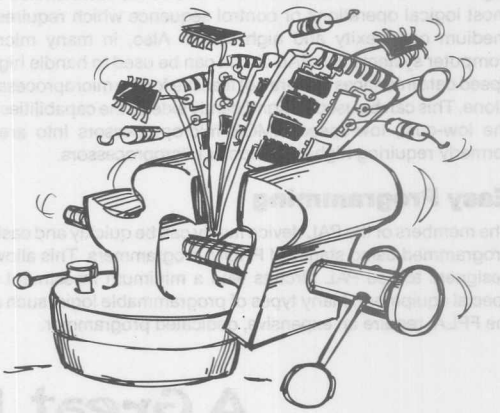
Direct Logic Replacement

In both new and existing designs the PAL circuit can be used to replace various logic functions. This allows the designer to optimize a circuit in many ways never before possible. The PAL circuit is particularly effective when used to provide interfaces required by many LSI functions. PAL circuit flexibility combined with LSI function density makes a powerful team.

Design Flexibility

The PAL circuit offers the systems logic designer a whole new world of options. Until now, the decision on logic system implementation was usually between SSI/MSI logic functions on one hand and microprocessors on the other. In many cases the function required is too awkward to implement the first way and too simple to justify the second. Now the PAL circuit offers the designer high functional density, high speed, and low cost. Even better, PAL devices come in a varity of sizes and functions, thereby further increasing the designer's options.

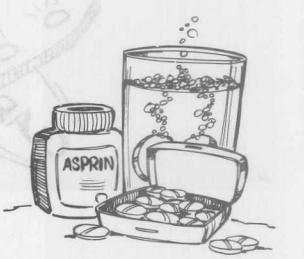
Space Efficiency



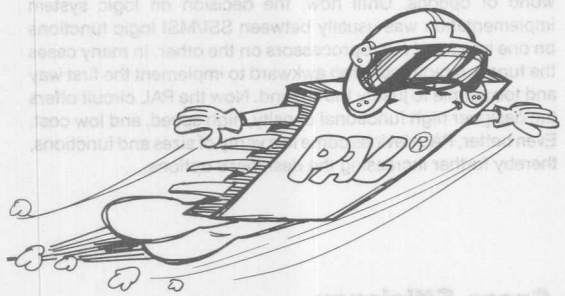
By allowing designers to replace many simple logic functions with single packages, the PAL device allows more compact P.C. board layouts. The PAL space-saving 20-pin and 24-pin "SKINNYDIP" package helps to further reduce board area while simplifying board layout and fabrication. This means that many multi-card systems can now be reduced to one or two cards, and that can make the difference between a profitable success or an expensive disaster.

Smaller Inventory

The PAL device family can be used to replace up to 90% of the conventional TTL family. This considerably lowers both shelving and inventory cataloging requirements. Even better, small custom modifications to the standard functions are easy for PAL device users, not so easy for standard TTL users.



High Speed

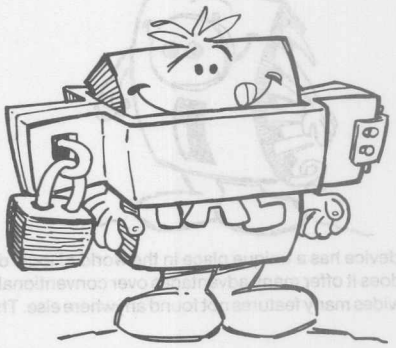


The PAL device family runs faster or equal to the best of bipolar logic circuits. This makes the PAL circuit the ideal choice for most logical operations or control sequence which requires a medium complexity and high speed. Also, in many microcomputer systems, the PAL circuit can be used to handle high-speed data interfaces that are not feasible for the microprocessor alone. This can be used to significantly extend the capabilities of the low-cost, low-speed NMOS microprocessors into areas formerly requiring high-cost bipolar microprocessors.

Easy Programming

The members of the PAL device family can be quickly and easily programmed using standard PROM programmers. This allows designers to use PAL circuits with a minimum investment in special equipment. Many types of programmable logic, such as the FPLA, require an expensive, dedicated programmer.

Secure Data

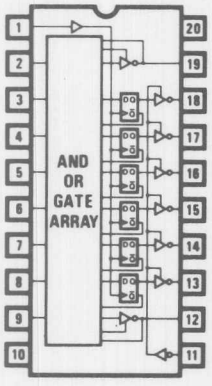


The PAL device verification logic can be completely disabled by blowing out a special "last link." This prevents the unauthorized copying of valuable data, and makes the PAL circuit perfect for use in any application where data integrity must be carefully guarded.

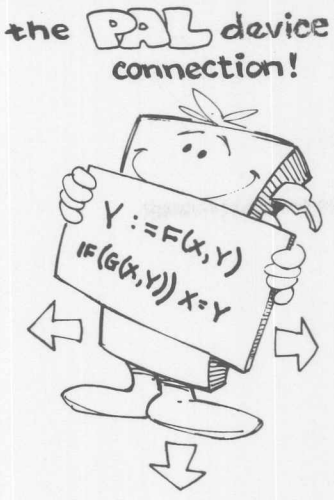
Summary

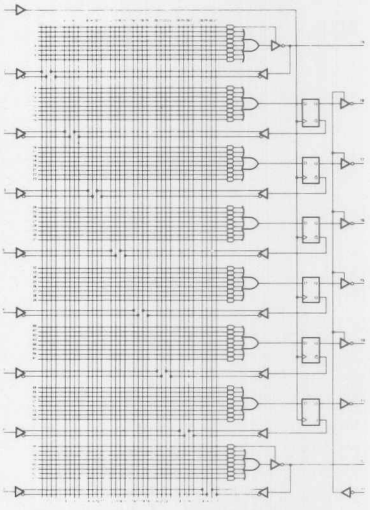
The PAL device family of logic devices offers designers new options in the implementation of sequential and combinatorial logic designs. The family is fast, compact, flexible, and easy to use in both new and existing designs. It promises to reduce costs in most areas of design and production with a corresponding increase in product profitability.

A Great Performer!

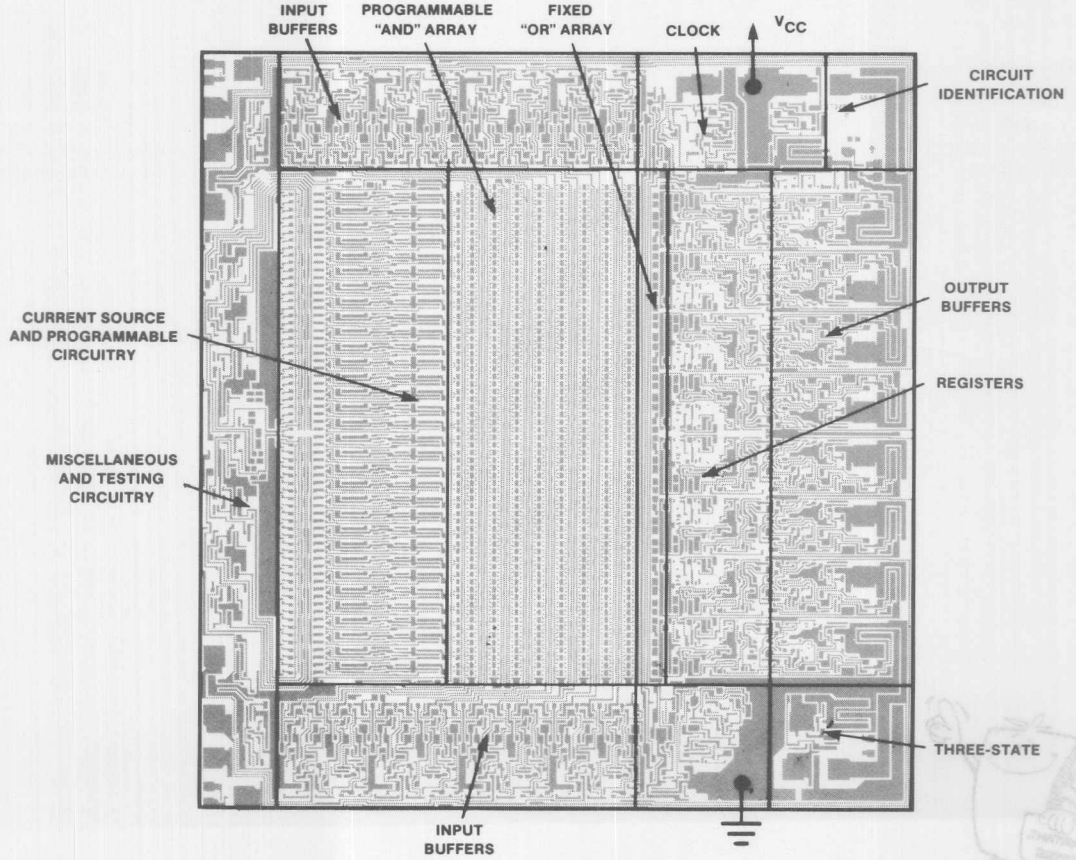


PAL16R6 Logic Symbols



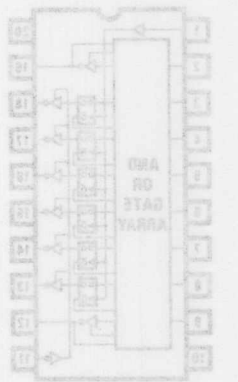


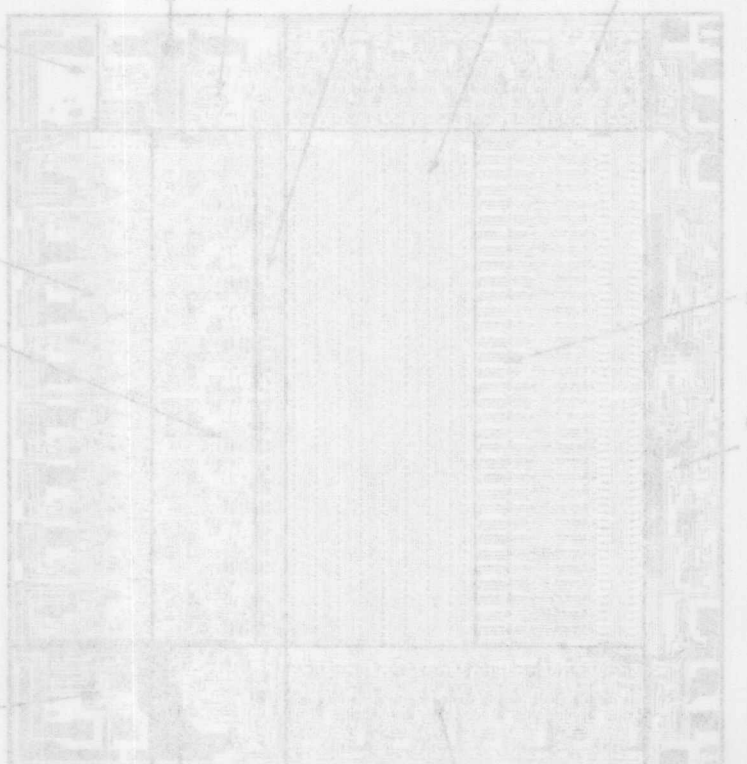
PAL16R6 Logic Diagram



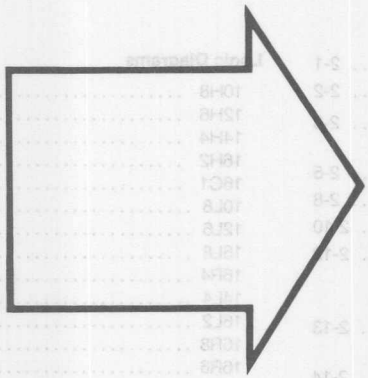
PAL 16R6 Metalization


IMPORTANT NOTES






IMPORTANT NOTES



Contents Section 2

PAL® Device - Programmable Array Logic
HAL® Device - Hard Array Logic

Features/Benefits

- **Reduces SSI/MSI chip count greater than 5 to 1**
- **Saves space with SKINNYDIP® packages**
- **Reduces IC inventories substantially**
- **Expedites and simplifies prototyping and board layout**
- **PALASM™ silicon compiler provides auto routing and test vectors**
- **Security fuse reduces possibility of copying by competitors**

Description

The PAL device family utilizes an advanced Schottky TTL process and the Bipolar PROM fusible link technology to provide user programmable logic for replacing conventional SSI/MSI gates and flip-flops at reduced chip count.

The HAL device family utilizes standard Low-Power Schottky TTL process and automated mask pattern generation directly from logic equations to provide a semicustom gate array for replacing conventional SSI/MSI gates and flip-flops at reduced chip count.

There are four different speed/power families offered. Choose from either the standard, high-speed, half-power, or quarter-power family to maximize design performance.

The PAL/HAL device family lets the systems engineer "design his own chip" by blowing fusible links to configure AND and OR gates to perform his desired logic function. Complex interconnections which previously required time-consuming layout are thus "lifted" from PC board etch and placed on silicon where they can be easily modified during prototype check-out or production.

The PAL device transfer function is the familiar sum of products. Like the PROM, the PAL device has a single array of fusible links. Unlike the PROM, the PAL device is a programmable AND array driving a fixed OR array (the PROM is a fixed AND array driving a programmable OR array).

The PAL device transfer function is the familiar sum of products. Like the PROM, the PAL device has a single array of fusible links. Unlike the PROM, the PAL device is a programmable AND array driving a fixed OR array (the PROM is a fixed AND array driving a programmable OR array).

In addition the PAL/HAL provides these options:

- **Variable input/output pin ratio**
- **Programmable three-state outputs**
- **Registers with feedback**
- **Arithmetic capability**
- **Exclusive-OR gates**
- **Other options identified on page 5-17**

Unused inputs are tied directly to V_{CC} or GND. Product terms with all fuses blown assume the logical high state, and product terms connected to both true and complement of any single input assume the logical low state. Registers consist of D-type flip-flops which are loaded on the low-to-high transition of the clock. PAL/HAL Circuit Logic Diagrams are shown with all fuses blown, enabling the designer to use the diagrams as coding sheets.

The entire PAL device family is programmed using inexpensive conventional PROM programmers with appropriate personality and socket adapter cards. Once the PAL device is programmed and verified, two additional fuses may be blown to defeat verification. This feature gives the user a proprietary circuit which is very difficult to copy.

To design a HAL device, the user first programs and debugs a PAL circuit using PALASM software and the "PAL DESIGN SPECIFICATION" standard format. This specification is submitted to Monolithic Memories where it is computer processed and assigned a bit pattern number, e.g., P01234.

Monolithic Memories will provide a PAL device sample for customer qualification. The user then submits a purchase order for a HAL device of the specified bit pattern number, e.g., HAL18L4 P01234. See Ordering Information below.

Ordering Information

PAL16L8 -2 M J SHRP P01234

- PAL = Programmable Family
- HAL = Hard Array Family
- NUMBER OF ARRAY INPUTS
- OUTPUT TYPE
  - H = Active High
  - L = Active Low
  - C = Complementary
  - P = Programmable
  - R = Registered
  - RA = Registered Asynchronous
  - S = Shared
  - X = Exclusive OR Registered
  - A = Arithmetic Registered
- NUMBER OF OUTPUTS
- SPEED/POWER
  - A = High Speed
  - -2 = 1/2 Power
  - -4 = 1/4 Power
  - A-2 = High Speed and 1/2 Power
  - A-4 = High Speed and 1/4 Power
- TEMPERATURE CODE
  - C = Commercial
  - M = Military
- PACKAGE
  - N = Plastic Dip
  - J = Ceramic Dip
  - F = Flat Pack
  - NL = Plastic Leadless Chip Carrier
  - NS = Plastic SKINNYDIP
  - JS = Ceramic SKINNYDIP
  - L = Leadless Chip Carrier
  - P = Pin Grid Array
- OPTIONAL PROCESSING
  - SHRP = Commercial Reliability Enhanced
  - XXXX = Hi-Rel
- BIT PATTERN NUMBER

PAL®, HAL® and SKINNYDIP® are registered trademarks of Monolithic Memories.
PMSI™ and HMSI™ are trademarks of Monolithic Memories.

Register Bypass

Outputs within a bank must either be all registered or all combinatorial. Whether or not a bank of registers is bypassed depends on how the outputs are defined in the equations. A colon followed by an equal sign [:=] specifies a registered output with feedback which is updated after the low-to-high transition of the clock. An equal sign [=] defines a combinatorial output which bypasses the register. Registers are bypassed in banks of eight. Bypassing a bank of registers eliminates the feedback lines for those outputs.

Output Polarity

Output polarity is defined by comparison of the pin list and the equations. If the logic sense of a specific output in the pin list is different from the logic sense of that output as defined by its equation, the output is inverted or active low polarity. If the logic sense of a specific output in the pin list is the same as the logic sense of that output as defined by its equation, the output is active high polarity.

Product Term Sharing

The basic configuration is sixteen product terms shared between two output cells. For a typical output pair, each product term can be used by either output; but, since product term sharing is exclusive, a product term can be used by only one output, not both. If equations call for an output pair to use the same product term, two product terms are generated, one for each output. This should be taken into account when writing equations. PAL circuit assemblers configure product terms automatically.

This example uses the 84-pin package. Four output equations are shown to demonstrate functionality. Pin names are arbitrary.

Product Term Editing

A unique feature of product term sharing is the ability to edit the design after the device has been programmed. Without this feature, a new PAL device had to be programmed if the user needed to change his design. Product term editing allows the user to delete an unwanted product term and reprogram a previously unused product term to the desired fuse pattern. This feature is made possible by the product term sharing architecture. Since each product term can be routed to either output in a given pair by selecting one of two steering fuses, it is possible to blow both of the steering fuses thereby completely disabling that product term. Once disabled, that product term is powered down, saving typically 0.25 mA. The desired change may now be programmed into one of the previously unused product terms corresponding to that output pair. Additional edits can be made as long as there are unused product terms for the output in question.

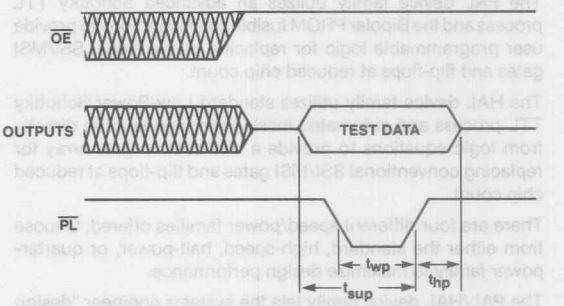
PRESET Feature (PAL64R32 device only)

Register banks of eight may be PRESET to all highs on the outputs by setting the PRESET pin (PS) to a Low level. Note from the Logic Diagram that when the state of an output is High, the state of the register is Low due to the inverting tri-state buffer.

PAL Device Testability Features

Preload pins have been added to enable the testability of each state in state-machine design. Typically, for a modulo-n counter or a state machine there are many unreachable states for the registers. These states, and the logic which controls them are untestable without a way to "set-in" the desired starting state of the registers. In addition, long test sequences are sometimes needed to test a state machine simply to reach those starting states which are legal. Since complete logic verification is needed to ensure the proper exit from "illegal" or unused states, a way to enter these states must be provided. The ability to preload a given bank of registers is provided in this device.

To use the preload feature, several steps must be followed. First, a high level on an assertive-low output enable pin disables the outputs for that bank of registers. Next, the data to be loaded is presented at the output pins. This data is then loaded into the register by placing a low level on the PRELOAD pin. PRELOAD is asynchronous with respect to the clock.



Programmable Set and Reset (PAL20RA10 only)

In each SMAC, two product lines are dedicated to asynchronous set and reset. If the set product line is high, the register output becomes a logic 1. If the reset product line is high, the register output becomes a logic 0 and the output pin a logic 1 due to output buffer inversion. The operation of the programmable set and reset overrides the clock.

Individually Programmable Register Bypass (PAL20RA10 only)

If both the set and reset product lines are high, the sum-of-products bypasses the register and appears immediately at the output, thus making the output combinatorial. This allows each output to be configured in the registered or combinatorial mode.

Programmable Clock (PAL20RA10 only)

One of the product lines in each group is connected to the clock. This provides the user with the additional flexibility of a programmable clock, so each output can be clocked independently of all the others.

GENERIC LOGIC	PINS	PACKAGE	DESCRIPTION	PART NUMBER			
				STANDARD	HIGH SPEED	1/2 POWER	1/4 POWER
10H8	20	N,J,F,L,NL	Octal 10 Input And-Or Gate Array	PAL10H8 HAL10H8		PAL10H8-2 HAL10H8-2	
12H6	20	N,J,F,L,NL	Hex 12 Input And-Or Gate Array	PAL12H6 HAL12H6		PAL12H6-2 HAL12H6-2	
14H4	20	N,J,F,L,NL	Quad 14 Input And-Or Gate Array	PAL14H4 HAL14H4		PAL14H4-2 HAL14H4-2	
16H2	20	N,J,F,L,NL	Dual 16 Input And-Or Gate Array	PAL16H2 HAL16H2		PAL16H2-2 HAL16H2-2	
16C1	20	N,J,F,L,NL	16 Input And-Or/Nor Gate Array	PAL16C1 HAL16C1		PAL16C1-2 HAL16C1-2	
10L8	20	N,J,F,L,NL	Octal 10 Input And-Or Invert Gate Array	PAL10L8 HAL10L8		PAL10L8-2 HAL10L8-2	
12L6	20	N,J,F,L,NL	Hex 12 Input And-Or-Invert Gate Array	PAL12L6 HAL12L6		PAL12L6-2 HAL12L6-2	
14L4	20	N,J,F,L,NL	Quad 14 Input And-Or-Invert Gate Array	PAL14L4 HAL14L4		PAL14L4-2 HAL14L4-2	
16L2	20	N,J,F,L,NL	Dual 16 Input And-Or-Invert Gate Array	PAL16L2 HAL16L2		PAL16L2-2 HAL16L2-2	
16L8	20	N,J,F,L,NL	Octal 16 Input And-Or-Invert Gate Array	PAL16L8 HAL16L8	PAL16L8A HAL16L8A	PAL16L8A-2 HAL16L8A-2	PAL16L8A-4 HAL16L8A-4
16R8	20	N,J,F,L,NL	Octal 16 Input Registered And-Or Invert Gate Array	PAL16R8 HAL16R8	PAL16R8A HAL16R8A	PAL16R8A-2 HAL16R8A-2	PAL16R8A-4 HAL16R8A-4
16R6	20	N,J,F,L,NL	Hex 16 Input Registered And-Or Invert Gate Array	PAL16R6 HAL16R6	PAL16R6A HAL16R6A	PAL16R6A-2 HAL16R6A-2	PAL16R6A-4 HAL16R6A-4
16R4	20	N,J,F,L,NL	Quad 16 Input Registered And-Or Invert Gate Array	PAL16R4 HAL16R4	PAL16R4A HAL16R4A	PAL16R4A-2 HAL16R4A-2	PAL16R4A-4 HAL16R4A-4
16X4	20	N,J,F,L,NL	Quad 16 Input Registered And-Or-Xor Invert Gate Array	PAL16X4 HAL16X4			
16A4	20	N,J,F,L,NL	Quad 16 Input Registered And-Carry-Or-Xor Invert Gate Array	PAL16A4 HAL16A4			
12L10	24 (28)	NS,JS,F,(L),(NL)	Deca 12 Input And-Or-Invert Gate Array	PAL12L10 HAL12L10			
14L8	24 (28)	NS,JS,F,(L),(NL)	Octal 14 Input And-Or-Invert Gate Array	PAL14L8 HAL14L8			
16L6	24 (28)	NS,JS,F,(L),(NL)	Hex 16 Input And-Or-Invert Gate Array	PAL16L6 HAL16L6			
18L4	24 (28)	NS,JS,F,(L),(NL)	Quad 18 Input And-Or-Invert Gate Array	PAL18L4 HAL18L4			
20L2	24 (28)	NS,JS,F,(L),(NL)	Dual 20 Input And-Or-Invert Gate Array	PAL20L2 HAL20L2			
20C1	24 (28)	NS,JS,F,(L),(NL)	20 Input And-Or/Nor Gate Array	PAL20C1 HAL20C1			
20L10	24 (28)	NS,JS,F,(L),(NL)	Deca 20 Input And-Or-Invert Gate Array	PAL20L10 HAL20L10			
20X10	24 (28)	NS,JS,F,(L),(NL)	Deca 20 Input Registered And-Or-Xor Invert Gate Array	PAL20X10 HAL20X10			
20X8	24 (28)	NS,JS,F,(L),(NL)	Octal 20 Input Registered And-Or-Xor Invert Gate Array	PAL20X8 HAL20X8			
20X4	24 (28)	NS,JS,F,(L),(NL)	Quad 20 Input Registered And-Or-Xor Invert Gate Array	PAL20X4 HAL20X4			
20L8	24 (28)	NS,JS,F,(L),(NL)	Octal 20 Input And-Or-Invert Gate Array		PAL20L8A HAL20L8A		
20R8	24 (28)	NS,JS,F,(L),(NL)	Octal 20 Input Registered And-Or Invert Gate Array		PAL20R8A HAL20R8A		
20R6	24 (28)	NS,JS,F,(L),(NL)	Hex 20 Input Registered And-Or Invert Gate Array		PAL20R6A HAL20R6A		
20R4	24 (28)	NS,JS,F,(L),(NL)	Quad 20 Input Registered And-Or Invert Gate Array		PAL20R4A HAL20R4A		

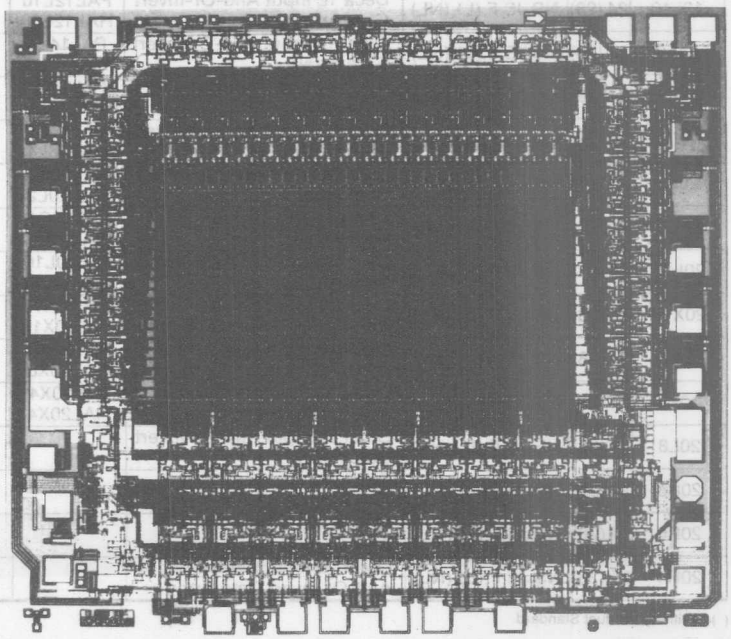
( ) = Military Product Standard.

PAL Device Input/Output/Function/Performance Chart

GENERIC LOGIC	PINS	PACKAGE	DESCRIPTION	PART NUMBER			
				STANDARD	HIGH SPEED	1/2 POWER	1/4 POWER
*16P8	20	N,J,L,NL	Octal 16 Input And-Or Array w/Programmable Polarity		PAL16P8A HAL16P8A		
*16RP8	20	N,J,L,NL	Octal 16 Input Registered And-Or Array w/Programmable Polarity		PAL16RP8A HAL16RP8A		
*16RP6	20	N,J,L,NL	Hex 16 Input Registered And-Or Array w/Programmable Polarity		PAL16RP6A HAL16RP6A		
*16RP4	20	N,J,L,NL	Quad 16 Input Registered And-Or Array w/Programmable Polarity		PAL16RP4A HAL16RP4A		
20S10	24 (28)	N,J,W,(L),(NL)	Deca 20 Input And-Or Array w/Product Term Sharing		PAL20S10 HAL20S10		
20RS10	24 (28)	N,J,W,(L),(NL)	Deca 20 Input Registered And-Or Array w/Product Term Sharing		PAL20RS10 HAL20RS10		
20RS8	24 (28)	N,J,W,(L),(NL)	Octal 20 Input Registered And-Or Array w/Product Term Sharing		PAL20RS8 HAL20RS8		
20RS4	24 (28)	N,J,W,(L),(NL)	Quad 20 Input Registered And-Or Array w/Product Term Sharing		PAL20RS4 HAL20RS4		
20RA10	24 (28)	N,J,W,(L),(NL)	Deca 20 Input Registered Asynchronous And-Or Array		PAL20RA10 HAL20RA10		
32R16	40 (44)	N,J,(L),(NL)	16 Output, 32 Input Registered And-Or Gate Array		PAL32R16 HAL32R16		
64R32	84 (88)	L,(P)	32 Output, 64 Input Registered And-Or Gate Array		PAL64R32 HAL64R32		

* Contact Factory for Flat Pack

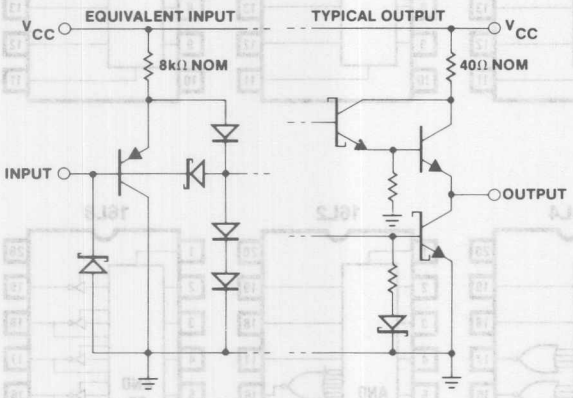
Die Configuration: PAL16L8



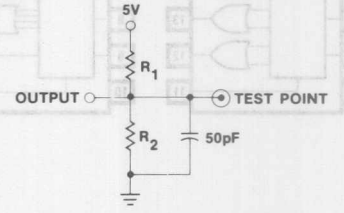
Absolute Maximum Ratings

	Operating	Programming
Supply Voltage, V_{CC}	-0.5V to 7.0V	-0.5V to 12.0V
Input Voltage	-1.5V to 5.5V	-1.0 to 22V
Off-state output Voltage	5.5V	12.0V
Storage temperature		-65° to +150°C

Schematic of Inputs and Outputs



Test Load



Other loads may be used.

Typical notes for all the following specifications (pages 7-22 — 7-30)

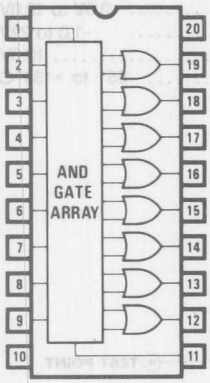
Notes: Apply to electrical and switching characteristics

† I/O pin leakage is the worst case of I_{OZX} or I_{IX} e.g., I_{IL} and I_{OZH}.

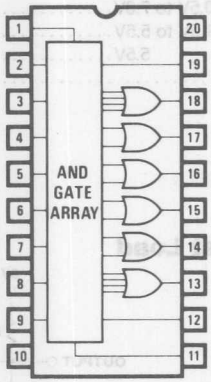
* These are absolute voltages with respect to the ground pin on the device and includes all overshoots due to system and/or tester noise. Do not attempt to test these values without suitable equipment.

* * Only one output shorted at a time.

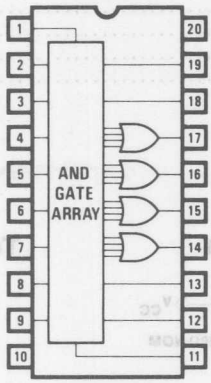
10H8



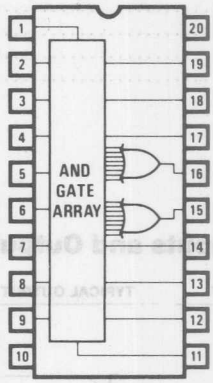
12H6



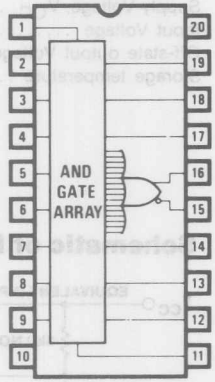
14H4



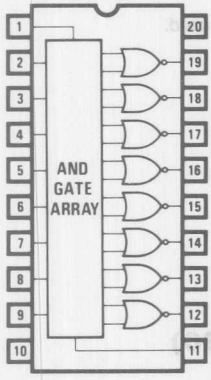
16H2



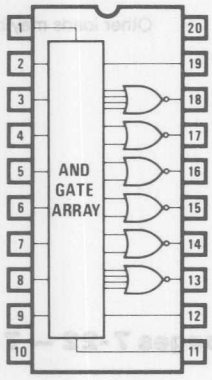
16C1



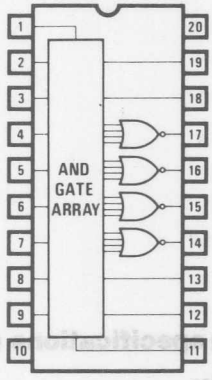
10L8



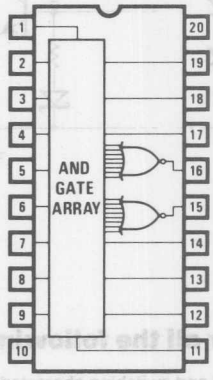
12L6



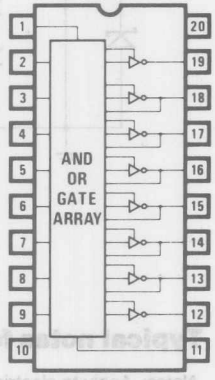
14L4



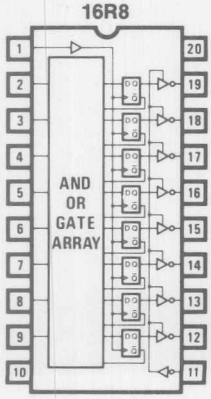
16L2



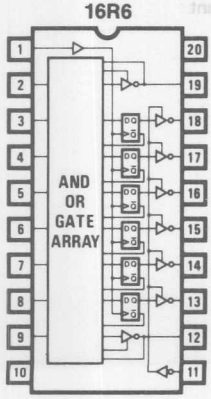
16L8



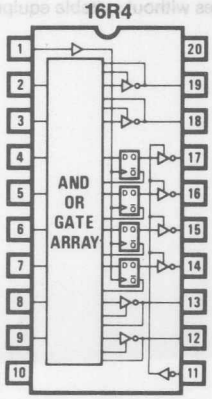
16R8



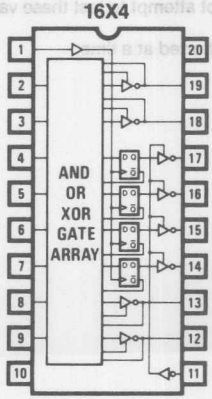
16R6



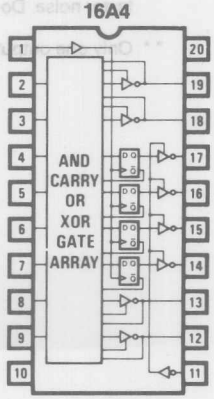
16R4



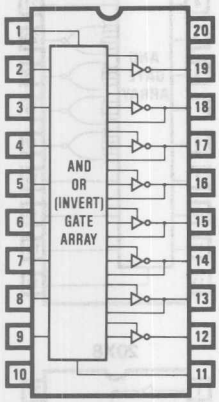
16X4



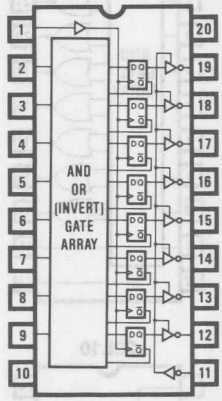
16A4



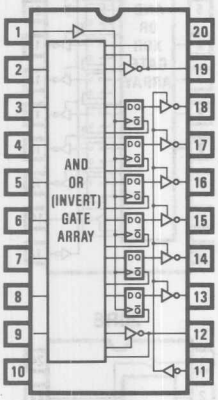
PAL16P8



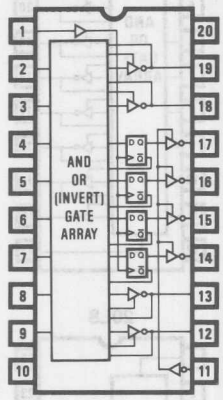
PAL16RP8



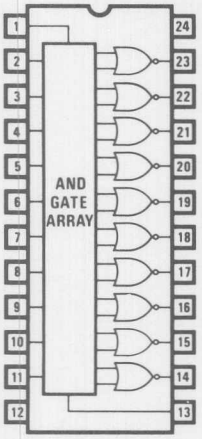
PAL16RP6



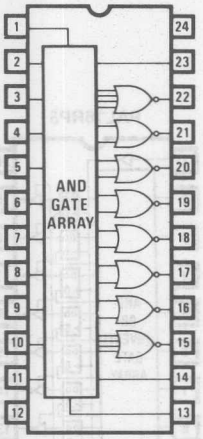
PAL16RP4



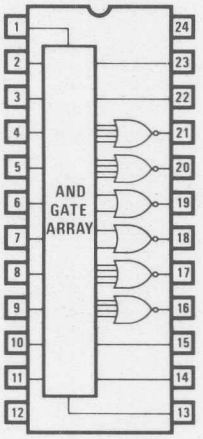
12L10



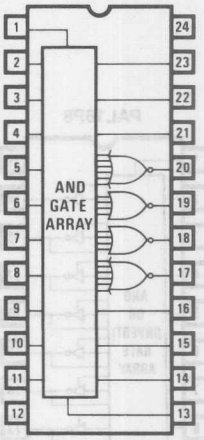
14L8



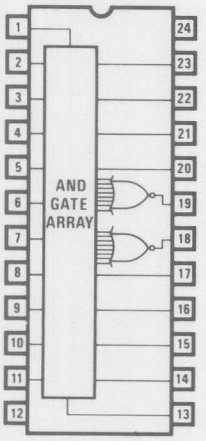
16L6



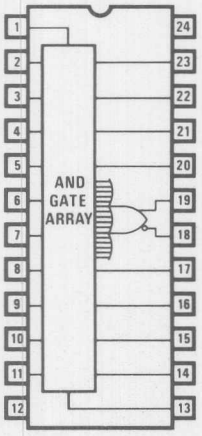
18L4



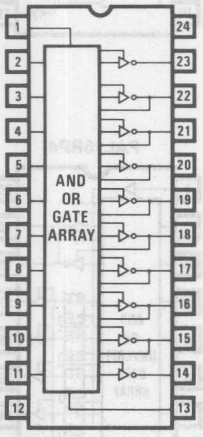
20L2



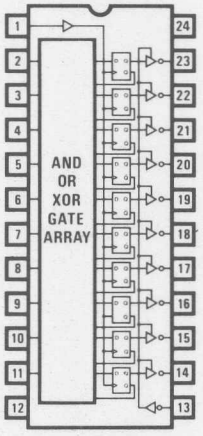
20C1



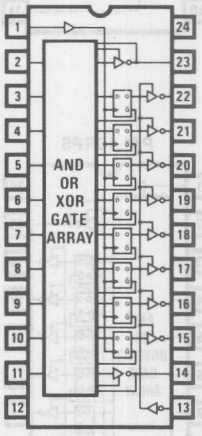
20L10



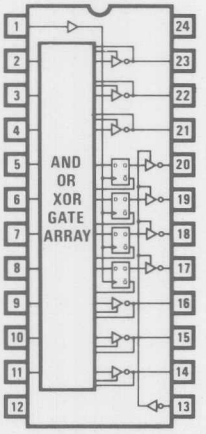
20X10



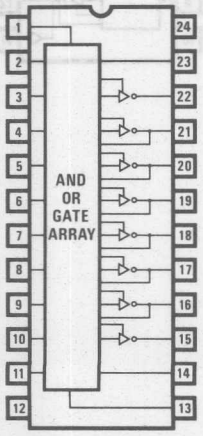
20X8



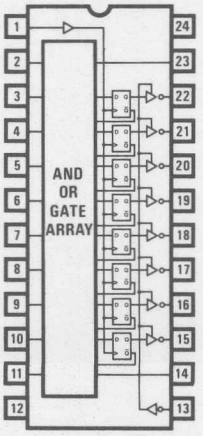
20X4



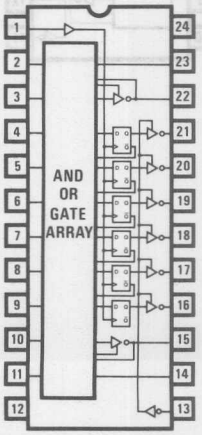
20L8



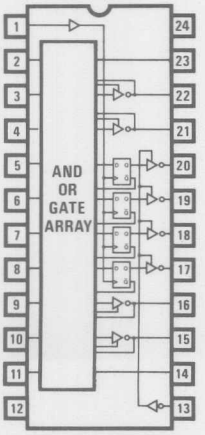
20R8

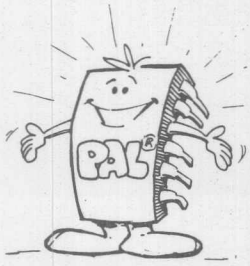


20R6

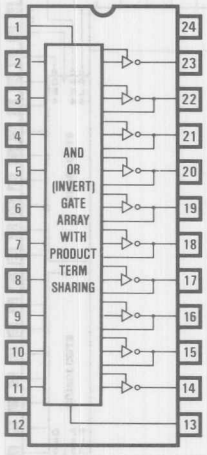


20R4

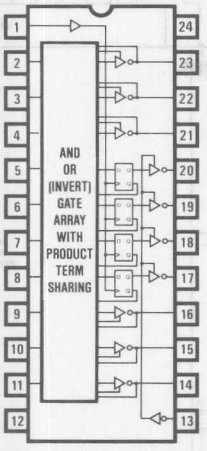




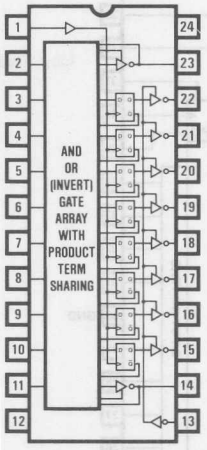
20S10



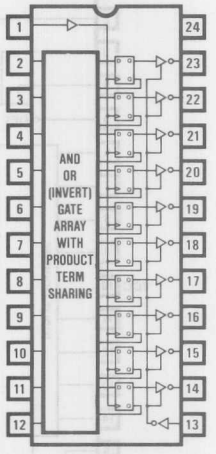
20RS4



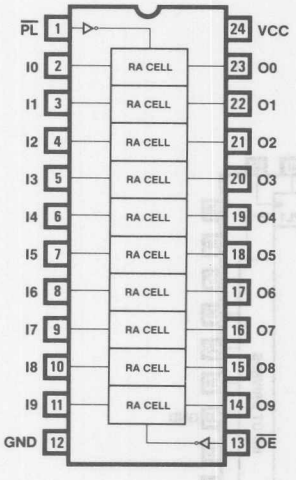
20RS8



20RS10

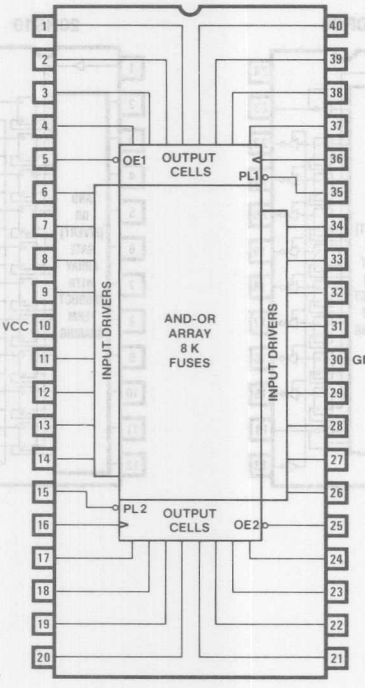


20RA10

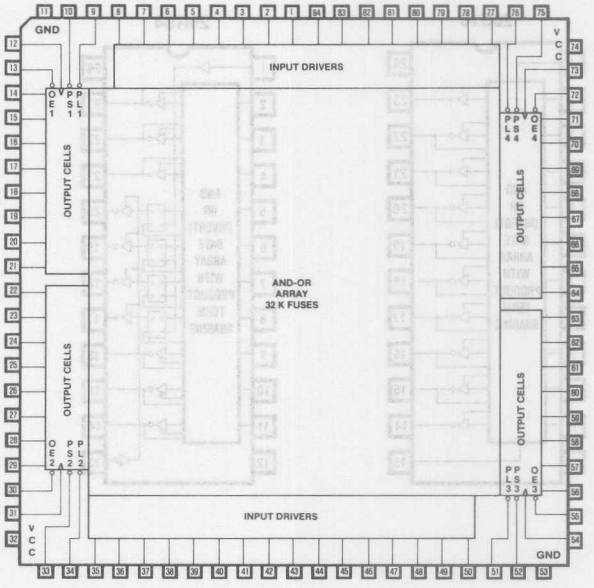


2

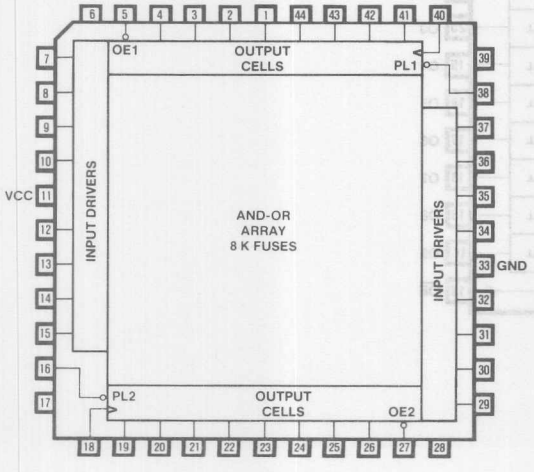
32R16



64R32



32R16



Operating Conditions

SYMBOL	PARAMETER	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
T_A	Operating free-air temperature	−55			0		75	°C
T_C	Operating case temperature			125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 8mA		0.3	0.5	V
			COM	I_{OL} = 8mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −2mA	2.4	2.8		V
			COM	I_{OH} = −3.2mA				
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX				55	90	mA

Switching Characteristics

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	Except 16C1	R1 = 560Ω, R2 = 1.1kΩ		25	45		25	35	ns
		16C1			25	45		25	40	

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	20	7		15	7		ns
		High	20	7		15	7		
t_{su}	Set up time from input or feedback to clock	20R8A 20R6A 20R4A	30	15		25	15		ns
t_h	Hold time		0	−10		0	−10		ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12mA		0.3	0.5	V
			COM	I_{OL} = 24mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −2mA	2.4	2.8		V
			COM	I_{OH} = −3.2mA				
I_{OZL}	Off-state output current †	V_{CC} = MAX		V_O = 0.4V			−100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	−30	−90	−130	mA
I_{CC}	Supply current	V_{CC} = MAX				160	210	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	20R6A 20R4A 20L8A	R_1 = 200Ω, R_2 = 390Ω		15	30		15	25	ns
t_{CLK}	Clock to output or feedback				10	20		10	15	ns
t_{PZX}	Pin 13 to output enable except 20L8A				10	25		10	20	ns
t_{PXZ}	Pin 13 to output disable except 20L8A				11	25		11	20	ns
t_{PZX}	Input to output enable	20R6A 20R4A 20L8A			10	30		10	25	ns
t_{PXZ}	Input to output disable	20R6A 20R4A 20L8A			13	30		13	25	ns
f_{MAX}	Maximum frequency	20R8A 20R6A 20R4A		20	40		28.5	40		MHz

Operating Conditions

SYMBOL	PARAMETER	MILITARY			COMMERCIAL			UNIT
		MIN	TYP	MAX	MIN	TYP	MAX	
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
T_A	Operating free-air temperature	–55			0		75	°C
T_C	Operating case temperature			125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = –18mA			–0.8	–1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4V			–0.02	–0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 8mA		0.3	0.5	V
			COM	I_{OL} = 8mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = –2mA	2.4	2.8		V
			COM	I_{OH} = –3.2mA				
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	–30	–70	–130	mA
I_{CC}	Supply current	V_{CC} = MAX				60	100	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS	MILITARY			COMMERCIAL			UNIT
			MIN	TYP	MAX	MIN	TYP	MAX	
t_{PD}	Input or feedback to output	R1 = 560Ω R2 = 1.1kΩ		25	45		25	40	ns

V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	25	10		25	10		ns
		High	25	10		25	10		
t_{su}	Set up time from input or feedback to clock	16R8 16R6 16R4	45	25		35	25		ns
		16X4 16A4	55	30		45	30		
t_h	Hold time		0	−15		0	−15		ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12mA		0.3	0.5	V
			COM	I_{OL} = 24mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −2mA	2.4	2.8		V
			COM	I_{OH} = −3.2mA				
I_{OZL}	Off-state output current †	V_{CC} = MAX		V_O = 0.4V			−100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX	16R4 16R6 16R8 16L8			120	180	mA
			16X4			160	225	
			16A4			170	240	

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	16R6 16R4 16L8	R_1 = 200Ω, R_2 = 390Ω		25	45		25	35	ns
		16X4 16A4			30	45		30	40	ns
t_{CLK}	Clock to output or feedback				15	25		15	25	ns
t_{PZX}	Pin 11 to output enable except 16L8				15	25		15	25	ns
t_{PXZ}	Pin 11 to output disable except 16L8				15	25		15	25	ns
t_{PZX}	Input to output enable	16R6 16R4 16L8			25	45		25	35	ns
		16X4 16A4			30	45		30	40	ns
t_{PXZ}	Input to output disable	16R6 16R4 16L8			25	45		25	35	ns
		16X4 16A4			30	45		30	40	ns
f_{MAX}	Maximum frequency	16R8 16R6 16R4		14	25		16	25		MHz
		16X4 16A4		12	22		14	22		

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	40	20		35	20		ns
		High	30	10		25	10		
t_{su}	Set up time from input or feedback to clock		60	38		50	38		ns
t_h	Hold time		0	–15		0	–15		ns
T_A	Operating free-air temperature		–55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = –18mA			–0.8	–1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			–0.02	–0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12mA		0.3	0.5	V
			COM	I_{OL} = 24mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = –2mA	2.4	2.8		V
			COM	I_{OH} = –3.2mA				
I_{OZL}	Off-state output current †	V_{CC} = MAX		V_O = 0.4V			–100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	–30	–70	–130	mA
I_{CC}	Supply current	V_{CC} = MAX	20X10, 20X8, 20X4			120	180	mA
I_{CC}	Supply current	V_{CC} = MAX	20L10			90	165	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	R_1 = 200Ω, R_2 = 390Ω		35	60		35	50	ns
t_{CLK}	Clock to output or feedback			20	35		20	30	ns
$t_{PXZ/ZX}$	Pin 13 to output disable/enable except 20L10			20	45		20	35	ns
t_{PZX}	Input to output enable except 20X10			35	55		35	45	ns
t_{PXZ}	Input to output disable except 20X10			35	55		35	45	ns
f_{MAX}	Maximum frequency		10.5	16		12.5	16		MHz

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	20	10		15	10		ns
		High	20	10		15	10		
t_{su}	Set up time from input or feedback to clock	16R8A 16R6A 16R4A 16RP8A 16RP6A 16RP4A	30	15		25	15		ns
t_h	Hold time		0	−10		0	−10		ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12mA		0.3	0.5	V
			COM	I_{OL} = 24mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −2mA	2.4	2.8		V
			COM	I_{OH} = −3.2mA				
I_{OZL}	Off-state output current †	V_{CC} = MAX		V_O = 0.4V			−100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current **	V_{CC} = 5V		V_O = 0V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX				120	180	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	16R6A 16R4A 16L8A 16RP6A 16RP4A 16P8A	R_1 = 200Ω, R_2 = 390Ω		15	30		15	25	ns
t_{CLK}	Clock to output or feedback				10	20		10	15	ns
t_{PZX}	Pin 11 to output enable except 16L8A 16P8A				10	25		10	20	ns
t_{PXZ}	Pin 11 to output disable except 16L8A 16P8A				11	25		11	20	ns
t_{PZX}	Input to output enable	16R6A 16R4A 16L8A 16RP6A 16RP4A 16P8A			10	30		10	25	ns
t_{PXZ}	Input to output disable	16R6A 16R4A 16L8A 16RP6A 16RP4A 16P8A			13	30		13	25	ns
f_{MAX}	Maximum frequency	16R8A 16R6A 16R4A 16RP8A 16RP6A 16RP4A		20	40		28.5	40		MHz

Operating Conditions

SYMBOL	PARAMETER	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage	4.5	5	5.5	4.75	5	5.25	V
T_A	Operating free-air temperature	−55		125	0		75	°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 4mA		0.3	0.5	V
			COM	I_{OL} = 4mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −1mA	2.4	2.8		V
			COM	I_{OH} = −1mA				
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX				30	45	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	R1 = 1.12kΩ R2 = 2.2kΩ		45	80		45	60	ns

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	25	10		25	10		ns
		High	25	10		25	10		
t_{su}	Set up time from input or feedback to clock	16R6A-2 16R4A-2 16R8A-2	50	25		35	25		ns
t_h	Hold time		0	–15		0	–15		ns
T_A	Operating free-air temperature		–55		125	0		75	°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = –18mA			–0.8	–1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			–0.02	–0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12mA		0.3	0.5	V
			COM	I_{OL} = 24mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = –2mA	2.4	2.8		V
			COM	I_{OH} = –3.2mA				
I_{OZL}	Off-state output current †	V_{CC} = MAX		V_O = 0.4V			–100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	–30	–70	–130	mA
I_{CC}	Supply current	V_{CC} = MAX				60	90	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	16L8A-2 16R6A-2 16R4A-2	R_1 = 200Ω, R_2 = 390Ω		25	50		25	35	ns
t_{CLK}	Clock to output or feedback				15	25		15	25	ns
$t_{PXZ/ZX}$	Pin 11 to output disable/enable except 16L8A-2				15	25		15	25	ns
t_{PZX}	Input to output enable	16L8A-2 16R6A-2 16R4A-2			25	45		25	35	ns
t_{PXZ}	Input to output disable	16R8A-2 16R6A-2 16R4A-2			25	45		25	35	ns
f_{MAX}	Maximum frequency	16R8A-2 16R6A-2 16R4A-2		14	25		16	25		MHz

Operating Conditions

SYMBOL	PARAMETER			MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage			4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	16R8A-4 16R6A-4 16R4A-4	Low	40	20		30	20		ns
			High	40	20		30	20		
t_{su}	Set up time from input or feedback to clock	16R8A-4 16R6A-4 16R4A-4		90	45		60	45		ns
t_h	Hold time			0	−15		0	−15		ns
T_A	Operating free-air temperature			−55		125	0		75	°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−0.8	−1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4V			−0.02	−0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 4mA		0.3	0.5	V
			COM	I_{OL} = 8mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −1mA	2.4	2.8		V
			COM	I_{OH} = −1 mA				
I_{OZL}	Off-state output current†	V_{CC} = MAX		V_O = 0.4V			−100	μA
I_{OZH}				V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX	16R4A-4 16R6A-4 16R8A-4 16L8A-4			30	50	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input or feedback to output	16R6A-4 16R4A-4 16L8A-4	R_1 = 800Ω, R_2 = 1.56kΩ		35	75		35	55	ns
t_{CLK}	Clock to output or feedback				20	45		20	35	ns
$t_{PXZ/ZX}$	Pin 11 to output disable/enable—except 16L8A-4				15	40		15	30	ns
t_{PZX}	Input to output enable	16R6A-4 16R4A-4 16L8A-4			30	65		30	50	ns
t_{PXZ}	Input to output disable	16R6A-4 16R4A-4 16L8A-4			30	65		30	50	ns
f_{MAX}	Maximum frequency	16R8A-4 16R6A-4 16R4A-4		8	18		11	18		MHz

SYMBOL	PARAMETER		MILITARY MIN	TYP	MAX	COMMERCIAL MIN	TYP	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock		25	13		20	13		ns
t_{wp}	Preload pulse width		45	15		35	15		ns
t_{su}	Setup time for input or feedback to clock		25	10		20	10		ns
t_{sup}	Preload setup time		30	5		25	5		ns
t_h	Hold time	Polarity fuse intact	10	−2		10	−2		ns
		Polarity fuse blown	0	−6		0	−6		
t_{hp}	Preload hold time		30	5		25	5		ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITION		MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage					0.8	V
V_{IH}*	High-level input voltage			2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18 mA		−0.8	−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4 V		−0.02	−0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4 V			25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5 V			1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	I_{OL} = 8 mA		0.3	0.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN	I_{OH}: Mil-2 mA Com-3.2 mA	2.4	2.8		V
I_{OZ}	Off-state output current	V_{CC} = MAX	V_O = 2.4 V/V_O = 0.4 V	−100		100	μA
I_{OS}	Output short-circuit current**	V_{CC} = 5 V	V_O = 0 V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX			155	200	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	TYP	MAX	COMMERCIAL MIN	TYP	MAX	UNIT
t_{PD}	Input or feedback to output	Polarity fuse intact	R_1 = 560 Ω, R_2 = 1.1 KΩ		20	35		20	30	ns
		Polarity fuse blown			25	40		25	35	
t_{CLK}	Clock to output or feedback			10	17	35	10	17	30	ns
t_S	Input to asynchronous set				22	40		22	35	ns
t_R	Input to asynchronous reset				27	45		27	40	ns
t_{PZX}	Pin 13 to output enable				10	25		10	20	ns
t_{PXZ}	Pin 13 to output disable				10	25		10	20	ns
t_{PZX}	Input to output enable				18	35		18	30	ns
t_{PXZ}	Input to output disable				15	35		15	30	ns
f_{MAX}	Maximum frequency			16	35		20	35		MHz

Operating Conditions

SYMBOL	PARAMETER			MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage			4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low		20	10		15	10		ns
		High		20	10		15	10		
t_{su}	Setup time from input or feedback to clock	20RS10 20RS8 20RS4		40	25		35	25		ns
t_h	Hold time			0	−10		0	−10		ns
T_A	Operating free-air temperature			−55			0		75	°C
T_C	Operating case temperature					125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITION			MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage						0.8	V
V_{IH}*	High-level input voltage				2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18 mA			−0.8	−1.5	V
I_{IL}	Low-level input current †	V_{CC} = MAX	V_I = 0.4 V			−0.02	−0.25	mA
I_{IH}	High-level input current †	V_{CC} = MAX	V_I = 2.4 V				25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5 V				1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL	I_{OL} = 12 mA		0.3	0.5	V
			COM	I_{OL} = 24 mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL	I_{OH} = −2 mA	2.4	2.8		V
			COM	I_{OH} = −3.2 mA				
I_{OZL}	Off-state output current†	V_{CC} = MAX		V_O = 0.4 V			−100	μA
I_{OZH}				V_{OL} = 2.4 mA			100	
I_{OS}	Output short-circuit current**	V_{CC} = 5V		V_O = 0 V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX				175	240	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	20S10, 20RS8, 20RS4 Input or feedback to output	Polarity fuse intact	R_1 = 200 Ω R_2 = 390 KΩ		25	40		25	35	ns
		Polarity fuse blown			30	45		30	40	
t_{CLK}	Clock to output or feedback				12	20		12	17	ns
t_{PZX}	Pin 13 to output enable except 20S10				10	25		10	20	ns
t_{PXZ}	Pin 13 to output disable except 20S10				11	25		11	20	ns
t_{PZX}	Input to output enable	20S10, 20RS8, 20RS4			25	35		25	35	ns
t_{PXZ}	Input to output disable	20S10, 20RS8 20RP4			13	25		13	25	ns
f_{MAX}	20RS10, 20RS8, 20RS4 Maximum frequency			18	28		20	28		MHz

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	25			20			ns
		High	25			20			
t_{wp}	Preload pulse width		45			35			ns
t_{su}	Setup time for input to clock	Polarity fuse intact	50			40			ns
		Polarity fuse blown	50			40			
t_{sup}	Preload setup time		30			25			ns
t_h	Hold time		0		−10	0		−10	ns
t_{hp}	Preload hold time		10			5			ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITION		MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage					0.8	V
V_{IH}*	High-level input voltage			2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18 mA		−0.8	−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4 V		−0.02	−0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4 V			25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5 V			1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL I_{OL} = 8 mA		0.3	0.5	V
			COM I_{OL} = 8 mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL I_{OH} = −2 mA	2.4	2.8		V
			COM I_{OH} = −3.2 mA				
I_{OZL}	Off-state output current	V_{CC} = MAX	V_O = 0.4 V			−100	μA
I_{OZH}			V_O = 2.4 V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = MAX	V_O = 0 V	−30	−70	−130	mA
I_{CC}	Supply current	V_{CC} = MAX			200	280	mA

Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input to output	Polarity fuse intact	R_1 = 560 Ω, R_2 = 1.1 KΩ			50			40	ns
		Polarity fuse blown				55			45	
t_{CLK}	Clock to output or feedback					30			25	ns
t_{PZX}	Output enable					25			20	ns
t_{PXZ}	Output disable					25			20	ns
f_{MAX}	Maximum frequency			14			16			MHz

Operating Conditions

SYMBOL	PARAMETER		MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	4.75	5	5.25	V
t_w	Width of clock	Low	25			20			ns
		High							
t_{su}	Setup time for input to clock	Polarity fuse intact	50			40			ns
		Polarity fuse blown							
t_h	Hold time		0	−10		0	−10		ns
T_A	Operating free-air temperature		−55			0		75	°C
T_C	Operating case temperature				125				°C

2

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITION		MIN	TYP	MAX	UNIT
V_{IL}*	Low-level input voltage					0.8	V
V_{IH}*	High-level input voltage			2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18 mA		−0.8	−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4 V		−0.02	−0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4 V			25	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5 V			1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	MIL I_{OL} = 8 mA		0.3	0.5	V
			COM I_{OL} = 8 mA				
V_{OH}	High-level output voltage	V_{CC} = MIN	MIL I_{OH} = −0.4 mA	2.4	2.8		V
			COM I_{OH} = −0.4 mA				
I_{OZL}	Off-state output current	V_{CC} = MAX	V_O = 0.4 V			−100	μA
I_{OZH}			V_O = 2.4 V			100	μA
I_{OS}	Output short-circuit current**	V_{CC} = MAX	V_O = 0 V	−10	−40	−60	mA
I_{CC}	Supply current	V_{CC} = MAX			400	640	mA

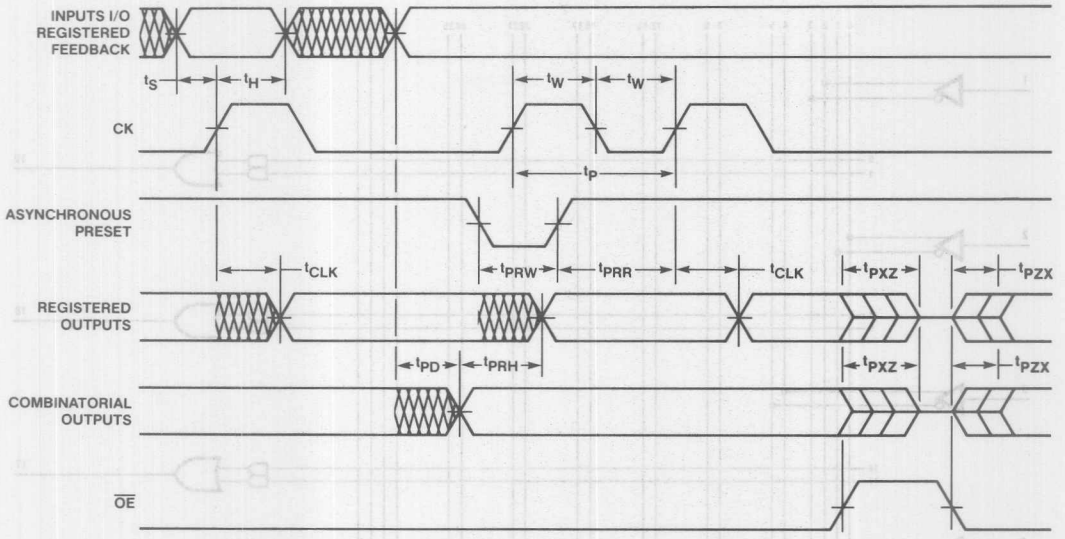
Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITIONS	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{PD}	Input to output	Polarity fuse intact	R_1 = 560Ω R_2 = 1.1 KΩ			55			50	ns
		Polarity fuse blown				60			55	
t_{CLK}	Clock to output or feedback					30			22	ns
t_{PZX}	Output enable					35			30	ns
t_{PXZ}	Output disable					35			30	ns
t_{PRH}	Preset to output					40			35	ns
f_{MAX}	Maximum frequency			12.5			16	20		MHz

Testing Conditions

SYMBOL	PARAMETER	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNIT
t_{wp}	Preload pulse width	45			35			ns
t_{sup}	Preload setup time	60			50			ns
t_{hp}	Preload hold time	10			5			ns
t_{PRW}	Preset pulse width	30			25			ns
t_{PRR}	Preset recovery time	40			35			ns

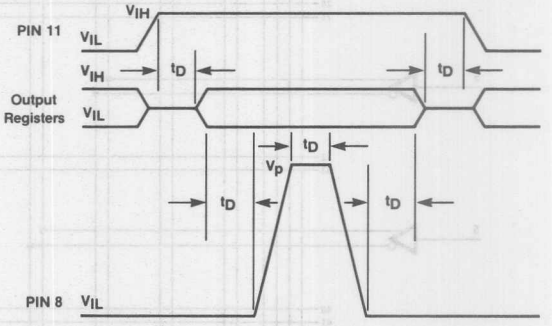
Switching Waveforms



Output Register PRELOAD Series 20AP

The PRELOAD function allows the register to be loaded from data placed on the output pins. This feature aids functional testing which would otherwise require a state sequencer for test coverage. The procedure for PRELOAD is as follows:

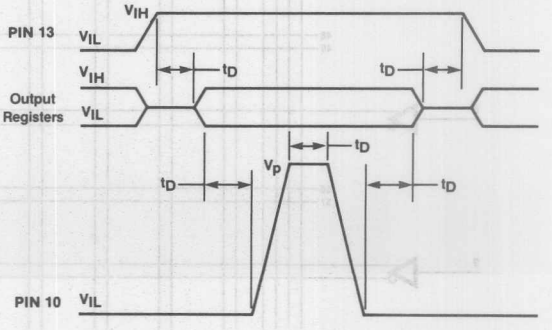
1 Raise V_{CC} to 4.5 V.
2 Disable output registers by setting pin 11 to V_{IH}.
3 Apply V_{IL}/V_{IH} to all output registers.
4 Pulse pin 8 to V_p. Then back to 0 V.
5 Remove V_{IL}/V_{IH} from all output registers.
6 Lower pin 11 to V_{IL} to enable the output registers.
7 Verify for V_{OL}/V_{OH} at all output registers.

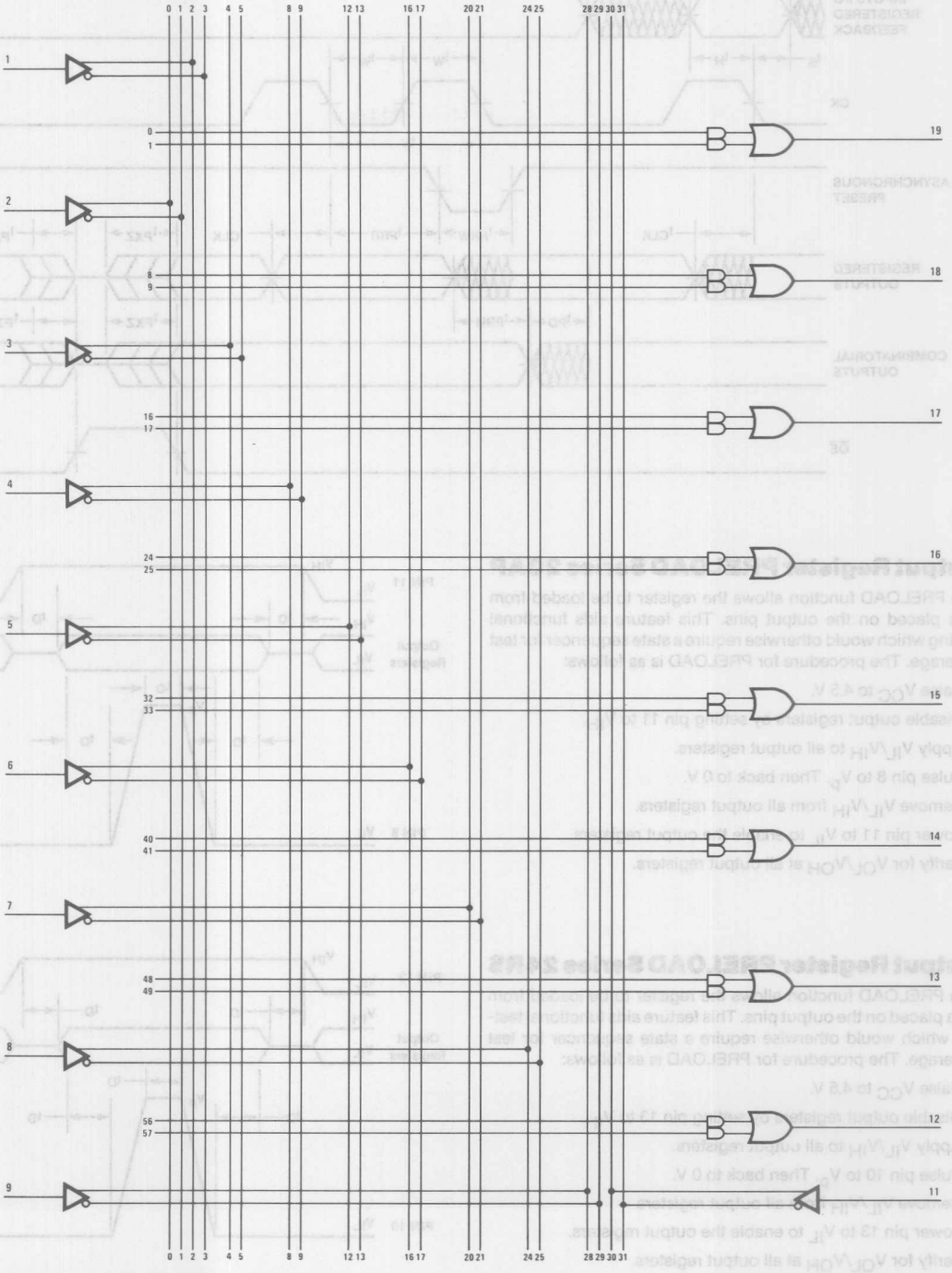


Output Register PRELOAD Series 24RS

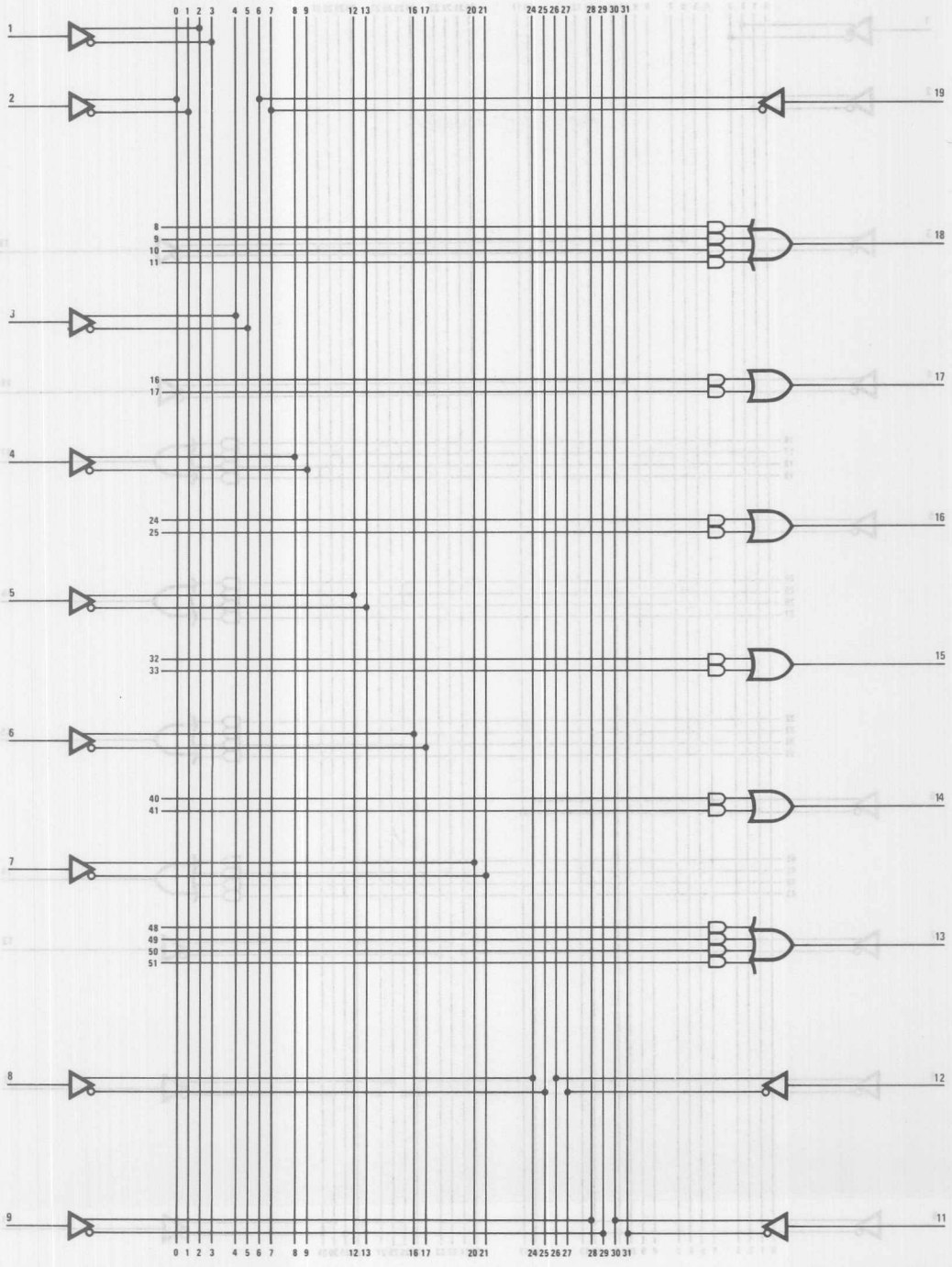
The PRELOAD function allows the register to be loaded from data placed on the output pins. This feature aids functional testing which would otherwise require a state sequencer for test coverage. The procedure for PRELOAD is as follows:

1 Raise V_{CC} to 4.5 V.
2 Disable output registers by setting pin 13 to V_{IH}.
3 Apply V_{IL}/V_{IH} to all output registers.
4 Pulse pin 10 to V_p. Then back to 0 V.
5 Remove V_{IL}/V_{IH} from all output registers.
6 Lower pin 13 to V_{IL} to enable the output registers.
7 Verify for V_{OL}/V_{OH} at all output registers.

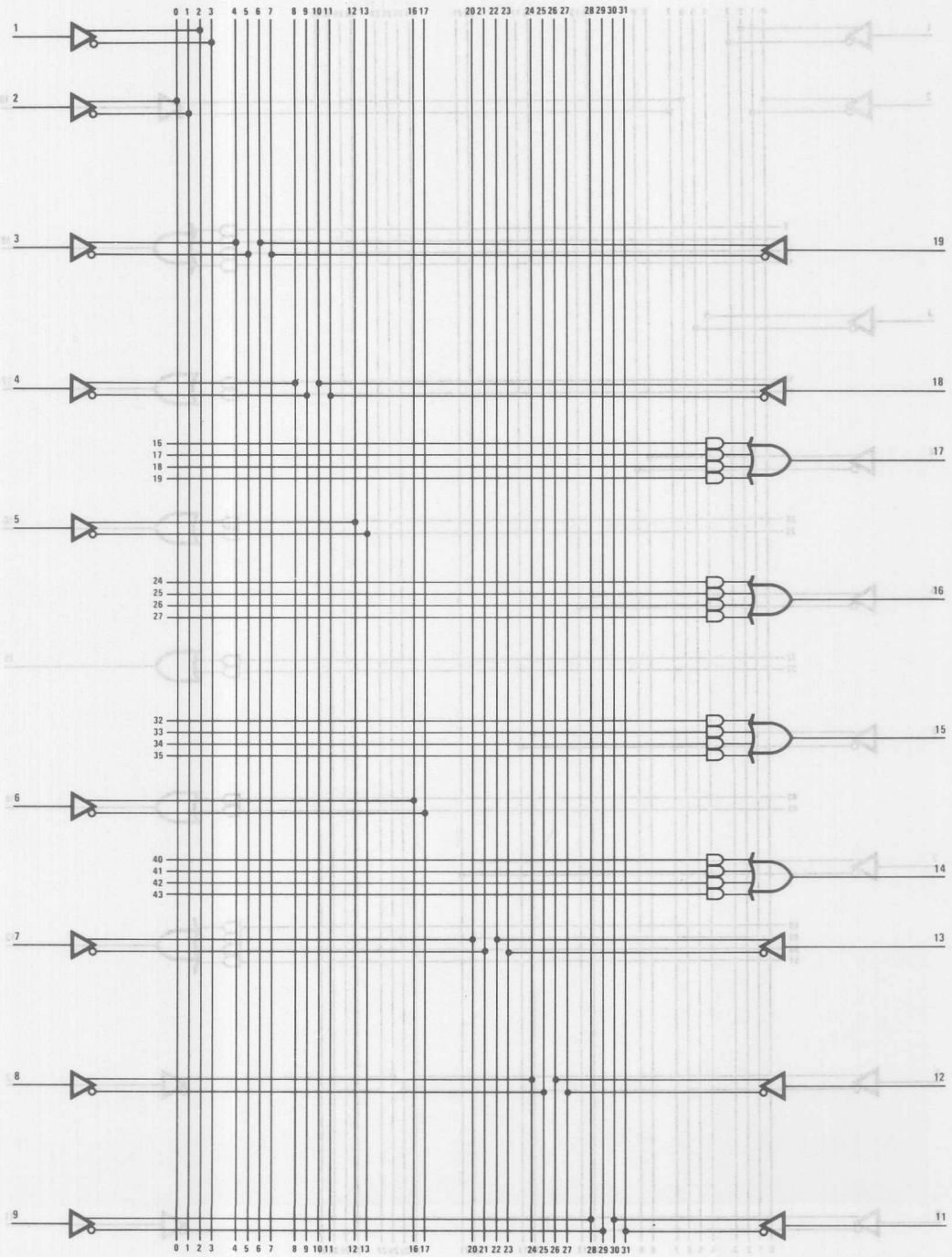



0 1 2 3 4 5 8 9 12 13 16 17 20 21 24 25 28 29 30 31
1
2
3
4
5
6
7
8
9
0
1
8
9
16
17
24
25
32
33
40
41
48
49
56
57
19
18
17
16
15
14
13
12
11

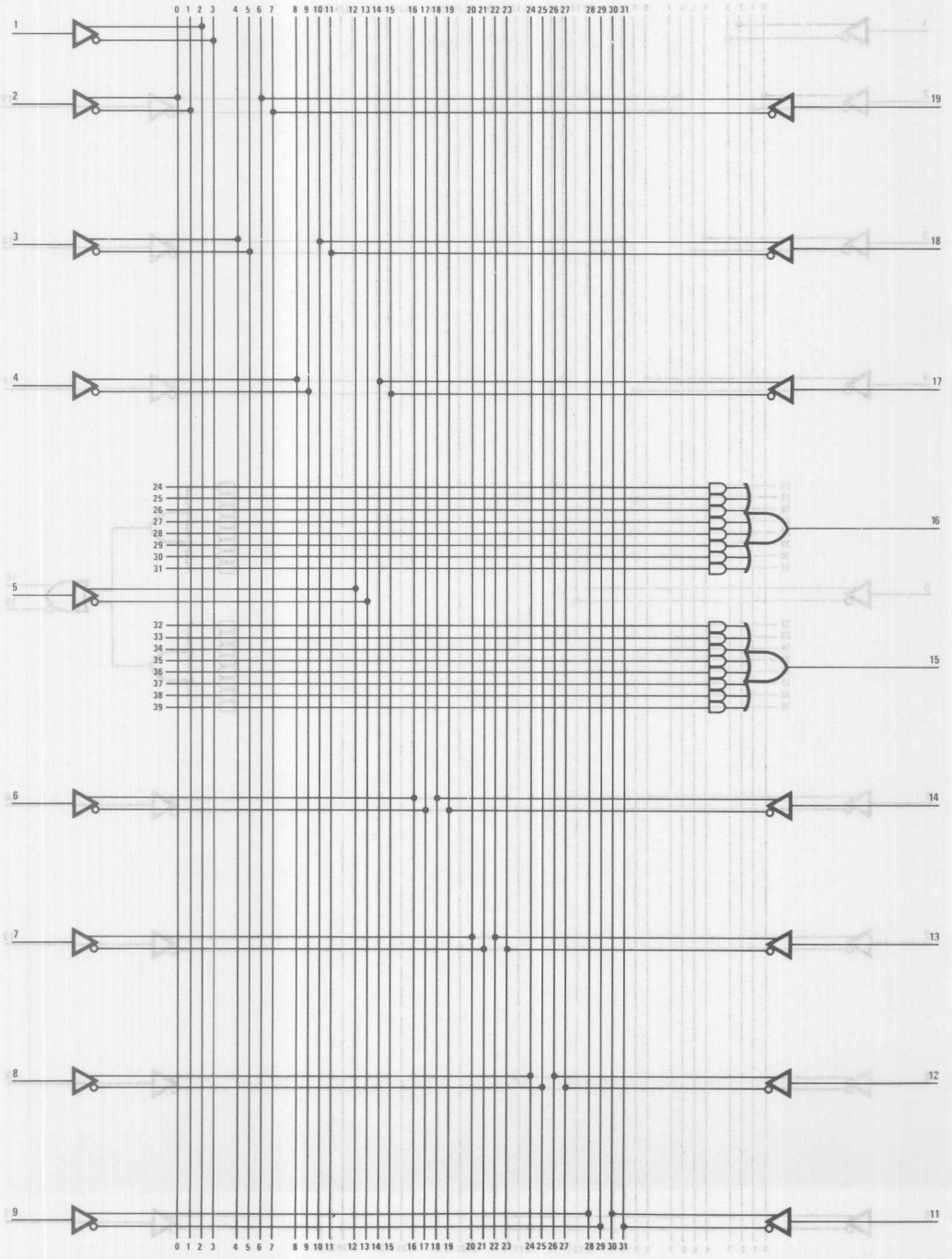
12H6


0 1 2 3 4 5 6 7 8 9 12 13 16 17 20 21 24 25 26 27 28 29 30 31
1
2
19
8
9
10
11
18
3
16
17
17
4
24
25
16
5
32
33
15
6
40
41
14
7
48
49
50
51
13
8
12
9
11
0 1 2 3 4 5 6 7 8 9 12 13 16 17 20 21 24 25 26 27 28 29 30 31

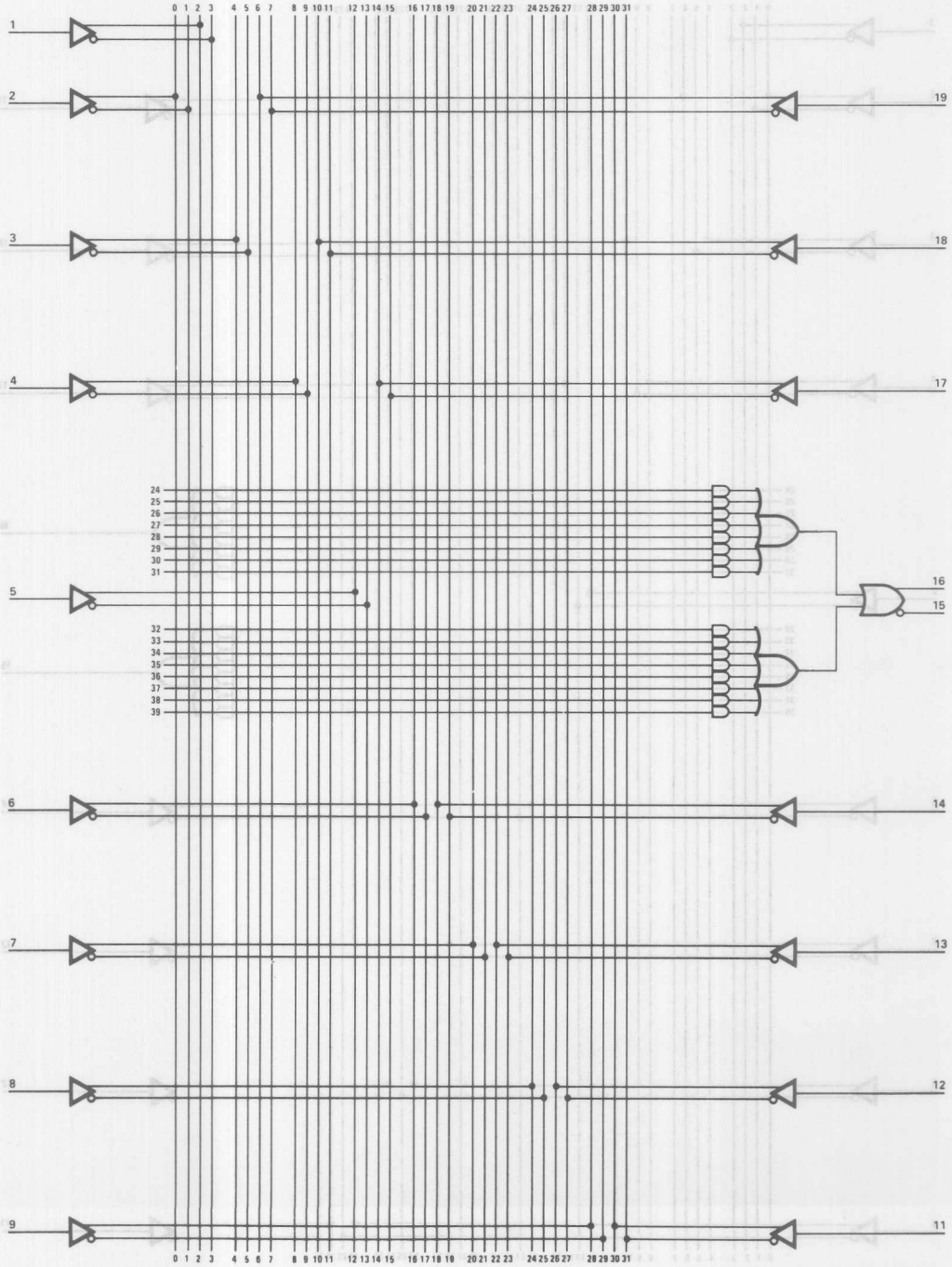
14H4


0 1 2 3 4 5 6 7 8 9 10 11 12 13 16 17 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
16 17 18 19
24 25 26 27
32 33 34 35
40 41 42 43

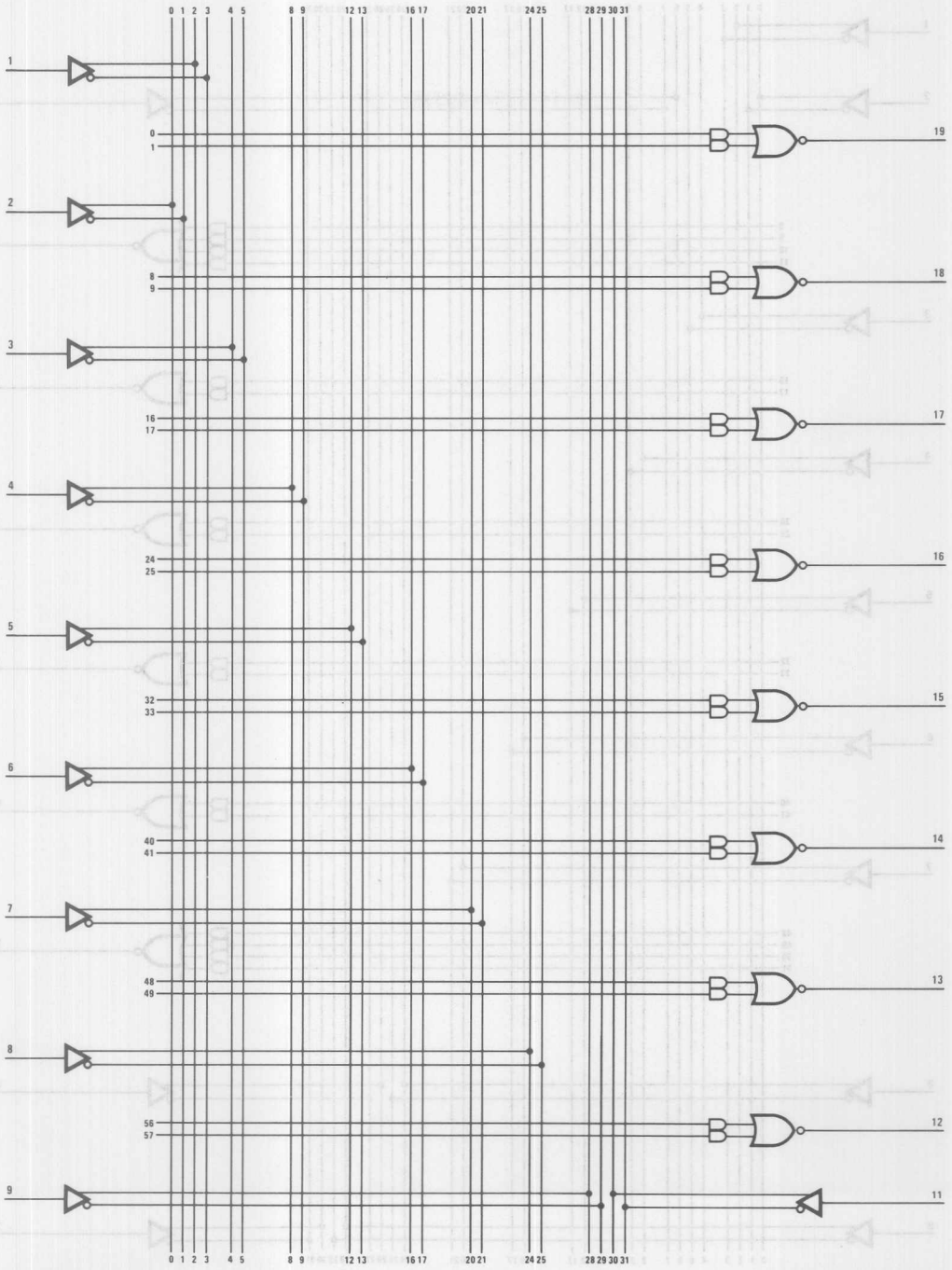
16H2


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39

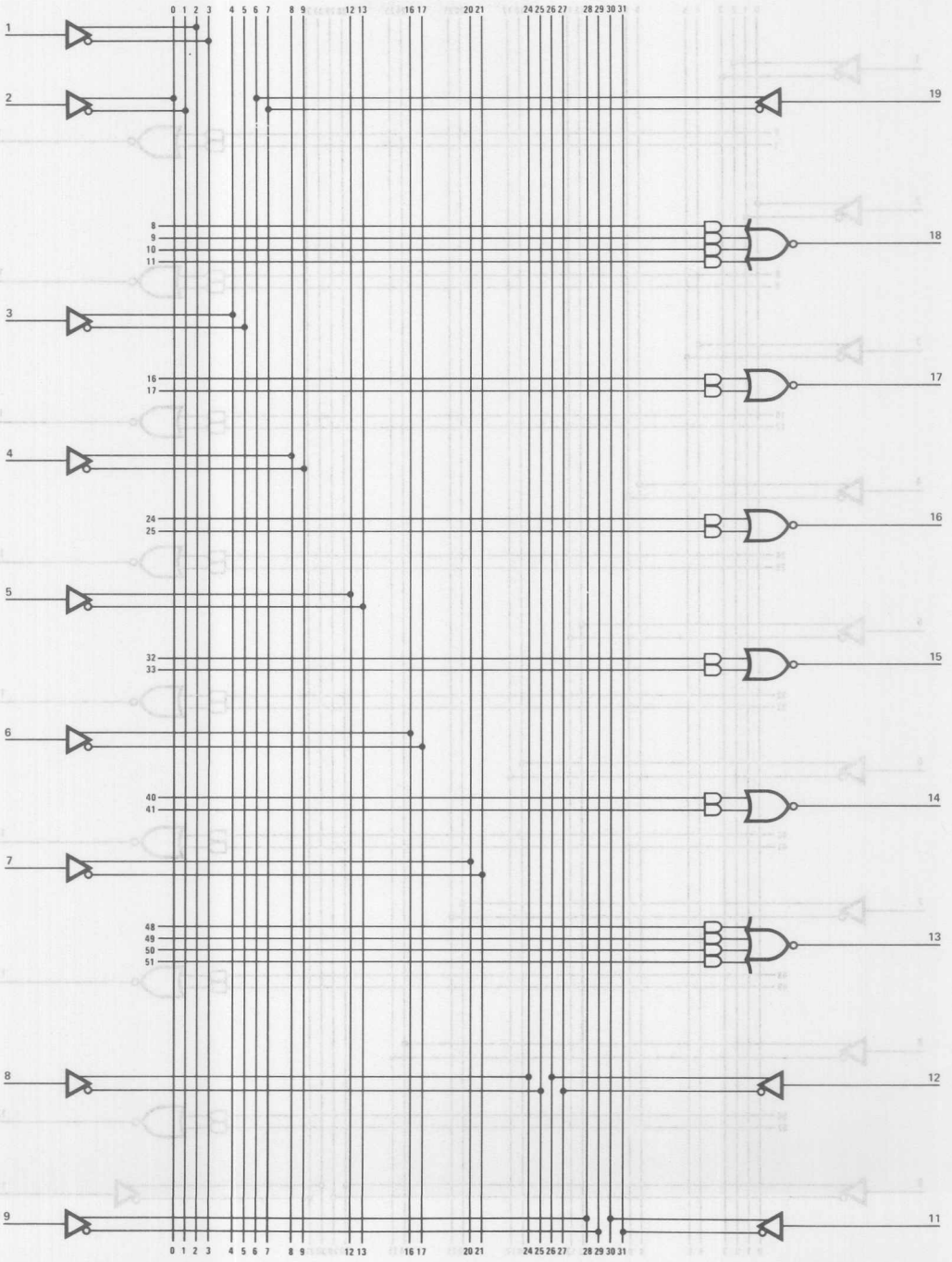
16C1


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

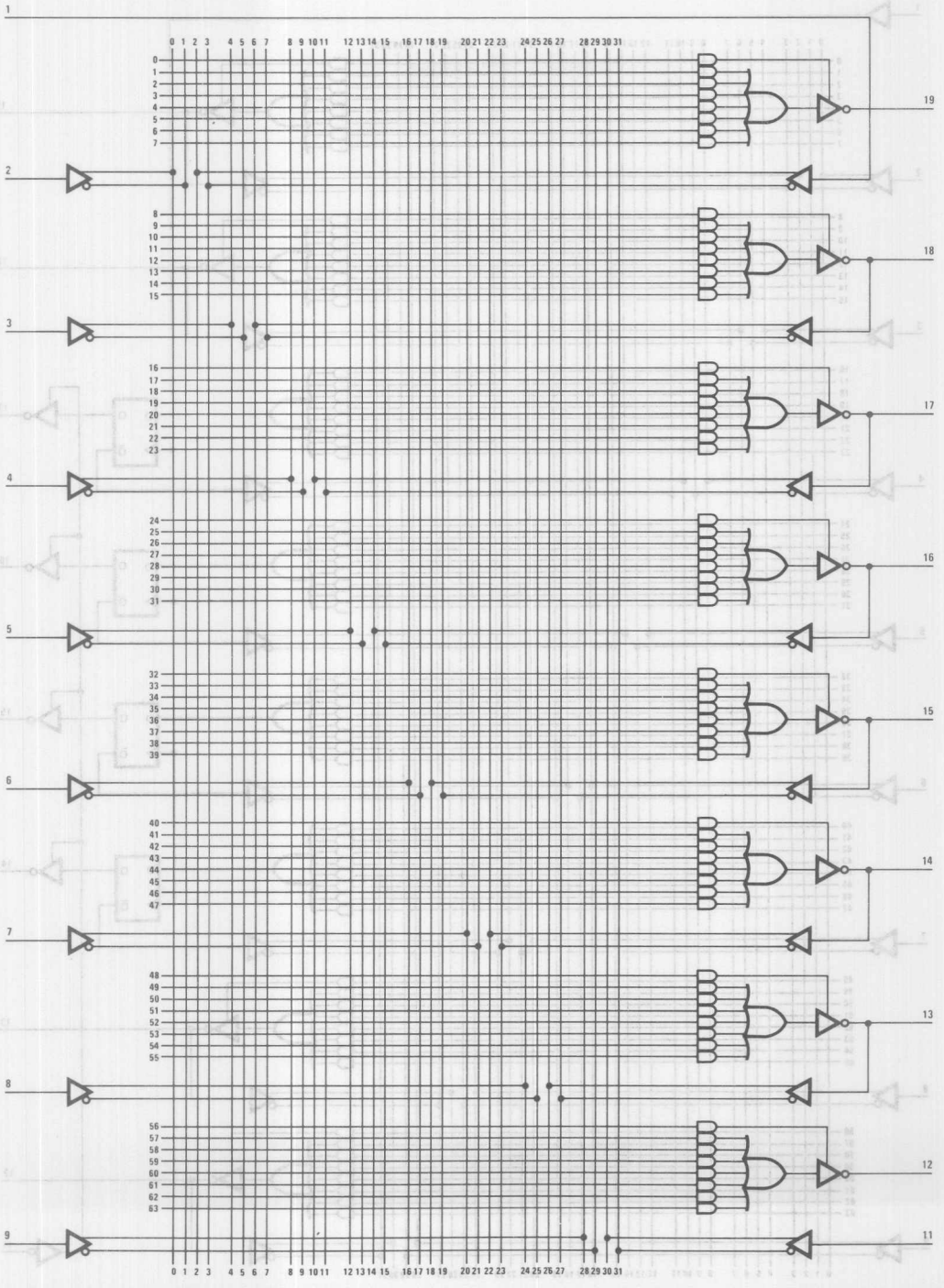
10L8


0 1 2 3 4 5 8 9 12 13 16 17 20 21 24 25 28 29 30 31
1
0
1
19
2
8
9
18
3
16
17
17
4
24
25
16
5
32
33
15
6
40
41
14
7
48
49
13
8
56
57
12
9
11

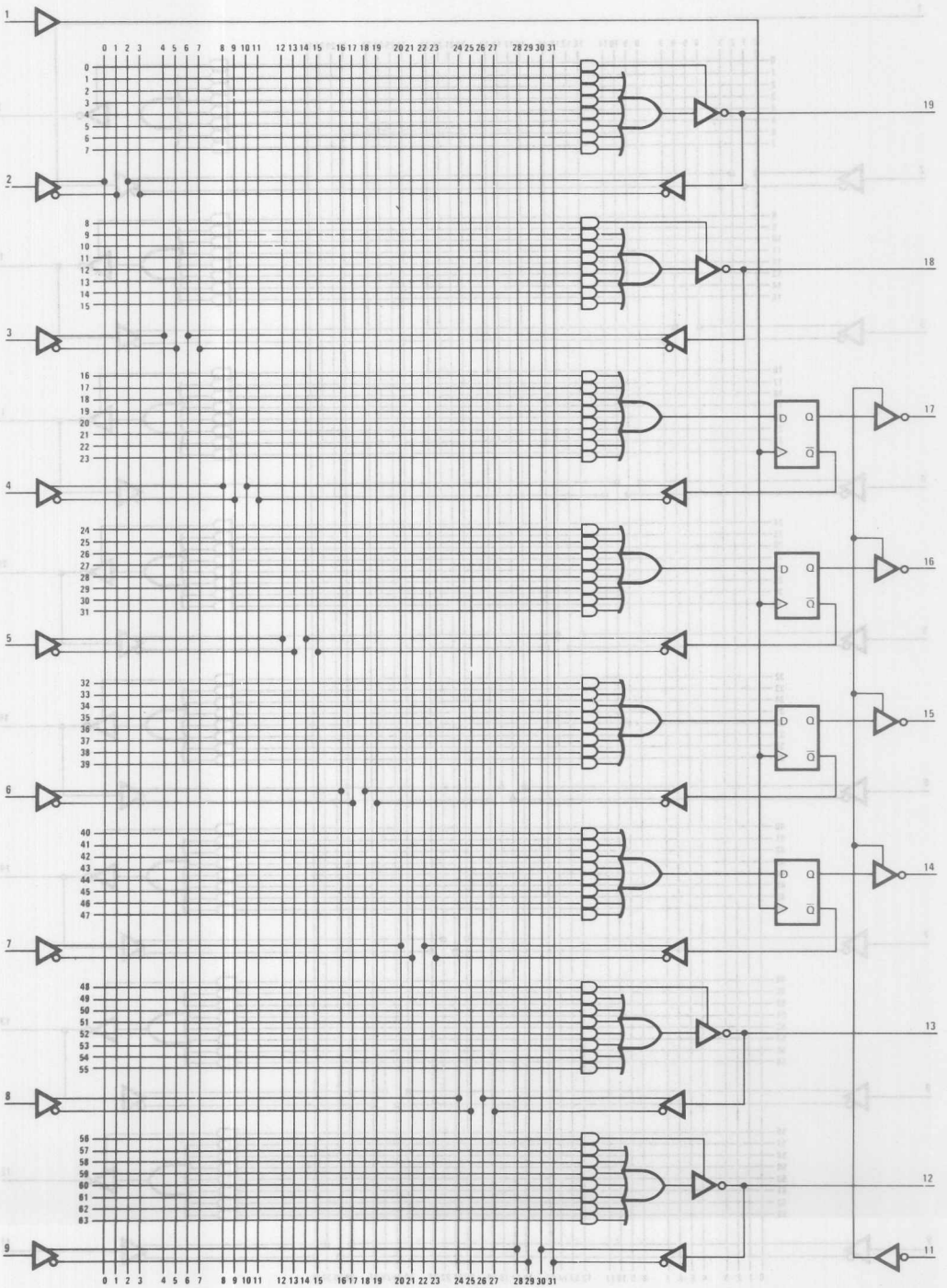
12L6


0 1 2 3 4 5 6 7 8 9 12 13 16 17 20 21 24 25 26 27 28 29 30 31
1
2
19
8
9
10
11
18
3
16
17
17
4
24
25
16
5
32
33
15
6
40
41
14
7
48
49
50
51
13
8
12
9
11

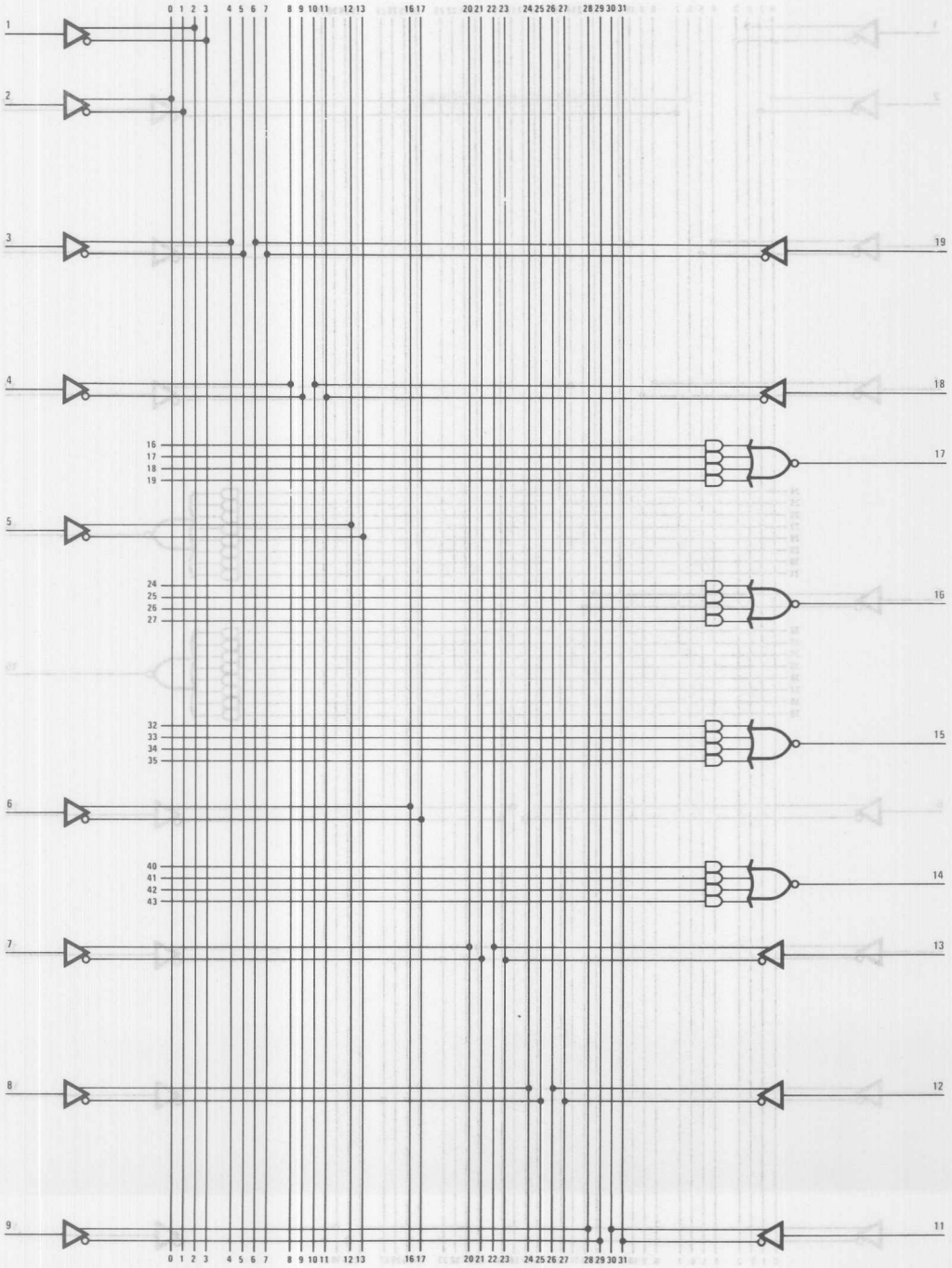
16L8


1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0
1
2
3
4
5
6
7
19
2
8
9
10
11
12
13
14
15
18
3
16
17
18
19
20
21
22
23
17
4
24
25
26
27
28
29
30
31
16
5
32
33
34
35
36
37
38
39
15
6
40
41
42
43
44
45
46
47
14
7
48
49
50
51
52
53
54
55
13
8
56
57
58
59
60
61
62
63
12
9
11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

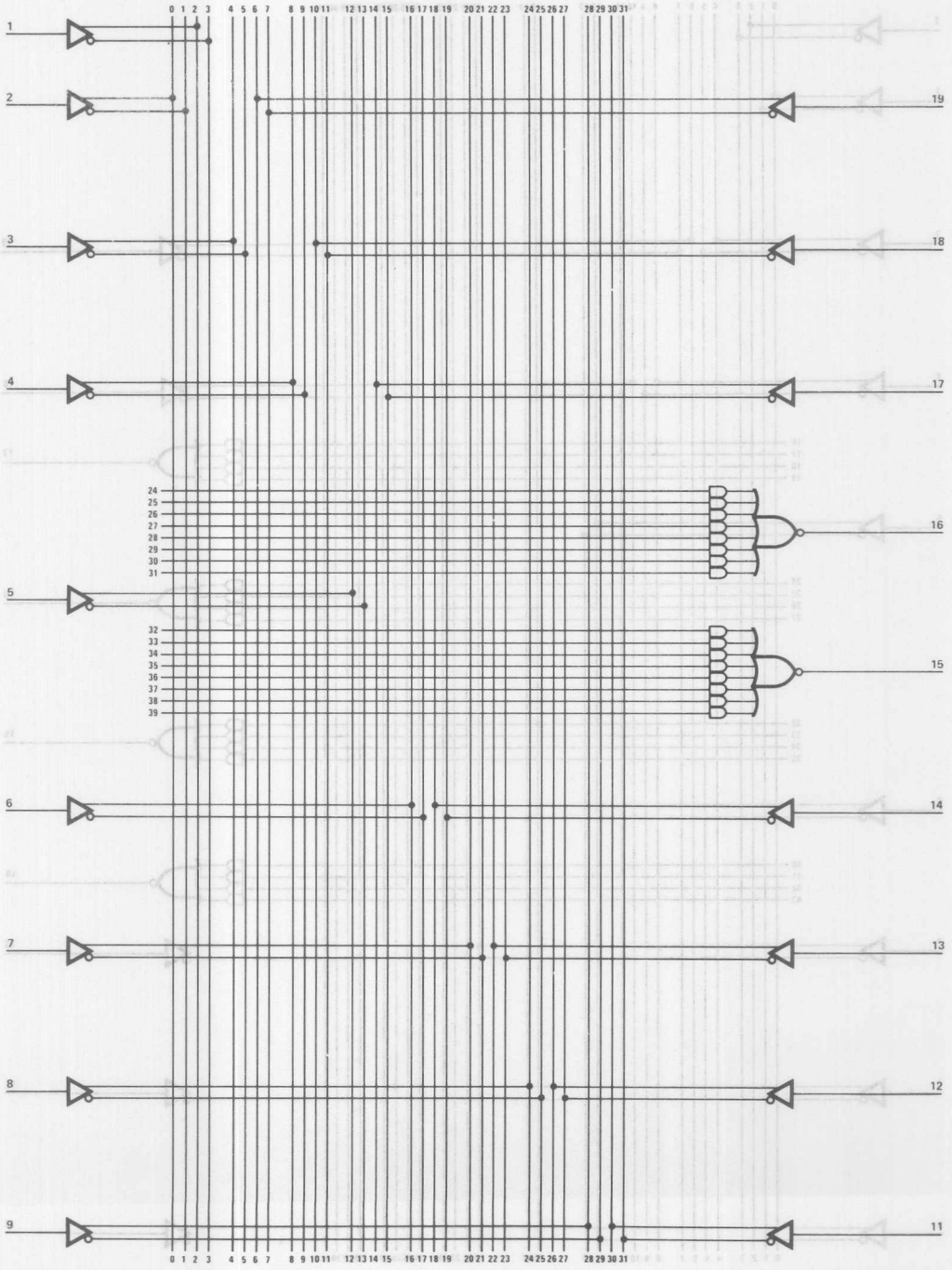
2


1
2
3
4
5
6
7
8
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
19
18
17
16
15
14
13
12
11
D
Q
Q̄

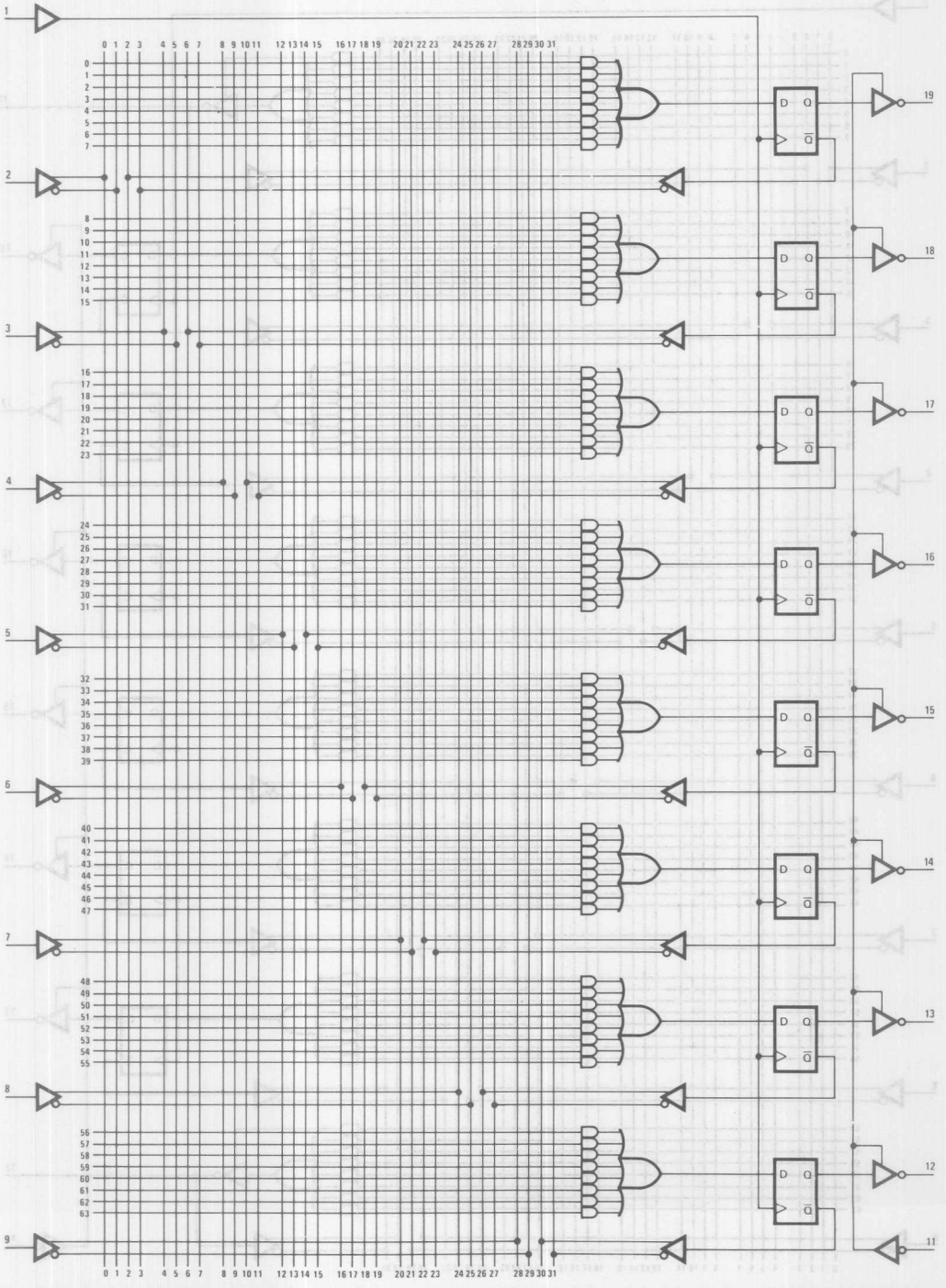
14L4

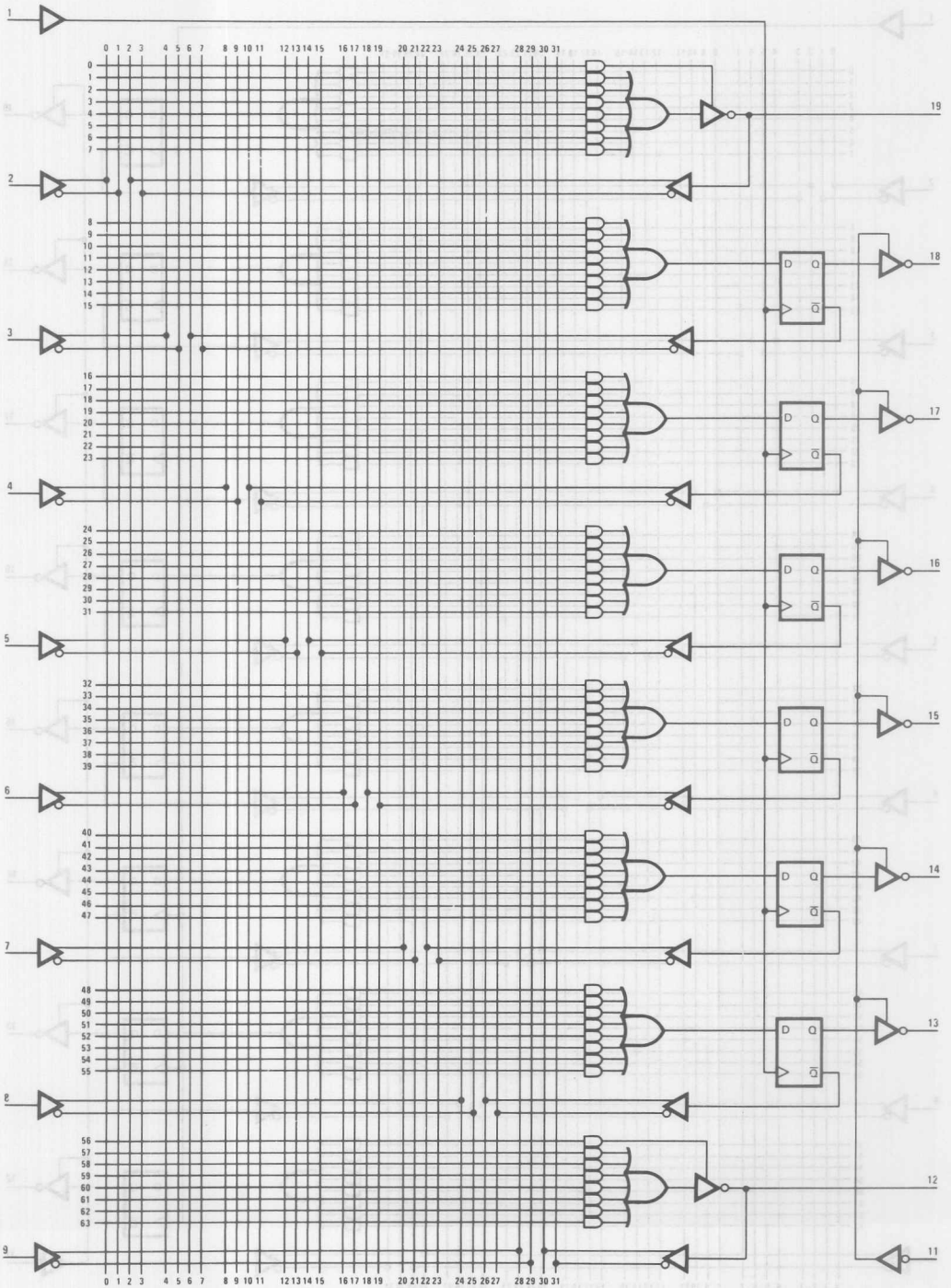

0 1 2 3 4 5 6 7 8 9 10 11 12 13 16 17 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
16 17 18 19
24 25 26 27
32 33 34 35
40 41 42 43

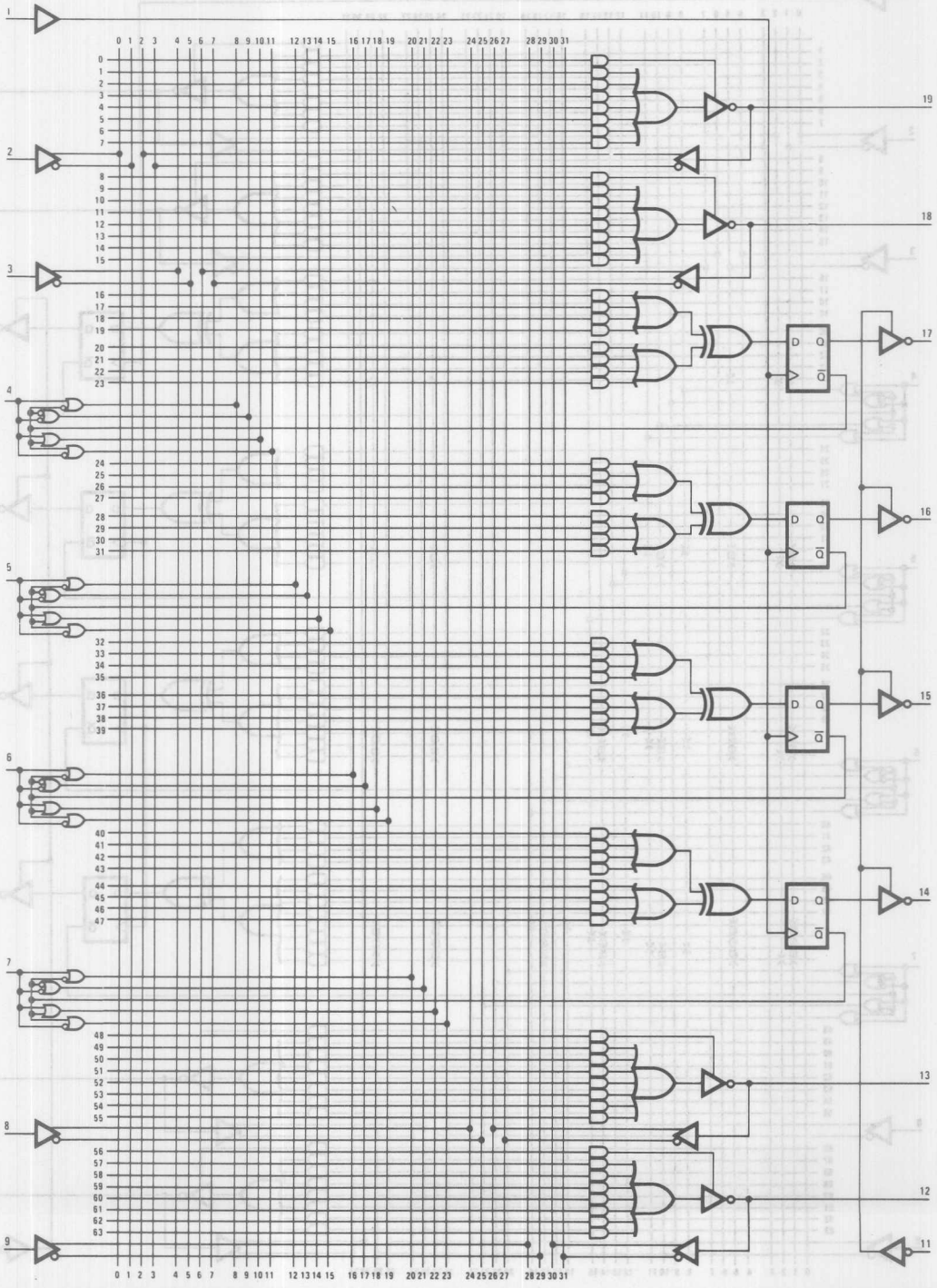
16L2


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39

16R8

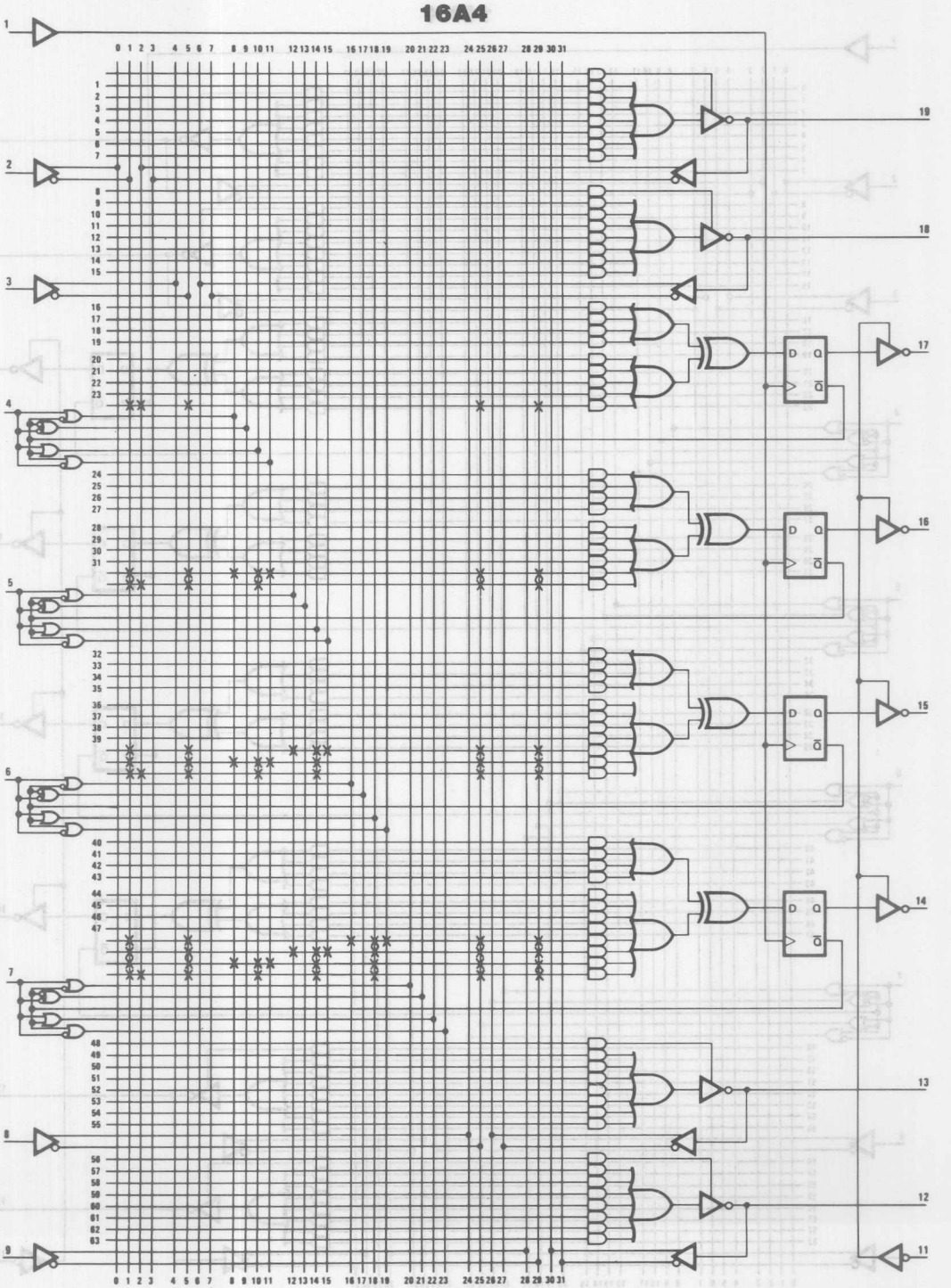

1
2
3
4
5
6
7
8
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
D Q
Q
19
18
17
16
15
14
13
12
11


1
2
3
4
5
6
7
8
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
D Q
Q̄
19
18
17
16
15
14
13
12
11

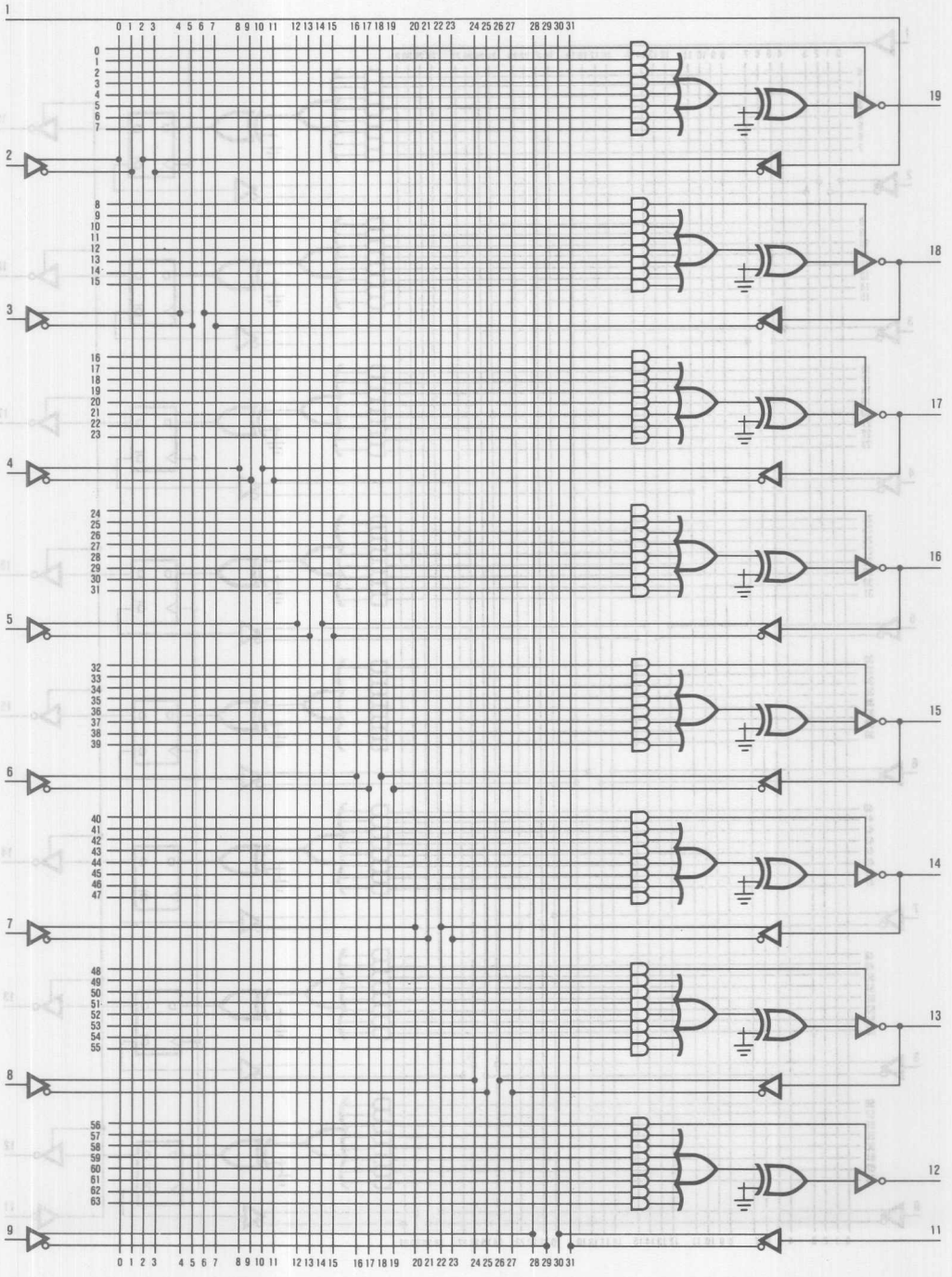

1
2
3
4
5
6
7
8
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
19
18
17
16
15
14
13
12
11
D Q
Q

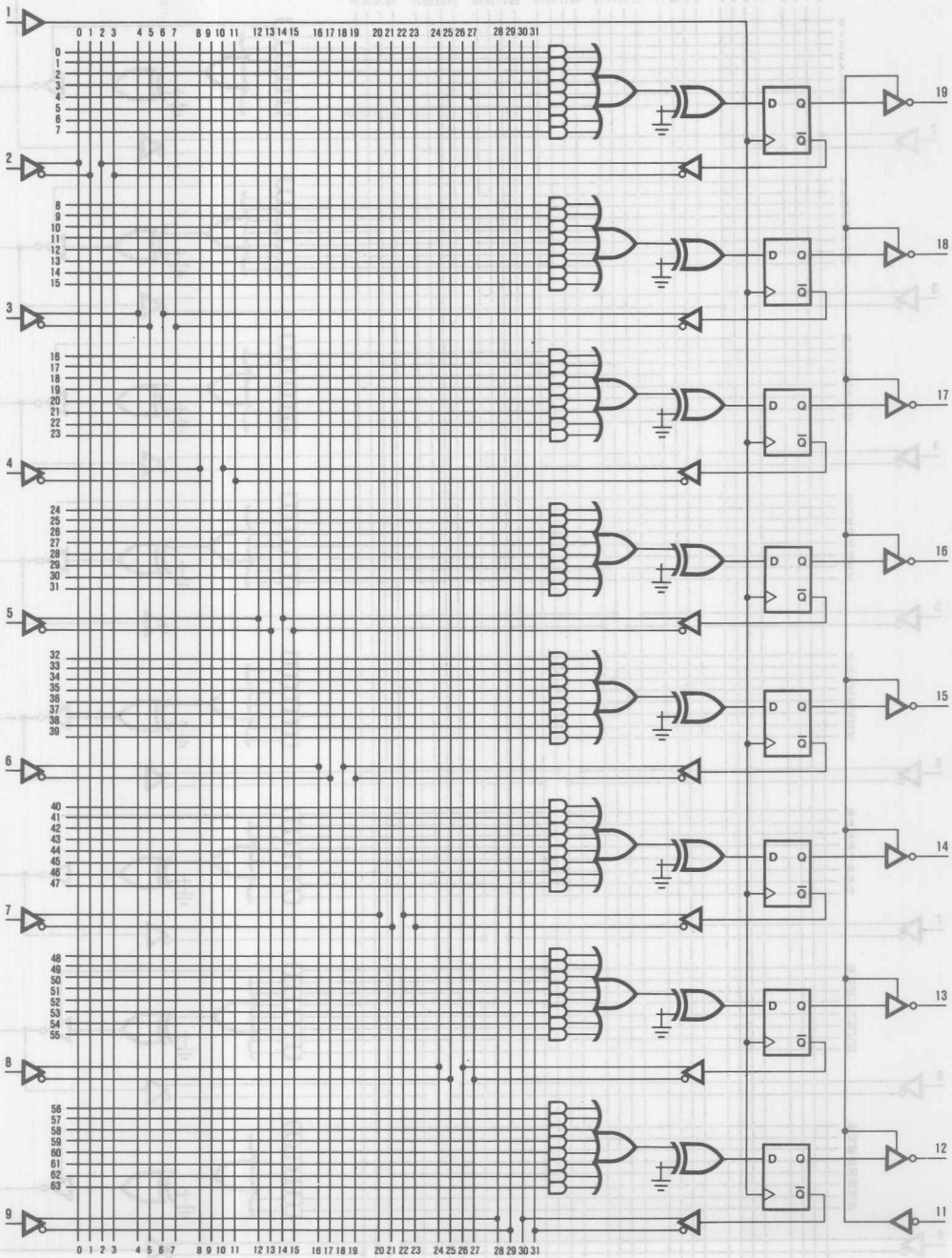
2

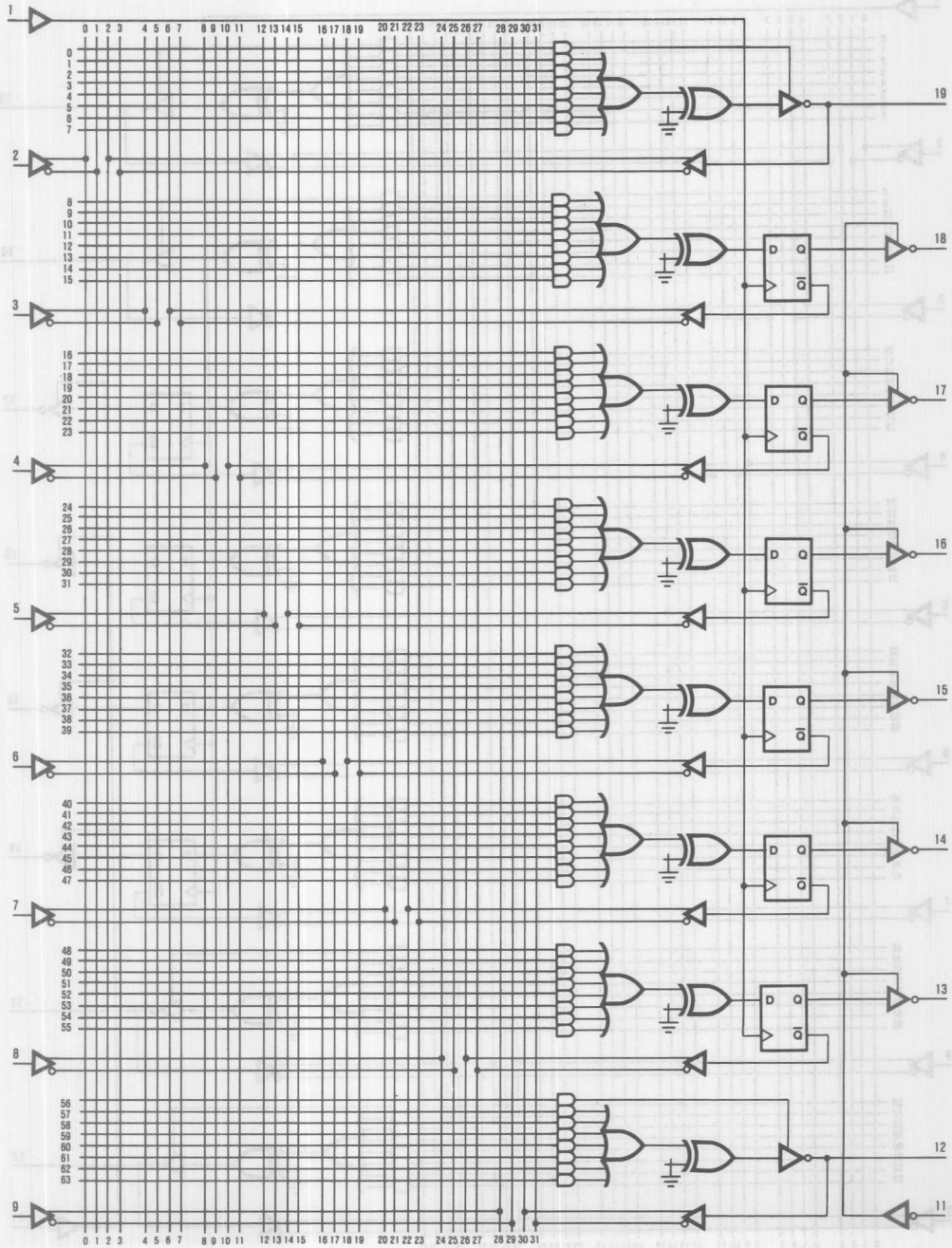
16A4



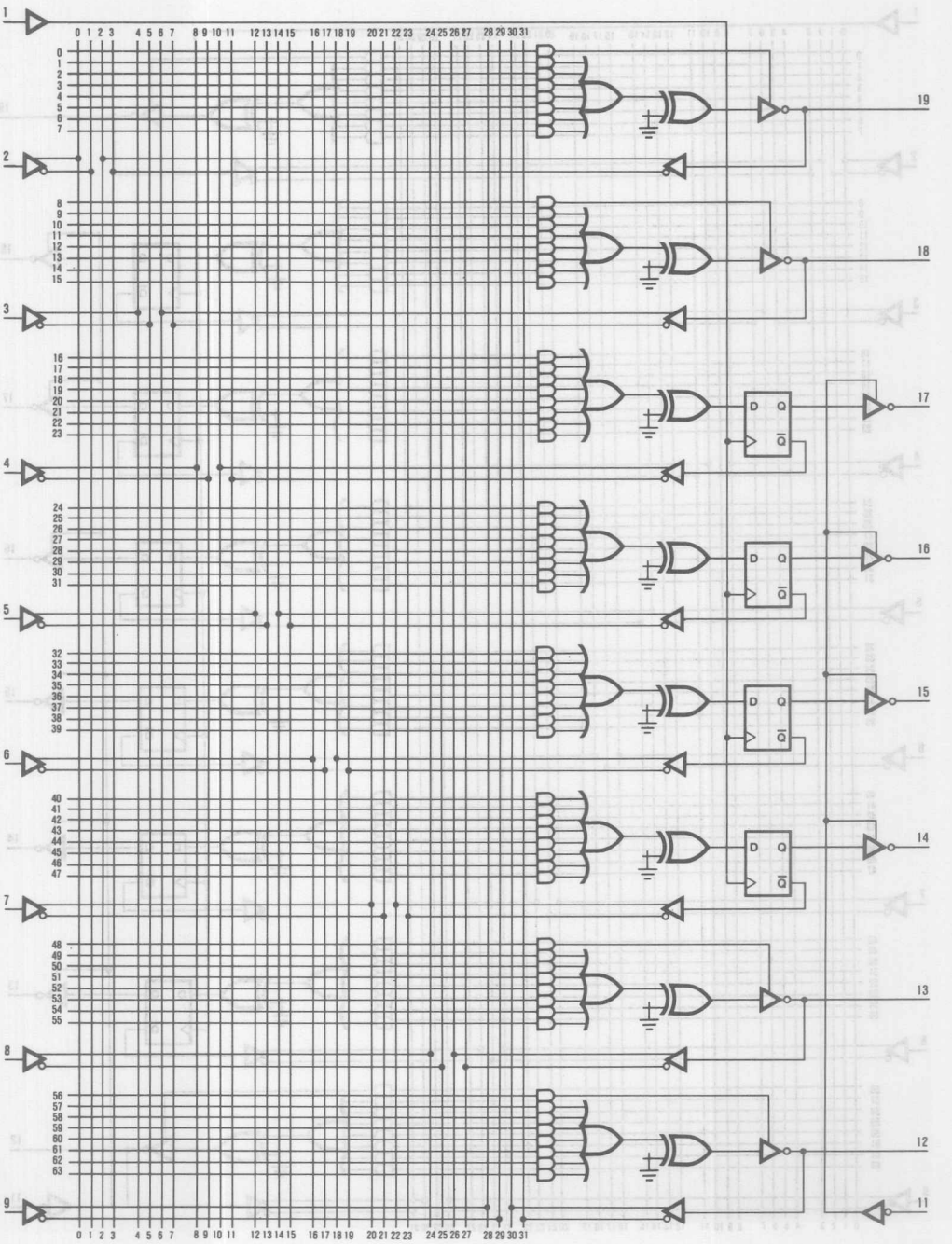
16P8


1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0
1
2
3
4
5
6
7
2
8
9
10
11
12
13
14
15
3
16
17
18
19
20
21
22
23
4
24
25
26
27
28
29
30
31
5
32
33
34
35
36
37
38
39
6
40
41
42
43
44
45
46
47
7
48
49
50
51
52
53
54
55
8
56
57
58
59
60
61
62
63
9
19
18
17
16
15
14
13
12
11

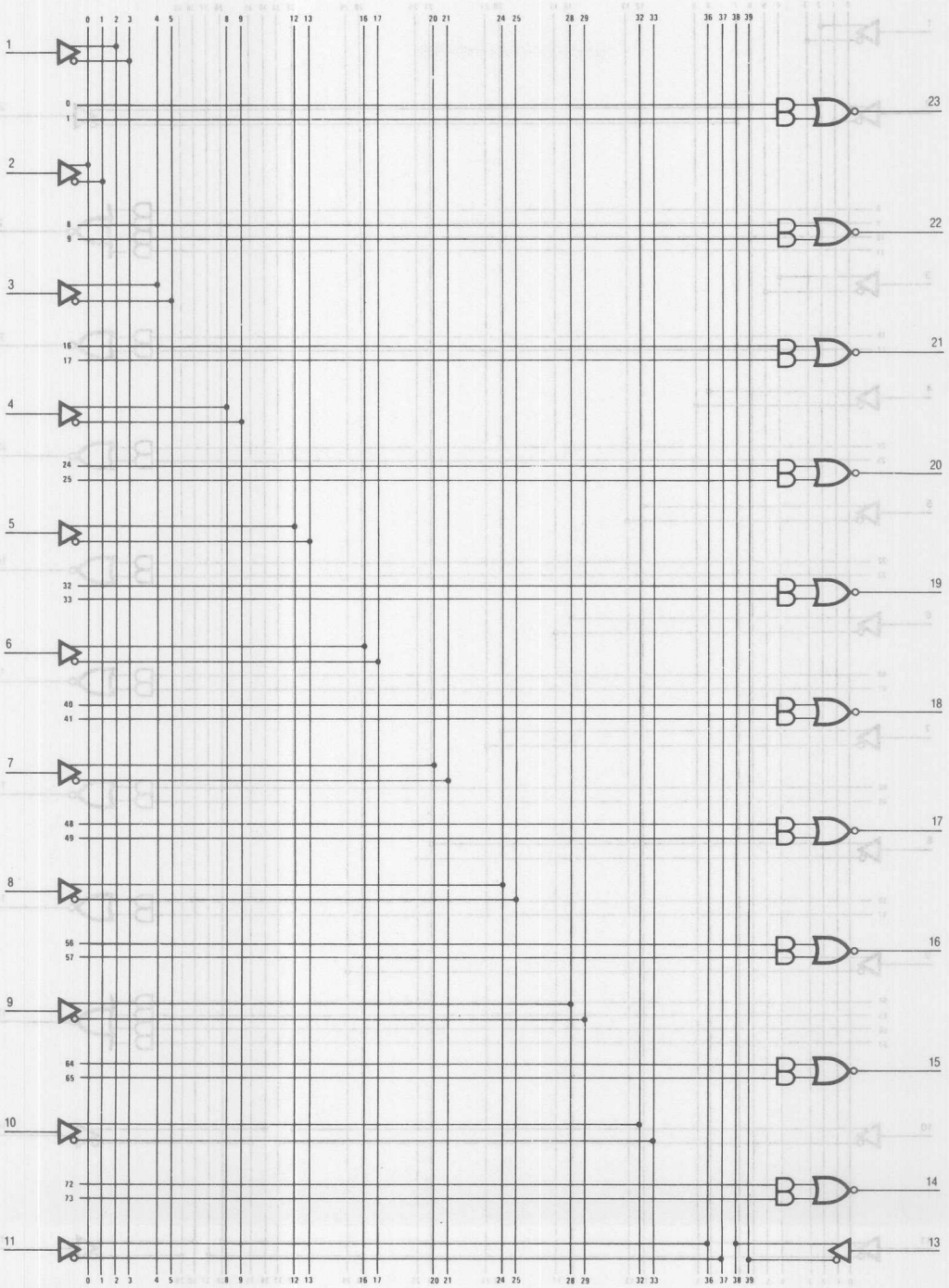

1
2
3
4
5
6
7
8
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
D
Q
Q̄
19
18
17
16
15
14
13
12
11

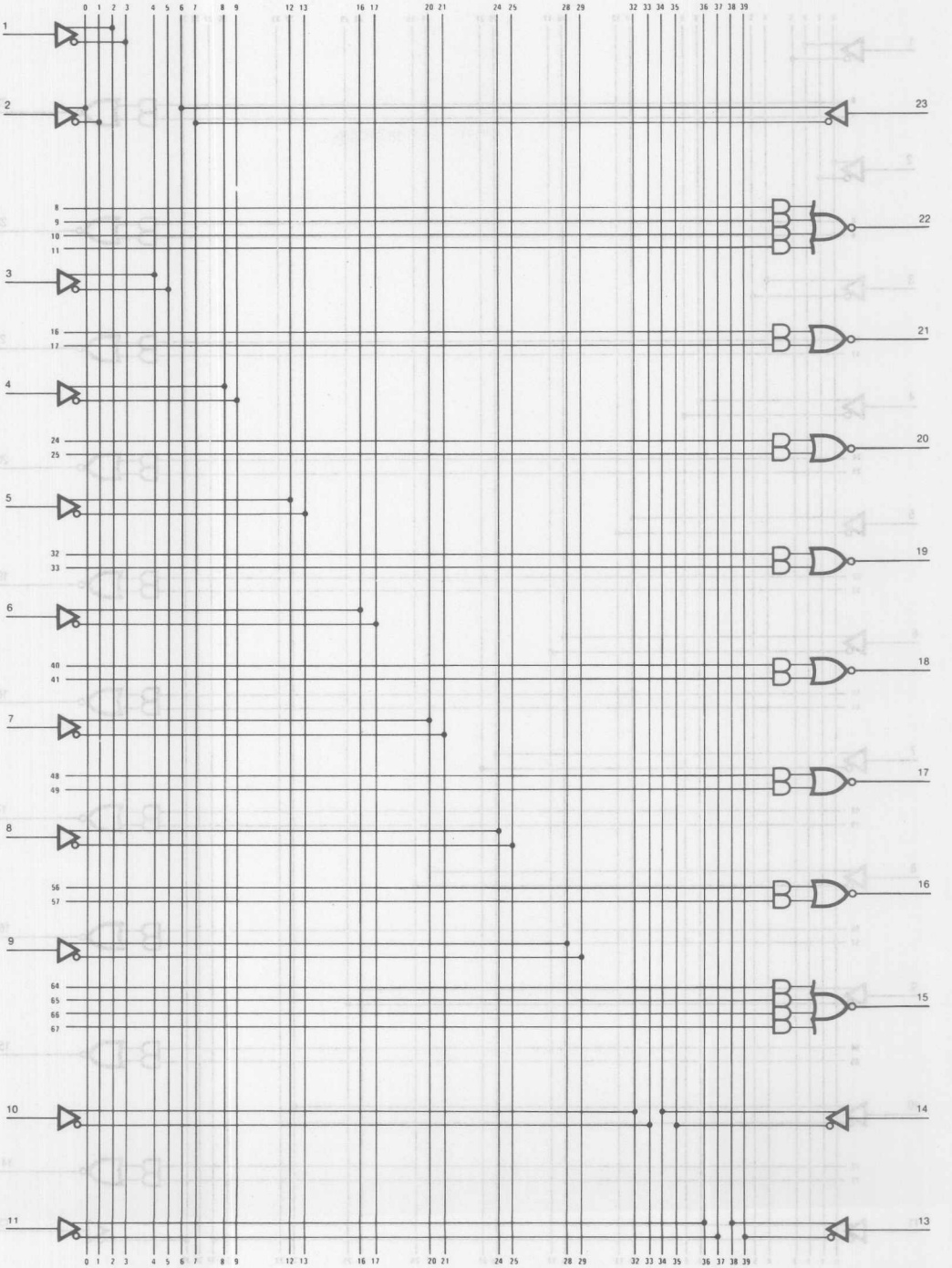

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
D
Q
Q

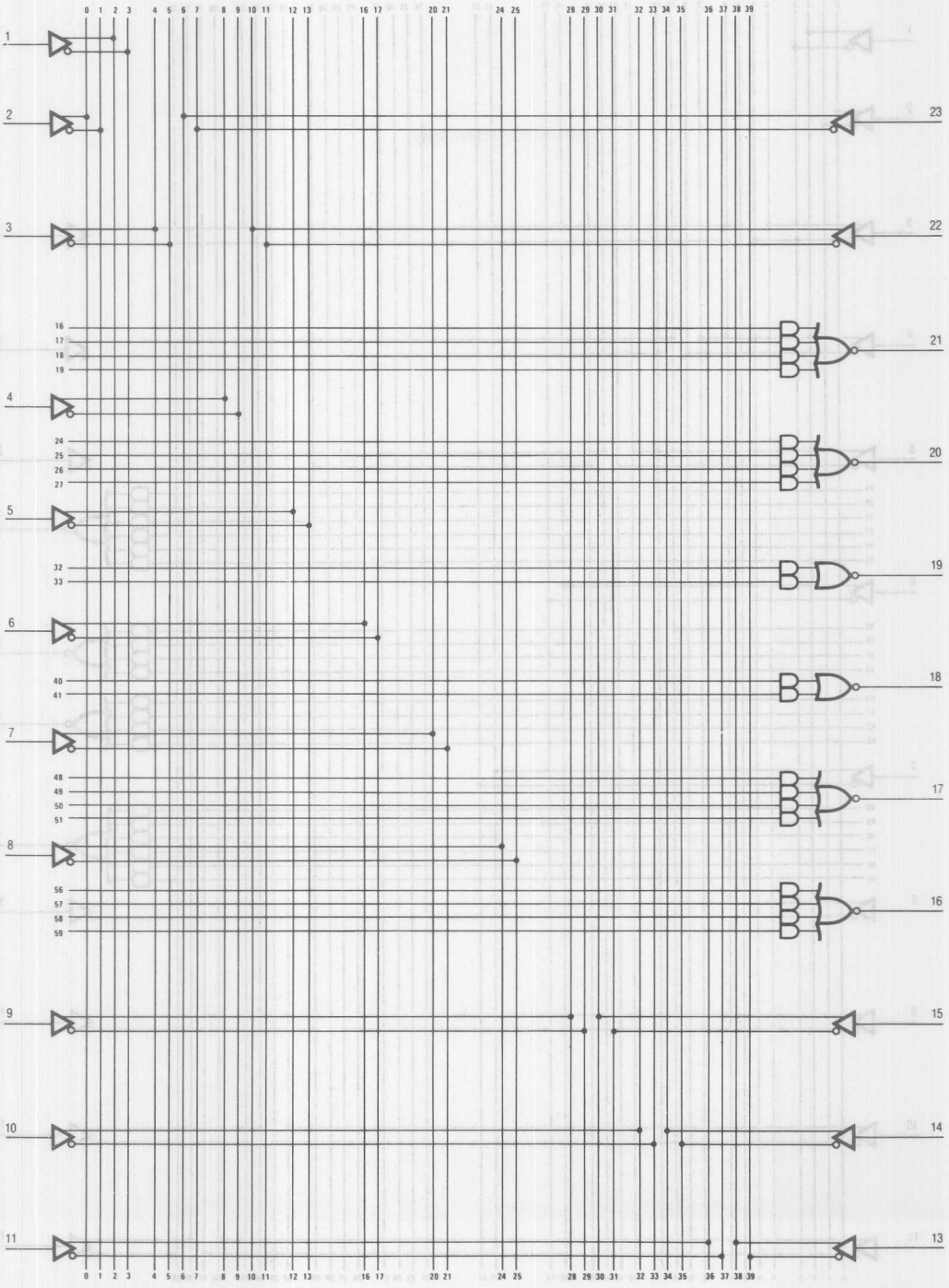
16RP4


1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 1 2 3 4 5 6 7
2
8 9 10 11 12 13 14 15
3
16 17 18 19 20 21 22 23
4
24 25 26 27 28 29 30 31
5
32 33 34 35 36 37 38 39
6
40 41 42 43 44 45 46 47
7
48 49 50 51 52 53 54 55
8
56 57 58 59 60 61 62 63
9
19
18
17
16
15
14
13
12
11
D
Q
Q

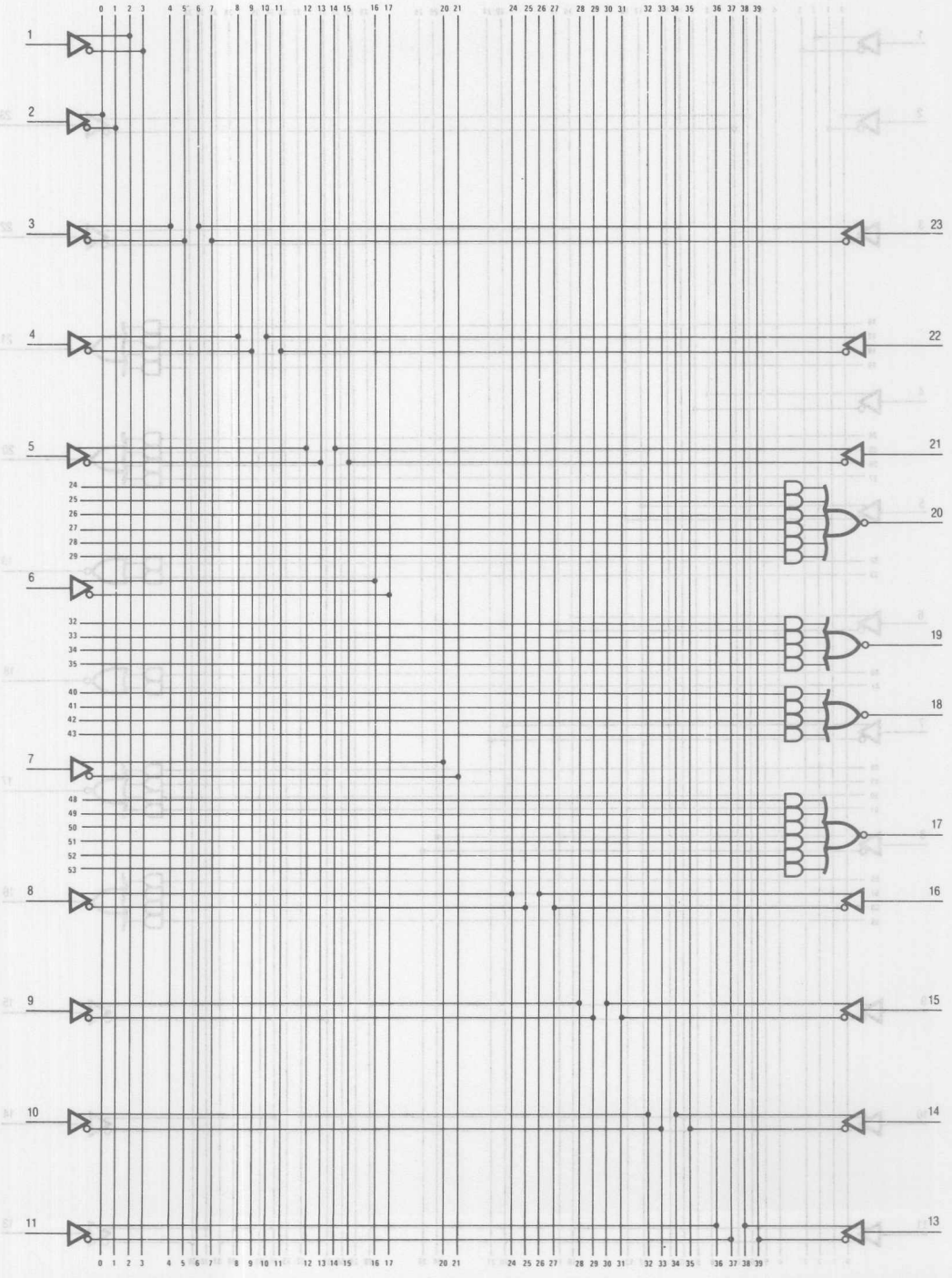
12L10


0 1 2 3 4 5 8 9 12 13 16 17 20 21 24 25 28 29 32 33 36 37 38 39
1
2
3
4
5
6
7
8
9
10
11
0
1
8
9
16
17
24
25
32
33
40
41
48
49
56
57
64
65
72
73
23
22
21
20
19
18
17
16
15
14
13

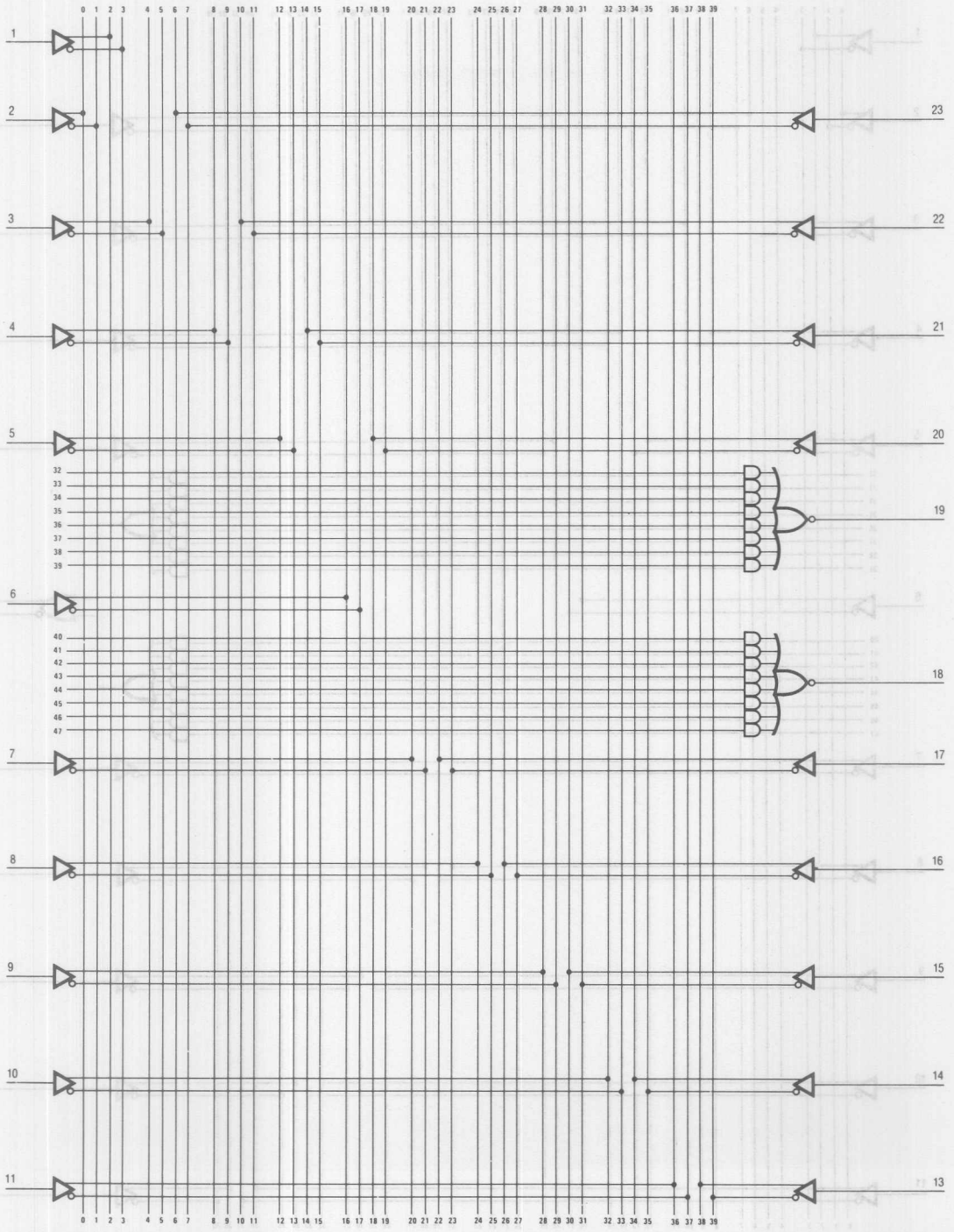

0 1 2 3
4 5 6 7
8 9
12 13
16 17
20 21
24 25
28 29
32 33 34 35
36 37 38 39
1
2
23
8
9
10
11
22
3
16
17
21
4
24
25
20
5
32
33
19
6
40
41
18
7
48
49
17
8
56
57
16
9
64
65
66
67
15
10
14
11
13

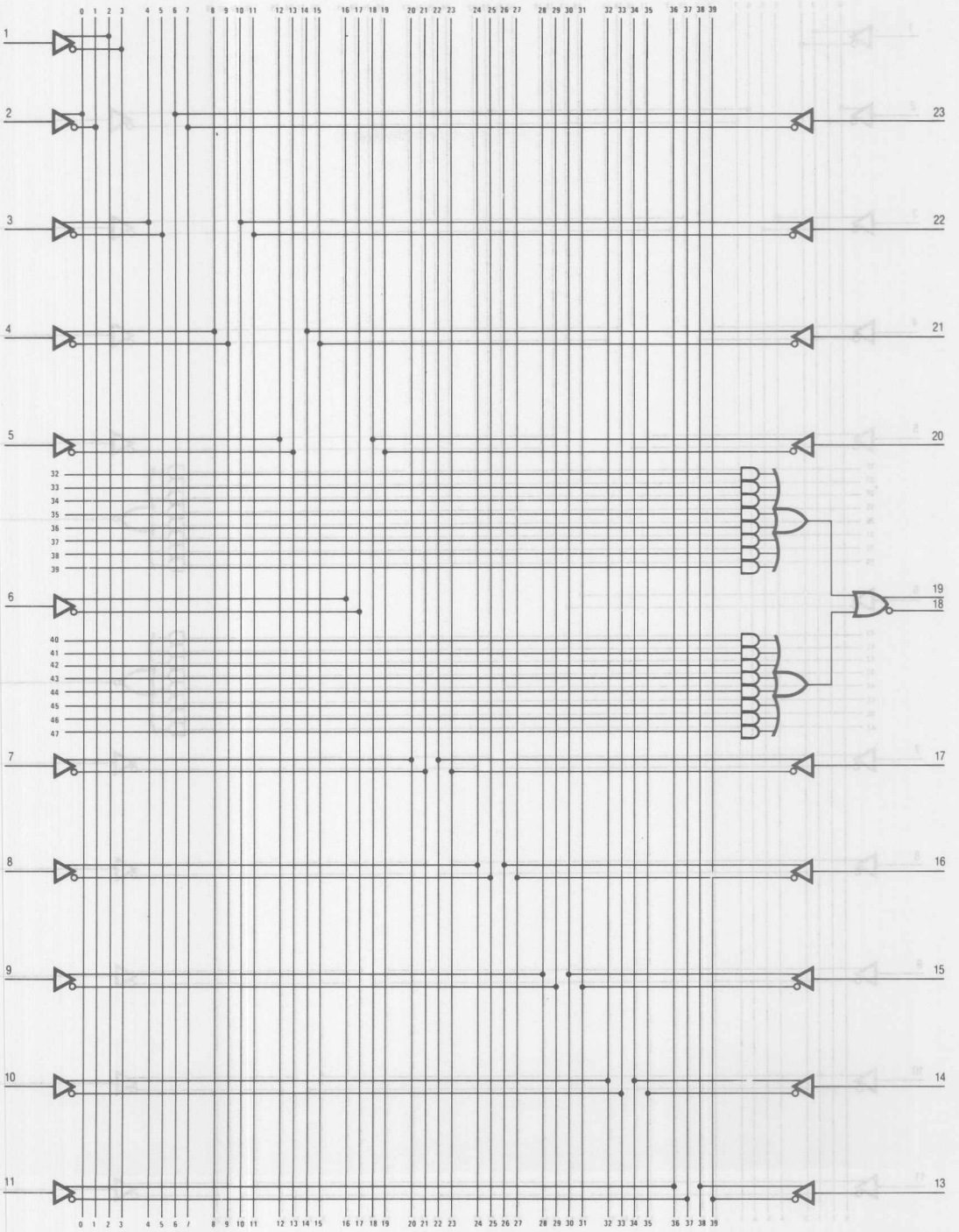

0 1 2 3
4 5 6 7
8 9 10 11
12 13
16 17
20 21
24 25
28 29 30 31
32 33 34 35
36 37 38 39
1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
16 17 18 19
24 25 26 27
32 33
40 41
48 49 50 51
56 57 58 59

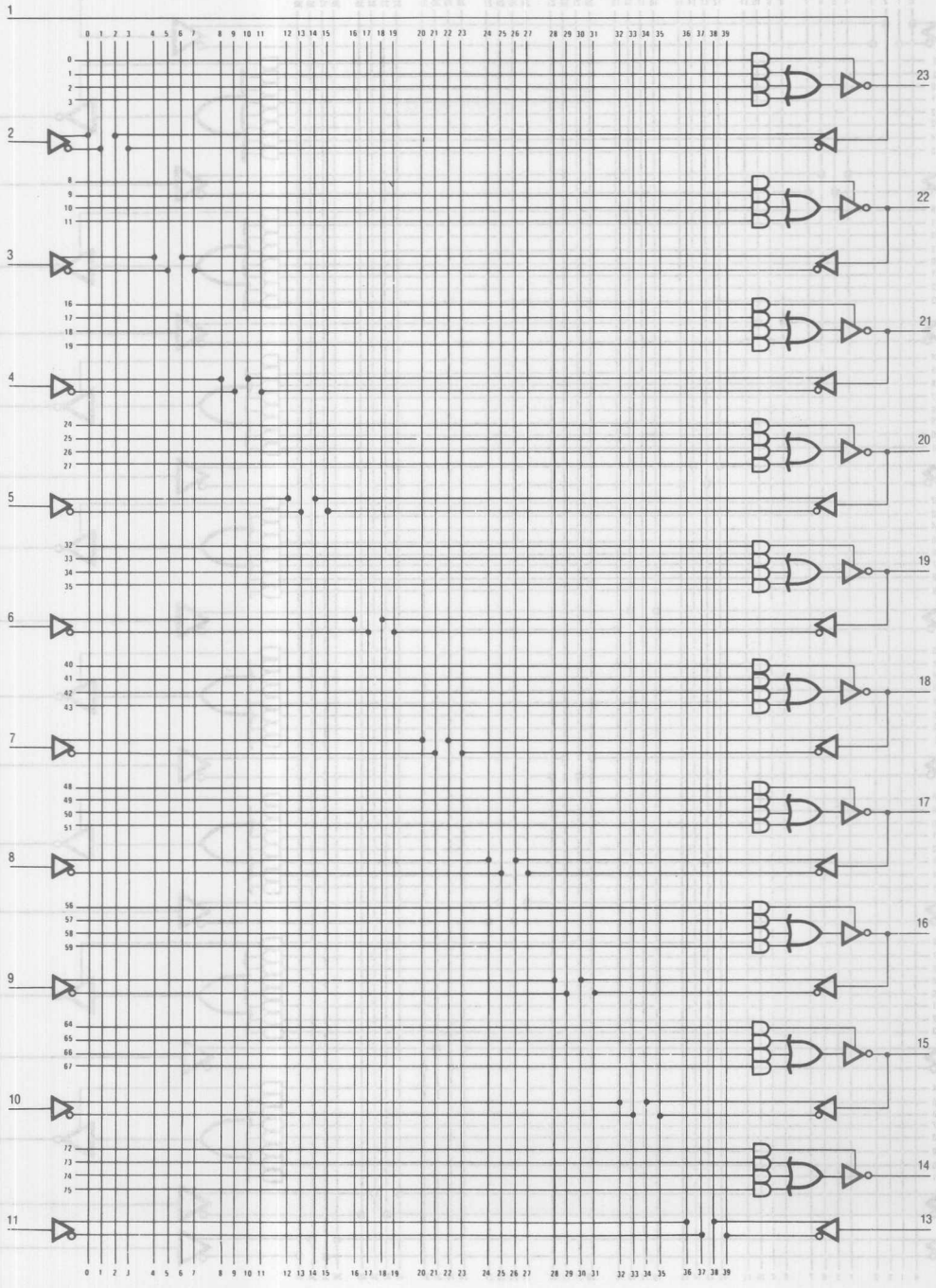
18L4


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 20 21 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
24 25 26 27 28 29
32 33 34 35
40 41 42 43
48 49 50 51 52 53

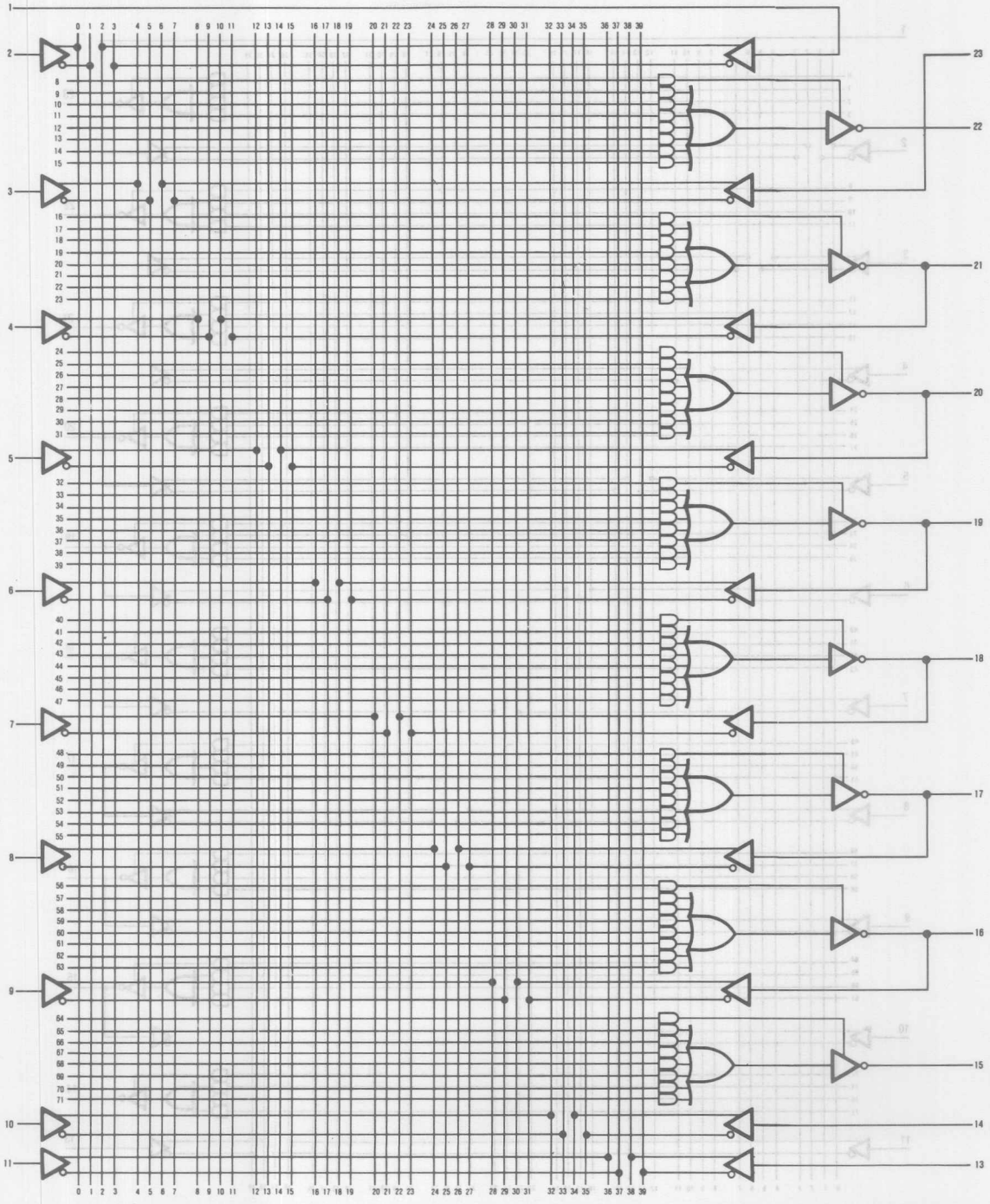
20L2


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47

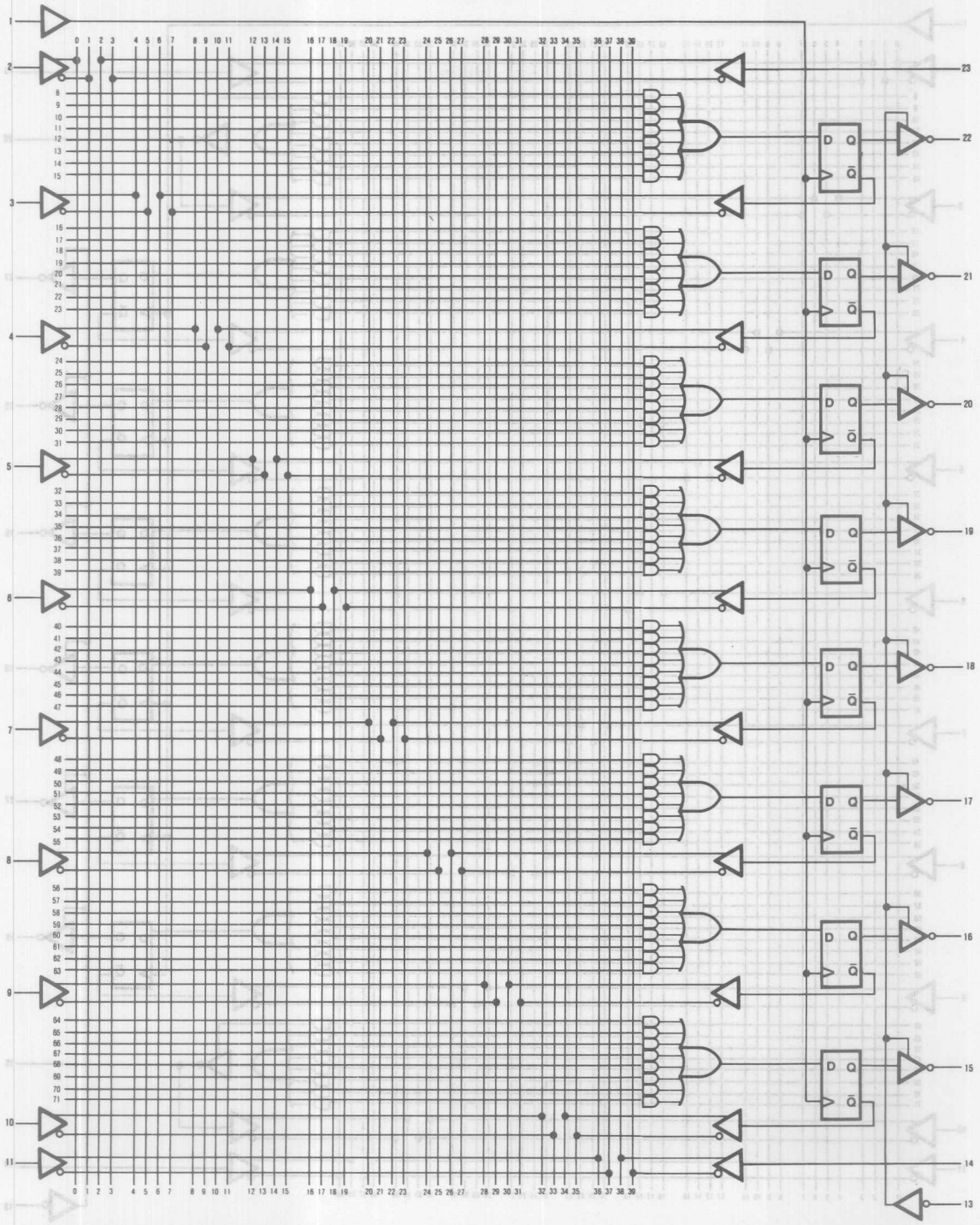

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47

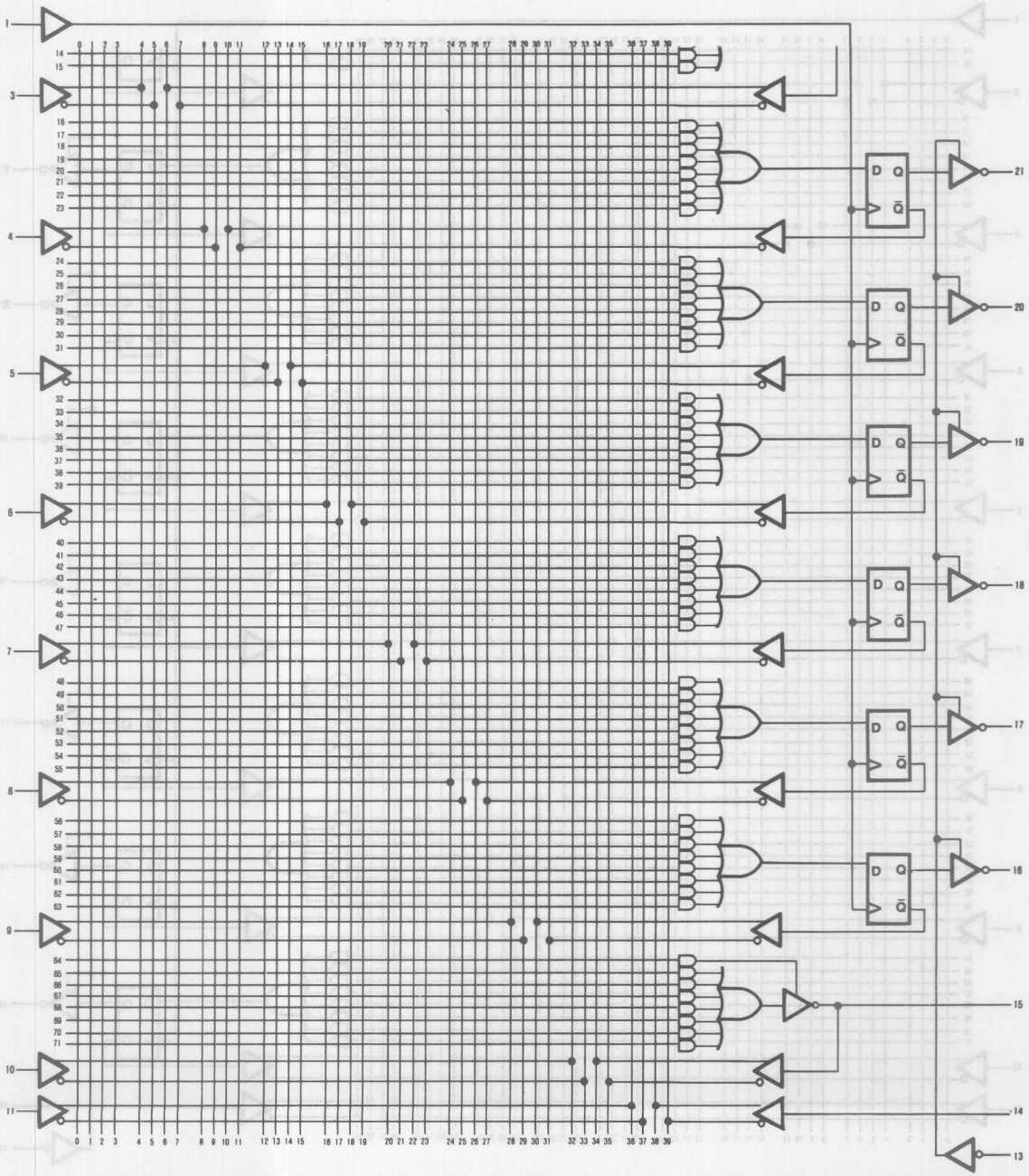

1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39

20L8

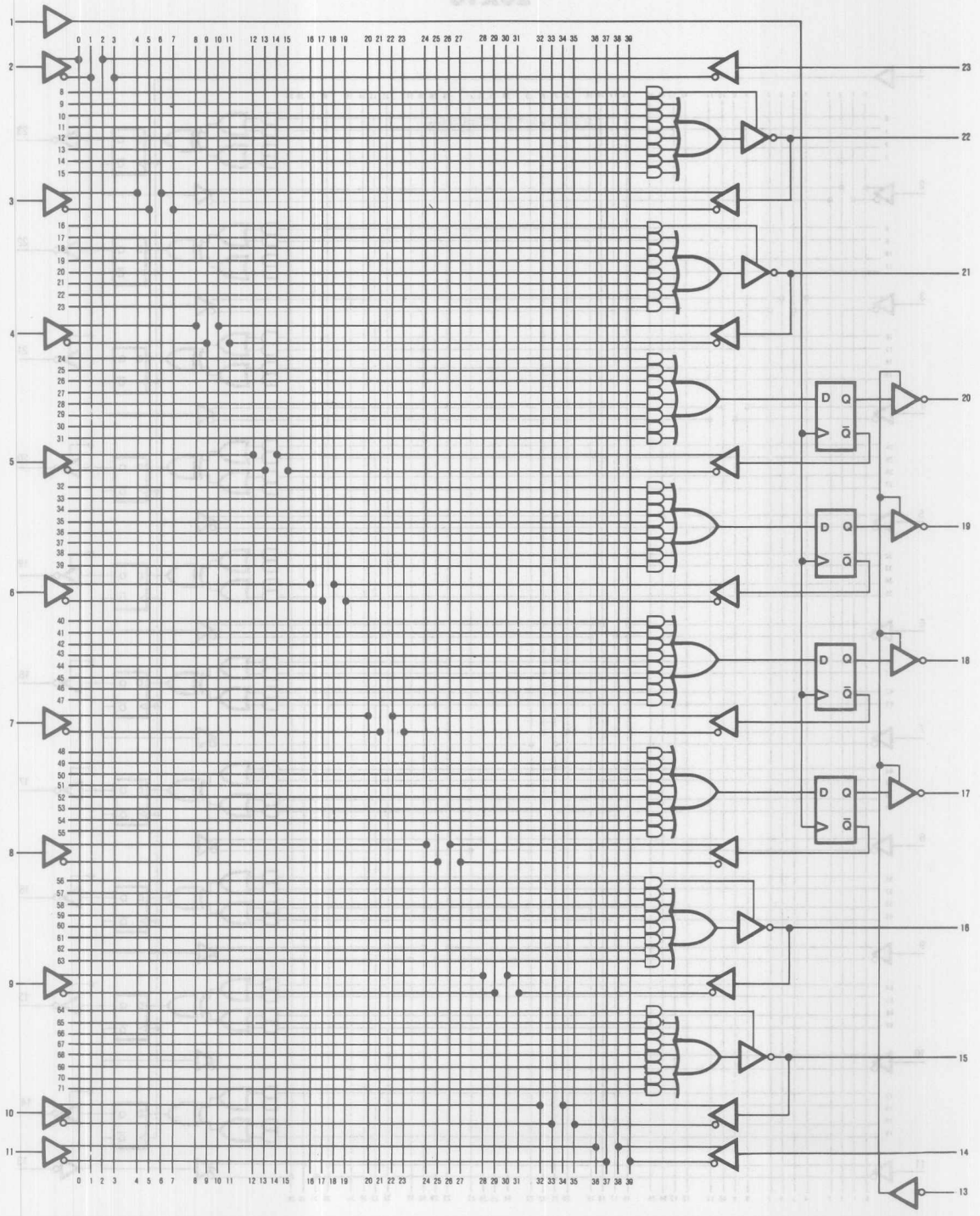

1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71

20R8

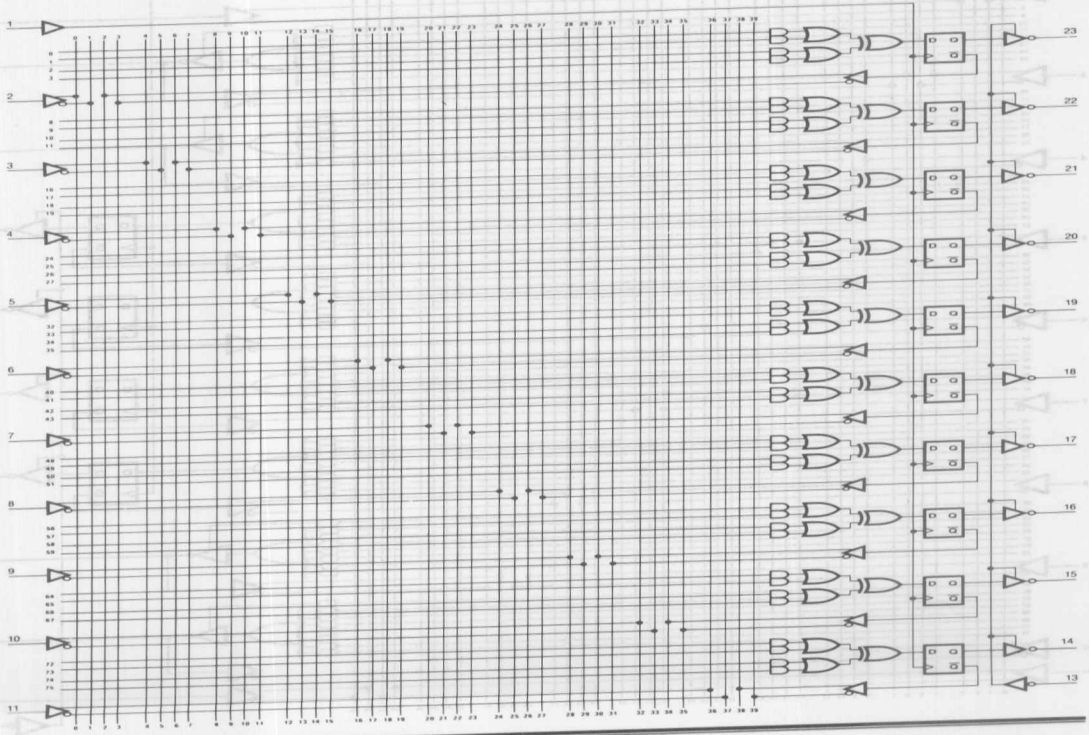



1
3
4
5
6
7
8
9
10
11
21
20
19
18
17
16
15
14
13
D
Q
Q̄

20R4

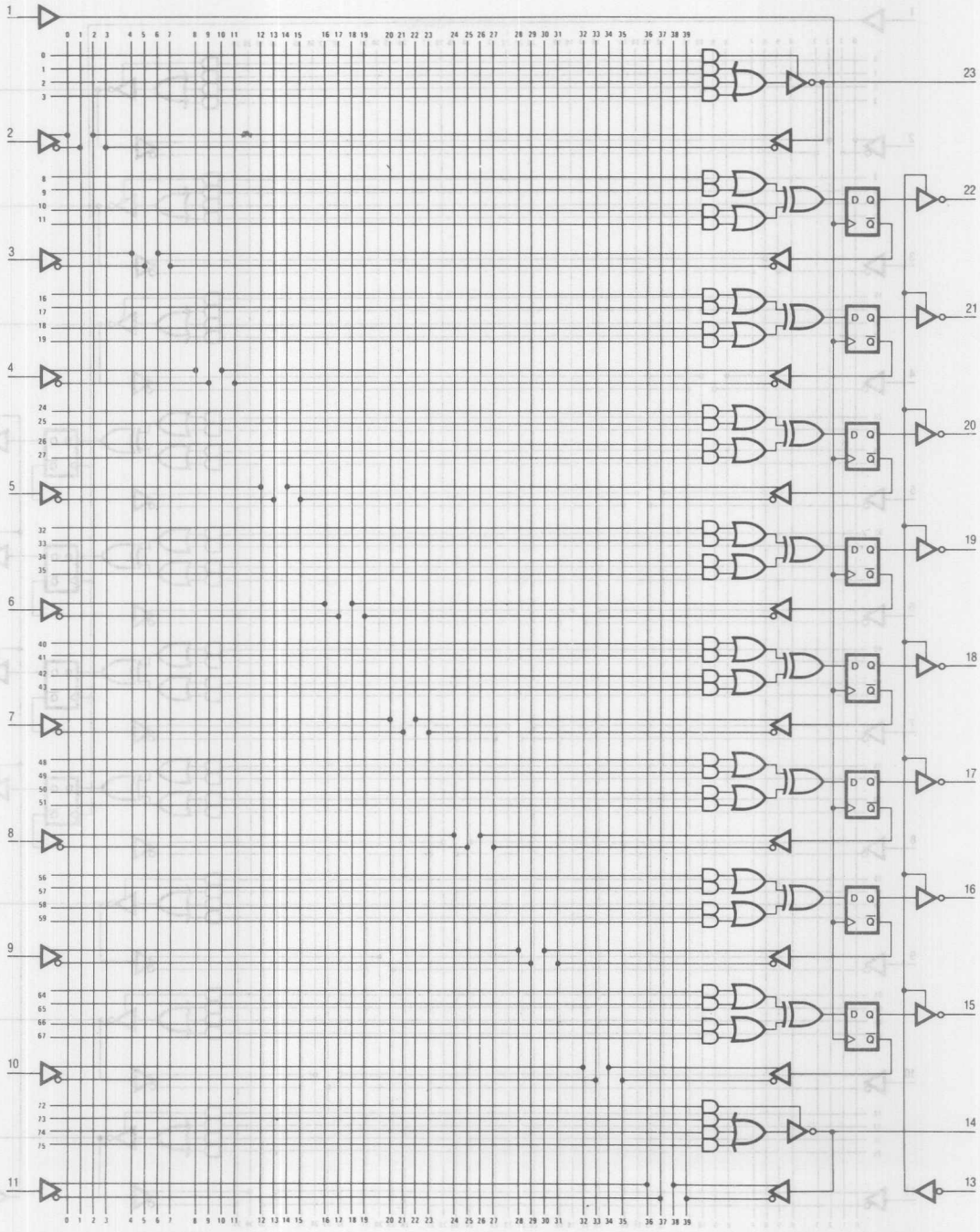


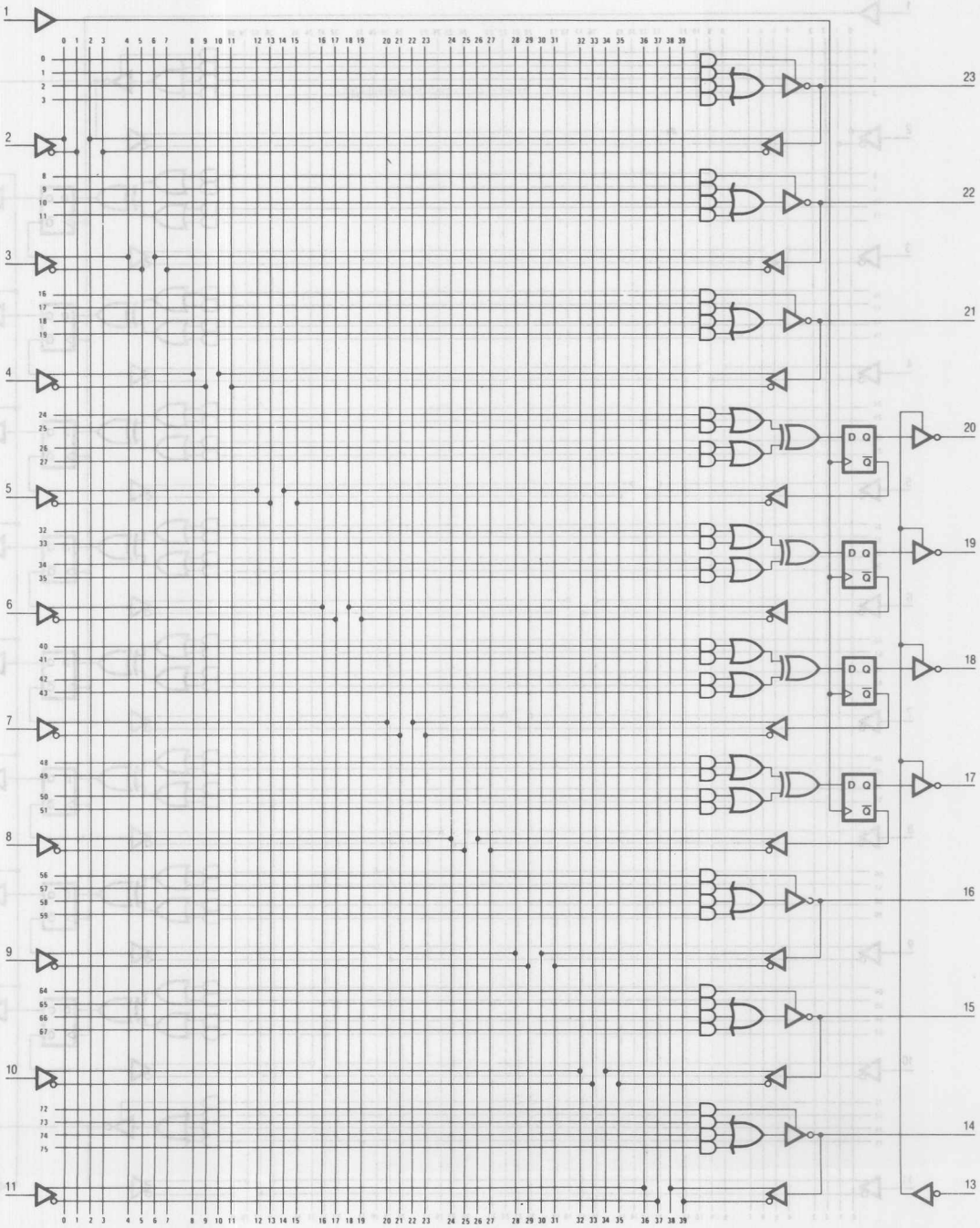
20X10



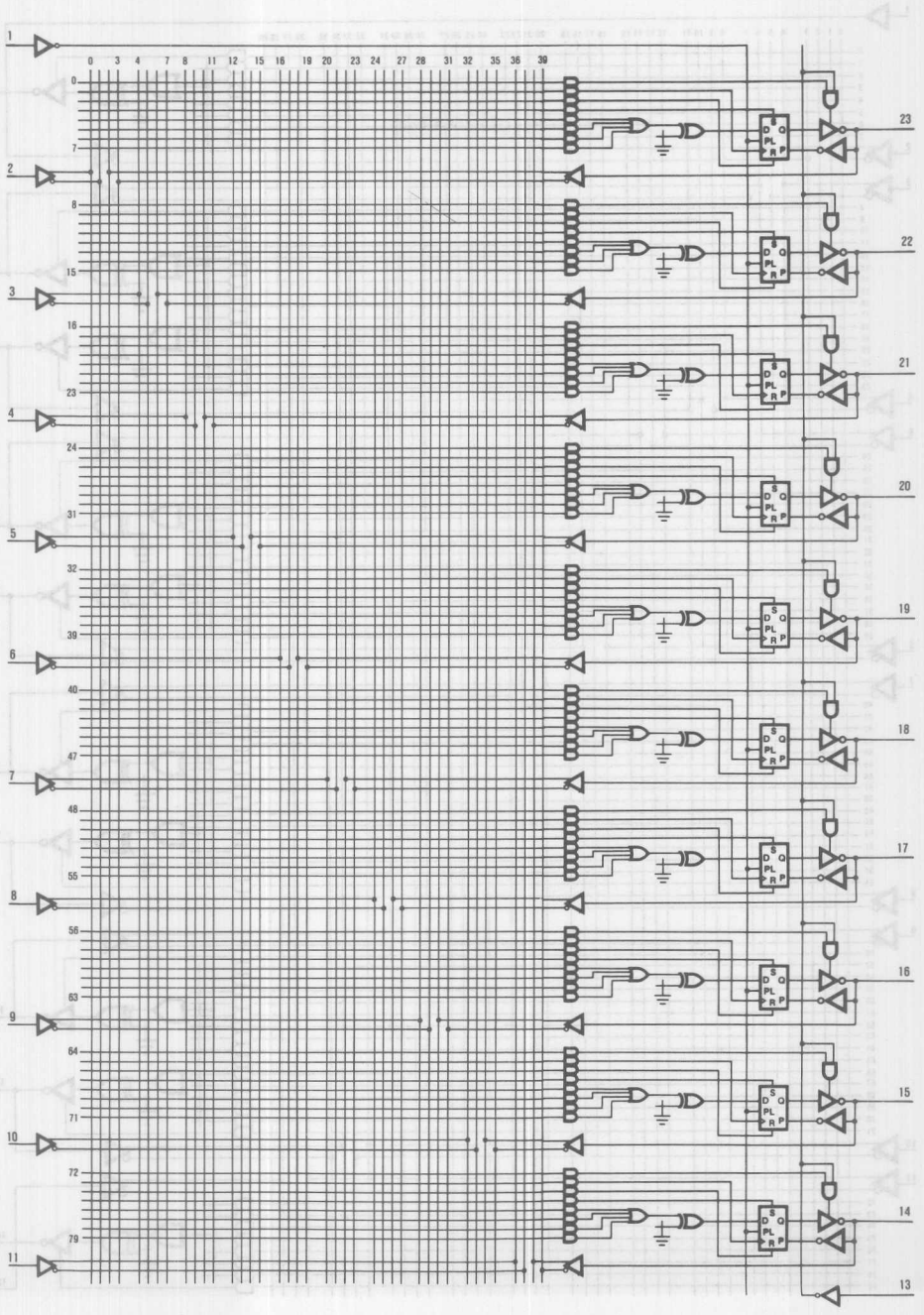
PAL/HAL Device Logic Diagram

20X8

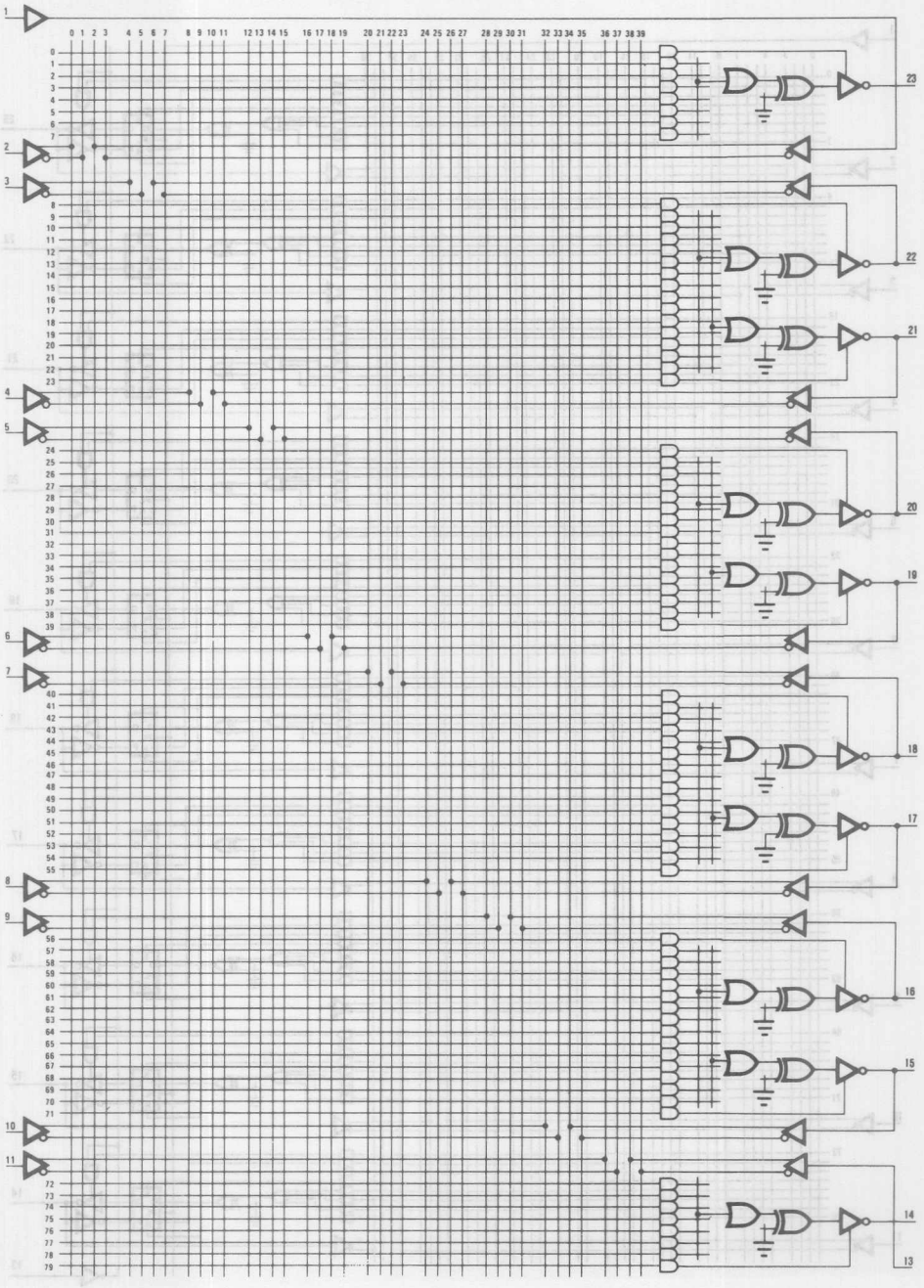

1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39


1
2
3
4
5
6
7
8
9
10
11
23
22
21
20
19
18
17
16
15
14
13

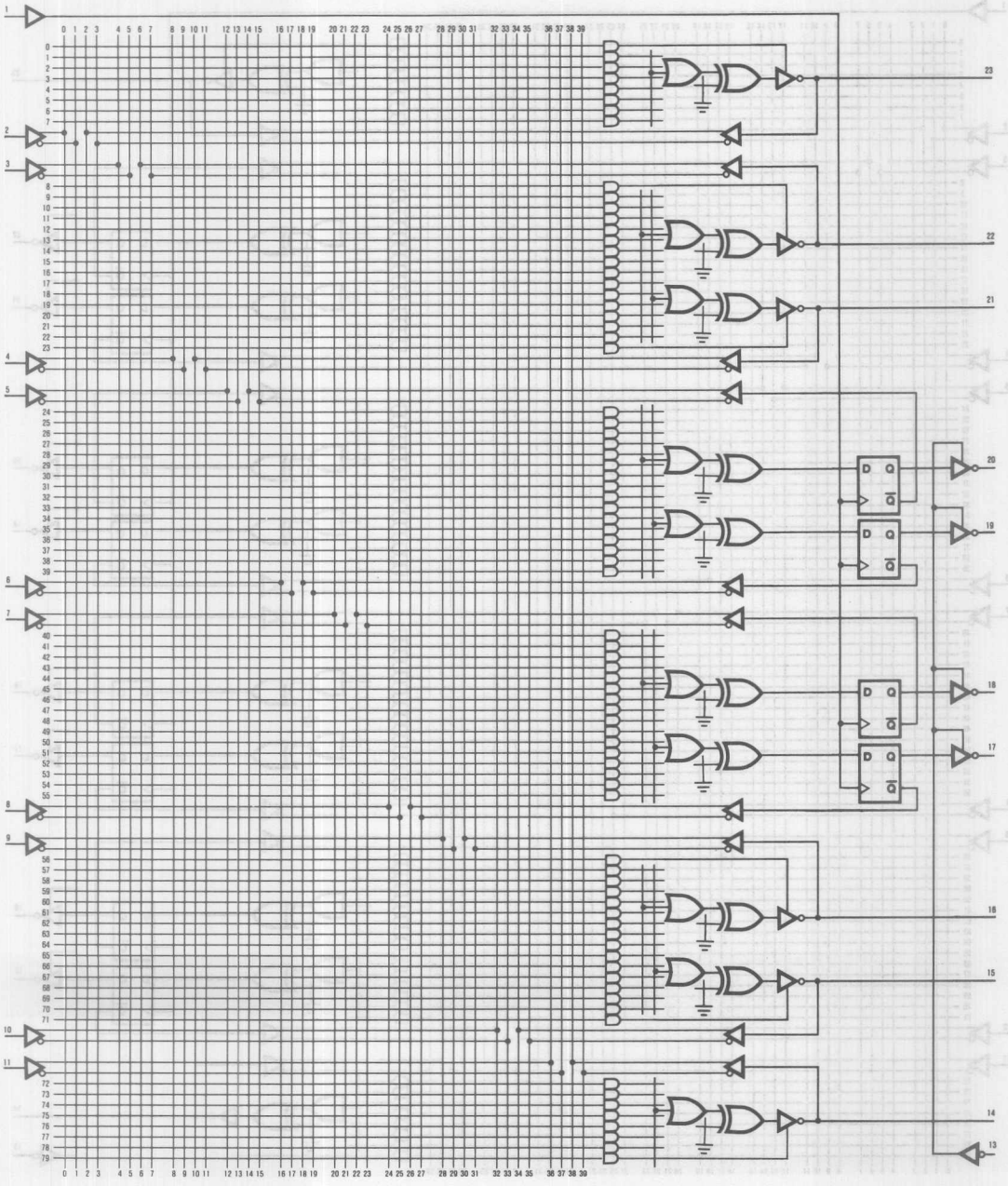
20RA10

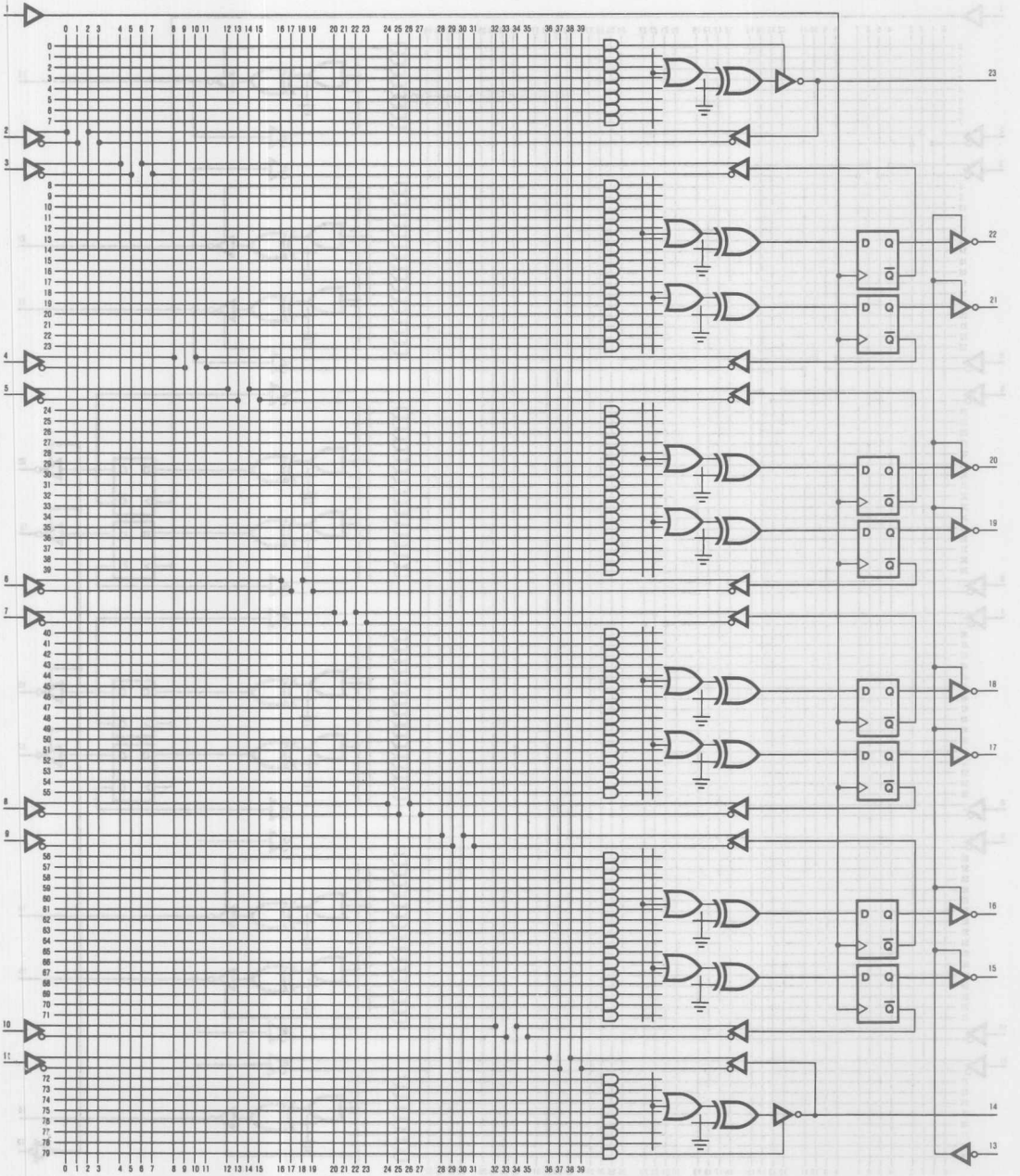

1
2
3
4
5
6
7
8
9
10
11
0 3 4 7 8 11 12 15 16 19 20 23 24 27 28 31 32 35 36 39
0
7
8
15
16
23
24
31
32
39
40
47
48
55
56
63
64
71
72
79
S
D
PL
R
Q
P
23
22
21
20
19
18
17
16
15
14
13

20S10

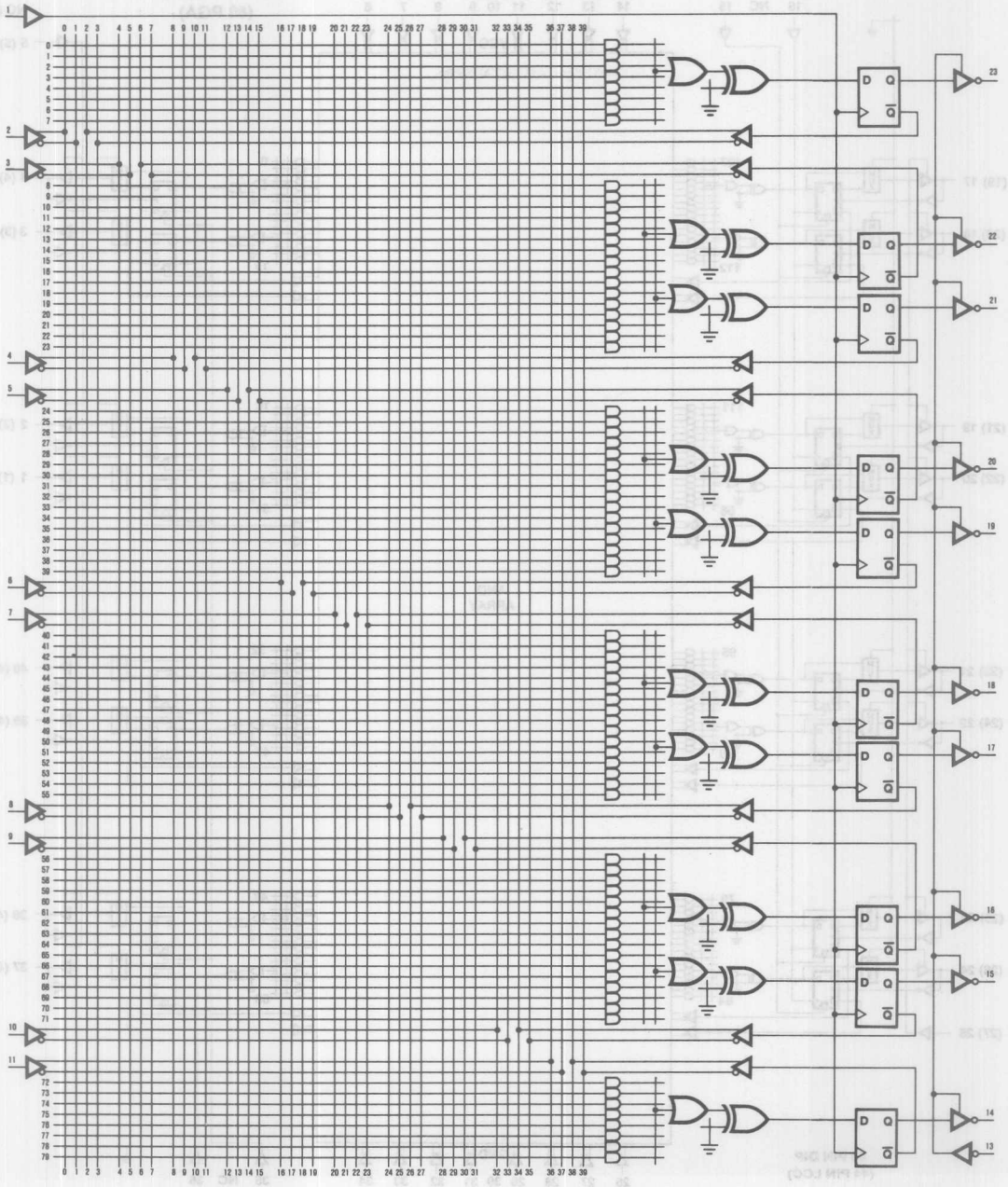


20RS4

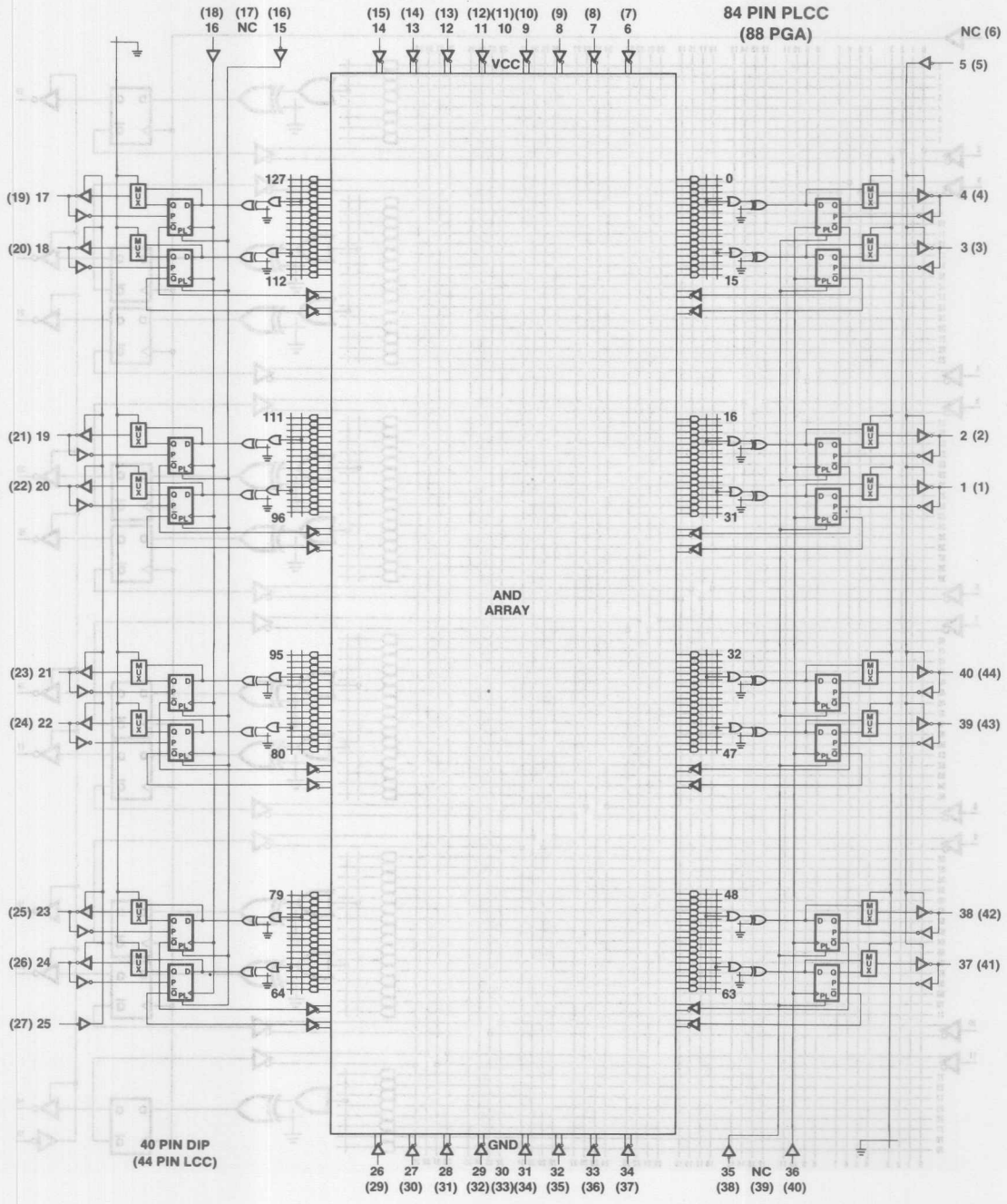




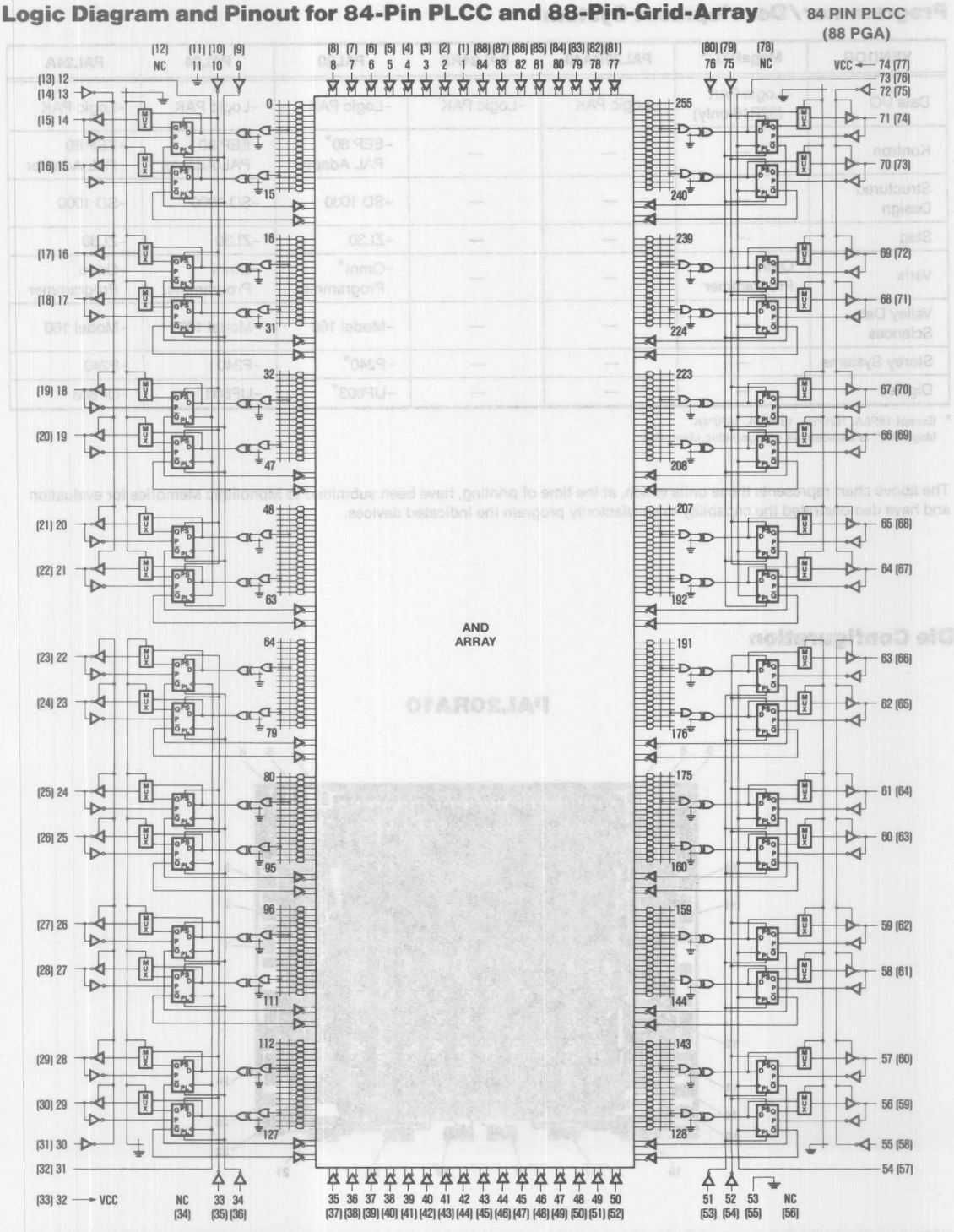
20RS10



Logic Diagram and Pinout for 84-Pin PLCC and 88-Pin-Grid-Array



Logic Diagram and Pinout for 84-Pin PLCC and 88-Pin-Grid-Array



2

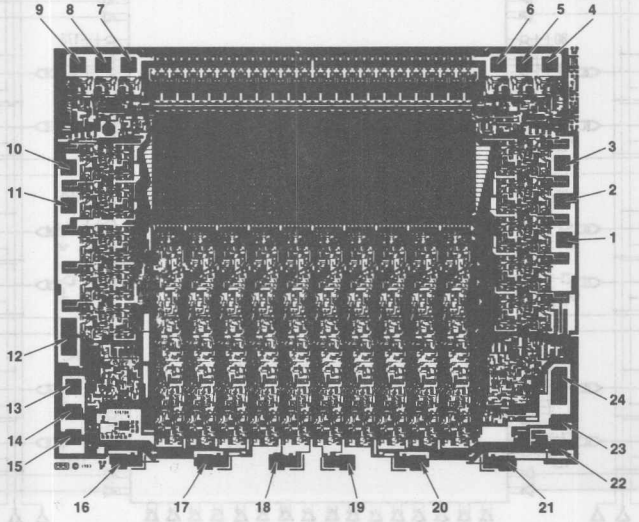
VENDOR	MegaPAL™	PAL20RA10	PAL24RS	PAL20	PAL24	PAL24A
Data I/O	-Logic PAK (32R16 only)	-Logic PAK	-Logic PAK	-Logic PAK	-Logic PAK	-Logic PAK
Kontron	—	—	—	-EEP 80* PAL Adapter	-EEP 80 PAL Adapter	-EEP 80 PAL Adapter
Structured Design	—	—	—	-SD 1000	-SD 1000	-SD 1000
Stag	—	—	—	-ZL30	-ZL30	-ZL30
Varix	Omni Programmer	—	—	-Omni* Programmer	-Omni Programmer	-Omni Programmer
Valley Data Sciences	—	—	—	-Model 160	-Model 160	-Model 160
Storey Systems	—	—	—	-P240*	-P240	-P240
Digelec	—	—	—	-UP803*	-UP803	-UP803

* Except 16P8A, 16RP8A, 16RP6A, 16RP4A
MegaPAL™ is a trademark of Monolithic Memories.

The above chart represents those units which, at the time of printing, have been submitted to Monolithic Memories for evaluation and have demonstrated the capability to satisfactorily program the indicated devices.

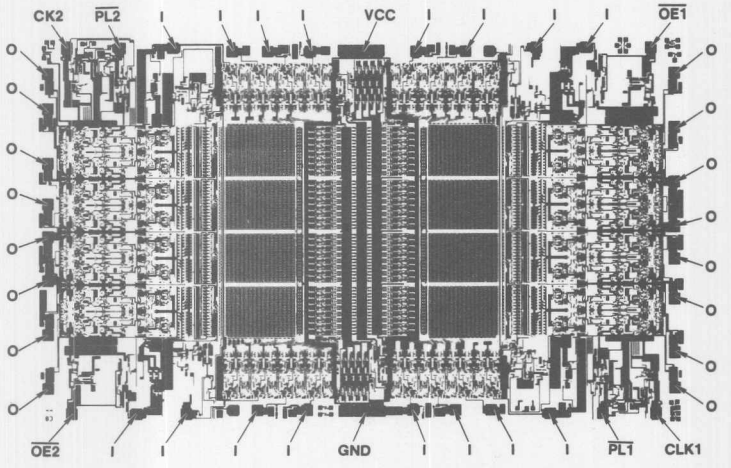
Die Configuration

PAL20RA10

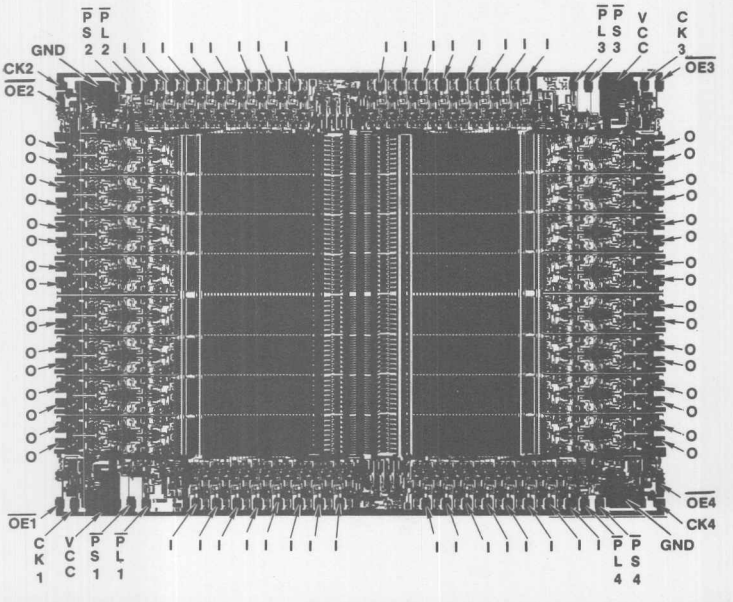


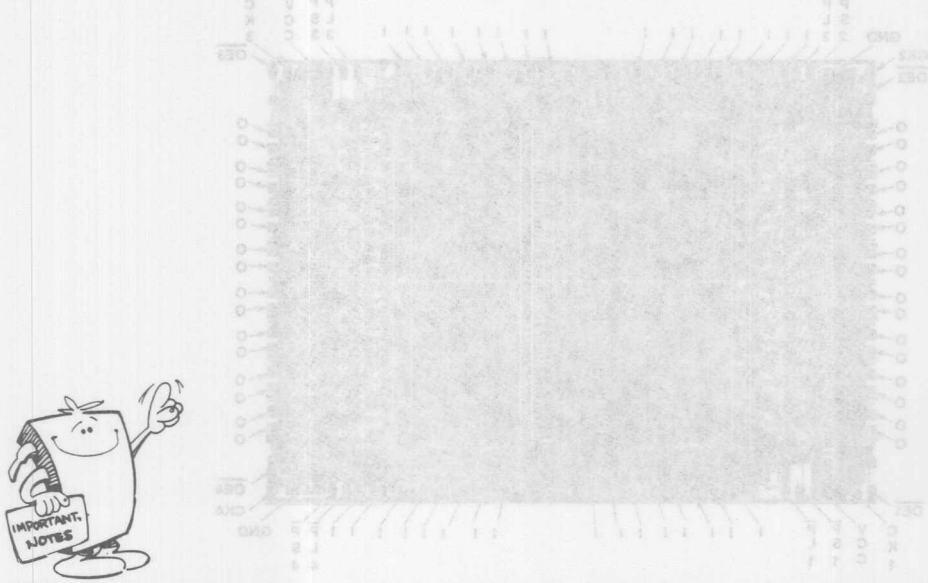
Die Configuration

PAL32R16

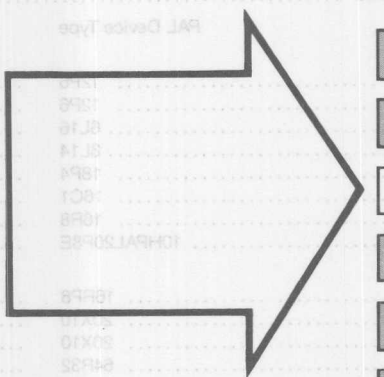


PAL64R32





IMPORTANT
NOTES



Contents Section 3

A WORK SESSION USING PALASM2 MENU

Given below is an example of how to use the PALASM2 Menu to Assemble and Simulate your Pal Design Specification(PDS). The PDS file name appears after the title "work file" and its related extensions after the title "Ext". Each Menu command has a highlighted character and is in upper case. So on pressing a highlighted character, the corresponding Menu command is executed.

In the following example, we assume the user already has created the PDS file, BGATESP.PDS and called the Menu.

```
Monolithic  MMI Memories
work file:  Ext:

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand
```

ENTER MENU COMMAND > W

```
Monolithic  MMI Memories
work file:  Ext:

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand
```

ENTER WORKFILE > BGATESP.PDS

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand
```

ENTER MENU COMMAND > L

```
BGATESP.PDS
```

ENTER A HIGHLIGHTED CHARACTER > P

```
TITLE       Basic Gates (Positive Logic)
PATTERN     BGates.pds
REVISION    B
AUTHOR      A G Gilbert
COMPANY     Monolithic Memories Inc., Santa Clara, CA
DATE        1/7/85

;PALs which feature programable output polarity such as
;the PAL12P6 in this example offer the designer superior
;flexability compared to previous PALS.The subtle advantage
;of the output polarity fuse is that it allows logic to
;be implemented in either positive or negative form within
;the same PAL.This means that the number of product terms
;required to realize an output may be reduced by optimal
;choise of positive or negative logic. If Karnough Maps
;are used by the designer to simplify the logic equation
;for an output, either clumps of 1's or 0's may be circled
;to reduce the equation.

CHIP  BASIC_GATES PAL12P6

C D F G M N P Q I GND
J K L R S H E B A VCC

-- More --

EQUATIONS

B = /A                  ;NOT FUNCTION   A : B
                        ;               -------
                        ;               0 : 1 *
                        ;               1 : 0

E = C*D                 ;AND FUNCTION   C   D : E
                        ;               -----------
                        ;               0   0 : 0
                        ;               0   1 : 0
                        ;               1   0 : 0
                        ;               1   1 : 1 *

H = F+G                 ;OR FUNCTION    F   G : H
                        ;               -----------
                        ;               0   0 : 0
                        ;               0   1 : 1 *
                        ;               1   0 : 1 *
                        ;               1   1 : 1 *

L = /J+/K               ;NAND FUNCTION  J   K : L
                        ;               -----------
                        ;               0   0 : 1 *
                        ;               0   1 : 1 *
                        ;               1   0 : 1 *
                        ;               1   1 : 0

S = /M*/N               ;NOR FUNCTION   M   N : S
                        ;               -----------
                        ;               0   0 : 1 *
                        ;               0   1 : 0
                        ;               1   0 : 0
                        ;               1   1 : 0

R = P*/Q + /P*Q         ;XOR FUNCTION   P   Q : R
                        ;               -----------
                        ;               0   0 : 0
                        ;               0   1 : 1 *
                        ;               1   0 : 1 *
                        ;               1   1 : 0

SIMULATION

-- More --

TRACE_ON A B C D E F G H J K L M N S P Q R
SETF /A /C /D /F /G /J /K /M /N /P /Q
SETF A /C D /F G /J K /M N /P Q
SETF C /D F /G J /K M /N P /Q
SETF C D F G J K M N P Q

;This simulation exercises the Basic Gates EQUATION section
;by evaluating the output for all input combinations.


Strike a key when ready . . .
```

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > A
```

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS,XPT,JED

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > L
```

```
          ******************
          PALASM Version 2.0
          ******************
...
Please wait, PALASM2 Syntax Checking.

PALASM FRONTEND, V2.06 - BETA RELEASE
(C) - COPYRIGHT MONOLITHIC MEMORIES INC, 1984

Source file name [default: specs.dat] :BGATESP.PDS
Processing BGATESP.PDS
Create error file  [default: No] ?

No error file created

Echo Palasm Design File to terminal [default: No] ?
...
Please wait, XPLOT & JEDEC File Generation.

PALASM XPLOT, V2.06 - BETA RELEASE
(C) - COPYRIGHT MONOLITHIC MEMORIES INC., 1984
```

```
BGATESP.PDS
BGATESP.XPT
BGATESP.JED

ENTER A HIGHLIGHTED CHARACTER > X
```

```
Equation being processed is for output==>> B
Equation being processed is for output==>> E
Equation being processed is for output==>> H
Equation being processed is for output==>> L
Equation being processed is for output==>> S
Equation being processed is for output==>> R

The fuseplot is stored in ===>BGATESP.xpt
The jedec is stored in    ===>BGATESP.jed
All done!
Strike a key when ready . . .
```

```
PALASM XPLOT, V2.06 - BETA RELEASE
(C) - COPYRIGHT MONLITHIC MEMORIES INC., 1984

Title     :  Basic Gates (Positive)
Pattern   :  BGates.pds
Revision  :  B
Author    :  A G Gilbert
Company   :  Monolithic Memories Inc., Santa Clara, CA
Date      :  1/7/85
```

```
PAL12P6
BASIC_GATES

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

  0 0000 0000 0000 0000 0000 0000 0000 0000
  1 0000 0000 0000 0000 0000 0000 0000 0000
  2 0000 0000 0000 0000 0000 0000 0000 0000
  3 0000 0000 0000 0000 0000 0000 0000 0000
  4 0000 0000 0000 0000 0000 0000 0000 0000
  5 0000 0000 0000 0000 0000 0000 0000 0000
  6 0000 0000 0000 0000 0000 0000 0000 0000
  7 0000 0000 0000 0000 0000 0000 0000 0000
-- More --

  8 ---- ---X --00 --00 --00 --00 ---- ----
  9 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 10 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 11 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 12 0000 0000 0000 0000 0000 0000 0000 0000
 13 0000 0000 0000 0000 0000 0000 0000 0000
 14 0000 0000 0000 0000 0000 0000 0000 0000
 15 0000 0000 0000 0000 0000 0000 0000 0000

 16 X-X- ---- --00 --00 --00 --00 ---- ----
 17 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 18 0000 0000 0000 0000 0000 0000 0000 0000
 19 0000 0000 0000 0000 0000 0000 0000 0000
 20 0000 0000 0000 0000 0000 0000 0000 0000
 21 0000 0000 0000 0000 0000 0000 0000 0000
 22 0000 0000 0000 0000 0000 0000 0000 0000
 23 0000 0000 0000 0000 0000 0000 0000 0000

 24 ---- X--- --00 --00 --00 --00 ---- ----
 25 ---- ---- X-00 --00 --00 --00 ---- ----
 26 0000 0000 0000 0000 0000 0000 0000 0000
 27 0000 0000 0000 0000 0000 0000 0000 0000
-- More --

 28 0000 0000 0000 0000 0000 0000 0000 0000
 29 0000 0000 0000 0000 0000 0000 0000 0000
 30 0000 0000 0000 0000 0000 0000 0000 0000
 31 0000 0000 0000 0000 0000 0000 0000 0000

 32 ---- ---- --00 -X00 -X00 --00 ---- ----
 33 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 34 0000 0000 0000 0000 0000 0000 0000 0000
 35 0000 0000 0000 0000 0000 0000 0000 0000
 36 0000 0000 0000 0000 0000 0000 0000 0000
 37 0000 0000 0000 0000 0000 0000 0000 0000
 38 0000 0000 0000 0000 0000 0000 0000 0000
 39 0000 0000 0000 0000 0000 0000 0000 0000

 40 ---- ---- --00 --00 --00 X-00 -X-- ----
 41 ---- ---- --00 --00 --00 -X00 X--- ----
 42 0000 0000 0000 0000 0000 0000 0000 0000
 43 0000 0000 0000 0000 0000 0000 0000 0000
 44 0000 0000 0000 0000 0000 0000 0000 0000
 45 0000 0000 0000 0000 0000 0000 0000 0000
 46 0000 0000 0000 0000 0000 0000 0000 0000
 47 0000 0000 0000 0000 0000 0000 0000 0000
 -- More --

 48 ---- ---- --00 --00 --00 --00 ---- ---X
 49 ---- ---- --00 --00 --00 --00 ---X ----
 50 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 51 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 52 0000 0000 0000 0000 0000 0000 0000 0000
 53 0000 0000 0000 0000 0000 0000 0000 0000
 54 0000 0000 0000 0000 0000 0000 0000 0000
 55 0000 0000 0000 0000 0000 0000 0000 0000

 56 0000 0000 0000 0000 0000 0000 0000 0000
 57 0000 0000 0000 0000 0000 0000 0000 0000
 58 0000 0000 0000 0000 0000 0000 0000 0000
 59 0000 0000 0000 0000 0000 0000 0000 0000
 60 0000 0000 0000 0000 0000 0000 0000 0000
 61 0000 0000 0000 0000 0000 0000 0000 0000
 62 0000 0000 0000 0000 0000 0000 0000 0000
 63 0000 0000 0000 0000 0000 0000 0000 0000

     OUTPUT PINS: 111111
                  345678
   POLARITY FUSE: ------
-- More --
```

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS,XPT,JED

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > L
```

```
BGATESP.PDS
BGATESP.XPT
BGATESP.JED

ENTER A HIGHLIGHTED CHARACTER > J
```

```
PALASM XPLOT, V2.06 - BETA RELEASE
(C) - COPYRIGHT MONOLITHIC MEMORIES INC., 1984

Title    :  Basic Gates (Positive)
Pattern  :  BGates.pds
Revision :  B
Author   :  A G Gilbert
Company  :  Monolithic Memories Inc., Santa Clara, CA
```

```
PAL12P6
BASIC_GATES*
G0*F0*
L0000
11111110111111111111111
00000000000000000000000
00000000000000000000000
00000000000000000000000
01011111111111111111111
00000000000000000000000
11110111111111111111111
11111111011111111111111
11111111111010111111111
00000000000000000000000
-- More --

11111111111110110111111
11111111111111001111111
11111111111111111111110
11111111111111111101111
00000000000000000000000
00000000000000000000000
111111
*C191B*
55ED

Strike a key when ready . . .
```

3

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS,XPT,JED

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > S
```

```
BGATESP.PDS
BGATESP.XPT
BGATESP.JED
BGATESP.HST
BGATESP.TRF

ENTER A HIGHLIGHTED CHARACTER > F
```

```
PALASM2 SIMULATION, V0.00 -- ALPHA RELEASE
(C) - COPYRIGHT  MONOLITHIC MEMORIES INC., 1985

TRACE_ON
SETF
SETF
SETF
SETF

END OF SIMULATION
Total memory used        ==>   4326 bytes

Simulation result (History) is in ===>BGATESP.hst
Simulation result (Trace)   is in ===>BGATESP.trf

Strike a key when ready . . .
```

```
Page :  1
   gggg
 A LHHH
 B HLLL
 C LLHH
 D LHLH
 E LLLH
 F LLHH
 G LHLH
 H LHHH
 J LLHH
 K LHLH
 L HHHL
 M LLHH
 N LHLH
 S HLLL
 P LLHH
-- More --
 Q LHLH
 R LHHL
```

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS,XPT,JED,HST,TRF

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > L
```

```
Monolithic  MMI Memories
work file: BGATESP.PDS Ext: PDS,XPT,JED,HST,TRF

                    PALASM2     MENU
Workfile            Assemble    Programmer interface
Edit                Simulate    Convert palasm1 to palasm2
List                Zhal        Quit
Directory
delete File
print docUment
dos coMmand

ENTER MENU COMMAND > Q
```

PAL Device Design Specification

```
TITLE     Basic Gates (Negative Logic)
PATTERN   BGatesn.pds
REVISION  A
AUTHOR    A G GILBERT
COMPANY   Monolithic Memories Inc., Santa Clara, CA
DATE      1/9/85

;It is worthwhile for the novice PAL designer to compare
;the number of product terms required to realize these
;basic functions implemented in negative logic with the
;same functions implemented in positive logic.

CHIP  BASIC_GATES PAL12P6

C D F G M N P Q I GND
J K L R S H E B A VCC

EQUATIONS

/B = A                  ;NOT FUNCTION       A | B
                        ;                  -------
                        ;                   0 | 1
                        ;                   1 | 0 *

/E = /C+/D              ;AND FUNCTION       C   D | E
                        ;                  -----------
                        ;                   0   0 | 0 *
                        ;                   0   1 | 0 *
                        ;                   1   0 | 0 *
                        ;                   1   1 | 1

/H = /F*/G              ;OR FUNCTION        F   G | H
                        ;                  -----------
                        ;                   0   0 | 0 *
                        ;                   0   1 | 1
                        ;                   1   0 | 1
                        ;                   1   1 | 1

/L = J*K                ;NAND FUNCTION      J   K | L
                        ;                  -----------
                        ;                   0   0 | 1
                        ;                   0   1 | 1
                        ;                   1   0 | 1
                        ;                   1   1 | 0 *

/S = M+N                ;NOR FUNCTION       M   N | S
                        ;                  -----------
                        ;                   0   0 | 1
                        ;                   0   1 | 0 *
                        ;                   1   0 | 0 *
                        ;                   1   1 | 0 *

/R = P*Q + /P*/Q        ;XOR FUNCTION       P   Q | R
                        ;                  -----------
                        ;                   0   0 | 0 *
                        ;                   0   1 | 1
                        ;                   1   0 | 1
                        ;                   1   1 | 0 *

SIMULATION

TRACE_ON A B C D E F G H I J K L M N P Q R S

SETF A C D F G J K M N P Q
SETF /A /C /F /J /M /P
SETF /D /G /K /N /Q
SETF C F J M P
```

Simulation Results

```
Page :  1
   gggg
 A HLLL
 B LHHH
 C HLLH
 D HHLL
 E HLLL
 F HLLH
 G HHLL
 H HHLH
 I XXXX
 J HLLH
 K HHLL
 L LHHH
 M HLLH
 N HHLL
 P HLLH
 Q HHLL
 R LHLH
 S LLHL
```

XPLOT Output

```
Title    : Basic Gates (Negative Logic)
Pattern  : BGatesn.pds
Revision : A
Author   : A G Gilbert
Company  : Monolithic Memories Inc., Santa Clara, CA
Date     : 1/9/85

PAL12P6
BASIC_GATES

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

 0 0000 0000 0000 0000 0000 0000 0000 0000
 1 0000 0000 0000 0000 0000 0000 0000 0000
 2 0000 0000 0000 0000 0000 0000 0000 0000
 3 0000 0000 0000 0000 0000 0000 0000 0000
 4 0000 0000 0000 0000 0000 0000 0000 0000
 5 0000 0000 0000 0000 0000 0000 0000 0000
 6 0000 0000 0000 0000 0000 0000 0000 0000
 7 0000 0000 0000 0000 0000 0000 0000 0000

 8 ---- --X- --OO --OO --OO --OO ---- ----
 9 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
10 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
11 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
12 0000 0000 0000 0000 0000 0000 0000 0000
13 0000 0000 0000 0000 0000 0000 0000 0000
14 0000 0000 0000 0000 0000 0000 0000 0000
15 0000 0000 0000 0000 0000 0000 0000 0000

16 ---X ---- --OO --OO --OO --OO ---- ----
17 -X-- ---- --OO --OO --OO --OO ---- ----
18 0000 0000 0000 0000 0000 0000 0000 0000
19 0000 0000 0000 0000 0000 0000 0000 0000
20 0000 0000 0000 0000 0000 0000 0000 0000
21 0000 0000 0000 0000 0000 0000 0000 0000
22 0000 0000 0000 0000 0000 0000 0000 0000
23 0000 0000 0000 0000 0000 0000 0000 0000

24 ---- -X-- -XOO --OO --OO --OO ---- ----
25 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
26 0000 0000 0000 0000 0000 0000 0000 0000
27 0000 0000 0000 0000 0000 0000 0000 0000
28 0000 0000 0000 0000 0000 0000 0000 0000
29 0000 0000 0000 0000 0000 0000 0000 0000
30 0000 0000 0000 0000 0000 0000 0000 0000
31 0000 0000 0000 0000 0000 0000 0000 0000

32 ---- ---- --OO X-OO --OO --OO ---- ----
33 ---- ---- --OO --OO X-OO --OO ---- ----
34 0000 0000 0000 0000 0000 0000 0000 0000
35 0000 0000 0000 0000 0000 0000 0000 0000
36 0000 0000 0000 0000 0000 0000 0000 0000
37 0000 0000 0000 0000 0000 0000 0000 0000
38 0000 0000 0000 0000 0000 0000 0000 0000
39 0000 0000 0000 0000 0000 0000 0000 0000

40 ---- ---- --OO --OO --OO X-OO X--- ----
41 ---- ---- --OO --OO --OO -XOO -X-- ----
42 0000 0000 0000 0000 0000 0000 0000 0000
43 0000 0000 0000 0000 0000 0000 0000 0000
44 0000 0000 0000 0000 0000 0000 0000 0000
45 0000 0000 0000 0000 0000 0000 0000 0000
46 0000 0000 0000 0000 0000 0000 0000 0000
47 0000 0000 0000 0000 0000 0000 0000 0000
```

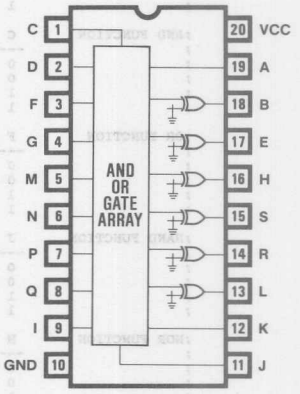
3

```
48 ---- ---- --OO --OO --OO --OO --X- --X-
49 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
50 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
51 XXXX XXXX XXOO XXOO XXOO XXOO XXXX XXXX
52 0000 0000 0000 0000 0000 0000 0000 0000
53 0000 0000 0000 0000 0000 0000 0000 0000
54 0000 0000 0000 0000 0000 0000 0000 0000
55 0000 0000 0000 0000 0000 0000 0000 0000

56 0000 0000 0000 0000 0000 0000 0000 0000
57 0000 0000 0000 0000 0000 0000 0000 0000
58 0000 0000 0000 0000 0000 0000 0000 0000
59 0000 0000 0000 0000 0000 0000 0000 0000
60 0000 0000 0000 0000 0000 0000 0000 0000
61 0000 0000 0000 0000 0000 0000 0000 0000
62 0000 0000 0000 0000 0000 0000 0000 0000
63 0000 0000 0000 0000 0000 0000 0000 0000


      OUTPUT PINS: 111111
                   345678
   POLARITY FUSE: XXXXXX

TOTAL FUSES BLOWN:   203
```

JEDEC Output

```
Title    : Basic Gates (Negative Logic)
Pattern  : BGatesn.pds
Revision : A
Author   : A G Gilbert
Company  : Monolithic Memories Inc., Santa Clara, CA
Date     : 1/9/85

PAL12P6
BASIC_GATES*
G0*F0*
L0000
11111101111111111111111111
00000000000000000000000000
00000000000000000000000000
00000000000000000000000000
11101111111111111111111111
10111111111111111111111111
11111011101111111111111111
00000000000000000000000000
11111111111011111111111111
11111111111101111111111111
11111111111111010111111111
11111111111111101011111111
11111111111111111111011101
00000000000000000000000000
00000000000000000000000000
00000000000000000000000000
000000
```

Logic Symbol



PAL Device Design Specification

```
Title     4to16 Decoder
Pattern   4-16DEC.PDS
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories, Santa Clara, CA
Date      1/9/85

CHIP Decoder PAL6L16

Q0 Q1 Q2 A B C D EN1 EN2 Q3 Q4 GND Q5
Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 VCC

EQUATIONS

/Q0  = /D*/C*/B*/A* EN1* EN2          ;Decode 0000
/Q1  = /D*/C*/B* A* EN1* EN2          ;Decode 0001
/Q2  = /D*/C* B*/A* EN1* EN2          ;Decode 0010
/Q3  = /D*/C* B* A* EN1* EN2          ;Decode 0011
/Q4  = /D* C*/B*/A* EN1* EN2          ;Decode 0100
/Q5  = /D* C*/B* A* EN1* EN2          ;Decode 0101
/Q6  = /D* C* B*/A* EN1* EN2          ;Decode 0110
/Q7  = /D* C* B* A* EN1* EN2          ;Decode 0111
/Q8  =  D*/C*/B*/A* EN1* EN2          ;Decode 1000
/Q9  =  D*/C*/B* A* EN1* EN2          ;Decode 1001
/Q10 =  D*/C* B*/A* EN1* EN2          ;Decode 1010
/Q11 =  D*/C* B* A* EN1* EN2          ;Decode 1011
/Q12 =  D* C*/B*/A* EN1* EN2          ;Decode 1100
/Q13 =  D* C*/B* A* EN1* EN2          ;Decode 1101
/Q14 =  D* C* B*/A* EN1* EN2          ;Decode 1110
/Q15 =  D* C* B* A* EN1* EN2          ;Decode 1111


SIMULATION

TRACE_ON  D B C A Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10
          Q11 Q12 Q13 Q14 Q15

SETF /D /C /B /A EN1 EN2
SETF A
SETF B
SETF C
SETF D
SETF /D
SETF /C
SETF /B
SETF /A
SETF /EN1                             ;Set outputs to high
SETF EN1 /EN2                         ;Set outputs to high
SETF /EN1                             ;Set outputs to high

;The 4 to 16 decoder, decodes four binary decoded inputs
;into one of 16 mutually exclusive outputs, whenever the
;two enable lines EN1 and EN2 are high. When one or both
;of the enable lines are low the outputs are all set to
;high values.
```

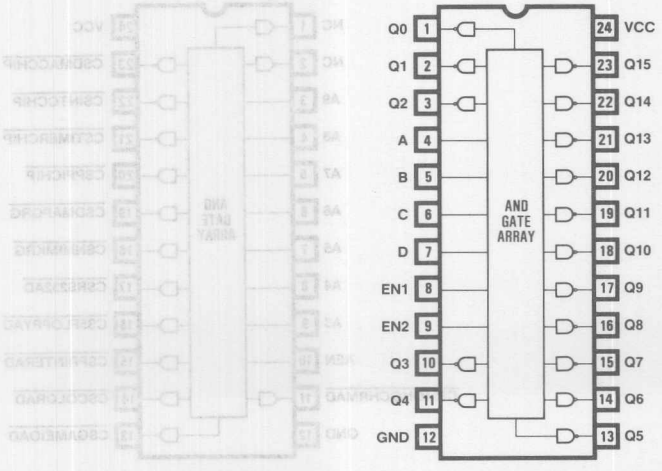
Simulation Results

```
Page :  1
      gggggggggg gg
  D   LLLLHLLLLL LL
  B   LLHHHHHLLL LL
  C   LLLHHHLLLL LL
  A   LHHHHHHHLL LL
  Q0  LHHHHHHHLH HH
  Q1  HLHHHHHLHH HH
  Q2  HHHHHHHHHH HH
  Q3  HHLHHHLHHH HH
  Q4  HHHHHHHHHH HH
  Q5  HHHHHHHHHH HH
  Q6  HHHHHHHHHH HH
  Q7  HHHHLHLHHH HH
  Q8  HHHHHHHHHH HH
  Q9  HHHHHHHHHH HH
  Q10 HHHHHHHHHH HH
  Q11 HHHHHHHHHH HH
  Q12 HHHHHHHHHH HH
  Q13 HHHHHHHHHH HH
  Q14 HHHHHHHHHH HH
  Q15 HHHHLHHHHH HH
```

XPLOT Output

```
Title    :    4to16 Decoder     Author   :   Mehrnaz Hada
Pattern  :   4-16DEC.PDS        Company  :   Monolithic Memories,
Revision :   A                  Date     :      1/9/85

PAL6L16
DECODER

                 11
     0123 4567 8901

  0 -X-X -X-X X-X-
  1 X-X- X-X- X-X-
  2 X--X -X-X X-X-
  3 -XX- -X-X X-X-
  4 -XX- X-X- X-X-
  5 X--X X-X- X-X-
  6 -X-X X-X- X-X-
  7 X-X- -XX- X-X-
  8 -XX- -XX- X-X-
  9 X--X -XX- X-X-
 10 -X-X -XX- X-X-
 11 X-X- -X-X X-X-
 12 X-X- X--X X-X-
 13 -XX- X--X X-X-
 14 -X-X X--X X-X-
 15 X--X X--X X-X-


TOTAL FUSES BLOWN:    96
```

Logic Symbol



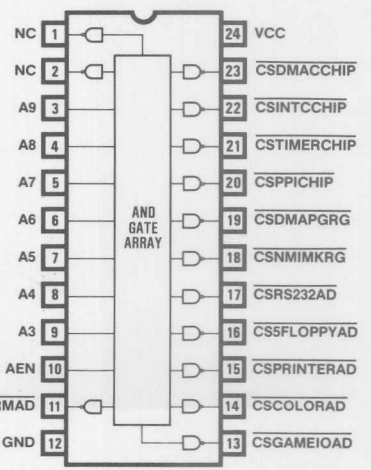
```
Title    PC I/O Mapper
Pattern  MemIO.pds
Revision A
Author   A G Gilbert
Company  Monolithic Memories Inc., Santa Clara,CA
Date     1/8/85

;Personal computers which are hardware compatible with the
;ubiquitous IBM PC share this I/O map.

CHIP PC_IO PAL8L14

NC NC A9 A8 A7 A6 A5 A4 A3 AEN /CSMONOCHRMAD GND
/CSGAMEIOAD /CSCOLORAD /CSPRINTERAD /CS5FLOPPYAD /CSRS232AD /CSNMIMKRG
/CSDMAPGRG /CSPPICHIP /CSTIMERCHIP /CSINTCCHIP /CSDMACCHIP VCC

Equations

CSDMACCHIP = /A9*/A8*/A7*/A6*/A5        ;DMA controller
           * /A4*/AEN                    ;Chip select
                                         ;HEX address 000-00F

CSINTCCHIP = /A9*/A8*/A7*/A6*A5          ;Interupt controller
           * /A4*/A3*/AEN                ;Chip select
                                         ;HEX address 020-021

CSTIMERCHIP = /A9*/A8*/A7*A6*/A5         ;Timer
            * /A4*/A3*/AEN               ;Chip select
                                         ;HEX address 040-043

CSPPICHIP = /A9*/A8*/A7*A6*A5            ;Parallel peripheral interface
          * /A4*/A3*/AEN                 ;Chip select
                                         ;HEX address 060-063

CSDMAPGRG = /A9*/A8*A7*/A6*/A5           ;DMA page register
          * /A4*/A3*/AEN                 ;Chip select
                                         ;HEX address 080-083

CSNMIMKRG = /A9*/A8*A7*/A6*A5            ;NMI mask register
          * /A4*/AEN                     ;Chip select
                                         ;HEX address 0AX

CSRS232AD = A9*A8*A7*A6*A5               ;RS 232 module
          * A4*A3*/AEN                   ;Device select
                                         ;HEX address 3F8-3FF

CS5FLOPPYAD = A9*A8*A7*A6*A5             ;5.25 floppy disk module
            * A4*/A3*/AEN                ;Device select
                                         ;HEX address 3F0-3F7

CSPRINTERAD = A9*/A8*/A7*A6*A5           ;Parallel printer module
            * A4*A3*/AEN                 ;Device select
                                         ;HEX address 378-37F

CSCOLORAD = A9*A8*A7*A6*/A5              ;Color graphics video module
          * A4*/AEN                      ;Device select
                                         ;HEX address 3D0-3DF

CSGAMEIOAD = A9*/A8*/A7*/A6*/A5          ;Game I/O module
           * /A4*/AEN                    ;Device select
                                         ;HEX address 200-20F

CSMONOCHRMAD = A9*A8*A7*/A6*A5           ;Monochrome video module
             * A4*/AEN                   ;Device select
                                         ;HEX address 3B0-3BF

SIMULATION

TRACE_ON  A9 A8 A7 A6 A5 A4 A3 AEN /CSMONOCHRMAD
          /CSGAMEIOAD /CSCOLORAD /CSPRINTERAD /CS5FLOPPYAD
          /CSRS232AD /CSNMIMKRG /CSDMAPGRG /CSPPICHIP
          /CSTIMERCHIP /CSINTCCHIP /CSDMACCHIP

SETF AEN
SETF /A9 /A8 /A7 /A6 /A5 /A4 /A3 /AEN
SETF  A5
SETF  A6
SETF /A5  A7
SETF  A5
SETF  A4 /A6
SETF  A9  A8  A7  A6  A5  A4  A3
SETF /A3
SETF /A5
SETF /A4 /A6 /A7 /A8
SETF  A9  A8  A7 /A6  A5  A4
```

```
Page :  1
                 gggggggggg gg
 A9              XLLLLLLHHH HH
 A8              XLLLLLHHH LH
 A7              XLLLHHHHHH LH
 A6              XLLHHHLHHH LL
 A5              XLHHLHHHHL LH
 A4              XLLLLLHHHH LH
 A3              XLLLLLLHLL LL
 AEN             HLLLLLLLLL LL
/CSMONOCHRMAD    HHHHHHHHHH HL
/CSGAMEIOAD      HHHHHHHHHH LH
/CSCOLORAD       HHHHHHHHHL HH
/CSPRINTERAD     HHHHHHHHHH HH
/CS5FLOPPYAD     HHHHHHHHLH HH
/CSRS232AD       HHHHHHHLHH HH
/CSNMIMKRG       HHHHHHHHHH HH
/CSDMAPGRG       HHHHHHHHHH HH
/CSPPICHIP       HHHLHHHHHH HH
/CSTIMERCHIP     HHHHHHHHHH HH
/CSINTCCHIP      HHLHHHHHHH HH
/CSDMACCHIP      HLHHHHHHHH HH
```

XPLOT Output

```
Title    : PC I/O Mapper
Pattern  : MemIO.pds
Revision : A
Author   : A G Gilbert
Company  : Monolithic Memories Inc
Date     : 1/8/85


PAL8L14
PC_IO

                    11 1111
     0123 4567 8901 2345

  0 XXXX XXXX XXXX XXXX
  1 -X-X -X-X -X-X ---X
  2 XXXX XXXX XXXX XXXX
  3 -X-X -X-X X--X -X-X
  4 -X-X -XX- -X-X -X-X
  5 -X-X -XX- X--X -X-X
  6 -X-X X--X -X-X -X-X
  7 -X-X X--X X--X ---X
  8 X-X- X-X- X-X- X--X
  9 X-X- X-X- X-X- -X-X
 10 X--X -XX- X-X- X--X
 11 X-X- X-X- -XX- ---X
 12 X-X- X--X X-X- ---X
 13 X--X -X-X -X-X ---X


TOTAL FUSES BLOWN:   101
```

Logic Symbol



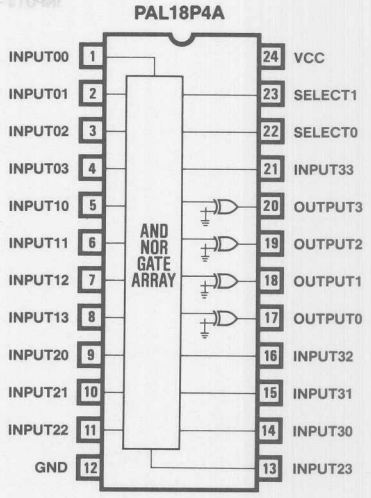
PAL Device Design Specification

```
TITLE     4:1 MUX
PATTERN   MUX4.PDS
REVISION  A
AUTHOR    John Birkner
COMPANY   Monolithic Memories Inc. Santa Clara, CA
DATE      1/8/85

;       The four to one multiplexor routes one of four 4-bit nibbles,
;       INPUT0n..INPUT3n, to the output, OUTPUT0..OUTPUT3

CHIP MUX4 PAL18P4
INPUT00 INPUT01 INPUT02 INPUT03
INPUT10 INPUT11 INPUT12 INPUT13
INPUT20 INPUT21 INPUT22 GND
INPUT23 INPUT30 INPUT31 INPUT32
OUTPUT0 OUTPUT1 OUTPUT2 OUTPUT3
INPUT33 SELECT0 SELECT1 VCC

EQUATIONS

OUTPUT0 =  INPUT00 * /SELECT1 * /SELECT0        ;SEL 0
        +  INPUT01 * /SELECT1 *  SELECT0        ;SEL 1
        +  INPUT02 *  SELECT1 * /SELECT0        ;SEL 2
        +  INPUT03 *  SELECT1 *  SELECT0        ;SEL 3

OUTPUT1 =  INPUT10 * /SELECT1 * /SELECT0        ;SEL 0
        +  INPUT11 * /SELECT1 *  SELECT0        ;SEL 1
        +  INPUT12 *  SELECT1 * /SELECT0        ;SEL 2
        +  INPUT13 *  SELECT1 *  SELECT0        ;SEL 3

OUTPUT2 =  INPUT20 * /SELECT1 * /SELECT0        ;SEL 0
        +  INPUT21 * /SELECT1 *  SELECT0        ;SEL 1
        +  INPUT22 *  SELECT1 * /SELECT0        ;SEL 2
        +  INPUT23 *  SELECT1 *  SELECT0        ;SEL 3

OUTPUT3 =  INPUT30 * /SELECT1 * /SELECT0        ;SEL 0
        +  INPUT31 * /SELECT1 *  SELECT0        ;SEL 1
        +  INPUT32 *  SELECT1 * /SELECT0        ;SEL 2
        +  INPUT33 *  SELECT1 *  SELECT0        ;SEL 3

SIMULATION

TRACE_ON
INPUT00 INPUT01 INPUT02 INPUT03
INPUT10 INPUT11 INPUT12 INPUT13
INPUT20 INPUT21 INPUT22 INPUT23
INPUT30 INPUT31 INPUT32 INPUT33
SELECT1 SELECT0
OUTPUT0 OUTPUT1 OUTPUT2 OUTPUT3

SETF    /SELECT1 /SELECT0

SETF    /INPUT00 /INPUT01 /INPUT02 /INPUT03     ;CLEAR ALL INPUTS
        /INPUT10 /INPUT11 /INPUT12 /INPUT13
        /INPUT20 /INPUT21 /INPUT22 /INPUT23
        /INPUT30 /INPUT31 /INPUT32 /INPUT33

SETF     INPUT00  INPUT10  INPUT20  INPUT30     ;SET SELECTED NIBBLE

SETF     INPUT00  INPUT01  INPUT02  INPUT03     ;SET ALL INPUTS
         INPUT10  INPUT11  INPUT12  INPUT13
         INPUT20  INPUT21  INPUT22  INPUT23
         INPUT30  INPUT31  INPUT32  INPUT33

SETF    /INPUT00 /INPUT10 /INPUT20 /INPUT30     ;CLEAR SELECTED NIBBLE


SETF     /SELECT1  SELECT0                      ;SELECT ADDRESS 1

SETF    /INPUT00 /INPUT01 /INPUT02 /INPUT03
        /INPUT10 /INPUT11 /INPUT12 /INPUT13
        /INPUT20 /INPUT21 /INPUT22 /INPUT23
        /INPUT30 /INPUT31 /INPUT32 /INPUT33

SETF     INPUT01  INPUT11  INPUT21  INPUT31     ;SET SELECTED NIBBLE

SETF     INPUT00  INPUT01  INPUT02  INPUT03     ;SET ALL INPUTS
         INPUT10  INPUT11  INPUT12  INPUT13
         INPUT20  INPUT21  INPUT22  INPUT23
         INPUT30  INPUT31  INPUT32  INPUT33

SETF    /INPUT01 /INPUT11 /INPUT21 /INPUT31     ;CLEAR SELECTED NIBBLE

SETF     SELECT1 /SELECT0                       ;SELECT ADDRESS 2

SETF    /INPUT00 /INPUT01 /INPUT02 /INPUT03     ;CLEAR ALL INPUTS
        /INPUT10 /INPUT11 /INPUT12 /INPUT13
        /INPUT20 /INPUT21 /INPUT22 /INPUT23
        /INPUT30 /INPUT31 /INPUT32 /INPUT33

SETF     INPUT02  INPUT12  INPUT22  INPUT32     ;SET SELECTED NIBBLE

SETF     INPUT00  INPUT01  INPUT02  INPUT03     ;SET ALL INPUTS
         INPUT10  INPUT11  INPUT12  INPUT13
         INPUT20  INPUT21  INPUT22  INPUT23
         INPUT30  INPUT31  INPUT32  INPUT33

SETF    /INPUT02 /INPUT12 /INPUT22 /INPUT32     ;CLEAR SELECTED NIBBLE

SETF     SELECT1  SELECT0                       ;SELECT ADDRESS 3

SETF    /INPUT00 /INPUT01 /INPUT02 /INPUT03     ;CLEAR ALL INPUTS
        /INPUT10 /INPUT11 /INPUT12 /INPUT13
        /INPUT20 /INPUT21 /INPUT22 /INPUT23
        /INPUT30 /INPUT31 /INPUT32 /INPUT33

SETF     INPUT03  INPUT13  INPUT23  INPUT33     ;SET SELECTED NIBBLE

SETF     INPUT00  INPUT01  INPUT02  INPUT03     ;SET ALL INPUTS
         INPUT10  INPUT11  INPUT12  INPUT13
         INPUT20  INPUT21  INPUT22  INPUT23
         INPUT30  INPUT31  INPUT32  INPUT33

SETF    /INPUT03 /INPUT13 /INPUT23 /INPUT33     ;CLEAR SELECTED NIBBLE
```

Simulation Results

```
Page :  1
          gggggggggg gggggggggg
INPUT00 XLHHLLLLHH HLLHHHLLHH
INPUT01 XLLHHHLHHL LLLHHHLLHH
INPUT02 XLLHHHLLHH HLHHLLLLHH
INPUT03 XLLHHHLLHH HLLHHHLHHL
INPUT10 XLHHLLLLHH HLLHHHLLHH
INPUT11 XLLHHHLHHL LLLHHHLLHH
INPUT12 XLLHHHLLHH HLHHLLLLHH
INPUT13 XLLHHHLLHH HLLHHHLHHL
INPUT20 XLHHLLLLHH HLLHHHLLHH
INPUT21 XLLHHHLHHL LLLHHHLLHH
INPUT22 XLLHHHLLHH HLHHLLLLHH
INPUT23 XLLHHHLLHH HLLHHHLHHL
INPUT30 XLHHLLLLHH HLLHHHLLHH
INPUT31 XLLHHHLHHL LLLHHHLLHH
INPUT32 XLLHHHLLHH HLHHLLLLHH
INPUT33 XLLHHHLLHH HLLHHHLHHL
SELECT1 LLLLLLLLLL HHHHHHHHHH
SELECT0 LLLLLHHHHH LLLLLHHHHH
OUTPUT0 XLHHLHLHHL HLHHLHLHHL
OUTPUT1 XLHHLHLHHL HLHHLHLHHL
OUTPUT2 XLHHLHLHHL HLHHLHLHHL
OUTPUT3 XLHHLHLHHL HLHHLHLHHL
```

Logic Symbol

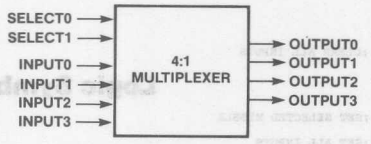


PAL Device Design Specification

```
TITLE     4:1 MUX
PATTERN   MUX4A.PDS
REVISION  A
AUTHOR    John Birkner
COMPANY   Monolithic Memories Inc. Santa Clara, CA
DATE      1/8/85

;       The four to one multiplexor routes one of four 4-bit nibbles,
;       INPUT[0,n]..INPUT[3,n], to the output, OUTPUT[0]..OUTPUT[3].
;       This example illustrates the use of high level macros to
;       save typing and improve accuracy.


CHIP MUX4A PAL18P4
INPUT[0..1,0..3] INPUT[2,0..2] GND
INPUT[2,3] INPUT[3,0..2]   OUTPUT[0..3]    INPUT[3,3] SELECT[0..1] VCC

EQUATIONS

OUTPUT[m=0..3] = OR [n=0..3] (INPUT[m,n] * BIN[n] ( SELECT[0] SELECT[1]

SIMULATION

TRACE_ON INPUT[0..3,0..3] SELECT[0..1] OUTPUT[0..3]

FOR n:=0 TO 3 DO
  BEGIN
        SETF  BIN[n](SELECT[1] SELECT[0])       ;SELECT ADDRESS
        SETF /INPUT[0..3,0..3]                  ;CLEAR ALL INPUTS
        SETF  INPUT[n,0..3]                     ;SET   SELECTED NIBBLE
        SETF  INPUT[0..3,0..3]                  ;SET   ALL INPUTS
        SETF /INPUT[n,0..3]                     ;CLEAR SELECTED NIBBLE
    END
```

Simulation Results

```
Page :  1
          gggggggggg gggggggggg
 INPUT00 XLHHLLLLHH HLLHHHLLHH
 INPUT01 XLLHHHLHHL LLLHHHLLHH
 INPUT02 XLLHHHLLHH HLHHLLLLHH
 INPUT03 XLLHHHLLHH HLLHHHLHHL
 INPUT10 XLHHLLLLHH HLLHHHLLHH
 INPUT11 XLLHHHLHHL LLLHHHLLHH
 INPUT12 XLLHHHLLHH HLHHLLLLHH
 INPUT13 XLLHHHLLHH HLLHHHLHHL
 INPUT20 XLHHLLLLHH HLLHHHLLHH
 INPUT21 XLLHHHLHHL LLLHHHLLHH
 INPUT22 XLLHHHLLHH HLHHLLLLHH
 INPUT23 XLLHHHLLHH HLLHHHLHHL
 INPUT30 XLHHLLLLHH HLLHHHLLHH
 INPUT31 XLLHHHLHHL LLLHHHLLHH
 INPUT32 XLLHHHLLHH HLHHLLLLHH
 INPUT33 XLLHHHLLHH HLLHHHLHHL
 SELECT1 LLLLLLLLLL HHHHHHHHHH
 SELECT0 LLLLLHHHHH LLLLLHHHHH
 OUTPUT0 XLHHLHLHHL HLHHLHLHHL
 OUTPUT1 XLHHLHLHHL HLHHLHLHHL
 OUTPUT2 XLHHLHLHHL HLHHLHLHHL
 OUTPUT3 XLHHLHLHHL HLHHLHLHHL
```



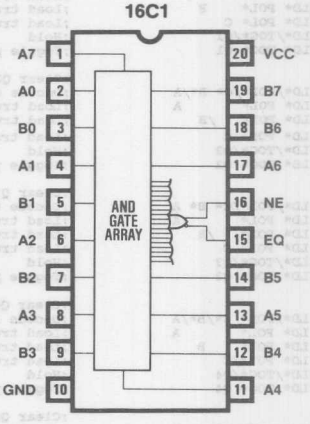
PAL Device Design Specification

```
Title    Octal_Comparator
Pattern  OctComp.pds
Revision A
Author   Mehrnaz Hada
Company  Monolithic Memories Inc., Santa Clara,CA
Date     1/29/85

;The octal comparator establishes when two 8-bit data
;strings (A7-A0) and (B7-B0) are equivalent (EQ=H) or
;equivalent (NE=H).

CHIP OctalComparato PAL16C1

A7 A0 B0 A1 B1 A2 B2 A3 B3 GND
A4 B4 A5 B5 EQ NE A6 B6 B7 VCC

EQUATIONS

NE =  A0*/B0  +  /A0* B0                ;A0 :+: B0
   +  A1*/B1  +  /A1* B1                ;A1 :+: B1
   +  A2*/B2  +  /A2* B2                ;A2 :+: B2
   +  A3*/B3  +  /A3* B3                ;A3 :+: B3
   +  A4*/B4  +  /A4* B4                ;A4 :+: B4
   +  A5*/B5  +  /A5* B5                ;A5 :+: B5
   +  A6*/B6  +  /A6* B6                ;A6 :+: B6
   +  A7*/B7  +  /A7* B7                ;A7 :+: B7


SIMULATION

TRACE_ON A7 A6 A5 A4 A3 A2 A1 A0 NE
         B7 B6 B5 B4 B3 B2 B1 B0

SETF  A7 /A6 /A5 /A4 /A3 /A2 /A1 /A0    ;A7=H, B7=L
     /B7 /B6 /B5 /B4 /B3 /B2 /B1 /B0
SETF /A7 A6                             ;A6=H, B6=L
SETF /A6 A5                             ;A5=H, B5=L
SETF /A5 A4                             ;A4=H, B4=L
SETF /A4 A3                             ;A3=H, B3=L
SETF /A3 A2                             ;A2=H, B2=L
SETF /A2 A1                             ;A1=H, B1=L
SETF /A1 A0                             ;A0=H, B0=L
SETF /A7 /A6 /A5 /A4 /A3 /A2 /A1 /A0    ;A7=L, B7=H
      B7
SETF /B7 B6                             ;A6=L, B6=H
SETF /B6 B5                             ;A5=L, B5=L
SETF /B5 B4                             ;A4=L, B4=H
SETF /B4 B3                             ;A3=L, B3=H
SETF /B3 B2                             ;A2=L, B2=H
SETF /B2 B1                             ;A1=L, B1=H
SETF /B1 B0                             ;A0=L, B0=H
SETF /B0                                ;Test all L's
SETF A7 A6 A5 A4 A3 A2 A1 A0            ;Test all H's
     B7 B6 B5 B4 B3 B2 B1 B0
SETF /A7 A6 /A5 A4 /A3 A2 /A1 A0        ;Test even ones
     /B7 B6 /B5 B4 /B3 B2 /B1 B0
SETF A7 /A6 A5 /A4 A3 /A2 A1 /A0        ;Test odd ones
     B7 /B6 B5 /B4 B3 /B2 B1 /B0


;Function Table for PALASM1

;A7 A6 A5 A4 A3 A2 A1 A0 B7 B6 B5 B4 B3 B2 B1 B0 NE EQ

; Input  A  Input  B  Outputs
; 76543210  76543210   NE EQ    Comments
;-----------------------------------------------------
; HLLLLLLL  LLLLLLLL   H   L    A7=H, B7=L
; LHLLLLLL  LLLLLLLL   H   L    A6=H, B6=L
; LLHLLLLL  LLLLLLLL   H   L    A5=H, B5=L
; LLLHLLLL  LLLLLLLL   H   L    A4=H, A5=L
; LLLLHLLL  LLLLLLLL   H   L    A3=H, B3=L
; LLLLLHLL  LLLLLLLL   H   L    A2=H, B2=L
; LLLLLLHL  LLLLLLLL   H   L    A1=H, B1=L
; LLLLLLLH  LLLLLLLL   H   L    A0=H, B0=L
; LLLLLLLL  HLLLLLLL   H   L    A7=L, B7=H
; LLLLLLLL  LHLLLLLL   H   L    A6=L, B6=H
; LLLLLLLL  LLHLLLLL   H   L    A5=L, B5=H
; LLLLLLLL  LLLHLLLL   H   L    A4=L, B4=H
; LLLLLLLL  LLLLHLLL   H   L    A3=L, B3=H
; LLLLLLLL  LLLLLHLL   H   L    A2=L, B2=H
; LLLLLLLL  LLLLLLHL   H   L    A1=L, B1=H
; LLLLLLLL  LLLLLLLH   H   L    A0=L, B0=H
; LLLLLLLL  LLLLLLLL   L   H    Test all L'S
; HHHHHHHH  HHHHHHHH   L   H    Test all H'S
; HLHLHLHL  HLHLHLHL   L   H    Test even checkerboard
; LHLHLHLH  LHLHLHLH   L   H    Test odd  checkerboard
;-----------------------------------------------------
```

Simulation Results

```
Page :  1
    gggggggggg gggggggggg
 A7 HLLLLLLLLL LLLLLLLHLH
 A6 LHLLLLLLLL LLLLLLHHLL
 A5 LLHLLLLLLL LLLLLLHLLH
 A4 LLLHLLLLLL LLLLLLHHLL
 A3 LLLLHLLLLL LLLLLLHLLH
 A2 LLLLLHLLLL LLLLLLHHLL
 A1 LLLLLLHLLL LLLLLLHLLH
 A0 LLLLLLLHLL LLLLLLHHLL
 NE HHHHHHHHHH HHHHHHLLLL
 B7 LLLLLLLLHL LLLLLLLHLH
 B6 LLLLLLLLLH LLLLLLHHLL
 B5 LLLLLLLLLL HLLLLLLHLH
 B4 LLLLLLLLLL LHLLLLHHLL
 B3 LLLLLLLLLL LLHLLLHLLH
 B2 LLLLLLLLLL LLLHLLHHLL
 B1 LLLLLLLLLL LLLLHLHLLH
 B0 LLLLLLLLLL LLLLLHLHHL
```

Logic Symbol



```
Title    3to8_Dmux
Pattern  3to8Dmux.pds
Revision A
Author   Mehrnaz Hada
Company  Monolithic Memories Inc., Santa Clara, CA
Date     1/29/85

;The 3-to-8 demultiplexer with control storage provides a
;conventional 8-bit demux function combined with control
;storage functions:load true, load complement, hold, toggle,
;polarity, clear and preset. Five inputs(/LD,/CLR,/PR,POL,
;TOG) select one of six operations. The six operations are
;summarized in the following operations table:

;Control   Functions   Polarity Inputs Outputs
;/OC CLK /CLR /PR /LD   POL  TOG  ABC    Q7-Q0  Operation
;-------------------------------------------------------------
; H   X    X   X   X     X    X    X       Z    HI-Z
; L   C    L   X   X     X    X    X       L    Clear
; L   C    H   L   X     X    X    X       H    PRESET
; L   C    H   H   L     H    X    I     MUX    Load true
; L   C    H   H   L     L    X    I    /MUX    Load COMP
; L   C    H   H   H     X    L    X       Q    Hold
; L   C    H   H   H     X    H    X      /Q    Tog polarity
;-------------------------------------------------------------

CHIP 3to8Dmux PAL16R8

CLK /CLR /PR A  B  C  /LD POL TOG GND
/OC  Q7   Q6 Q5 Q4 Q3  Q2 Q1  Q0  VCC

EQUATIONS

/Q0 :=  CLR                              ;Clear Q0
     + /PR* LD*/POL*/C*/B*/A             ;Decode 000
     + /PR* LD* POL*       A             ;Load true
     + /PR* LD* POL*    B                ;Load true
     + /PR* LD* POL* C                   ;Load true
     + /PR*/LD*/TOG*/Q0                  ;Hold
     + /PR*/LD* TOG* Q0                  ;Toggle polarity

/Q1 :=  CLR                              ;Clear Q1
     + /PR* LD*/POL*/C*/B* A             ;Decode 001
     + /PR* LD* POL*       /A            ;Load true
     + /PR* LD* POL*    B                ;Load true
     + /PR* LD* POL* C                   ;Load true
     + /PR*/LD*/TOG*/Q1                  ;Hold
     + /PR*/LD* TOG* Q1                  ;Toggle polarity

/Q2 :=  CLR                              ;Clear Q2
     + /PR* LD*/POL*/C* B*/A             ;Decode 010
     + /PR* LD* POL*       A             ;Load true
     + /PR* LD* POL*   /B                ;Load true
     + /PR* LD* POL* C                   ;Load true
     + /PR*/LD*/TOG*/Q2                  ;Hold
     + /PR*/LD* TOG* Q2                  ;Toggle polarity

/Q3 :=  CLR                              ;Clear Q3
     + /PR* LD*/POL*/C* B* A             ;Decode 011
     + /PR* LD* POL*       /A            ;Load true
     + /PR* LD* POL*   /B                ;Load true
     + /PR* LD* POL* C                   ;Load true
     + /PR*/LD*/TOG*/Q3                  ;Hold
     + /PR*/LD* TOG* Q3                  ;Toggle polarity

/Q4 :=  CLR                              ;Clear Q4
     + /PR* LD*/POL* C*/B*/A             ;Decode 100
     + /PR* LD* POL*       A             ;Load true
     + /PR* LD* POL*    B                ;Load true
     + /PR* LD* POL*/C                   ;Load true
     + /PR*/LD*/TOG*/Q4                  ;Hold
     + /PR*/LD* TOG* Q4                  ;Toggle polarity

/Q5 :=  CLR                              ;Clear Q5
     + /PR* LD*/POL* C*/B* A             ;Decode 101
     + /PR* LD* POL*       /A            ;Load true
     + /PR* LD* POL*    B                ;Load true
     + /PR* LD* POL*/C                   ;Load true
     + /PR*/LD*/TOG*/Q5                  ;Hold
     + /PR*/LD* TOG* Q5                  ;Toggle polarity

/Q6 :=  CLR                              ;Clear Q6
     + /PR* LD*/POL* C* B*/A             ;Decode 110
     + /PR* LD* POL*       A             ;Load true
     + /PR* LD* POL*   /B                ;Load true
     + /PR* LD* POL*/C                   ;Load true
     + /PR*/LD*/TOG*/Q6                  ;Hold
     + /PR*/LD* TOG* Q6                  ;Toggle polarity

/Q7 :=  CLR                              ;Clear Q7
     + /PR* LD*/POL* C* B* A             ;Decode 111
     + /PR* LD* POL*       /A            ;Load true
     + /PR* LD* POL*   /B                ;Load true
     + /PR* LD* POL*/C                   ;Load true
     + /PR*/LD*/TOG*/Q7                  ;Hold
     + /PR*/LD* TOG* Q7                  ;Toggle polarity

SIMULATION

TRACE_ON /OC /CLR /PR /LD POL TOG C B A
         Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

SETF OC CLR PR LD POL /TOG               ;Clear
CLOCKF CLK
SETF /CLR                                ;Preset
CLOCKF CLK
SETF /PR /C /B /A                        ;Load 0
CLOCKF CLK
SETF A                                   ;Load 1
CLOCKF CLK
SETF B /A                                ;Load 2
CLOCKF CLK
SETF A                                   ;Load 3
CLOCKF CLK
SETF /LD                                 ;Hold
SETF TOG                                 ;Toggle polarity
CLOCKF CLK
CLOCKF CLK                               ;Toggle polarity

SETF /POL LD /C /B /A                    ;Load 0 complement
CLOCKF CLK
SETF A                                   ;Load 1 complement
CLOCKF CLK
SETF /OC                                 ;Test HI-Z
CLOCKF CLK

;Function Table for PALASM1

;/OC CLK /CLR /PR /LD POL TOG C B A Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

;Control   Functions   Polarity  Input  Output
;/OC CLK /CLR /PR /LD  POL  TOG   CBA  Q7----Q0  Comments
;-------------------------------------------------------------
; L   C    L   L   L    H    L    XXX  LLLLLLLL  Clear
; L   C    H   L   L    H    L    XXX  HHHHHHHH  PRESET
; L   C    H   H   L    H    X    LLL  LLLLLLLH  Load 0
; L   C    H   H   L    H    X    LLH  LLLLLLHL  Load 1
; L   C    H   H   L    H    X    LHL  LLLLHLLL  Load 2
; L   C    H   H   L    H    X    LHH  LLLHLLLL  Load 3
; L   C    H   H   L    H    X    HLL  LLLHLLLL  Load 4
; L   C    H   H   L    H    X    HLH  LLHLLLLL  Load 5
; L   C    H   H   L    H    X    HHL  LHLLLLLL  Load 6
; L   C    H   H   L    H    X    HHH  HLLLLLLL  Load 7
; L   C    H   H   H    X    L    XXX  HLLLLLLL  Hold 7
; L   C    H   H   H    X    H    XXX  LHHHHHHH  Hold
; L   C    H   H   H    X    H    XXX  HLLLLLLL  Hold
; L   C    H   H   L    L    X    LLL  HHHHHHHL  Load 0
; L   C    H   H   L    L    X    LLH  HHHHHHLH  Load 1
; L   C    H   H   L    L    X    LHL  HHHHHLHH  Load 2
; L   C    H   H   L    L    X    LHH  HHHHLHHH  Load 3
; L   C    H   H   L    L    X    HLL  HHHLHHHH  Load 4
; L   C    H   H   L    L    X    HLH  HHLHHHHH  Load 5
; L   C    H   H   L    L    X    HHL  HLHHHHHH  Load 6
; L   C    H   H   L    L    X    HHH  LHHHHHHH  Load 7
; L   C    H   H   H    X    L    XXX  LHHHHHHH  Hold 7
; L   C    H   H   H    X    H    XXX  HLLLLLLL  Hold
; L   C    H   H   H    X    H    XXX  LHHHHHHH  Hold
; H   X    X   X   X    X    X    XXX  ZZZZZZZZ  Test HI-Z
;-------------------------------------------------------------
```

Simulation Results

```
Page :  1
        g  cgcgcgc gcgcgc cgc gcg   c
/OC   LLLLLLLLL LLLLLLLLL LLHHHHH
/CLR  LLLLHHHHH HHHHHHHHH HHHHHH
/PR   LLLLLLHHH HHHHHHHHH HHHHHH
/LD   LLLLLLLLL LLLLHHHHL LLLLLL
 POL  HHHHHHHHH HHHHHHHL LLLLLL
 TOG  LLLLLLLLL LLLLHHHHH HHHHHH
 C    XXXXXXLLL LLLLLLLLL LLLLLL
 B    XXXXXXLLL HHHHHHHHL LLLLLL
 A    XXXXXXLLH LLHHHHHHL HHHHHH
 Q7   XXXLLHHLL LLLLLHHLH HHHZZZ
 Q6   XXXLLHHLL LLLLLHHLH HHHZZZ
 Q5   XXXLLHHLL LLLLLHHLH HHHZZZ
 Q4   XXXLLHHLL LLLLLHHLH HHHZZZ
 Q3   XXXLLHHLL LLLHHLLHH HHHZZZ
 Q2   XXXLLHHLL LHHLLHHLH HHHZZZ
 Q1   XXXLLHHLH HLLLLHHLH HLLZZZ
 Q0   XXXLLHHHL LLLLLHHLL LHHZZZ
```

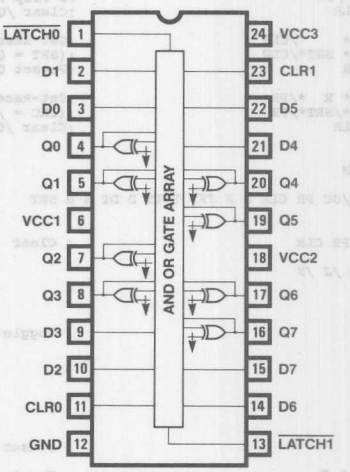
PAL Device Design Specification

```
Title     Octal Latch
Pattern   8latch.pds
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories Inc. Santa Clara, CA
Date      1/15/85

CHIP OctalLatch PAL20P8E

/LATCH0 D1 D0 Q0 Q1 VCC1 Q2 Q3 D3 D2 CLR0 GND
/LATCH1 D6 D7 Q7 Q6 VCC2 Q5 Q4 D4 D5 CLR1 VCC3

EQUATIONS

Q0 = D0 * /CLR0 * LATCH0                ;Load
   + D0 * /CLR0 * Q0                    ;Transition
   + Q0 * /CLR0 * /LATCH0               ;Hold

Q1 = D1 * /CLR0 * LATCH0                ;Load
   + D1 * /CLR0 * Q1                    ;Transition
   + Q1 * /CLR0 * /LATCH0               ;Hold

Q2 = D2 * /CLR0 * LATCH0                ;Load
   + D2 * /CLR0 * Q2                    ;Transition
   + Q2 * /CLR0 * /LATCH0               ;Hold

Q3 = D3 * /CLR0 * LATCH0                ;Load
   + D3 * /CLR0 * Q3                    ;Transition
   + Q3 * /CLR0 * /LATCH0               ;Hold

Q4 = D4 * /CLR1 * LATCH1                ;Load
   + D4 * /CLR1 * Q4                    ;Transition
   + Q4 * /CLR1 * /LATCH1               ;Hold

Q5 = D5 * /CLR1 * LATCH1                ;Load
   + D5 * /CLR1 * Q5                    ;Transition
   + Q5 * /CLR1 * /LATCH1               ;Hold

Q6 = D6 * /CLR1 * LATCH1                ;Load
   + D6 * /CLR1 * Q6                    ;Transtion
   + Q6 * /CLR1 * /LATCH1               ;Hold

Q7 = D7 * /CLR1 * LATCH1                ;Load
   + D7 * /CLR1 * Q7                    ;Transition
   + Q7 * /CLR1 * /LATCH1               ;Hold

SIMULATION

TRACE_ON CLR0 LATCH0 Q0 Q1 Q2 Q3 CLR1 LATCH1 Q4 Q5 Q6 Q7

SETF CLR0 LATCH0 D0 D1 D2 D3 D4 D5 D6 D7 VCC1 VCC2 VCC3

SETF CLR1 LATCH1                        ;Clear Latch

SETF /CLR0 /CLR1                        ;Clear Latches

SETF LATCH0 LATCH1
CHECK Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7

SETF /LATCH1 D0 /D1 /D2 D3

SETF /LATCH0 LATCH1 /D4 /D5 /D6 D7

SETF CLR0 CLR1                          ;Clear Latches

SETF /CLR0 /CLR1 /LATCH0 /LATCH1        ;Hold values

SETF CLR0                               ;Clear Q0,Q1,Q2,Q3

SETF CLR1                               ;Clear Q4,Q5,Q6,Q7

;The octal latch is an 8-bit latch with load, hold and clear
;capability. Clear sets all outputs to low and overrides
;hold. Load operation loads inputs (D0-D7) into the latch.
;The hold operation holds the previous values of (Q0-Q7).
```

Simulation Results

```
          gggggggggg
CLR0      HHLLLHLHH
LATCH0    HHHHLLLLL
Q0        LLHHHLHLL
Q1        LLHLLLLLL
Q2        LLHLLLLLL
Q3        LLHHHLHLL
CLR1      XHLLLHLLH
LATCH1    XHHLHHLLL
Q4        XLHHLLLLL
Q5        XLHHLLLLL
Q6        XLHHLLLLL
Q7        XLHHHLHHL
```

Logic Symbol



3

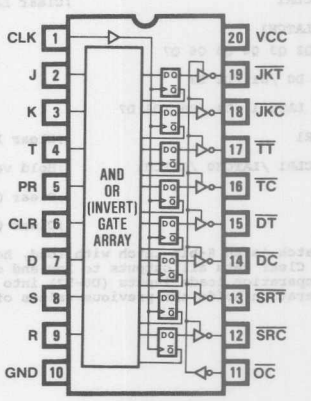
PAL Device Design Specification

```
Title     Basic Flip Flops
Pattern   FlipFlop.pds
Revision A
Author    Vincent Coli
Company   Monolithic Memories Inc., Santa Clara, CA
Date      2/28/85

CHIP FlipFlop PAL16RP8

CLK J K T PR CLR D S R GND
/OC /SRC /SRT /DC /DT /TC /TT /JKC /JKT VCC

EQUATIONS

JKT :=  J*/JKT*/CLR                    ;JK Flio-Flop
     + /K* JKT*/CLR                    ;(JKC = /Q)
     +  PR                             ;Preset Q

JKC := /J* K  */PR                     ;JK Flip-Flop
     + /J*/JKT*/PR                     ;(JK = /Q)
     +  K* JKT*/PR
     +  CLR                           ;Clear /Q

TT  :=  T*/TT*/CLR                     ;T Flip-Flop
     + /T* TT*/CLR                     ;(TT = Q)
     +  PR                            ;Preset Q

TC  := /T*/TT*/CLR                     ;T Flip-Flop
     +  T* TT*/PR                     ;(TC = /Q)
     +  CLR                           ;Clear /Q

DT  :=  D*/CLR                         ;D Flip-Flop
     +  PR                            ;Preset Q

DC  := /D*/PR                          ;D Flip-Flop
     +  CLR                           ;Clear /Q

SRT :=  S*     /CLR                    ;Set-Reset Flip-Flop
     + /R* SRT*/CLR                    ;(SRT = Q)
     +  PR                            ;Preset Q

SRC := /S* R  */PR                     ;Set-Reset Flip-Flop
     + /S*/SRT*/PR                     ;(SRC = /Q)
     +  CLR                           ;Clear /Q


SIMULATION

TRACE_ON /OC PR CLR J K JKT T TT D DT S R SRT


SETF OC /PR CLR                        ; Clear
CLOCKF
SETF /CLR /J /K
CLOCKF
SETF K
CLOCKF
SETF J                                 ; Toggle
CLOCKF
SETF /K
CLOCKF
SETF /J
CLOCKF
SETF K
CLOCKF
SETF PR                                ; Preset
CLOCKF
SETF /PR J K                           ; Toggle
CLOCKF
SETF /K
CLOCKF


SETF CLR                               ; Clear
CLOCKF
SETF /CLR /T
CLOCKF
SETF T                                 ; Toggle
CLOCKF
SETF /T                                ; Toggle
CLOCKF


SETF CLR                               ; Clear
CLOCKF
SETF /D
CLOCKF
SETF D
CLOCKF
SETF /D
CLOCKF
SETF CLR                               ; Clear
CLOCKF
SETF /CLR /S /R
CLOCKF
SETF S                                 ; Set
CLOCKF
SETF /S R                              ; Reset
CLOCKF
SETF /S R                              ; Hold
CLOCKF
SETF /OC                               ; HI-Z
CLOCKF
```

Simulation Results

```
     g  cg cg c g cg cg cg  cg cg cg  cg cg cg c
/OC  LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 PR  LLLLLLLLLL LLLLLLLLLL LLHHHLLLLL LLLLLLLLLL
 CLR HHHHLLLLLL LLLLLLLLLL LLLLLLLLLL LHHHLLLLLL
 J   XXXXLLLLLL HHHHHHLLLL LLLLLHHHHH HHHHHHHHHH
 K   XXXXLLLHHH HHHLLLLLLH HHHHHHHHLL LLLLLLLLLL
 JKT XXXLLLLLLL LLHHHHHHHH HLLLHHHLLL HHHLLLHHHH
 T   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXLLLHHH
 TT  XXXLLLXXXX XXXXXXXXXX XXXXHHHXXX XXXLLLLLLH
 D   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
 DT  XXXLLLXXXX XXXXXXXXXX XXXXHHHXXX XXXLLLXXXX
 S   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
 R   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
 SRT XXXLLLXXXX XXXXXXXXXX XXXXHHHXXX XXXLLLXXXX

Page :  2
     g cg cg cg  cg cgcg c g cg cgcg
/OC  LLLLLLLLLL LLLLLLLLLL LLLLLLLLHH
 PR  LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 CLR LLLHHHHHHH HHHHHHLLL LLLLLLLLLL
 J   HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 K   LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 JKT HHHHHLLLLL LLLLLLLLLH HHHHHHHHHZ
 T   LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 TT  HHHHHLLLLL LLLLLLLLLL LLLLLLLLLZ
 D   XXXXXXLLLH HHLLLLLLLL LLLLLLLLLL
 DT  XXXXXLLLLL LLLLLLLLLL LLLLLLLLLZ
 S   XXXXXXXXXX XXXXXXXLLL HHHLLLLLLL
 R   XXXXXXXXXX XXXXXXXLLL LLLHHHHHHH
 SRT XXXXXLLLLL LLLLLLLLLL LLHHHLLLLZ
```

Logic Symbol



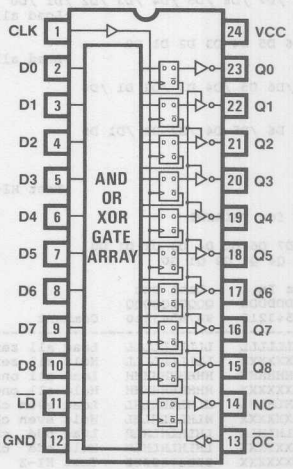
PAL Device Design Specification

```
Title     9BitRegister
Pattern   9BitReg.pds
Revision A
Author    Vincent Coli/Mehrnaz Hada
Company   Monolithic Memories Inc., Santa Clara, CA
Date      1/30/85

;This is a design of a 9-bit register with parallel load
;and hold capabilities. The operations of this register are
;summarized in the following operations table:
;
;       /OC   CLK   /LD   D8-D0   Q8-Q0     Operation
;       ---------------------------------------------
;        H     X     X      X       Z       HI-Z
;        L     1     H      X       Q       Hold
;        L     1     L      D       D       Load
;       ---------------------------------------------

CHIP 9BitRegister PAL20X10

CLK D0 D1 D2 D3 D4 D5 D6 D7 D8 /LD GND
/OC NC Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1  Q0 VCC

EQUATIONS

/Q0 := /D0* LD                          ;Load D0
     + /Q0*/LD                          ;Hold Q0

/Q1 := /D1* LD                          ;Load D1
     + /Q1*/LD                          ;Hold Q1

/Q2 := /D2* LD                          ;Load D2
     + /Q2*/LD                          ;Hold Q2

/Q3 := /D3* LD                          ;Load D3
     + /Q3*/LD                          ;Hold Q3

/Q4 := /D4* LD                          ;Load D4
     + /Q4*/LD                          ;Hold Q4

/Q5 := /D5* LD                          ;Load D5
     + /Q5*/LD                          ;Hold Q5

/Q6 := /D6* LD                          ;Load D6
     + /Q6*/LD                          ;Hold Q6

/Q7 := /D7* LD                          ;Load D7
     + /Q7*/LD                          ;Hold Q7

/Q8 := /D8* LD                          ;Load D8
     + /Q8*/LD                          ;Hold Q8

SIMULATION

TRACE_ON /OC CLK /LD D8 D7 D6 D5 D4 D3 D2 D1 D0
         Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

SETF  OC LD /D8 /D7 /D6 /D5 /D4 /D3     ;Load zeros
     /D2 /D1 /D0
CLOCKF CLK

SETF /LD                                ;Hold zeros
CLOCKF CLK

SETF LD D8 D7 D6 D5 D4 D3 D2 D1 D0      ;Load ones
CLOCKF CLK

SETF /LD                                ;Hold ones
CLOCKF CLK

SETF LD /D8 D7 /D6 D5 /D4 D3 /D2 D1 /D0
CLOCKF CLK

SETF /LD                                ;Hold even ones
CLOCKF CLK

SETF  LD D8 /D7 D6 /D5 D4 /D3 D2 /D1 D0
CLOCKF CLK

SETF /LD                                ;Hold odd ones
CLOCKF CLK

SETF OC
CLOCKF CLK                              ;Test HI-Z

;Function Table for PALASM1

;/OC CLK D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
;Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
```

Simulation Results

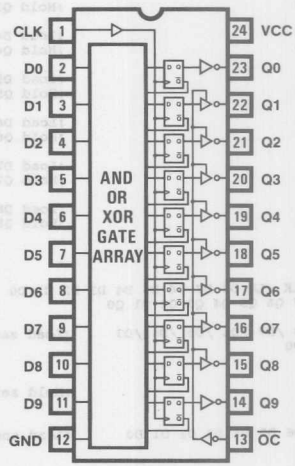
```
;            Data In      Data Out
;Control     DDDDDDDDDD   QQQQQQQQQQ
;/OC CLK     9876543210   9876543210   Comment
;-----------------------------------------------------------
; L   C      LLLLLLLLLL   LLLLLLLLLL   Load all zeros
; L   L      XXXXXXXXXX   LLLLLLLLLL   Hold all zeros
; L   C      HHHHHHHHHH   HHHHHHHHHH   Load all ones
; L   L      XXXXXXXXXX   HHHHHHHHHH   Hold all ones
; L   C      HLHLHLHLHL   HLHLHLHLHL   Load even checkerboard
; L   L      XXXXXXXXXX   HLHLHLHLHL   Hold even checkerboard
; L   C      LHLHLHLHLH   LHLHLHLHLH   Load odd  checkerboard
; L   L      XXXXXXXXXX   LHLHLHLHLH   Hold odd  checkerboard
; H   X      XXXXXXXXXX   ZZZZZZZZZZ   Test HI-Z
;-----------------------------------------------------------
```

```
Page :  1
        g  cgcgcgc gcgcgcgcg    c
 /OC   LLLLLLLLL LLLLLLLHH HHH
  CLK XXHLHLHLHL HLHLHLHLL HHL
 /LD   LLLLHHLLHH LLHHLLHHHH HHH
  D8   LLLLLLHHHH LLLLHHHHHH HHH
  D7   LLLLLLHHHH HHHHLLLLLL LLL
  D6   LLLLLLHHHH LLLLHHHHHH HHH
  D5   LLLLLLHHHH HHHHLLLLLL LLL
  D4   LLLLLLHHHH LLLLHHHHHH HHH
  D3   LLLLLLHHHH HHHHLLLLLL LLL
  D2   LLLLLLHHHH LLLLHHHHHH HHH
  D1   LLLLLLHHHH HHHHLLLLLL LLL
  D0   LLLLLLHHHH LLLLHHHHHH HHH
  Q8   XXXLLLLHHH HLLLLHHHHZ ZZZ
  Q7   XXXLLLLHHH HHHHHLLLLZ ZZZ
  Q6   XXXLLLLHHH HLLLLHHHHZ ZZZ
  Q5   XXXLLLLHHH HHHHHLLLLZ ZZZ
  Q4   XXXLLLLHHH HLLLLHHHHZ ZZZ
  Q3   XXXLLLLHHH HHHHHLLLLZ ZZZ
  Q2   XXXLLLLHHH HLLLLHHHHZ ZZZ
  Q1   XXXLLLLHHH HHHHHLLLLZ ZZZ
  Q0   XXXLLLLHHH HLLLLHHHHZ ZZZ
```



```
Revision A
Author   Vincent Coli/Mehrnaz Hada
Company  Monolithic Memories Inc., Santa Clara, CA
Date     1/28/85

;The 10-bit register loads the data (D9-D0) on the rising
;edge of the clock(CLK) into the register(Q9-Q0). The data
;is held in the register until the next posiyive edge of
;the clock.

;
;      /OC  CLK  D9-D0  Q9-Q0    Operation
;      ------------------------------------
;       H    X     X       Z      HI-Z
;       L    C     D       D      Load
;       L    L     X       Q      Hold
;      ------------------------------------


CHIP 10BitReg PAL20X10

CLK D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 GND
/OC Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 VCC

EQUATIONS

/Q0 := /D0                              ;Load D0

/Q1 := /D1                              ;Load D1

/Q2 := /D2                              ;Load D2

/Q3 := /D3                              ;Load D3

/Q4 := /D4                              ;Load D4

/Q5 := /D5                              ;Load D5

/Q6 := /D6                              ;Load D6

/Q7 := /D7                              ;Load D7

/Q8 := /D8                              ;Load D8

/Q9 := /D9                              ;Load D9


SIMULATION

TRACE_ON /OC CLK Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
         D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

SETF OC /D9 /D8 /D7 /D6 /D5 /D4 /D3 /D2 /D1 /D0
CLOCKF CLK                              ;Load all zeros

SETF D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
CLOCKF CLK                              ;Load all ones

SETF D9 /D8 D7 /D6 D5 /D4 D3 /D2 D1 /D0
CLOCKF CLK

SETF /D9 D8 /D7 D6 /D5 D4 /D3 D2 /D1 D0
CLOCKF CLK

SETF /OC
CLOCKF CLK                              ;Test HI-Z

;Function Table for PALASM1

;/OC CLK D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
;Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

;            Data In      Data Out
;Control     DDDDDDDDDD   QQQQQQQQQQ
;/OC CLK     9876543210   9876543210   Comment
;-------------------------------------------------------
; L   C      LLLLLLLLLL   LLLLLLLLLL   Load all zeros
; L   L      XXXXXXXXXX   LLLLLLLLLL   Hold all zeros
; L   C      HHHHHHHHHH   HHHHHHHHHH   Load all ones
; L   L      XXXXXXXXXX   HHHHHHHHHH   Hold all ones
; L   C      HLHLHLHLHL   HLHLHLHLHL   Load even checkerboard
; L   L      XXXXXXXXXX   HLHLHLHLHL   Hold even checkerboard
; L   C      LHLHLHLHLH   LHLHLHLHLH   Load odd  checkerboard
; L   L      XXXXXXXXXX   LHLHLHLHLH   Hold odd  checkerboard
; H   X      XXXXXXXXXX   ZZZZZZZZZZ   Test HI-Z
;-------------------------------------------------------
```

```
          g   cg cg   cg cg   c
/OC  LLLLLLLLLL LLLLLHHHHH
 CLK XXHHLLHLLH HLLHLLLHHL
  Q9 XXXLLLLHHH HHHHLLZZZZ
  Q8 XXXLLLLHHH LLLLHHZZZZ
  Q7 XXXLLLLHHH HHHHLLZZZZ
  Q6 XXXLLLLHHH LLLLHHZZZZ
  Q5 XXXLLLLHHH HHHHLLZZZZ
  Q4 XXXLLLLHHH LLLLHHZZZZ
  Q3 XXXLLLLHHH HHHHLLZZZZ
  Q2 XXXLLLLHHH LLLLHHZZZZ
  Q1 XXXLLLLHHH HHHHLLZZZZ
  Q0 XXXLLLLHHH LLLLHHZZZZ
  D9 LLLLLHHHHH HHLLLLLLLL
  D8 LLLLLHHHLL LLHHHHHHHH
  D7 LLLLLHHHHH HHLLLLLLLL
  D6 LLLLLHHHLL LLHHHHHHHH
  D5 LLLLLHHHHH HHLLLLLLLL
  D4 LLLLLHHHLL LLHHHHHHHH
  D3 LLLLLHHHHH HHLLLLLLLL
  D2 LLLLLHHHLL LLHHHHHHHH
  D1 LLLLLHHHHH HHLLLLLLLL
  D0 LLLLLHHHLL LLHHHHHHHH
```



```
Pattern   Barrel.pds
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories Inc. Santa Clara, CA
Date      1/15/85

;The 16-bit barrel shifter will shift 16 bits of data
;(D15-D0) a number of locations into the output pins, as
;specified by the binary encoded input. A compacted
;equation can be used to specify this design. It can be
;specified as following:
;
;Q[J=0..15] :=
;      OR[K=0..15](D[(J+K)-((J+K)/16)*16]*BIN[K,I=3..0]S(I))
;
;Inputs are shown by D. Si are shift amount inputs and
;Qj are outputs. 16 product terms in each output pair
;are directed to one output; thus only 16 out of 32
;output pins are used.

CHIP BarrelShift PAL64R32

D7 D6 D5 D4 D3 D2 D1 D0 /PL1 /PS1 GND CLK1
/OC1 Q0 NC Q1 NC Q2 NC Q3 NC Q4 NC Q5 NC Q6
NC Q7 NC /OC2 CLK2 VCC /PS2 /PL2 NC NC NC
NC NC S0 S1 S2 S3 NC NC NC NC NC NC NC
/PL3 /PS3 GND CLK3 /OC3 NC Q8 NC Q9 NC Q10
NC Q11 NC Q12 NC Q13 NC Q14 NC Q15 /OC4
CLK4 VCC /PS4 /PL4 D15 D14 D13 D12 D11 D10
D9 D8

EQUATIONS

Q0 := /S3 * /S2 * /S1 * /S0 * D0        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D1        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D2        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D3        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D4        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D5        ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D6        ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D7        ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D8        ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D9        ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D10       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D11       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D12       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D13       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D14       ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D15       ; Shift 15 spaces

Q1 := /S3 * /S2 * /S1 * /S0 * D1        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D2        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D3        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D4        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D5        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D6        ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D7        ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D8        ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D9        ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D10       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D11       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D12       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D13       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D14       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D15       ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D0        ; Shift 15 spaces

Q2 := /S3 * /S2 * /S1 * /S0 * D2        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D3        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D4        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D5        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D6        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D7        ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D8        ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D9        ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D10       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D11       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D12       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D13       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D14       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D15       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D0        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D1        ; Shift 15 spaces

Q3 := /S3 * /S2 * /S1 * /S0 * D3        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D4        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D5        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D6        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D7        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D8        ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D9        ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D10       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D11       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D12       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D13       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D14       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D15       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D0        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D1        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D2        ; Shift 15 spaces

Q4 := /S3 * /S2 * /S1 * /S0 * D4        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D5        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D6        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D7        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D8        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D9        ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D10       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D11       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D12       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D13       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D14       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D15       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D0        ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D1        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D2        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D3        ; Shift 15 spaces

Q5 := /S3 * /S2 * /S1 * /S0 * D5        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D6        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D7        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D8        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D9        ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D10       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D11       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D12       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D13       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D14       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D15       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D0        ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D1        ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D2        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D3        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D4        ; Shift 15 spaces

Q6 := /S3 * /S2 * /S1 * /S0 * D6        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D7        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D8        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D9        ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D10       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D11       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D12       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D13       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D14       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D15       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D0        ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D1        ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D2        ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D3        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D4        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D5        ; Shift 15 spaces

Q7 := /S3 * /S2 * /S1 * /S0 * D7        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D8        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D9        ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D10       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D11       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D12       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D13       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D14       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D15       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D0        ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D1        ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D2        ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D3        ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D4        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D5        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D6        ; Shift 15 spaces

Q8 := /S3 * /S2 * /S1 * /S0 * D8        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D9        ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D10       ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D11       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D12       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D13       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D14       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D15       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D0        ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D1        ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D2        ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D3        ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D4        ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D5        ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D6        ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D7        ; Shift 15 spaces

Q9 := /S3 * /S2 * /S1 * /S0 * D9        ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D10       ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D11       ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D12       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D13       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D14       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D15       ; Shift 6 spaces
```

```
   +  /S3 *  S2 *  S1 *  S0 * D0       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D1       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D2       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D3       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D4       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D5       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D6       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D7       ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D8       ; Shift 15 spaces

Q10 :=/S3 * /S2 * /S1 * /S0 * D10      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D11      ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D12      ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D13      ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D14      ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D15      ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D0       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D1       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D2       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D3       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D4       ; Shift 10 Spaces
   +   S3 * /S2 *  S1 *  S0 * D5       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D6       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D7       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D8       ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D9       ; Shift 15 spaces

Q11 :=/S3 * /S2 * /S1 * /S0 * D11      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D12      ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D13      ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D14      ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D15      ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D0       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D1       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D2       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D3       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D4       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D5       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D6       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D7       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D8       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D9       ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D10      ; Shift 15 spaces

Q12 :=/S3 * /S2 * /S1 * /S0 * D12      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D13      ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D14      ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D15      ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D0       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D1       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D2       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D3       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D4       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D5       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D6       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D7       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D8       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D9       ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D10      ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D11      ; Shift 15 spaces

Q13 :=/S3 * /S2 * /S1 * /S0 * D13      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D14      ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D15      ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D0       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D1       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D2       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D3       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D4       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D5       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D6       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D7       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D8       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D9       ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D10      ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D11      ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D12      ; Shift 15 spaces

Q14 :=/S3 * /S2 * /S1 * /S0 * D14      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D15      ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D0       ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D1       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D2       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D3       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D4       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D5       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D6       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D7       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D8       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D9       ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D10      ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D11      ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D12      ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D13      ; Shift 15 spaces

Q15 :=/S3 * /S2 * /S1 * /S0 * D15      ; No shift
   +  /S3 * /S2 * /S1 *  S0 * D0       ; Shift 1 space
   +  /S3 * /S2 *  S1 * /S0 * D1       ; Shift 2 spaces
   +  /S3 * /S2 *  S1 *  S0 * D2       ; Shift 3 spaces
   +  /S3 *  S2 * /S1 * /S0 * D3       ; Shift 4 spaces
   +  /S3 *  S2 * /S1 *  S0 * D4       ; Shift 5 spaces
   +  /S3 *  S2 *  S1 * /S0 * D5       ; Shift 6 spaces
   +  /S3 *  S2 *  S1 *  S0 * D6       ; Shift 7 spaces
   +   S3 * /S2 * /S1 * /S0 * D7       ; Shift 8 spaces
   +   S3 * /S2 * /S1 *  S0 * D8       ; Shift 9 spaces
   +   S3 * /S2 *  S1 * /S0 * D9       ; Shift 10 spaces
   +   S3 * /S2 *  S1 *  S0 * D10      ; Shift 11 spaces
   +   S3 *  S2 * /S1 * /S0 * D11      ; Shift 12 spaces
   +   S3 *  S2 * /S1 *  S0 * D12      ; Shift 13 spaces
   +   S3 *  S2 *  S1 * /S0 * D13      ; Shift 14 spaces
   +   S3 *  S2 *  S1 *  S0 * D14      ; Shift 15 spaces

SIMULATION

TRACE_ON CLK1 CLK2 CLK3 CLK4 OC1 OC2 OC3 OC4
         PL1 PL2 PL3 PL4 PS1 PS2 PS3 PS4 S3
         S2 S1 S0 D0 D1 D2 D3 D4 D5 D6 D7 D8
         D9 D10 D11 D12 D13 D14 D15 Q0 Q1 Q2
         Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13
         Q14 Q15

SETF OC1 OC2 OC3 OC4 /PS1 /PS2 /PS3 /PS4
    /PL1 /PL2 /PL3 /PL4 /S3 /S2 /S1 /S0 D0
    /D1 /D2 /D3 /D4 /D5 /D6 /D7 /D8 /D9 /D10
    /D11 /D12 /D13 /D14 /D15

CLOCKF CLK1 CLK2 CLK3 CLK4                    ;Clock

SETF /S3 /S2 /S1 S0                           ;Shift 1
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 /S2 S1 /S0                           ;Shift 2
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 /S2 S1 S0                            ;Shift 3
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 S2 /S1 /S0                           ;Shift 4
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 S2 /S1 S0                            ;Shift 5
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 S2 S1 /S0                            ;Shift 6
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF /S3 S2 S1 S0                             ;Shift 7
CLOCKF CLK1 CLK2 CLK3 CLK4

SETF S3 /S2 /S1 /S0                           ;Shift 8
CLOCKF CLK1 CLK2 CLK3 CLK4

;The 16-bit barrel shifter will shift 16 bits of data
;(D15-D0) a number of locations into the output pins, as
;specified by the binary encoded input. A compacted
;equation can be used to specify this design. It can be
;specified as following:
;
;Q[J=0..15] :=
;  OR[K=0..15]{D[(J+K)-((J+K)/16)*16]*BIN[K,I=3..0]S(I)}
;
;Inputs are shown by D. Si are shift amount inputs and
;Qj are outputs. 16 product terms in each output pair
;are directed to one output; thus only 16 out of 32
;output pins are used.
```

Simulation Results

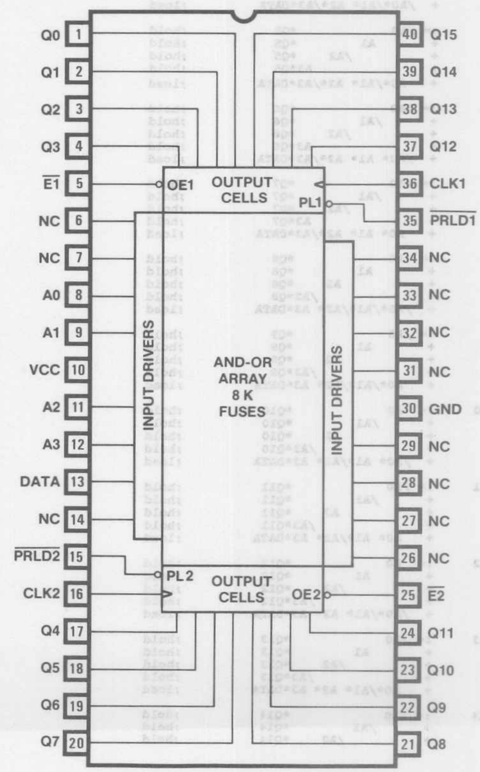
PAL Device Design Specification

```
TITLE     16-BIT ADDRESSABLE REGISTER
PATTERN   ADREG16.PDS
REVISION  A
AUTHOR    John Birkner
COMPANY   Monolithic Memories Inc. Santa Clara, CA
DATE      2/11/85

;       The 16-bit addressable register loads one of 16 registers
;       selected by ADDR[0..3] with data input, DATA.


CHIP ADREG16 PAL32R16
Q0 Q1 Q2 Q3 /E1 NC NC A0 A1 VCC A2 A3 DATA NC /PRLD2 CLK2
Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 /E2 NC NC NC NC GND NC NC NC NC /PRLD1 CLK1
Q12 Q13 Q14 Q15


EQUATIONS

Q0      :=  A0              *Q0           ;hold
        +       A1          *Q0           ;hold
        +           A2      *Q0           ;hold
        +             A3*Q0               ;hold
        +  /A0*/A1*/A2*/A3*DATA           ;load

Q1      := /A0              *Q1           ;hold
        +       A1          *Q1           ;hold
        +           A2      *Q1           ;hold
        +             A3*Q1               ;hold
        +   A0*/A1*/A2*/A3*DATA           ;load

Q2      :=  A0              *Q2           ;hold
        +      /A1          *Q2           ;hold
        +           A2      *Q2           ;hold
        +             A3*Q2               ;hold
        +  /A0* A1*/A2*/A3*DATA           ;load

Q3      := /A0              *Q3           ;hold
        +      /A1          *Q3           ;hold
        +           A2      *Q3           ;hold
        +             A3*Q3               ;hold
        +   A0* A1*/A2*/A3*DATA           ;load

Q4      :=  A0              *Q4           ;hold
        +       A1          *Q4           ;hold
        +          /A2      *Q4           ;hold
        +             A3*Q4               ;hold
        +  /A0*/A1* A2*/A3*DATA           ;load

Q5      := /A0              *Q5           ;hold
        +       A1          *Q5           ;hold
        +          /A2      *Q5           ;hold
        +             A3*Q5               ;hold
        +   A0*/A1* A2*/A3*DATA           ;load

Q6      :=  A0              *Q6           ;hold
        +      /A1          *Q6           ;hold
        +          /A2      *Q6           ;hold
        +             A3*Q6               ;hold
        +  /A0* A1* A2*/A3*DATA           ;load

Q7      := /A0              *Q7           ;hold
        +      /A1          *Q7           ;hold
        +          /A2      *Q7           ;hold
        +             A3*Q7               ;hold
        +   A0* A1* A2*/A3*DATA           ;load

Q8      :=  A0              *Q8           ;hold
        +       A1          *Q8           ;hold
        +           A2      *Q8           ;hold
        +             /A3*Q8              ;hold
        +  /A0*/A1*/A2* A3*DATA           ;load

Q9      := /A0              *Q9           ;hold
        +       A1          *Q9           ;hold
        +           A2      *Q9           ;hold
        +             /A3*Q9              ;hold
        +   A0*/A1*/A2* A3*DATA           ;load

Q10     :=  A0              *Q10          ;hold
        +      /A1          *Q10          ;hold
        +           A2      *Q10          ;hold
        +             /A3*Q10             ;hold
        +  /A0* A1*/A2* A3*DATA           ;load

Q11     := /A0              *Q11          ;hold
        +      /A1          *Q11          ;hold
        +           A2      *Q11          ;hold
        +             /A3*Q11             ;hold
        +   A0* A1*/A2* A3*DATA           ;load

Q12     :=  A0              *Q12          ;hold
        +       A1          *Q12          ;hold
        +          /A2      *Q12          ;hold
        +             /A3*Q12             ;hold
        +  /A0*/A1* A2* A3*DATA           ;load

Q13     := /A0              *Q13          ;hold
        +       A1          *Q13          ;hold
        +          /A2      *Q13          ;hold
        +             /A3*Q13             ;hold
        +   A0*/A1* A2* A3*DATA           ;load

Q14     :=  A0              *Q14          ;hold
        +      /A1          *Q14          ;hold
        +          /A2      *Q14          ;hold
        +             /A3*Q14             ;hold
        +  /A0* A1* A2* A3*DATA           ;load

Q15     := /A0              *Q15          ;hold
        +      /A1          *Q15          ;hold
        +          /A2      *Q15          ;hold
        +             /A3*Q15             ;hold
        +   A0* A1* A2* A3*DATA           ;load

SIMULATION

TRACE_ON Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15
         A0 A1 A2 A3 DATA

SETF E1 E2 /DATA /PRLD1 /PRLD2

        SETF /A0 /A1 /A2 /A3
        CLOCKF CLK1 CLK2

        SETF  A0 /A1 /A2 /A3
        CLOCKF CLK1 CLK2

        SETF /A0  A1 /A2 /A3
        CLOCKF CLK1 CLK2

        SETF  A0  A1 /A2 /A3
        CLOCKF CLK1 CLK2

        SETF /A0 /A1  A2 /A3
        CLOCKF CLK1 CLK2

        SETF  A0 /A1  A2 /A3
        CLOCKF CLK1 CLK2

        SETF /A0  A1  A2 /A3
        CLOCKF CLK1 CLK2

        SETF  A0  A1  A2 /A3
        CLOCKF CLK1 CLK2

        SETF /A0 /A1 /A2  A3
        CLOCKF CLK1 CLK2

        SETF  A0 /A1 /A2  A3
        CLOCKF CLK1 CLK2

        SETF /A0  A1 /A2  A3
        CLOCKF CLK1 CLK2

        SETF  A0  A1 /A2  A3
        CLOCKF CLK1 CLK2

        SETF /A0 /A1  A2  A3
        CLOCKF CLK1 CLK2

        SETF  A0 /A1  A2  A3
        CLOCKF CLK1 CLK2

        SETF /A0  A1  A2  A3
        CLOCKF CLK1 CLK2

        SETF  A0  A1  A2  A3
        CLOCKF CLK1 CLK2

        SETF DATA

        SETF /A0 /A1 /A2 /A3
        CLOCKF CLK1 CLK2
```

Simulation Results

```
Page :  1
        g g cgcgcg cgcgcgcgcg cgcgcgcgcg cgcgcgc
  Q0    XXXXLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLH
  Q1    XXXXXXLLLL LLLLLLLLLL LLLLLLLLLL LLLLLL
  Q2    XXXXXXXXLL LLLLLLLLLL LLLLLLLLLL LLLLLL
  Q3    XXXXXXXXXX LLLLLLLLLL LLLLLLLLLL LLLLLL
  Q4    XXXXXXXXXX XXLLLLLLLL LLLLLLLLLL LLLLLL
  Q5    XXXXXXXXXX XXXXLLLLLL LLLLLLLLLL LLLLLL
  Q6    XXXXXXXXXX XXXXXXLLLL LLLLLLLLLL LLLLLL
  Q7    XXXXXXXXXX XXXXXXXXLL LLLLLLLLLL LLLLLL
  Q8    XXXXXXXXXX XXXXXXXXXX LLLLLLLLLL LLLLLL
  Q9    XXXXXXXXXX XXXXXXXXXX XXLLLLLLLL LLLLLL
  Q10   XXXXXXXXXX XXXXXXXXXX XXXXLLLLLL LLLLLL
  Q11   XXXXXXXXXX XXXXXXXXXX XXXXXXLLLL LLLLLL
  Q12   XXXXXXXXXX XXXXXXXXXX XXXXXXXXLL LLLLLL
  Q13   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX LLLLLL
  Q14   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXLLLL
  Q15   XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXLL
  A0    XXLLLHHLLH HLLHHLLHHL LHHLLHHLLH HLLHHL
  A1    XXLLLLLHHH HLLLLHHHHL LLLHHHHLLL LHHHHL
  A2    XXLLLLLLLL LHHHHHHHHL LLLLLLHHH HHHHHL
  A3    XXLLLLLLLL LLLLLLLLLH HHHHHHHHHH HHHHHL
 DATA   LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLHH
```

```
(C) - COPYRIGHT MONLITHIC MEMORIES INC., 1984

Title     : 16-BIT Addressable Register
Pattern   : ADREG16.PDS
Revision  : A
Author    : John Birkner
Company   : Monolithic Memories Inc
Date      : 2/11/85

PAL32R16
ADREG16

            111111 11112222 22222233 33333333 44444444 44555555 55556
   01234567 89012345 67890123 45678901 23456789 01234567 89012345 67890

 0 -------- -------- -------- -------- -------- -------- -------- -X---
 1 -------- -------- -------- -------- -------- -------- -------- -----
 2 X------- -------- -------- -------- -------- -------- -------- -----
 3 ----X--- -------- -------- -------- -------- -------- -------- -----
 4 -X---X-- X------- -------- -------- -------- -------- -------- X---X
 5 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
 6 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
 7 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
 8 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
 9 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
10 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
11 -X---X-- X------- -------- -------- -------- -------- -------- -X--X
12 ----X--- -------- -------- -------- -------- -------- -------- --X--
13 X------- -------- -------- -------- -------- -------- -------- --X--
14 -------- -------- -------- -------- -------- -------- -------- --X--
15 -------- -------- -------- -------- -------- -------- -------- X-X--

16 -------- -------- -------- -------- -------- -------- ------X- -X---
17 -------- -------- -------- -------- -------- -------- ------X- ----X
18 X------- -------- -------- -------- -------- -------- ------X- -----
19 ----X--- -------- -------- -------- -------- -------- ------X- -----
20 -X---X-- X------- -------- -------- -------- -------- -------- X----
21 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
22 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
23 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
24 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
25 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
26 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
27 -X---X-- X------- -------- -------- -------- -------- -------- -X---
28 ----X--- -------- -------- -------- -------- -------- --X----- -----
29 X------- -------- -------- -------- -------- -------- --X----- -----
30 -------- -------- -------- -------- -------- -------- --X----- ----X
31 -------- -------- -------- -------- -------- -------- --X----- X----

32 -------- -------- -------- -------- -------- ------X- -------- -X---
33 -------- -------- -------- -------- -------- ------X- -------- -----
34 -X------ -------- -------- -------- -------- ------X- -------- -----
35 -----X-- -------- -------- -------- -------- ------X- -------- -----
36 X---X--- X------- -------- -------- -------- -------- -------- X---X
37 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
38 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
39 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
40 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
41 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
42 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
43 X---X--- X------- -------- -------- -------- -------- -------- -X--X
44 -----X-- -------- -------- -------- -------- --X----- -------- -----
45 -X------ -------- -------- -------- -------- --X----- -------- -----
46 -------- -------- -------- -------- -------- --X----- -------- -----
47 -------- -------- -------- -------- -------- --X----- -------- X----

48 -------- -------- -------- -------- ------X- -------- -------- -X---
49 -------- -------- -------- -------- ------X- -------- -------- ----X
50 -X------ -------- -------- -------- ------X- -------- -------- -----
51 -----X-- -------- -------- -------- ------X- -------- -------- -----
52 X---X--- X------- -------- -------- -------- -------- -------- X----
53 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
54 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
55 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
56 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
57 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
58 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
59 X---X--- X------- -------- -------- -------- -------- -------- -X---
60 -----X-- -------- -------- -------- --X----- -------- -------- -----
61 -X------ -------- -------- -------- --X----- -------- -------- -----
62 -------- -------- -------- -------- --X----- -------- -------- ----X
63 -------- -------- -------- -------- --X----- -------- -------- X----

64 -------- -------- -------- ------X- -------- -------- -------- -X---
65 -------- -------- -------- ------X- -------- -------- -------- -----
66 X------- -------- -------- ------X- -------- -------- -------- -----
67 -----X-- -------- -------- ------X- -------- -------- -------- -----
68 -X--X--- X------- -------- -------- -------- -------- -------- X---X
69 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
70 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
71 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
72 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
73 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
74 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
75 -X--X--- X------- -------- -------- -------- -------- -------- -X--X
76 -----X-- -------- -------- --X----- -------- -------- -------- -----
77 X------- -------- -------- --X----- -------- -------- -------- -----
78 -------- -------- -------- --X----- -------- -------- -------- -----
79 -------- -------- -------- --X----- -------- -------- -------- X----

80 -------- -------- ------X- -------- -------- -------- -------- -X---
81 -------- -------- ------X- -------- -------- -------- -------- ----X
82 X------- -------- ------X- -------- -------- -------- -------- -----
83 -----X-- -------- ------X- -------- -------- -------- -------- -----
84 -X--X--- X------- -------- -------- -------- -------- -------- X----
85 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
86 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
87 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
88 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
89 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
90 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
91 -X--X--- X------- -------- -------- -------- -------- -------- -X---
92 -----X-- -------- --X----- -------- -------- -------- -------- -----
93 X------- -------- --X----- -------- -------- -------- -------- -----
94 -------- -------- --X----- -------- -------- -------- -------- ----X
95 -------- -------- --X----- -------- -------- -------- -------- X----

[illegible] -------- -------- -X---
 97 -------- ------X- -------- -------- -------- -------- -------- -----
 98 -X------ ------X- -------- -------- -------- -------- -------- -----
 99 ----X--- ------X- -------- -------- -------- -------- -------- -----
100 X----X-- X------- -------- -------- -------- -------- -------- X---X
101 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
102 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
103 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
104 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
105 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
106 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
107 X----X-- X------- -------- -------- -------- -------- -------- -X--X
108 ----X--- --X----- -------- -------- -------- -------- -------- -----
109 -X------ --X----- -------- -------- -------- -------- -------- -----
110 -------- --X----- -------- -------- -------- -------- -------- -----
111 -------- --X----- -------- -------- -------- -------- -------- X----

112 ------X- -------- -------- -------- -------- -------- -------- -X---
113 ------X- -------- -------- -------- -------- -------- -------- ----X
114 -X----X- -------- -------- -------- -------- -------- -------- -----
115 ----X-X- -------- -------- -------- -------- -------- -------- -----
116 X----X-- X------- -------- -------- -------- -------- -------- X----
117 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
118 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
119 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
120 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
121 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
122 XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXXX
123 X----X-- X------- -------- -------- -------- -------- -------- -X---
124 --X-X--- -------- -------- -------- -------- -------- -------- -----
125 -XX----- -------- -------- -------- -------- -------- -------- -----
126 --X----- -------- -------- -------- -------- -------- -------- ----X
127 --X----- -------- -------- -------- -------- -------- -------- X----

      OUTPUT PINS:     111222223334
                    1234789012347890
   POLARITY FUSE:  ----------------

      OUTPUT  BANK:   4-40   17-24
        FLUSH FUSE:     X        X

TOTAL FUSES BLOWN:  5008
```



Traffic Signal Controller

Figure 1 illustrates a simple traffic intersection consisting of two one-way streets, direction 1 and direction 2. Each direction has a signal consisting of red, yellow, and green lamps which are activated with appropriately named active high signals. Also each direction has a sensor which provides an active high signal indicating the presence of an oncoming vehicle. Our controller is to manage this intersection with the sensors as inputs and the lamps as outputs, as shown in Figure 2.

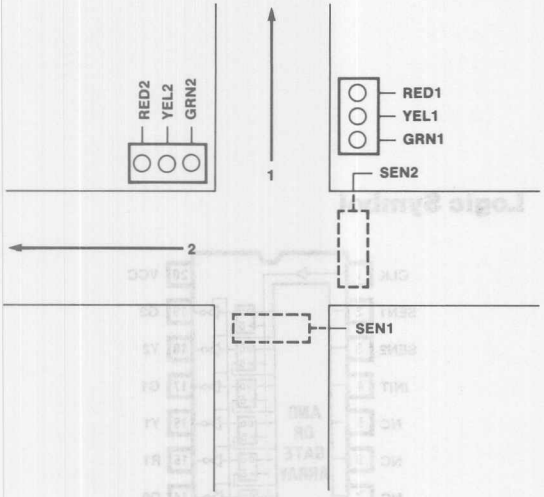


Figure 1. Traffic Intersection

Figure 2 also includes the system clock and an initialize (or reset) signal, which drives the controller to a predefined initial state. This raises two important issues in designing sequential logic with PAL devices. First, all circuit implementations of sequential logic with PAL devices are totally synchronous. This implies that all state variables (flip-flops) change at the same time, precisely after the rising edge of the clock. Second, PAL sequential logic designs should include a means for initialization to implement test programs and ensure reliable circuit operation. The specifics of the controller operations are detailed with a state diagram shown in Figure 3.

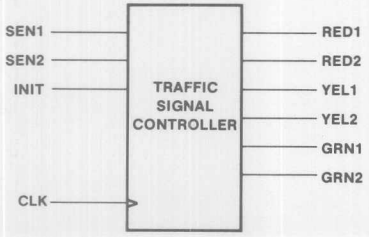


Figure 2. Traffic Signal Controller

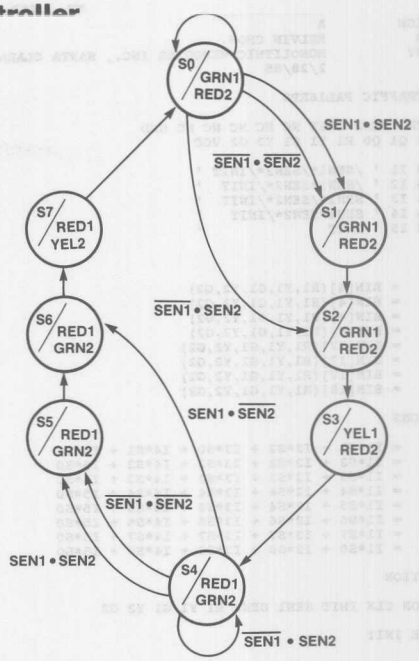


Figure 3. State Diagram — Traffic Signal Controller

Each circle in Figure 3 represents a stable state, i.e. an output configuration lasting at least one clock cycle. Inside the circles is the name of the state (S0-S7) and the outputs associated with that state. For the sake of simplicity in the state diagram, the transitions involving INIT are omitted; INIT simply drives the circuit to S0 from any state, regardless of other inputs.

Since RED1 = / RED2, RED1 is implemented with one flip-flop and RED2 with an external inverter.

3

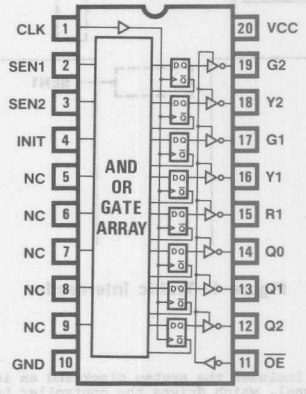
PAL Device Design Specification

```
TITLE           TRAFFIC SIGNAL CONTROLLER
PATTERN         TRAFFIC1.PDS
REVISION        A
AUTHOR          KELVIN CHOW
COMPANY         MONOLITHIC MEMORIES INC., SANTA CLARA
DATE            2/28/85

CHIP TRAFFIC PAL16RP8

CLK SEN1 SEN2 INIT NC NC NC NC NC GND
/OE Q2 Q1 Q0 R1 Y1 G1 Y2 G2 VCC

STRING I1 ' /SEN1*/SEN2*/INIT '
STRING I2 ' /SEN1*SEN2*/INIT  '
STRING I3 ' SEN1*/SEN2*/INIT  '
STRING I4 ' SEN1*SEN2*/INIT   '
STRING I5 ' INIT              '

STATE

S0      = BIN[4](R1,Y1,G1,Y2,G2)
S1      = BIN[4](R1,Y1,G1,Y2,G2)
S2      = BIN[4](R1,Y1,G1,Y2,G2)
S3      = BIN[8](R1,Y1,G1,Y2,G2)
S4      = BIN[17](R1,Y1,G1,Y2,G2)
S5      = BIN[17](R1,Y1,G1,Y2,G2)
S6      = BIN[17](R1,Y1,G1,Y2,G2)
S7      = BIN[18](R1,Y1,G1,Y2,G2)

EQUATIONS

S0      = I1*S1 + I2*S2 + I3*S0 + I4*S1 + I5*S0
S1      = I1*S2 + I2*S2 + I3*S2 + I4*S2 + I5*S0
S2      = I1*S3 + I2*S3 + I3*S3 + I4*S3 + I5*S0
S3      = I1*S4 + I2*S4 + I3*S4 + I4*S4 + I5*S0
S4      = I1*S5 + I2*S4 + I3*S6 + I4*S5 + I5*S0
S5      = I1*S6 + I2*S6 + I3*S6 + I4*S6 + I5*S0
S6      = I1*S7 + I2*S7 + I3*S7 + I4*S7 + I5*S0
S7      = I1*S0 + I2*S0 + I3*S0 + I4*S0 + I5*S0

SIMULATION

TRACE_ON CLK INIT SEN1 SEN2 R1 Y1 G1 Y2 G2

SETF OE INIT
CLOCKF
CLOCKF
CHECK /R1 /Y1 G1 /Y2 /G2

SETF /INIT /SEN1 /SEN2
CLOCKF

SETF SEN1 /SEN2
CLOCKF
CHECK /R1 /Y1 G1 /Y2 /G2

SETF /SEN1 SEN2
CLOCKF
CHECK /R1 /Y1 G1 /Y2 /G2

SETF SEN1 SEN2
CLOCKF
CHECK /R1 Y1 /G1 /Y2 /G2

SETF /SEN1 /SEN2
CLOCKF
CHECK R1 /Y1 /G1 /Y2 G2

SETF /SEN1 SEN2
CLOCKF
CHECK R1 G2

CLOCKF
CHECK R1 G2

CLOCKF
CHECK R1 /Y1 /G1 Y2 /G2

CLOCKF
CHECK /R1 /Y1 G1 /Y2 /G2

CLOCKF
CLOCKF
CLOCKF
CLOCKF

; This simulation was done using the alpha release version
; of Palasm2 software.
```

Simulation Results

```
     g  c  cg c g cg cg cg  cg c c c  c c c  c
CLK  XXHLHHLLHL LHLLHLLHLL HLLHLHLHLH LHLHLHHL
INIT HHHHHHHLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLL
SEN1 XXXXXXXLLL HHHLLLHHHL LLLLLLLLLL LLLLLLLL
SEN2 XXXXXXXLLL LLLHHHHHHL LLHHHHHHHH HHHHHHHH
R1   XXXXXLLLLL LLLLLLLLLL LHHHHHHHHH LLLLLLHH
Y1   XXXXXLLLLL LLLLLLLLHH HLLLLLLLLL LLLLHHLL
G1   XXXXXHHHHH HHHHHHHHLL LLLLLLLLLL HHHHLLLL
Y2   XXXXXLLLLL LLLLLLLLLL LLLLLLLLHH LLLLLLLL
G2   XXXXXLLLLL LLLLLLLLLL LHHHHHHHLL LLLLLLHH
```

Logic Symbol



State Machine Design Example

A typical control logic problem is the memory-to-processor handshake on memory transfer used in many computer architectures. The processor makes a transfer request by activating a request line (REQ) and specifies a read or write operation on a Read/Write line (R/W).

During a read operation, the processor waits for a Data Available signal at which time the data bus is sampled and the request line lowered, thus completing the cycle. During a write operation, the processor places data on the bus and waits for a Write Complete signal after the write cycle is finished. Upon write complete, the request line is lowered, hence completing the cycle. Table 1 shows the state assignments and the appropriate outputs. The state diagram is shown in Figure 1. Also the handshaking operation is illustrated in the timing diagram of Figure 2.

The memory-board logic to implement this function may be designed with gates and edge-triggered flip-flops as shown in Figure 3. This particular design would require about five SSI/MSI packages, but the same design can be implemented by a single PAL16RP6. The PAL design specification using state equations is shown on the next page.

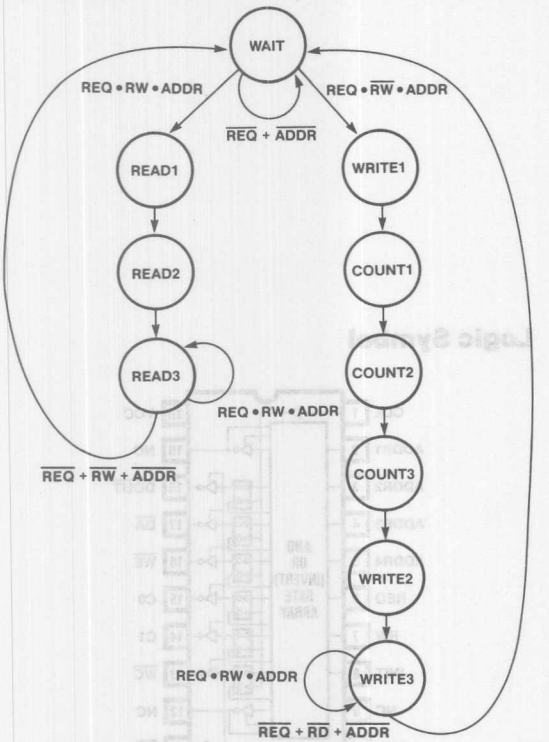


Figure 1. State Diagram — Memory Handshake Logic

STATE	DOUT	DA	WE	WC	C0	C1
WAIT	0	0	0	0	0	0
READ1	1	0	0	0	0	0
READ2	1	1	0	0	0	0
READ3	0	0	0	0	0	0
COUNT1	0	0	1	0	1	0
COUNT2	0	0	1	0	0	1
COUNT3	0	0	1	0	1	1
WRITE1	0	0	1	0	0	0
WRITE2	0	0	1	1	0	0
WRITE3	0	0	0	1	0	0

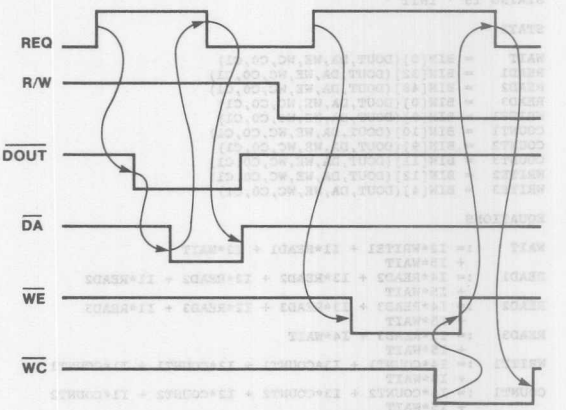


Figure 2. Memory Handshake Timing

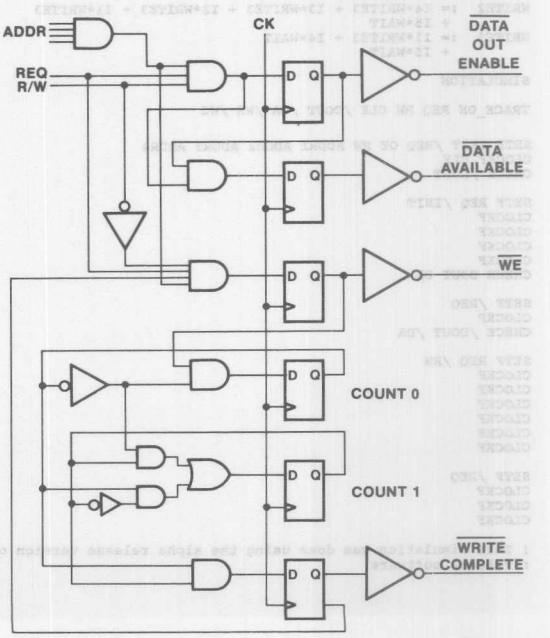
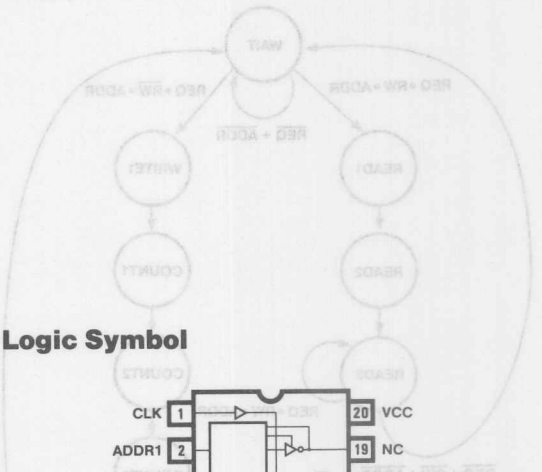


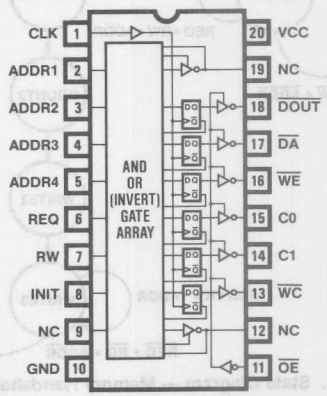
Figure 3. Memory Handshake Logic

```
TITLE           MEMORY HANDSHAKE LOGIC
PATTERN         MEMORY1.PDS
REVISION        A
AUTHOR          KELVIN CHOW
COMPANY         MONOLITHIC MEMORIES INC., SANTA CLARA, CA
DATE            2/28/85

CHIP MEMORY PAL16RP6

CLK ADDR1 ADDR2 ADDR3 ADDR4 REQ RW INIT NC GND
/OE NC /WC C1 C0 /WE /DA /DOUT NC VCC

STRING I1 ' REQ*RW*ADDR1*ADDR2*ADDR3*ADDR4*/INIT '
STRING I2 ' REQ*/RW*ADDR1*ADDR2*ADDR3*ADDR4*/INIT'
STRING I3 ' (/REQ+/ADDR1+/ADDR2+/ADDR3+/ADDR4) * /INIT '
STRING I4 ' (/REQ+/RW+/ADDR1+/ADDR2+/ADDR3+/ADDR4) * /INIT '
STRING I5 ' INIT '

STATE

WAIT    = BIN[0](DOUT,DA,WE,WC,C0,C1)
READ1   = BIN[32](DOUT,DA,WE,WC,C0,C1)
READ2   = BIN[48](DOUT,DA,WE,WC,C0,C1)
READ3   = BIN[0](DOUT,DA,WE,WC,C0,C1)
WRITE1  = BIN[8](DOUT,DA,WE,WC,C0,C1)
COUNT1  = BIN[10](DOUT,DA,WE,WC,C0,C1)
COUNT2  = BIN[9](DOUT,DA,WE,WC,C0,C1)
COUNT3  = BIN[11](DOUT,DA,WE,WC,C0,C1)
WRITE2  = BIN[12](DOUT,DA,WE,WC,C0,C1)
WRITE3  = BIN[4](DOUT,DA,WE,WC,C0,C1)

EQUATIONS

WAIT    := I2*WRITE1 + I1*READ1 + I3*WAIT
         + I5*WAIT
READ1   := I4*READ2 + I3*READ2 + I2*READ2 + I1*READ2
         + I5*WAIT
READ2   := I4*READ3 + I3*READ3 + I2*READ3 + I1*READ3
         + I5*WAIT
READ3   := I1*READ3 + I4*WAIT
         + I5*WAIT
WRITE1  := I4*COUNT1 + I3*COUNT1 + I2*COUNT1 + I1*COUNT1
         + I5*WAIT
COUNT1  := I4*COUNT2 + I3*COUNT2 + I2*COUNT2 + I1*COUNT2
         + I5*WAIT
COUNT2  := I4*COUNT3 + I3*COUNT3 + I2*COUNT3 + I1*COUNT3
         + I5*WAIT
COUNT3  := I4*WRITE2 + I3*WRITE2 + I2*WRITE2 + I1*WRITE2
         + I5*WAIT
WRITE2  := I4*WRITE3 + I3*WRITE3 + I2*WRITE3 + I1*WRITE3
         + I5*WAIT
WRITE3  := I1*WRITE3 + I4*WAIT
         + I5*WAIT

SIMULATION

TRACE_ON REQ RW CLK /DOUT /DA /WE /WC

SETF INIT /REQ OE RW ADDR1 ADDR2 ADDR3 ADDR4
CLOCKF CLK
CHECK /DOUT

SETF REQ /INIT
CLOCKF
CLOCKF
CLOCKF
CLOCKF
CHECK DOUT DA

SETF /REQ
CLOCKF
CHECK /DOUT /DA

SETF REQ /RW
CLOCKF
CLOCKF
CLOCKF
CLOCKF
CLOCKF
CLOCKF

SETF /REQ
CLOCKF
CLOCKF
CLOCKF

; This simulation was done using the alpha release version of
; Palasm2 software.
```

```
Page :  1
      g  cg   c   c c cg   cg    c c c c   c cg c c
 CLK  XXHLLHHLHH LHLHLLHHLL HHLHLHLHLH LHLLHLHL
 REQ  LLLLHHHHHH HHHHHLLLLH HHHHHHHHHH HHHLLLLL
  RW  HHHHHHHHHH HHHHHHHHHL LLLLLLLLLL LLLLLLLL
/DOUT XXXHHHLLLL LLLLLLLHHH HHHHHHHHHH HHHHHHHH
 /DA  XXXHHHHHHL LLLLLLLHHH HHHHHHHHHH HHHHHHHH
 /WE  XXXHHHHHHH HHHHHHHHHH HLLLLLLLLL LLHHHHHH
 /WC  XXXXXXHHHH HHHHHHHHHH HHHHHHHHHH LLHHHHHH
```



Logic Symbol



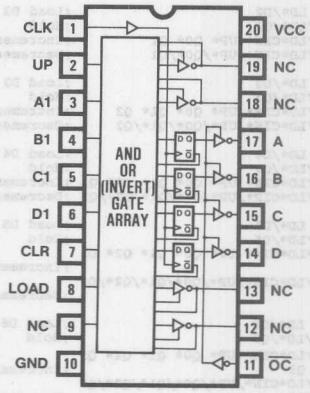
PAL Device Design Specification

```
Title     4Bit_Counter
Pattern   4cnt.pds
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories Inc. Santa Clara, CA
Date      1/14/85

CHIP 4BitCounter PAL16RP4

CLK UP AI BI CI DI CLR LOAD NC GND
/OC NC NC D C B A NC NC VCC

EQUATIONS

A   := /A*/B*/C*/D*/UP*/LOAD*/CLR       ;When CLR=1, A=0.
     + /A* B* C* D* UP*/LOAD*/CLR       ;Else it will count
     +  A* B*   /D*/UP*/LOAD*/CLR       ;UP or DOWN.
     +  A*/B* C*    UP*/LOAD*/CLR
     +  A*   /C*    UP*/LOAD*/CLR
     +  A*       D*/UP*/LOAD*/CLR
     +                 LOAD*/CLR* AI    ;New value is loaded
                                        ;when LOAD=1, CLR=0.

B   :=    /B*/C*/D*/UP*/LOAD*/CLR       ;When CLR=1, B=0.
     +    /B* C* D* UP*/LOAD*/CLR       ;Else it will count.
     +     B* C*/D*    /LOAD*/CLR
     +     B*/C*    UP*/LOAD*/CLR
     +     B*    D*/UP*/LOAD*/CLR
     +                 LOAD*/CLR* BI    ;New value is loaded
                                        ;when LOAD=1, CLR=0.

C   :=       /C*/D*/UP*/LOAD*/CLR       ;When CLR=1, C=0.
     +       /C* D* UP*/LOAD*/CLR       ;Else it will count.
     +        C*/D* UP*/LOAD*/CLR
     +        C* D*/UP*/LOAD*/CLR
     +                 LOAD*/CLR* CI    ;New value is loaded
                                        ;when LOAD=1, CLR=0.

D   :=          /D*    /LOAD*/CLR      ;Count
     +                 LOAD*/CLR* DI    ;New value is loaded
                                        ;when LOAD=1, CLR=0.

SIMULATION

TRACE_ON  AI BI CI DI LOAD CLR UP A B C D

SETF LOAD /CLR AI BI CI DI OC           ;Load all registers
CLOCKF CLK                              ;to HIGH and count up

SETF CLR                                ;Clear all registers
CLOCKF CLK

SETF /CLR UP /LOAD                      ;Start Counting up

FOR I:= 1 TO 16 DO                      ;Count up 16 clock
 BEGIN                                  ;cycles
   CLOCKF CLK
 END

SETF LOAD /CLR /UP AI BI CI DI          ;Load all registers
CLOCKF                                  ;to HIGH and count
SETF /LOAD                              ;down
  FOR I:= 1 TO 16 DO                    ;Count down 16 clock
   BEGIN                                ;cycles
     CLOCKF CLK
   END

SETF LOAD CLR AI /BI CI /DI             ;Test setting LOAD
CLOCKF CLK                              ;and CLR on at the
                                        ;same time.
SETF /OC

;The 4-bit counter counts up or down and has the clear and
;load capability. The clear operation overrides count and
;load. The counter counts up when CLR=low, LOAD=low, and
;UP=high. It counts down whenever CLR=low, LOAD=low, and
```

Simulation Results

```
Page :  1
          g  cgcgc c  c c c c c  c c c c c  c c c cgc
 AI     HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 BI     HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 CI     HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 DI     HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 LOAD   HHHHHHLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLHH
 CLR    LLLLHHLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 UP     XXXXXXHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHLLL
 A      XXXHHLLLLL LLLLLLLLLL LHHHHHHHHH HHHHLLHHHH
 B      XXXHHLLLLL LLLHHHHHHH HLLLLLLLLH HHHHHHHLLH
 C      XXXHHLLLLH HHHLLLHHHH HLLLLHHHHL LLLHHHHLLH
 D      XXXHHLLHHL LHHLLHHLLH HLLHHLLHHL LHHLLHHLLH
```

Logic Symbol



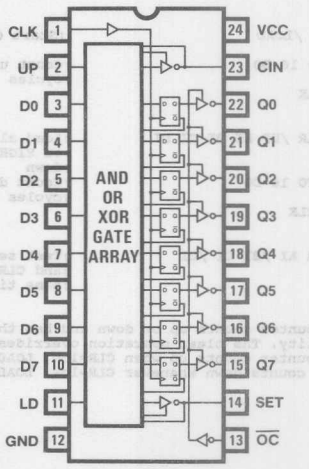
PAL Device Design Specification

```
Title     8Count
Pattern   8count.pds
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories Inc. Santa Clara, CA
Date      1/15/85

;This 8-bit up/down counter has the hold and load
;capabilities. It sets all the outputs high if SET=high.
;It loads new value when SET=low and LOAD=high. Else it
;counts up if UP=high and counts down if UP=low.

CHIP 8BitCounter PAL20X8

CLK UP D0 D1 D2 D3 D4 D5 D6 D7 LD GND
/OC SET Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 CIN VCC

EQUATIONS

/Q0 := /SET* LD*/D0                          ;Load D0
    +  /SET*/LD*/Q0                          ;Hold
   :+: /SET*/LD*CIN* UP                      ;Increment
    +  /SET*/LD*CIN*/UP                      ;Decrement

/Q1 := /SET* LD*/D1                          ;Load D1
    +  /SET*/LD*/Q1                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0                  ;Increment
    +  /SET*/LD*CIN*/UP*/Q0                  ;Decrement

/Q2 := /SET* LD*/D2                          ;Load D2
    +  /SET*/LD*/Q2                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1              ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1              ;Decrement

/Q3 := /SET* LD*/D3                          ;Load D3
    +  /SET*/LD*/Q3                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1* Q2          ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1*/Q2          ;Decrement

/Q4 := /SET* LD*/D4                          ;Load D4
    +  /SET*/LD*/Q4                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1* Q2* Q3      ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1*/Q2*/Q3      ;Decrement

/Q5 := /SET* LD*/D5                          ;Load D5
    +  /SET*/LD*/Q5                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1* Q2* Q3
       * Q4                                  ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1*/Q2*/Q3
       */Q4                                  ;Decrement

/Q6 := /SET* LD*/D6                          ;Load D6
    +  /SET*/LD*/Q6                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1* Q2* Q3
       * Q4* Q5                              ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1*/Q2*/Q3
       */Q4*/Q5                              ;Decrement

/Q7 := /SET* LD*/D7                          ;Load D7
    +  /SET*/LD*/Q7                          ;Hold
   :+: /SET*/LD*CIN* UP* Q0* Q1* Q2* Q3
       * Q4* Q5* Q6                          ;Increment
    +  /SET*/LD*CIN*/UP*/Q0*/Q1*/Q2*/Q3
       */Q4*/Q5*/Q6                          ;Decrement


SIMULATION

TRACE_ON  SET LD CIN UP
          D0 D1 D2 D3 D4 D5 D6 D7
          Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7

SETF OC SET
CLOCKF CLK
CHECK Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0                ;All outputs high
SETF /SET UP CIN /LD                         ;Counting up

FOR I:=1 TO 9 DO
 BEGIN
  CLOCKF CLK
  IF I=8 THEN                                ;Checking after 8
   BEGIN                                     ;clock pulses
    CHECK /Q7 /Q6 /Q5 /Q4 /Q3 Q2 Q1 Q0
   END
 END

SETF /CIN                                    ;Holding
CLOCKF CLK
CLOCKF CLK                                   ;The outputs hold to
                                             ;their values

SETF /UP CIN                                 ;Counting down

SETF LD /D7 D6 /D5 D4 /D3 D2 /D1 D0          ;Loading some data
CLOCKF CLK
CHECK /Q7 Q6 /Q5 Q4 /Q3 Q2 /Q1 Q0            ;Checking the output
                                             ;for the loaded data
SETF /LD UP                                  ;Counting up after
FOR I:=1 TO 5 DO                             ;removing HOLD, count
 BEGIN                                       ;up 3 cycles, count
  CLOCKF CLK                                 ;down for 2 cycles.
  IF I=3 THEN
   BEGIN
    SETF /UP
   END
 END
```

Simulation Results

```
Page :  1
        g  cgc c c  c c c c c  cgc cgcgc  c cgc c
 SET HHHHLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLL
 LD  XXXXLLLLLL LLLLLLLLLL LLLLLHHLL LLLLLLLL
 CIN XXXXHHHHHH HHHHHHHHHH HHLLLLHHHH HHHHHHHH
 UP  XXXXHHHHHH HHHHHHHHHH HHHHHLLHH HHHHLLLL
 D0  XXXXXXXXXX XXXXXXXXXX XXXXXXHHHH HHHHHHHH
 D1  XXXXXXXXXX XXXXXXXXXX XXXXXXLLLL LLLLLLLL
 D2  XXXXXXXXXX XXXXXXXXXX XXXXXXHHHH HHHHHHHH
 D3  XXXXXXXXXX XXXXXXXXXX XXXXXXLLLL LLLLLLLL
 D4  XXXXXXXXXX XXXXXXXXXX XXXXXXHHHH HHHHHHHH
 D5  XXXXXXXXXX XXXXXXXXXX XXXXXXLLLL LLLLLLLL
 D6  XXXXXXXXXX XXXXXXXXXX XXXXXXHHHH HHHHHHHH
 D7  XXXXXXXXXX XXXXXXXXXX XXXXXXLLLL LLLLLLLL
 Q0  XXXHHLLHHL LHHLLHHLLH HLLLLLLHHL LHHLLHHL
 Q1  XXXHHLLLLH HHHLLLHHHH HLLLLLLLLH HHHLLHHH
 Q2  XXXHHLLLLL LLLHHHHHHH HLLLLLLHHH HHHLLHHH
 Q3  XXXHHLLLLL LLLLLLLLLL LHHHHHHLLL LLLHHLLL
 Q4  XXXHHLLLLL LLLLLLLLLL LLLLLLHHH HHHHHHHH
 Q5  XXXHHLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLL
 Q6  XXXHHLLLLL LLLLLLLLLL LLLLLLHHH HHHHHHHH
 Q7  XXXHHLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLL
```

Logic Symbol



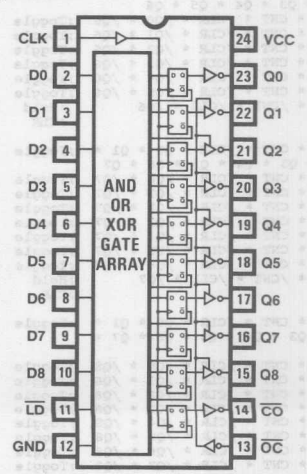
PAL Device Design Specification

```
Title    9BitCounter
Pattern  9BitCnt.pds
Revision A
Author   Mehrnaz Hada
Company  Monolithic Memories Inc., Santa Clara, CA
Date     1/28/85


;The 9-bit synchronous counter has parallel load, increment,
;and hold capabilities. The carry out pin (/CO) shows how to
;implement a carry out using a register by anticipated one
;count before the terminal count if counting and the terminal
;count if loading.

;Operations Table

;      /OC    CLK    /LD    D8-D0    Q8-Q0      Operation
;      ---------------------------------------------------
;       H      X      X       X        Z        HI-Z
;       L      L      X       X        Q        Hold
;       L      C      L       D        D        Load
;       L      C      H       X    Q PLUS 1     Increment
;      ---------------------------------------------------

CHIP 9BitCounter PAL20X10

CLK  D0 D1 D2 D3 D4 D5 D6 D7 D8 /LD GND
/OC /CO Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1  Q0 VCC

EQUATIONS

/Q0 :=  /LD*/Q0                                        ;Hold Q0
     +   LD*/D0                                        ;Load D0 (LSB)
    :+: /LD                                            ;Count

/Q1 :=  /LD*/Q1                                        ;Hold Q1
     +   LD*/D1                                        ;Load D1
    :+: /LD* Q0                                        ;Count

/Q2 :=  /LD*/Q2                                        ;Hold Q2
     +   LD*/D2                                        ;Load D2
    :+: /LD* Q0* Q1                                    ;Count

/Q3 :=  /LD*/Q3                                        ;Hold Q3
     +   LD*/D3                                        ;Load D3
    :+: /LD* Q0* Q1* Q2                                ;Count

/Q4 :=  /LD*/Q4                                        ;Hold Q4
     +   LD*/D4                                        ;Load D4
    :+: /LD* Q0* Q1* Q2* Q3                            ;Count

/Q5 :=  /LD*/Q5                                        ;Hold Q5
     +   LD*/D5                                        ;Load D5
    :+: /LD* Q0* Q1* Q2* Q3* Q4                        ;Count

/Q6 :=  /LD*/Q6                                        ;Hold Q6
     +   LD*/D6                                        ;Load D6
    :+: /LD* Q0* Q1* Q2* Q3* Q4* Q5                    ;Count

/Q7 :=  /LD*/Q7                                        ;Hold Q7
     +   LD*/D7                                        ;Load D7
    :+: /LD* Q0* Q1* Q2* Q3* Q4* Q5* Q6                ;Count

/Q8 :=  /LD*/Q8                                        ;Hold Q8
     +   LD*/D8                                        ;Load D8 (MSB)
    :+: /LD* Q0* Q1* Q2* Q3* Q4* Q5* Q6* Q7            ;Count

 CO :=  /LD*/Q0* Q1* Q2* Q3* Q4* Q5* Q6* Q7* Q8        ;Carry out (Anticipate
     +   LD* D0* D1* D2* D3* D4* D5* D6* D7* D8        ;Carry out (Anticipate

SIMULATION

TRACE_ON /OC /LD D8 D7 D6 D5 D4 D3 D2 D1 D0
         /CO Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

SETF OC LD /D8 /D7 /D6 /D5 /D4 /D3 /D2 /D1 /D0     ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

SETF LD /D8 /D7 /D6 /D5 /D4 /D3 /D2 /D1 D0         ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

SETF LD /D8 /D7 /D6 /D5 /D4 /D3 D2 D1 D0           ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

SETF LD /D8 /D7 /D6 /D5 /D4 D3 D2 D1 D0            ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

SETF LD /D8 /D7 /D6 /D5 D4 D3 D2 D1 D0             ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

SETF LD /D8  D7  D6  D5 D4 D3 D2 D1 D0             ;Load
CLOCKF CLK

SETF /LD                                           ;Increment
CLOCKF CLK

;Function Table

;CLK /OC /LD D8 D7 D6 D5 D4 D3 D2 D1 D0 /CO
;            Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0

;              Data In          Data  Out
;  Control     DDDDDDDDD        QQQQQQQQQ
;CLK /OC /LD 876543210  /CO  876543210  Comment
;-------------------------------------------------------
; C   L   L  LLLLLLLLL   H   LLLLLLLLL  Load
; C   L   H  XXXXXXXXX   H   LLLLLLLLH  Increment
; C   L   L  LLLLLLLLH   H   LLLLLLLLH  Load
; C   L   H  XXXXXXXXX   H   LLLLLLLHL  Increment
; C   L   L  LLLLLLLHH   H   LLLLLLLHH  Load
; C   L   H  XXXXXXXXX   H   LLLLLLHLL  Increment
; C   L   L  LLLLLLHHH   H   LLLLLLHHH  Load
; C   L   H  XXXXXXXXX   H   LLLLLHLLL  Increment
; C   L   L  LLLLLHHHH   H   LLLLLHHHH  Load
; C   L   H  XXXXXXXXX   H   LLLLHLLLL  Increment
; C   L   L  LLLLHHHHH   H   LLLLHHHHH  Load
; C   L   H  XXXXXXXXX   H   LLLHLLLLL  Increment
; C   L   L  LLLHHHHHH   H   LLLHHHHHH  Load
; C   L   H  XXXXXXXXX   H   LLHLLLLLL  Increment
; C   L   L  LLHHHHHHH   H   LLHHHHHHH  Load
; C   L   H  XXXXXXXXX   H   LHLLLLLLL  Increment
; C   L   L  LHHHHHHHH   H   LHHHHHHHH  Load
; C   L   H  XXXXXXXXX   H   HLLLLLLLL  Increment
; C   L   L  HHHHHHHHH   L   HHHHHHHHH  Load(Carry out)
; C   L   H  XXXXXXXXX   H   LLLLLLLLL  Increment(Roll over)
; C   L   L  HHHHHHHLL   H   HHHHHHHLL  Load
; C   L   H  XXXXXXXXX   H   HHHHHHHLH  Increment
; L   L   H  XXXXXXXXX   H   HHHHHHHLH  Hold
; C   L   H  XXXXXXXXX   H   HHHHHHHHL  Increment
; C   L   H  XXXXXXXXX   L   HHHHHHHHH  Increment(Carry out)
; C   L   H  XXXXXXXXX   H   LLLLLLLLL  Increment(Roll over)
; X   H   X  XXXXXXXXX   Z   ZZZZZZZZZ  Test HI-Z
;-------------------------------------------------------
```

Simulation Results

```
Page :  1
    g  cgcgcgc gcgcgcgcgc gcgcg c
/OC LLLLLLLLL LLLLLLLLL LLLLLL
/LD LLLLHHLHH LLHHLLHHLL HHLLHHH
 D8 LLLLLLLLL LLLLLLLLL LLLLLL
 D7 LLLLLLLLL LLLLLLLLL LLHHHHH
 D6 LLLLLLLLL LLLLLLLLL LLHHHHH
 D5 LLLLLLLLL LLLLLLLLL LLHHHHH
 D4 LLLLLLLLL LLLLLLLHH HHHHHHH
 D3 LLLLLLLLL LLLLHHHHHH HHHHHHH
 D2 LLLLLLLLL HHHHHHHHHH HHHHHHH
 D1 LLLLLLLLL HHHHHHHHHH HHHHHHH
 D0 LLLLLLHHH HHHHHHHHHH HHHHHHH
/CO XXXHHHHHH HHHHHHHHHH HHHHHHH
 Q8 XXXLLLLLL LLLLLLLLL LLLLHH
 Q7 XXXLLLLLL LLLLLLLLL LLLHHLL
 Q6 XXXLLLLLL LLLLLLLLL LLLHHLL
 Q5 XXXLLLLLL LLLLLLLLL LHHHLL
 Q4 XXXLLLLLL LLLLLLHHH HLLHHLL
 Q3 XXXLLLLLL LLLHHHHLH HLLHHLL
 Q2 XXXLLLLLL LHHLLHHLH HLLHHLL
 Q1 XXXLLLLLH HHHLLHHLH HLLHHLL
 Q0 XXXLLHHHH LHHLLHHLH HLLHHLL
```

Logic Symbol

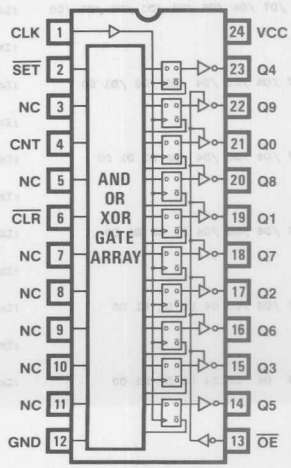


```
Revision  A
Author    Mehrnaz Hada
Company   Monolithic Memories Inc. Santa Clara, CA
Date      1/15/85

;The 10-bit counter increments on the rising edge of the
;clock input (CLK), if CNT input is high. The outputs are
;HIGH-Z when the enable line (/OE) is high and enabled
;when the enable line (/OE) is low. The counter is
;cleared (all lows) if CLR=HIGH.

CHIP 10BitCount PAL20RS10

CLK /SET NC CNT NC /CLR NC NC NC NC GND
/OE Q5 Q3 Q6 Q2 Q7 Q1 Q8 Q0 Q9 Q4 VCC

EQUATIONS

/Q0  := /SET *  CNT * /CLR * Q0          ;Toggle
      + /SET * /CNT * /CLR * /Q0         ;Hold
      + CLR                              ;CLR

/Q1  := /SET * CNT * /CLR * Q0 * Q1      ;Toggle
      + /SET * CNT * /CLR * /Q0 * /Q1    ;Toggle
      + /SET * /CNT * /CLR * /Q1         ;Hold
      + CLR                              ;CLR

/Q2  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2
      + /SET * CNT * /CLR * /Q0 * /Q2    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q2    ;Toggle
      + /SET * /CNT * /CLR * /Q2         ;Hold
      + CLR                              ;CLR

/Q3  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3
      + /SET * CNT * /CLR * /Q0 * /Q3    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q3    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q3    ;Toggle
      + /SET * /CNT * /CLR * /Q3         ;Hold
      + CLR                              ;CLR

/Q4  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4
      + /SET * CNT * /CLR * /Q0 * /Q4    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q4    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q4    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q4    ;Toggle
      + /SET * /CNT * /CLR * /Q4         ;Hold
      + CLR                              ;CLR

/Q5  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4 * Q5
      + /SET * CNT * /CLR * /Q0 * /Q5    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q5    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q5    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q5    ;Toggle
      + /SET * CNT * /CLR * /Q4 * /Q5    ;Toggle
      + /SET * /CNT * /CLR * /Q5         ;Hold
      + CLR                              ;CLR

/Q6  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4 * Q5 * Q6
      + /SET * CNT * /CLR * /Q0 * /Q6    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q6    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q6    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q6    ;Toggle
      + /SET * CNT * /CLR * /Q4 * /Q6    ;Toggle
      + /SET * CNT * /CLR * /Q5 * /Q6    ;Toggle
      + /SET * /CNT * /CLR * /Q6         ;Hold
      + CLR                              ;CLR

/Q7  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4 * Q5 * Q6 * Q7
      + /SET * CNT * /CLR * /Q0 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q4 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q5 * /Q7    ;Toggle
      + /SET * CNT * /CLR * /Q6 * /Q7    ;Toggle
      + /SET * /CNT * /CLR * /Q7         ;Hold
      + CLR                              ;CLR

/Q8  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4 * Q5 * Q6 * Q7 *
        Q8
      + /SET * CNT * /CLR * /Q0 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q4 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q5 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q6 * /Q8    ;Toggle
      + /SET * CNT * /CLR * /Q7 * /Q8    ;Toggle

/Q9  := /SET * CNT * /CLR * Q0 * Q1 *    ;Toggle
        Q2 * Q3 * Q4 * Q5 * Q6 * Q7 *
        Q8 * Q9
      + /SET * CNT * /CLR * /Q0 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q1 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q2 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q3 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q4 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q5 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q6 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q7 * /Q9    ;Toggle
      + /SET * CNT * /CLR * /Q8 * /Q9    ;Toggle
      + /SET */CNT * /CLR * /Q9          ;Hold
      + CLR                              ;CLR

SIMULATION

TRACE_ON OE CLK SET CLR CNT Q0 Q1 Q2 Q3
         Q4 Q5 Q6 Q7 Q8 Q9

SETF OE /CLR SET                         ;Set high all the
CLOCKF CLK                               ;registers

SETF CLR                                 ;Clear all the
CLOCKF CLK                               ;registers

SETF /SET CNT /CLR                       ;Start counting

FOR I:= 1 TO 5 DO                        ;Count for five
  BEGIN                                  ;cycles. At the count
    CLOCKF CLK                           ;of four check for
                                         ;four on the output.
    IF I=4 THEN
     BEGIN
      CHECK /Q0 /Q1 Q2
     END
  END

SETF /CNT                                ;Hold
CLOCKF CLK
```

Simulation Results

```
Page :  1
        g  cg cg c   c c c cg  c
 OE   HHHHHHHHHH HHHHHHHHHH H
 CLK  XXHLLHLLHL HLHLHLHLLH L
 SET  HHHHHHHLLL LLLLLLLLLL L
 CLR  LLLLHHHLLL LLLLLLLLLL L
 CNT  XXXXXXXHHH HHHHHHHLL L
 Q0   XXXHHHLLLH HLLHHLLHHH H
 Q1   XXXHHHLLLL LHHHLLLLL L
 Q2   XXXHHHLLLL LLLLLHHHHH H
 Q3   XXXHHHLLLL LLLLLLLLL L
 Q4   XXXHHHLLLL LLLLLLLLL L
 Q5   XXXHHHLLLL LLLLLLLLL L
 Q6   XXXHHHLLLL LLLLLLLLL L
 Q7   XXXHHHLLLL LLLLLLLLL L
 Q8   XXXHHHLLLL LLLLLLLLL L
 Q9   XXXHHHLLLL LLLLLLLLL L
```

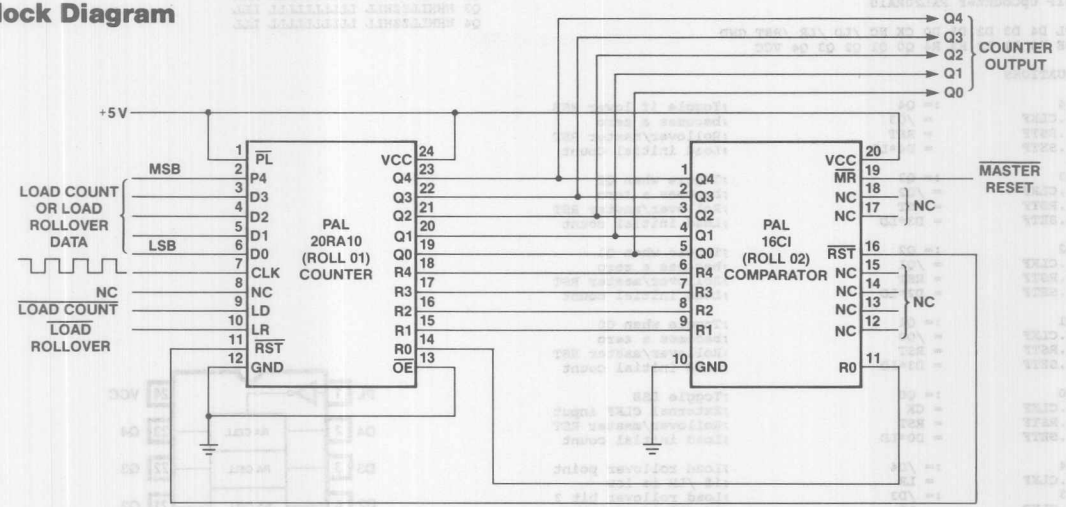
Logic Symbol



Functional Description

Shown below is a schematic of two PAL devices implementing a 5-bit up asynchronous ring counter with programmable rollover, asynchronous load, and reset. Initial count point can be loaded by asserting /LD low. Rollover point is loaded by asserting /LR low. Q4...Q0 and R4...R0 are compared in the PAL16C1 which is implemented as a comparator. The result of the comparison is fed back from the PAL16C1 to the PAL20RA10 device through the /RST line. Note that a Master Reset must be executed first before an initial count point can be loaded.

Block Diagram



5-Bit Up Counter

PAL Device Design Specification

```
PAL16C1
ROLL02
COMPARATOR
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
Q4 Q3 Q2 Q1 Q0 R4 R3 R2 R1 GND
R0 NC NC NC NC /RST NC NC /MR VCC

EQUATIONS

/RST     = Q4*/R4*/MR + /Q4*R4*/MR      ;COMPARE Q4 & R4 MSB
         + Q3*/R3*/MR + /Q3*R3*/MR      ;COMPARE Q3 & R3
         + Q2*/R2*/MR + /Q2*R2*/MR      ;COMPARE Q2 & R2
         + Q1*/R1*/MR + /Q1*R1*/MR      ;COMPARE Q1 & R1
         + Q0*/R0*/MR + /Q0*R0*/MR      ;COMPARE Q0 & R0 LSB

FUNCTION TABLE

Q4 Q3 Q2 Q1 Q0 R4 R3 R2 R1 R0 /MR /RST

;              /
;            /  R
;QQQQQ RRRRR M  S
;43210 43210 R  T
-------------------------------------------------
 HLLLL LLLLL H  H     1  Q4=H, R4=L
 LHLLL LLLLL H  H     2  Q3=H, R3=L
 LLHLL LLLLL H  H     3  Q2=H, R2=L
 LLLHL LLLLL H  H     4  Q1=H, R1=L
 LLLLH LLLLL H  H     5  Q0=H, R0=L
 LLLLL HLLLL H  H     6  Q4=L, R4=H
 LLLLL LHLLL H  H     7  Q3=L, R3=H
 LLLLL LLHLL H  H     8  Q2=L, R2=H
 LLLLL LLLHL H  H     9  Q1=L, R1=H
 LLLLL LLLLH H  H    10  Q0=L, R0=H
 LLLLL LLLLL H  L    11  TEST ALL LOWS
 HHHHH HHHHH H  L    12  TEST ALL HIGHS
 LHLHL LHLHL H  L    13  TEST EVEN CHECKERBOARD
 HLHLH HLHLH H  L    14  TEST ODD CHECKERBOARD
 LLLLL HHHHH L  L    15  TEST MASTER RESET
 HHHHH LLLLL L  L    16  TEST MASTER RESET
-------------------------------------------------
```

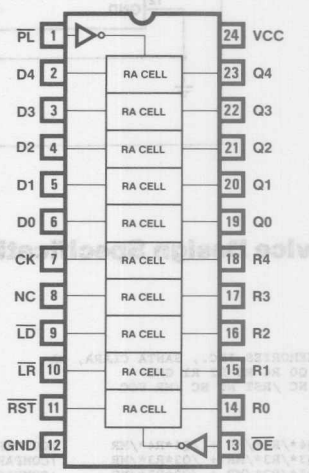
```
DESCRIPTION

PAL16C1 DESIGN SPECIFICATION
5-BIT COMPARATOR WITH MASTER RESET OVERRIDE
(SECOND OF THE TWO PALS SOLUTION ON THE 5-BIT COUNTER WITH
PROGRAMMABLE ROLLOVER, ASYNCHRONOUS LOAD AND RESET.)
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
BILL KARKULA                7/19/84

THIS DEVICE COMPARES 5 BITS OF DATA (R4...R0) WITH Q4...Q0
AND ASSERTS /RST IF THEY ARE EQUAL.  THEREFORE /RST
GOES LOW WHEN PROGRAMMED ROLLOVER POINT R4...R0
MATCHES COUNT Q4...Q0.  /RST ALSO GOES LOW WHEN /MR
GOES LOW, INDICATING A MASTER RESET.

NOTE:  THIS PAL DESIGN SPEC WAS ASSEMBLED ON PALASM V1.7.
```

PAL Device Design Specification

```
Title     5-Bit Up Counter
Pattern   UpCount.pds
Revision  A
Author    Bill Karkula
Company   Monolithic Memories Inc., Santa Clara, CA
Date      7/19/84

CHIP UpCounter PAL20RA10

/PL D4 D3 D2 D1 D0 CK NC /LD /LR /RST GND
/OE R0 R1 R2 R3 R4 Q0 Q1 Q2 Q3 Q4 VCC

EQUATIONS

/Q4             := Q4                   ;Toggle if lower MSB
Q4.CLKF          = /Q3                  ;becomes a zero
Q4.RSTF          = RST                  ;Rollover/master RST
Q4.SETF          = D4*LD                ;Load initial Count

/Q3             := Q3                   ;Toggle when Q2
Q3.CLKF          = /Q2                  ;becomes a zero
Q3.RSTF          = RST                  ;Rollover/master RST
Q3.SETF          = D3*LD                ;Load initial count

/Q2             := Q2                   ;Toggle when Q1
Q2.CLKF          = /Q1                  ;becomes a zero
Q2.RSTF          = RST                  ;Rollover/master RST
Q2.SETF          = D2*LD                ;Load initial count

/Q1             := Q1                   ;Toggle when Q0
Q1.CLKF          = /Q0                  ;becomes a zero
Q1.RSTF          = RST                  ;Rollover/master RST
Q1.SETF          = D3*LD                ;Load initial count

/Q0             := Q0                   ;Toggle LSB
Q0.CLKF          = CK                   ;External CLKF input
Q0.RSTF          = RST                  ;Rollover/master RST
Q0.SETF          = D0*LD                ;Load initial count

/R4             := /D4                  ;Load rollover point
R4.CLKF          = LR                   ;if /LR is low
/R3             := /D3                  ;Load rollover bit 3
R3.CLKF          = LR                   ;if /LR is asserted
/R2             := /D2                  ;Load rollover bit 2
R2.CLKF          = LR                   ;if /LR is asserted
/R1             := /D1                  ;Load rollover bit 1
R1.CLKF          = LR                   ;if /LR is asserted
/R0             := /D0                  ;Load rollover bit 0
R0.CLKF          = LR                   ;if /LR is asserted

SIMULATION

TRACE_ON PL OE CK Q0 Q1 Q2 Q3 Q4

SETF RST /LD                            ;Test SET function
                                        ;of registers
CHECK  Q0  Q1  Q2  Q3  Q4               ;Check for high
                                        ;on reg. outputs
SETF /RST                               ;Deassert SET funct

SETF D0 D1 D2 D3 D4 LD                  ;Test RESET function
CHECK /Q0 /Q1 /Q2 /Q3 /Q4

SETF /OE Q4 Q3 Q2 Q1 /Q0 /LD            ;Disable RESET(LD=0)
                                        ;load registers w/
                                        ;HHHHL (Q4..Q0),
                                        ;tristate registers
SETF PL                                 ;Load reg.'s w/ data
                                        ;on output bus.
SETF OE /PL                             ;Disable PRELOAD &
                                        ;TRISTATE funct.
SETF CK
SETF /CK

SETF CK
SETF /CK

SETF CK
SETF /CK

SETF CK
SETF /CK
```

Simulation Results

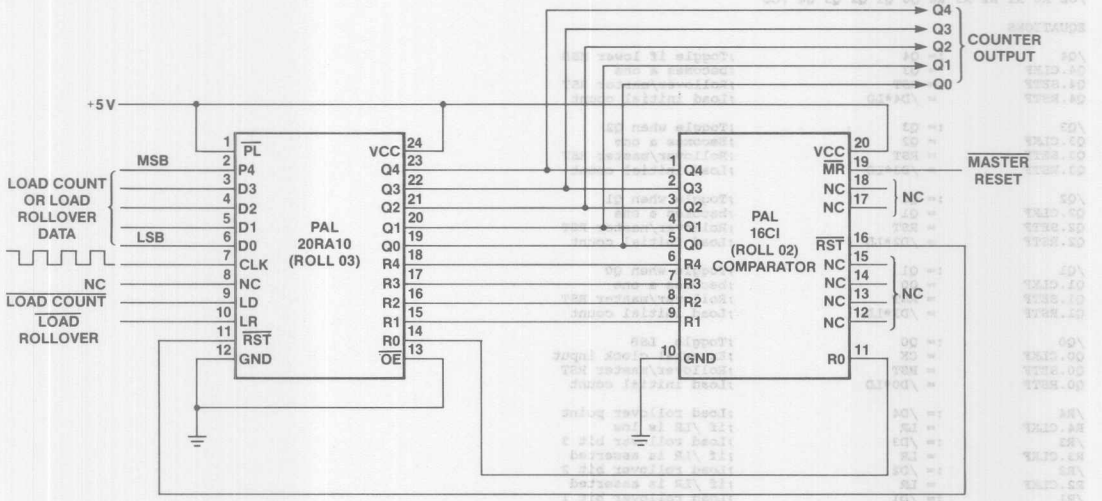
```
        g gg g ggg    gg gg    gg
PL  XXXXXXXHLL LLLLLLLLL LLL
OE  XXXXXLLLHH HHHHHHHHH HHH
CK  XXXXXXXXXH HHLHHLHHHH LHH
Q0  HHHLLZZLHH LLLLHHHLLL LLH
Q1  HHHLLZZHLL LHHHHHHLL LLL
Q2  HHHLLZZHLL LLLLLLLLH HHH
Q3  HHHLLZZHLL LLLLLLLLL LLL
Q4  HHHLLZZHLL LLLLLLLLL LLL
```



Functional Description

Shown below is a schematic of two PAL devices implementing a 5-bit up asynchronous ring counter with programmable rollover, asynchronous load, and reset. Initial count point can be loaded by asserting /LD low. Rollover point is loaded by asserting /LR low. Q4...Q0 and R4...R0 are compared in the PAL16C1 device which is implemented as a comparator. The result of the comparison is fed back from the PAL16C1 logic circuit to the PAL20RA10 device through the /RST line. Note that a Master Reset must be executed first before an initial "countdown" point can be loaded.

Block Diagram



5-Bit Down Counter

PAL Device Design Specification

```
PAL16C1
ROLL02
COMPARATOR
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
Q4 Q3 Q2 Q1 Q0 R4 R3 R2 R1 GND
R0 NC NC NC /RST NC NC /MR VCC

EQUATIONS

/RST     = Q4*/R4*/MR + /Q4*R4*/MR      ;COMPARE Q4 & R4 MSB
         + Q3*/R3*/MR + /Q3*R3*/MR      ;COMPARE Q3 & R3
         + Q2*/R2*/MR + /Q2*R2*/MR      ;COMPARE Q2 & R2
         + Q1*/R1*/MR + /Q1*R1*/MR      ;COMPARE Q1 & R1
         + Q0*/R0*/MR + /Q0*R0*/MR      ;COMPARE Q0 & R0 LSB

FUNCTION TABLE

Q4 Q3 Q2 Q1 Q0 R4 R3 R2 R1 R0 /MR /RST

;             /
;           /  R
;QQQQQ RRRRR M  S
;43210 43210 R  T
-------------------------------------------------
 HLLLL LLLLL H  H     1  Q4=H, R4=L
 LHLLL LLLLL H  H     2  Q3=H, R3=L
 LLHLL LLLLL H  H     3  Q2=H, R2=L
 LLLHL LLLLL H  H     4  Q1=H, R1=L
 LLLLH LLLLL H  H     5  Q0=H, R0=L
 LLLLL HLLLL H  H     6  Q4=L, R4=H
 LLLLL LHLLL H  H     7  Q3=L, R3=H
 LLLLL LLHLL H  H     8  Q2=L, R2=H
 LLLLL LLLHL H  H     9  Q1=L, R1=H
 LLLLL LLLLH H  H    10  Q0=L, R0=H
 LLLLL LLLLL H  L    11  TEST ALL LOWS
 HHHHH HHHHH H  L    12  TEST ALL HIGHS
 LHLHL LHLHL H  L    13  TEST EVEN CHECKERBOARD
 HLHLH HLHLH H  L    14  TEST ODD CHECKERBOARD
 LLLLL HHHHH L  L    15  TEST MASTER RESET
 HHHHH LLLLL L  L    16  TEST MASTER RESET
-------------------------------------------------
```

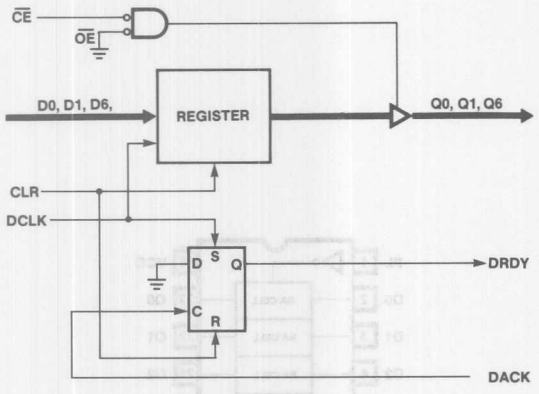
```
DESCRIPTION

PAL16C1 DESIGN SPECIFICATION
5-BIT COMPARATOR WITH MASTER RESET OVERRIDE
(SECOND OF THE TWO PALS SOLUTION ON THE 5-BIT COUNTER WITH
PROGRAMMABLE ROLLOVER, ASYNCHRONOUS LOAD AND RESET.)
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
BILL KARKULA                    7/19/84

THIS DEVICE COMPARES 5 BITS OF DATA (R4...R0) WITH Q4...Q0
AND ASSERTS /RST IF THEY ARE EQUAL.  THEREFORE /RST
GOES LOW WHEN PROGRAMMED ROLLOVER POINT R4...R0
MATCHES COUNT Q4...Q0.  /RST ALSO GOES LOW WHEN /MR
GOES LOW, INDICATING A MASTER RESET.

NOTE:  THIS PAL DESIGN SPEC WAS ASSEMBLED ON PALASM V1.7.
```

```
                5 Bit Down Counter
Pattern   DCount.pds
Revision  A
Author    Bill Karkula
Company   Monolithic Memories Inc., Sanat Clara, CA
Date      7/19/84

CHIP DN_COUNTER PAL20RA10

/PL D4 D3 D2 D1 D0 CK NC /LD /LR /RST GND
/OE R0 R1 R2 R3 R4 Q0 Q1 Q2 Q3 Q4 VCC

EQUATIONS

/Q4             := Q4                   ;Toggle if lower MSB
Q4.CLKF          = Q3                   ;becomes a one
Q4.SETF          = RST                  ;Rollover/master RST
Q4.RSTF          = /D4*LD               ;Load initial count

/Q3             := Q3                   ;Toggle when Q2
Q3.CLKF          = Q2                   ;Becomes a one
Q3.SETF          = RST                  ;Rollover/master RST
Q3.RSTF          = /D3*LD               ;Load initial count

/Q2             := Q2                   ;Toggle when Q1
Q2.CLKF          = Q1                   ;becomes a one
Q2.SETF          = RST                  ;Rollover/master RST
Q2.RSTF          = /D2*LD               ;Load initial count

/Q1             := Q1                   ;Toggle when Q0
Q1.CLKF          = Q0                   ;becomes a one
Q1.SETF          = RST                  ;Rollover/master RST
Q1.RSTF          = /D1*LD               ;Load initial count

/Q0             := Q0                   ;Toggle  LSB
Q0.CLKF          = CK                   ;External clock input
Q0.SETF          = RST                  ;Rollover/master RST
Q0.RSTF          = /D0*LD               ;Load initial count

/R4             := /D4                  ;Load rollover point
R4.CLKF          = LR                   ;if /LR is low
/R3             := /D3                  ;Load rollover bit 3
R3.CLKF          = LR                   ;if /LR is asserted
/R2             := /D2                  ;Load rollover bit 2
R2.CLKF          = LR                   ;if /LR is asserted
/R1             := /D1                  ;Load rollover bit 1
R1.CLKF          = LR                   ;if /LR is asserted
/R0             := /D0                  ;Load rollover bit 0
R0.CLKF          = LR                   ;if /LR is asserted

SIMULATION

TRACE_ON PL OE CK Q0 Q1 Q2 Q3 Q4

SETF RST /LD                            ;Test SET function
                                        ;of registers
CHECK /Q0 /Q1 /Q2 /Q3 /Q4               ;Check for high on
                                        ;register outputs
SETF /RST                               ;Deassert SET funct
SETF /D0 /D1 /D2 /D3 /D4 LD             ;Test RESET funct.
CHECK Q0 Q1 Q2 Q3 Q4
SETF /OE /Q4 /Q3 /Q2 /Q1 Q0 /LD         ;Disable RESET, load
                                        ;registers w/ LLLLH,
                                        ;tristate registers
SETF  PL                                ;Load regs w/ data
                                        ;on output bus.
SETF OE /PL                             ;Disable PRELOAD &
                                        ;TRISTATE function.
FOR I:=1 TO 7 DO                        ;Initially load regs
 BEGIN                                  ;w/ LLLLH & clocked
  SETF CK                               ;7 times.
  SETF /CK                              ;Rollover at I=2
  IF I=2 THEN                           ;count goes LLLLL
   BEGIN                                ;to HHHHH.
    CHECK Q4 Q3 Q2 /Q1 /Q0              ;Check rollover pt.
   END
  IF I=6 THEN
   BEGIN
    CHECK Q4 Q3 /Q2 /Q1 /Q0
   END
  IF I=7 THEN
   BEGIN
    CHECK Q4 /Q3 Q2 Q1 Q0
   END
  END
```

```
   g gg g ggg   gg gg    gg gg  gg  gg   g
PL XXXXXXXHLL LLLLLLLLL LLLLLLLLL LLLLLL
OE XXXXXLLLHH HHHHHHHHH HHHHHHHHH HHHHHH
CK XXXXXXXXXH HHLHHLHHH LHHLHHLHH LHHHHL
Q0 LLLHHZZHLL HHHHLLLHHH HHLLLHHHHL LLHHHH
Q1 LLLHHZZLHH HLLLLLLLHH HHHHHLLLL LLLHHH
Q2 LLLHHZZLHH HHHHHHHHHL LLLLLLLLL LLLLHH
Q3 LLLHHZZLHH HHHHHHHHHH HHHHHHHHH HHHHLL
Q4 LLLHHZZLHH HHHHHHHHHH HHHHHHHHH HHHHHH
```

This application is for a seven-bit register with handshake logic. The chip can be used for interfacing between a microprocessor and its peripheral I/O. The on-chip flag flip-flop provides the handshaking capability required in typical demand-response-based data transfer. Both the register and the flag flip-flops are asynchronously cleared by CLR signal.

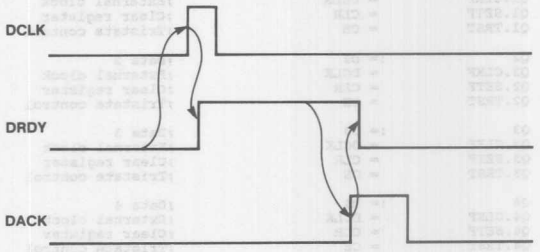
Block Diagram



Seven-Bit I/O Port with Handshake Logic

and at the same time, the event is signified by asserting DRDY signal. The DRDY signal indicates that the data is available in the register. By monitoring the DRDY signal when it is high, the stored input data can be transferred to Q output port by asserting /OE three-state control signal. After moving the data, DACK signal should be applied to clear the flag flip-flop.

Handshake Operation



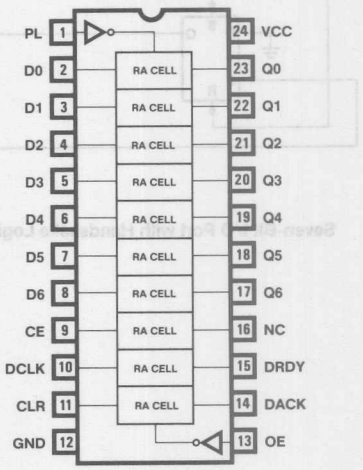
Seven-Bit I/O Port with Handshake Logic

PAL Device Design Specification

```
Title     7-Bit I/O Port with Handshake Logic
Pattern   Port.pds
Revision  A
Author    Sadahiro Horiko / Kelvin Chow
Company   Monolithic Memories Inc., Santa Clara, Ca
Date      3/1/85

CHIP IOPORT PAL20RA10

PL D0 D1 D2 D3 D4 D5 D6 CE DCLK CLR GND
OE DACK DRDY NC Q6 Q5 Q4 Q3 Q2 Q1 Q0 VCC

EQUATIONS

Q0              := D0                   ;LSB of 7-bit regs
Q0.CLKF          = DCLK                 ;External clock
Q0.SETF          = CLR                  ;Clear register
Q0.TRST          = CE                   ;Tristate control

Q1              := D1                   ;Data 1
Q1.CLKF          = DCLK                 ;External clock
Q1.SETF          = CLR                  ;Clear register
Q1.TRST          = CE                   ;Tristate control

Q2              := D2                   ;Data 2
Q2.CLKF          = DCLK                 ;External clock
Q2.SETF          = CLR                  ;Clear register
Q2.TRST          = CE                   ;Tristate control

Q3              := D3                   ;Data 3
Q3.CLKF          = DCLK                 ;External clock
Q3.SETF          = CLR                  ;Clear register
Q3.TRST          = CE                   ;Tristate control

Q4              := D4                   ;Data 4
Q4.CLKF          = DCLK                 ;External clock
Q4.SETF          = CLR                  ;Clear register
Q4.TRST          = CE                   ;Tristate control

Q5              := D5                   ;Data 5
Q5.CLKF          = DCLK                 ;External clock
Q5.SETF          = CLR                  ;Clear register
Q5.TRST          = CE                   ;Tristate control

Q6              := D6                   ;Data 6
Q6.CLKF          = DCLK                 ;External clock
Q6.SETF          = CLR                  ;Clear register
Q6.TRST          = CE                   ;Tristate control

DRDY            := GND                  ;Handshake logic
DRDY.CLKF        = DACK                 ;Cleared by DACK
DRDY.RSTF        = DCLK                 ;Clear
DRDY.SETF        = CLR                  ;Asserted by DCLK
DRDY.TRST        = VCC                  ;(External clock)

SIMULATION

TRACE_ON CLR Q0 Q1 Q2 Q3 Q4 Q5 Q6 DCLK DRDY DACK

SETF PL /CE /OE /D0 D1 /D2 D3 /D4 D5 /D6 CLR /DCLK /DACK
                                        ;Set input values
                                        ;Tristate outputs

SETF CE OE CLR                          ;Remove the tri-
                                        ;states on the
                                        ;outputs and clear
                                        ;registers

SETF CLR
SETF CLR

SETF /CLR                               ;Clock the data &
SETF DCLK                               ;set DRDY register
SETF DCLK

SETF /DCLK                              ;Remove the clock

SETF DACK                               ;Assert DACK
SETF DACK

SETF /DACK                              ;Lower DACK
SETF /DACK
```

Simulation Results

```
Page :  1
        g g gg gg  g
CLR   HHHHLLLLLL L
Q0    XZZLLLLLLL L
Q1    XZZLLLHHHH H
Q2    XZZLLLLLLL L
Q3    XZZLLLHHHH H
Q4    XZZLLLLLLL L
Q5    XZZLLLHHHH H
Q6    XZZLLLLLLL L
DCLK  LLLLLHHLLL L
DRDY  XLZZLHHHHL L
DACK  LLLLLLLLHH L
```



Functional Description

Original application was developed by LTT, Conflans Ste. Honorine, FRANCE. Part of the schematics, reprinted with courtesy of LTT, is used to control a serial data link based upon a specialized LSI chip.

Originally designed with six standard SSI/MSI circuits, this same function can now be implemented, not only into a single PAL20RA10 device, but with even more features and better performance. The function can be divided into three subfunctions:

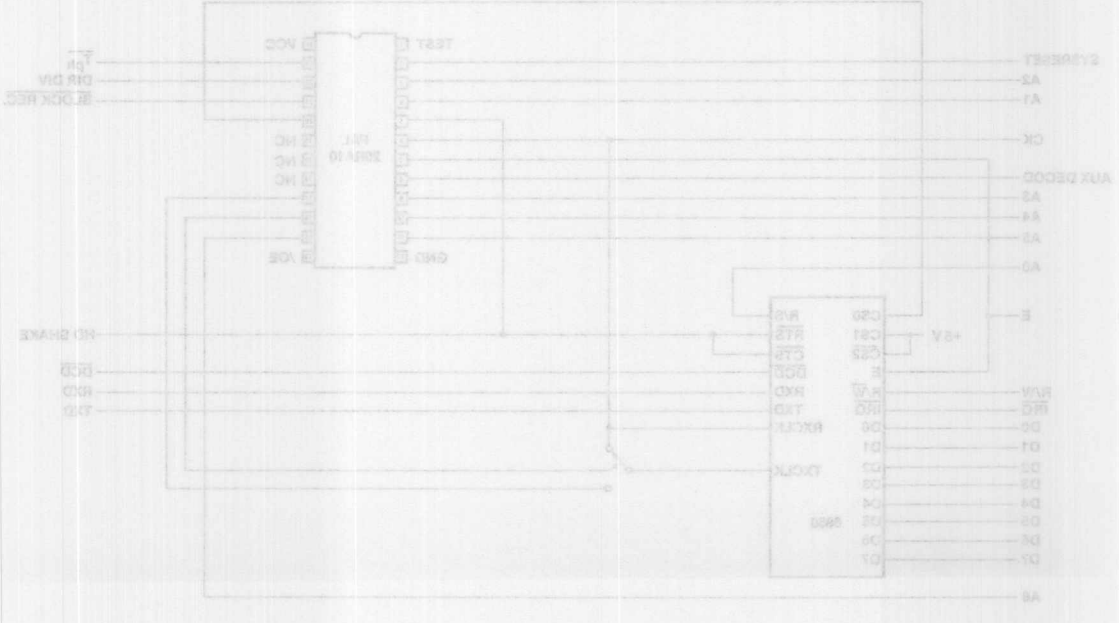
1. Address Decoding
2. Control Flags
3. Transmission Speed Selection

Up to four address lines are allowed (eight were actually used), plus two extra lines which are special decoding controls (MEM/IO selection, Enable Control . . .). Two flip-flop load flag conditions, from the address bus (A1 and A2), providing handshake between the 6850 UART and the communication lines. They have a common clock which also serves as Chip Select (CSO) for the UART.

The UART Transmit clock (TXCLK) can be directly connected to the Receive Clock (CK or RXCLK) or represents the Receive Clock value divided by sixteen. This function was performed by four D-type Flip-Flops connected as a 4-stage Asynchronous Divider. Since each basic cell, used in a PAL20RA10 device has four Product Terms available, this function could be implemented either asynchronously or synchronously. In the PAL Design Specification example, a 4-bit synchronous divider was used instead of the asynchronous circuit shown in the schematic.

Pin Description

1. TEST ..... Allows preload function for testing.
2. SYSRESET ..... Reset line from microprocessor.
3. A2 ..... Address line from address bus.
4. A1 ..... Address line from address bus.
5. HDSHAKE ..... Handshake line (CTS/RTS).
6. CK ..... External clock.
7. E ..... Enable line from microprocessor.
8. AUXDECOD ..... Extra decoding line (e.g. board level decoding).
9. A3 ..... Address line from address bus.
10. A4 ..... Address line from address bus.
11. A5 ..... Address line from address bus.
12. GND ..... Reference power supply ground.
13. /OE ..... Output enable line.
14. A6 ..... Address line from address bus.
15. SPEEDSEL ..... Speed selection line.
16. DIV4 ..... MSB 4-bit synchronous counter.
17. DIV3 ..... 3rd stage synchronous counter.
18. DIV2 ..... 2nd stage synchronous counter.
19. DIV1 ..... LSB 4-bit synchronous counter.
20. CS0 ..... UART chip select line (CSO).
21. BLOCREC ..... Bloc receive line.
22. DIR DIV ..... Direct or divided clock.
23. /TPH ..... External use flag.
24. VCC ..... 5 V power supply.



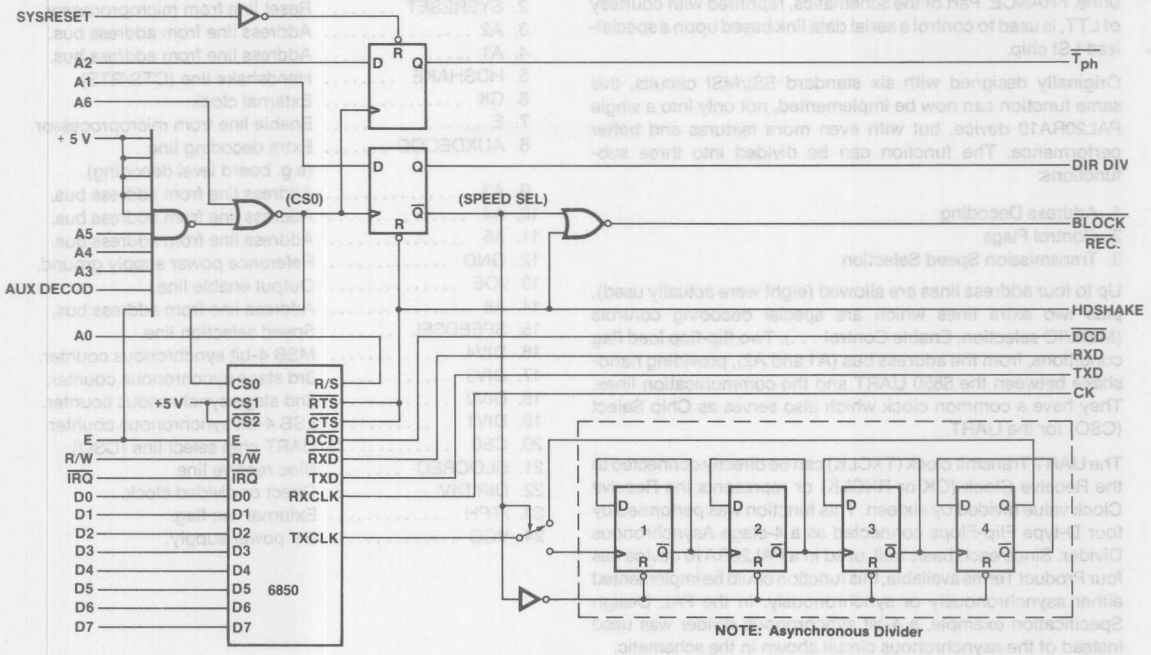


Figure 6.

After

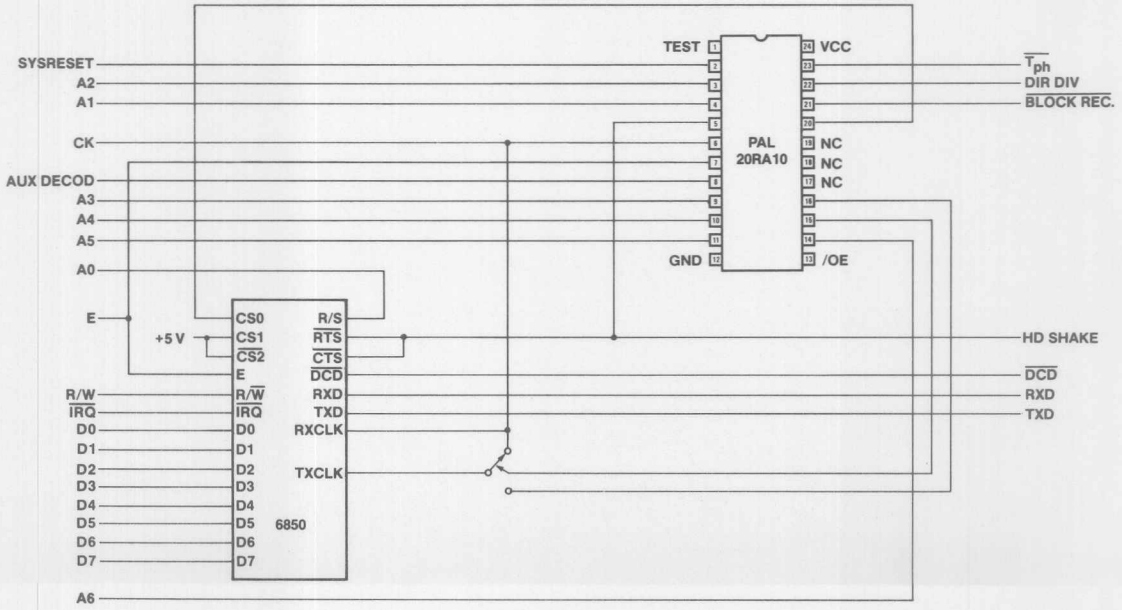


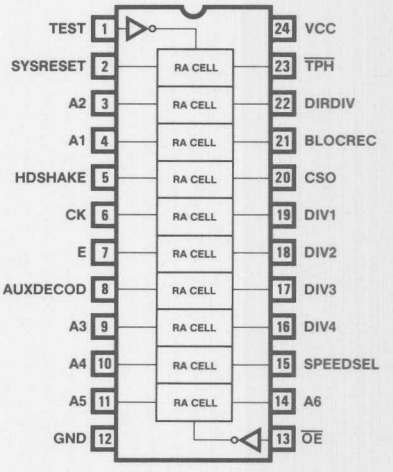
Figure 7.

PAL Device Design Specification

```
Title     Serial Data Link Controller
Pattern   Link.pds
Revision  A
Author    Jose Juntas / Kelvin Chow
Company   Monolithic Memories Inc., Santa Clara, Ca
Date      3/1/85

CHIP SE_CH_CNTRL  PAL20RA10

TEST SYSRESET A2 A1 HDSHAKE CK E AUXDECOD A3 A4 A5 GND
/OE A6 SPEEDSEL DIV4 DIV3 DIV2 DIV1 CSO BLOCREC DIRDIV
/TPH VCC

EQUATIONS

/TPH              := A2                    ;Load A2 as flag
/TPH.CLKF         = CSO                    ;CLK W/ ADDR. decode
/TPH.SETF         = SYSRESET               ;global system reset

DIRDIV            := A1                    ;Load speed ratio
DIRDIV.CLKF       = CSO                    ;CLK W/ ADDR. decode
DIRDIV.SETF       = /HDSHAKE               ;CLR by CTS/RTS line

/BLOCREC          = /DIRDIV                ;Controlled by speed
                  + HDSHAKE                ;option and CTS/RTS
                                           ;line
CSO               = /A6*A5*A4*A3*AUXDECOD*E
                                           ;UART address valid
/DIV1             := DIV1                  ;4-bit synchronous
                                           ;divider LSB
/DIV1.CLKF        = CK                     ;CLK by CK(external)
/DIV1.SETF        = /DIRDIV                ;CLR by speed option

/DIV2             := /DIV1*/DIV2           ;2ND stage of
                  + DIV1*DIV2              ;divider
/DIV2.CLKF        = CK                     ;CLK by CK(external)
/DIV2.SETF        = /DIRDIV                ;CLR by speed option

/DIV3             := /DIV2*/DIV3           ;3RD stage of
                  + /DIV1*/DIV3            ;divider
                  + DIV1*DIV2*DIV3
/DIV3.CLKF        = CK                     ;CLK by CK(external)
/DIV3.SETF        = /DIRDIV                ;CLR by speed option

/DIV4             := /DIV3*/DIV4           ;4TH stage of
                  + /DIV2*/DIV4            ;divider MSB
                  + /DIV1*/DIV4
                  + DIV1*DIV2*DIV3*DIV4
/DIV4.CLKF        = CK                     ;CLK by CK(external)
/DIV4.SETF        = /DIRDIV                ;CLR by speed option

SPEEDSEL          := /A1                   ;Load speed choice
SPEEDSEL.CLKF     = CSO                    ;CLK W/ ADDR. decode

SPEEDSEL.SETF     = /HDSHAKE               ;CLR by CTS/RTS line

SIMULATION

TRACE_ON A1,A2,A3,A4,A5,A6,E,              ;Signals to be
         AUXDECOD,SYSRESET,/TPH,HDSHAKE,          ;observed
         CSO,SPEEDSEL,DIRDIV,CK,
         DIV1,DIV2,DIV3,DIV4
SETF SYSRESET,/HDSHAKE                     ;Reset all regs

CHECK /SPEEDSEL,/DIRDIV,TPH
SETF /SYSRESET,A1,A2,A3,A4,A5,/A6,HDSHAKE,  ;Set decode
    E,AUXDECOD                             ;condition

CHECK /SPEEDSEL,DIRDIV                     ;Check SPEEDSEL and
                                           ;DIRDIV regs

FOR I:=1 TO 15 DO
 BEGIN                                     ;This portion
  SETF CK                                  ;simulates divide
                                           ;by four counter
  SETF /CK
 END
```

Simulation Results

```
Page :  1
             g g  g gg  gg gg gg g g gg gg gg  gg gg gg
 A1          XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 A2          XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 A3          XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 A4          XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 A5          XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 A6          XXLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 E           XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 AUXDECOD    XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 SYSRESET    HHLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
/TPH         LLLLHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 HDSHAKE     LLHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 CSO         XXHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 SPEEDSEL    LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 DIRDIV      LLLLHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 CK          XXXXXHHLHH LHHLHHLHHL HHLHHLHHLH HLHHLHHLHH
 DIV1        XLLLLHHHLL LLHHHLLLHH HLLLHHHLLL HHHLLLHHHL
 DIV2        XLLLLLLLLH HHHHHLLLLL LHHHHHLLLL LLLHHHHHHL
 DIV3        XLLLLLLLLL LLLLLHHHHH HHHHHHLLLL LLLLLLLLLH
 DIV4        XLLLLLLLLL LLLLLLLLLL LLLLLLHHHH HHHHHHHHHH

Page :  2
             gg gg gg g
 A1          HHHHHHHHHH
 A2          HHHHHHHHHH
 A3          HHHHHHHHHH
 A4          HHHHHHHHHH
 A5          HHHHHHHHHH
 A6          LLLLLLLLLL
 E           HHHHHHHHHH
 AUXDECOD    HHHHHHHHHH
 SYSRESET    LLLLLLLLLL
/TPH         HHHHHHHHHH
 HDSHAKE     HHHHHHHHHH
 CSO         HHHHHHHHHH
 SPEEDSEL    LLLLLLLLLL
 DIRDIV      HHHHHHHHHH
 CK          LHHLHHLHHL
 DIV1        LLHHHLLLHH
 DIV2        LLLLLHHHHH
 DIV3        HHHHHHHHHH
 DIV4        HHHHHHHHHH
```



Functional Description

One of the more widely used computer families is the Digital Equipment Corp.'s PDP-11 series. This family of computers uses the DEC unibus to communicate between cards. A specific protocol is required to interface a card to the unibus. This protocol is described in the available DEC literature.

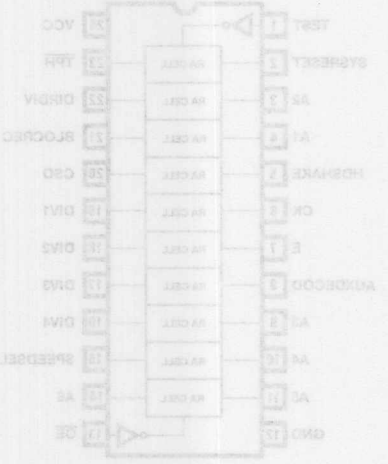
Since the unibus is an asynchronous bus, much of the interface circuitry consists of combinational logic to generate specific signals and flip-flops which are set and reset as flags. This tends to use a lot of SSI and MSI logic packages. Using Monolithic Memories' PAL devices, much of this logic can be condensed into a few packages. Figure 2 is the schematic diagram for an interrupt Controller to be used on the unibus. (p. 6-30 of the 1976 DEC PDP-11 Peripherals Handbook.)

Many cards communicate over the bus by taking control of the unibus with an interrupt request, and then do whatever they require before releasing control. As can be seen, this interrupt controller takes six special interface ICs, (380 and 8881 bus drivers and receivers) eight MSI, SSI ICs, (7400, 7402 nad 7474s) along with some transistors and discrete parts. This parts count can be considerably reduced by using PAL20RA10 and PAL20L10 devices.

Figure 1 shows how the circuit with the PAL devices would look. The two PAL devices allow almost all of the 7400, 7402 and 7474 packages to be removed. (Almost a 4-1 saving in chip count.) In addition the preload pin (PRLD) on the 20RA10 allows the flip-flops to be easily set to a known state on power up, or when re-initializing. So the PAL devices reduce the logic package count from eight chips to three.

This shows that by using PAL devices substantial space and circuit savings can be realized when interfacing to the unibus.

In the schematic shown, thre are three VLSI devices, three MSIs and two SSIs. Using a PAL20RA10 logic circuit, it is possible to replace three MSIs and one SSI device, thereby reducing the chip count by a factor of two. The ICs inside the enclosed loop were replaced.



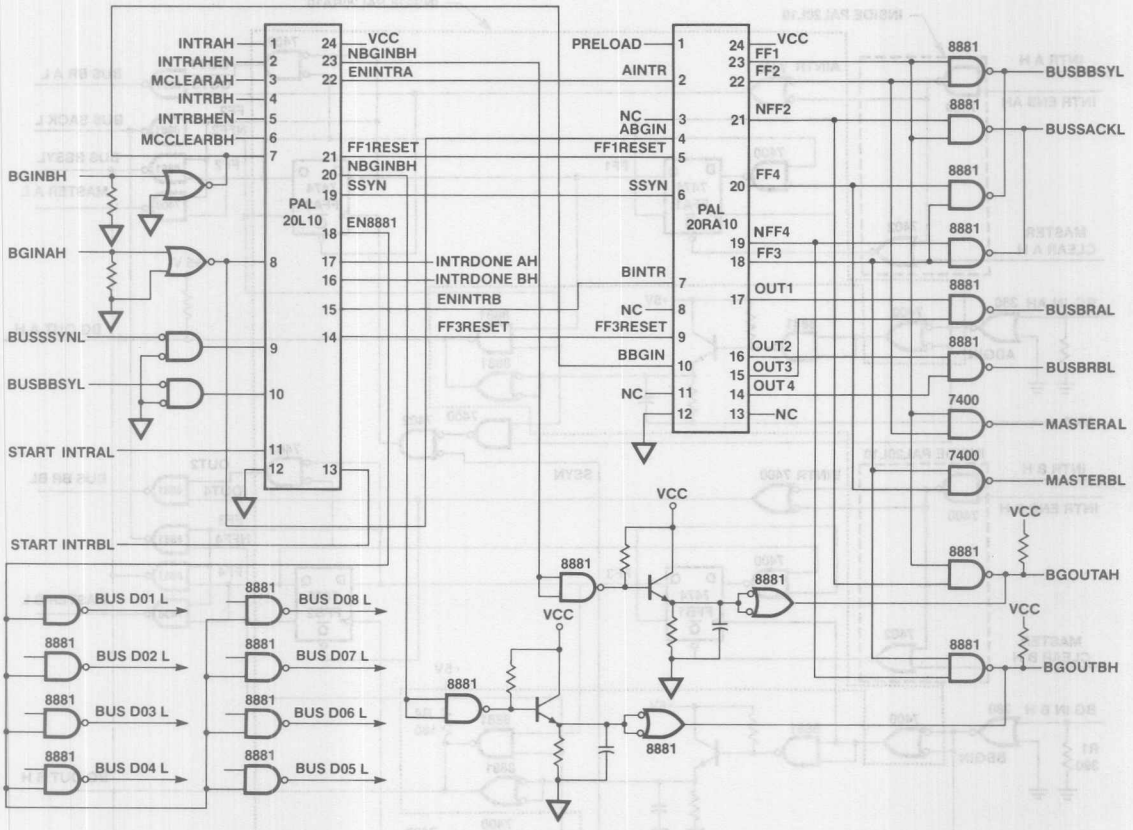


Figure 1.

PAL Design Specification

```
PAL20L10
INTRP01
INTERRUPT LOGIC
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
INTRAH INTRAHEN MCLEARAH INTRBH INTRBHEN MCLEARBH BGINBH
BGINAH BUSSSYNL BUSBBSYL STARTINTRAL GND
STARTINTRBL FF3RESET ENINTRB INTRDONEBH INTRDONEAH EN8881
SSYN NBGINBH FF1RESET ENINTRA NBGINAH VCC

EQUATIONS

/NBGINAH        =  BGINAH                ;FF1 CLK CONTROL
                                         ; BLOCK A
/FF1RESET       =  MCLEARAH+ENINTRA      ;SET FF1 CONTROL
                                         ; BLOCK A
/ENINTRA        =  INTRAHEN*INTRAH       ;ENABLE INTERRUPT A

/NBGINBH        =  BGINBH                ;FF3 CLOCK CONTROL
                                         ; BLOCK B
/SSYN           =  BUSSSYNL*BUSBBSYL     ;SYNCHRONIZE FF2 &
                                         ; FF4
/EN8881         = STARTINTRAL*STARTINTRBL
                                         ;INTERRUPT BUS
/INTRDONEAH     = BUSSSYNL+STARTINTRAL   ;SIGNAL INTERRUPT
                                         ; DONE
/ENINTRB        = INTRBH*INTRBHEN        ;ENABLE INTERRUPT B

/FF3RESET       = MCLEARBH+ENINTRB       ;SET FF3 CONTROL
                                         ; BLOCK B
/INTRDONEBH     = BUSSSYNL+STARTINTRBL   ;SIGNAL INTERRUPT
                                         ; DONE
```

```
DESCRIPTION

COMBINATORIAL LOGIC FOR PAL20RA10 INTERRUPT CONTROLLER
(1ST PART OF THE TWO PALS SOLUTION: PAL20L10 & PAL20RA10)
MONOLITHIC MEMORIES INC., SANTA CLARA, CA
DAN KINSELLA                7/19/84

NOTE: THIS PAL DESIGN SPEC WAS ASSEMBLED ON PALASM V1.7.
```

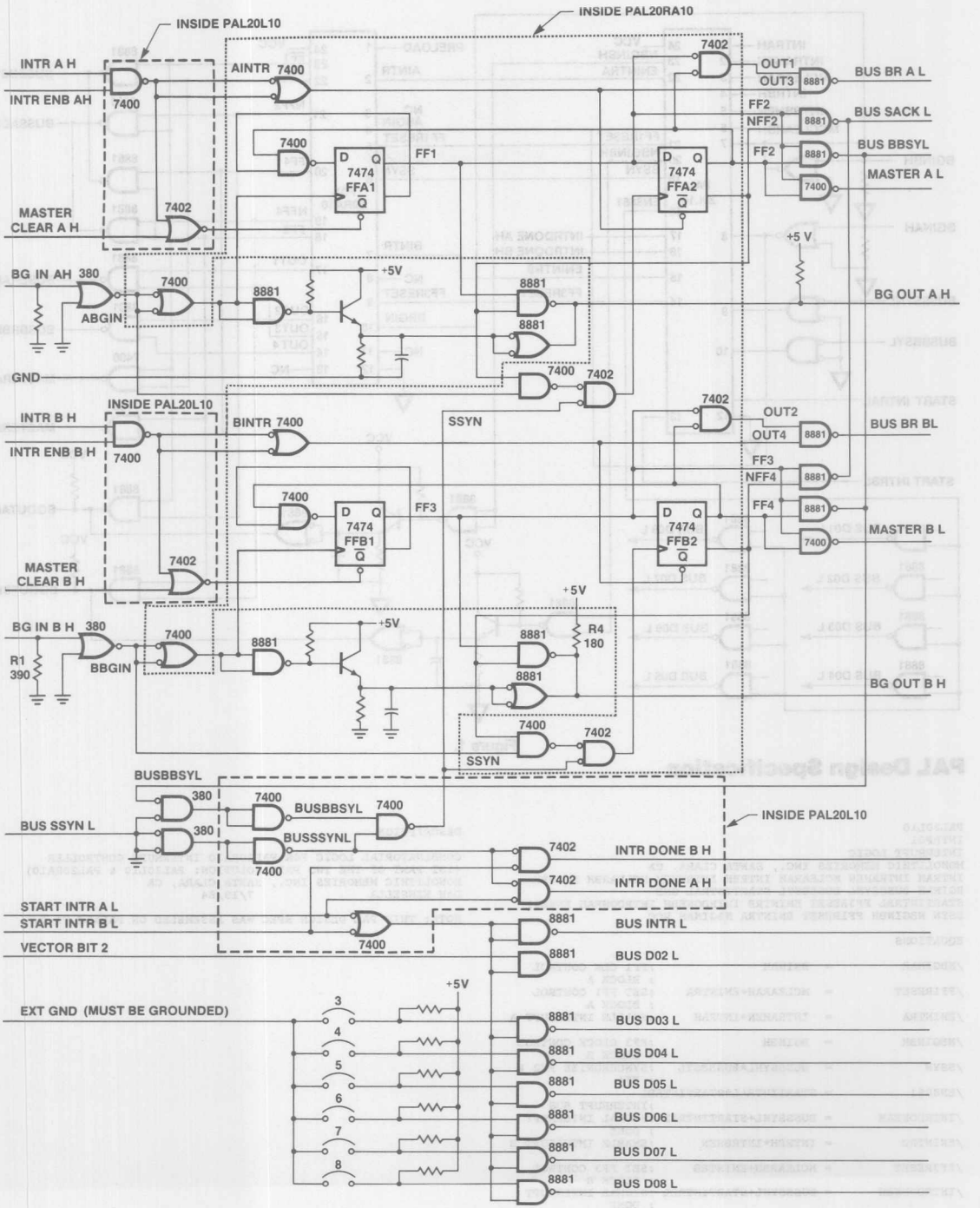
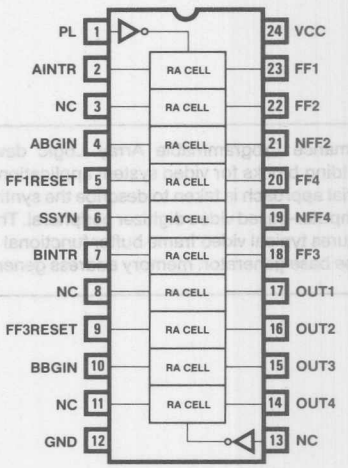

INSIDE PAL20L10
INSIDE PAL20RA10
INTR A H
INTR ENB AH
7400
AINTR
MASTER CLEAR A H
7402
FF1
7474 FFA1
7474 FFA2
OUT1
OUT3
8881
BUS BR A L
FF2
NFF2
BUS SACK L
BUS BBSYL
MASTER A L
+5 V
BG IN AH
380
ABGIN
+5V
BG OUT A H
GND
INTR B H
INTR ENB B H
BINTR
SSYN
OUT2
OUT4
BUS BR BL
FF3
NFF4
FF4
7474 FFB1
7474 FFB2
MASTER B L
MASTER CLEAR B H
BG IN B H
R1
390
BBGIN
R4
180
BG OUT B H
BUSBBSYL
BUS SSYN L
BUSSSYNL
INTR DONE B H
INTR DONE A H
START INTR A L
START INTR B L
VECTOR BIT 2
BUS INTR L
BUS D02 L
EXT GND (MUST BE GROUNDED)
BUS D03 L
BUS D04 L
BUS D05 L
BUS D06 L
BUS D07 L
BUS D08 L

Figure 2.

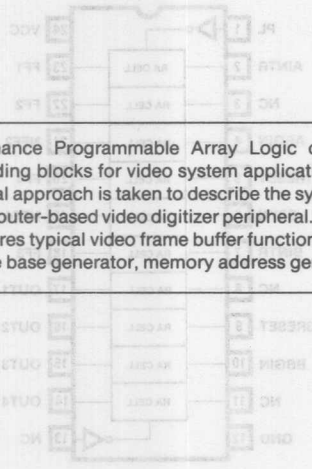
```
Title     DEC PDP-11 unibus interrupt controller
Pattern   Control.pds
Revision  A
Author    Dan Kinsella
Company   Monolithic Memories Inc., Santa Clara, CA
Date      3/1/85

CHIP INTR_CONTROL PAL20RA10

PL AINTR NC ABGIN FF1RESET SSYN BINTR NC FF3RESET BBGIN
NC GND
NC OUT4 OUT3 OUT2 OUT1 FF3 NFF4 FF4 NFF2 FF2 FF1 VCC

EQUATIONS

/FF1             := /FF1*FF2              ;Master control
FF1.SETF          = /FF1RESET             ;block A
FF1.CLKF          = /ABGIN

FF2              := FF1                   ;Bus Busy Signal
FF2.SETF          = /AINTR
FF2.CLKF          = ABGIN*FF2*/SSYN

/NFF2            := FF1                   ;Bus sack signal
NFF2.SETF         = /AINTR
NFF2.CLKF         = ABGIN*NFF2*/SSYN

/FF3             := /FF3*FF4              ;Master control
FF3.SETF          = /FF3RESET             ;block B
FF3.CLKF          = /BBGIN

FF4              := FF4                   ;Bus busy signal
FF4.SETF          = /BINTR
FF4.CLKF          = BBGIN*FF4*/SSYN

/NFF4            := FF3                   ;Bus sack signal
NFF4.SETF         = /BINTR
NFF4.CLKF         = BBGIN*NFF4*/SSYN

/OUT1             = FF1+FF2               ;Bus request signal
                                          ;block A
/OUT2             = FF4+FF3               ;Bus request signal
                                          ;block B
/OUT3             = AINTR                 ;Intr. signal for
                                          ;bus req. block A
/OUT4             = BINTR                 ;Intr. signal for
                                          ;bus req. block B

SIMULATION

TRACE_ON FF1RESET FF3RESET AINTR BINTR SSYN ABGIN BBGIN
         FF1 FF3 NFF2 NFF4 OUT1 OUT2 OUT3 OUT4

SETF /FF1RESET /FF3RESET AINTR BINTR     ;Reset all regs

SETF FF1RESET FF3RESET /AINTR /BINTR     ;Clock FF1 and FF3
     ABGIN BBGIN                         ;regs

SETF /SSYN                               ;Clock NFF and NFF3
                                         ;regs
```

```
Page :  1
                gg g
 FF1RESET       LHHH
 FF3RESET       LHHH
 AINTR          HLLL
 BINTR          HLLL
 SSYN           XXXL
 ABGIN          XHHH
 BBGIN          XHHH
 FF1            LLLL
 FF3            LLLL
 NFF2           XLLL
 NFF4           XLLL
 OUT1           XHHH
 OUT2           XHHH
 OUT3           LHHH
 OUT4           LHHH
```



Video Frame Grabber

Alfie Gilbert



High performance Programmable Array Logic devices are powerful building blocks for video system applications. In this paper a tutorial approach is taken to describe the synthesis of a personal computer-based video digitizer peripheral. The design exercise features typical video frame buffer functional modules such as a time base generator, memory address generator and memory control logic which are integrated into a single MegaPAL™ device. To maximize the information transfer from this tutorial the reader should be moderately familiar with general PAL design concepts, however, a minimal amount of video knowledge is required to grasp the design implementation.

HOW TO GET LUCKY IN CALIFORNIA

Hello my name is Alfie Gilbert. That is me on the right with my supervisor, John Birkner. John co-invented the PAL logic circuit back in 1977 and has championed these remarkable devices ever since. As a token of MMI's appreciation of John's effort, he drives a PORSCHE CARRERA which is quite a beautiful car. Not bad John , not bad at all!

If you take the time to ask John how one might be so lucky as to be handed the keys to a turbo, he will answer you in one word, INNOVATE. So how do I innovate my way into a Porsche you are probably wondering? If you are a mature electronic engineer, you no doubt have some strong ideas and some direct innovative work experiences to serve as guidelines. Students and recent graduates however face a little different challenge. Although these individuals typically have few preconceptions about what is feasible, which is quite an asset, they often have not experienced the design process.

exercise to familiarize the novice designer with some of the tradeoffs and "tricks-of-the-trade" involved with logic design. If you are a student, I hope this application will help focus your creativity into something rewarding like John's car. GO FOR IT!

As you probably noticed by now, this application note is written in the literary first person which is very unusual for technical subjects. I chose this style because I wanted to relate to you directly, designer to designer. Although I am presently employed as an applications specialist , I enjoy teaching electronics as a second career, which over the years has put me in direct contact with many students who seem to share one thing in common , the desire to invent something. PAL devices from the student perspective, are ideal vehicles for creative design because they directly realize digital logic equations in silicon. If a design can be described with Boolean algebra or state equations, it can be built with PAL devices. All this magic is made possible by systematically burning out a fuse array , the height of simplicity in ASIC design.

SOME PERSPECTIVE

PAL logic circuits participate in a segment of the semiconductor marketplace known as ASICs , which is an acronym for Application Specific Integrated Circuits. Programmable Array Logic, Gate Array, Standard Cell, and Full Custom technologies compete for market share in the ASIC arena. Typically, ASICs implement the functionality that would occupy a whole circuit board of standard MSI logic onto a single chip. Of the four ASIC technologies, PAL devices are by far the simplest to use. PAL-based designs typically can be implemented in a far shorter timeframe than with the other three alternatives. Development systems for PAL designs are less expensive as well. A typical PAL development system consists of a personal computer and a programming unit. This FRAME GRABBER design exercise, for example, was done using a COMPAQ computer, a VARIX programmer, and a TEKTRONIX scope. The pearl of wisdom which I would like to pass on to the reader is simple: No matter what you are designing or what technology (ASIC or standard logic) you use to design, it is easier to prototype the design with PAL devices. Any designer will tell you that PALs ARE GREAT BUILDING BLOCKS. Pros also know that true artists SHIP thier product FIRST. This is known as opening the window of opportunity and it is very crucial to the success of a product. The following is an excerpt from the Macintosh Design Case History article which appeared in IEEE SPECTRUM DECEMBER 1984. If you have a chance, read the entire article.

The computer's circuit density was one bottleneck. Mr. Smith had trouble paring enough circuitry off his first two prototype to squeeze them onto one logic board... Another problem with the Macintosh display was its limited dot density...
Mr. Smith could think of but one alternative: combine the video and other miscellaneous circuitry on a single custom n-channel MOS chip. Mr. Smith began designing such a chip in February 1982. During the next six months the size of the hypothetical chip kept growing...
After thinking about the problem for three months, Mr. Smith concluded in July 1982 that "the difference in size between this chip and the state of Rhode Island is not very great." He then set out to design the circuitry with higher-speed programmable array logic--as he had started to do six months earlier. He had assumed that higher resolution in the horizontal video required a faster clock speed. But he realized that he could achieve the same effect with clever use of faster bipolar-logic chips that had become available only a few months earlier. By adding several high speed logic circuits and a few ordinary circuits, he pushed the resolution to 512 dots.
Another advantage was that the PALs were a mature technology and their electrical parameters could tolerate large variations from the specified values, making the Macintosh more stable and more reliable--important characteristics for a so-called appliance product.

WHAT WIZARDS KNOW

The fellow referred to as Mr. Smith in the excerpt from SPECTRUM is Burrell Smith. It is amusing that Burrell's business card really does read "HARDWARE WIZARD" which is a fairly accurate title considering his handiwork. He and Andy Hertzfeld (computer cult heroes out here in Silicon Valley) wrote a great technical description of the Macintosh System Architecture and System Software for BYTE Magazine, February 1984 (Volume 9, Number 2). I encourage you to reference that issue to get some insight into how creatively PALs may be employed in system design. The six PAL devices which are designed into the Macintosh are a lot more than "glue logic". They form the core of the video graphics/bus management hardware. I can not elaborate too much more on the functionality of the Macintosh PAL set without touching on some proprietary subjects, so I won't, other than to mention that the MAC PAL devices are all from the "20 pin " family, PAL16R8-type parts. As you probably guessed, I own a MAC (as well as several other computers which I keep for historical reasons) really like it, and respect its designers. NICE WORK FOLKS!

OLD PAL DEVICES, NEW PAL DEVICES

Technology has a way of marching on, with us or without us, therefore a substantial part of the design challange is often in picking the technological alternative which is most in league with the future. When Burrell Smith designed the MAC's PAL set (back in 1982) the industry "workhorse" was the PAL16R8/PAL16L8 family of parts. The big advances at that time were in the speed area, fast (A) PAL devices which featured a propagation delay of only 25 ns were novel. Of course today in 1985 we have even faster (B) PAL logic circuits which save yet another 10 ns of the propagation delay resulting in a blazing fast TPD of only 15 ns. Over the past few years, we have also seen the power consumption of MMI PAL devices decrease by a factor of two (-2) and a factor of four (-4 parts). Most of these improvements and variations in speed and power for the 20-pin family have resulted from changes in our semiconductor process. All of these changes are certainly noteworthy, however, I think ARCHITECTURE is the area where MMI has really advanced by leaps and bounds. In 1984, MMI brought to market the first parts of our megaPAL family, the PAL64R32 and the PAL32R16 devices. MMI was flattered to be given the PRODUCT of the YEAR award (Electronic Products Magizine) for the PAL64R32 logic circuits. I chose it for the heart of this design exercise. Our megaPAL devices feature several significant architectural improvements over previous PAL devices most notably, product term sharing, programmable output polarity, and register bypass. The PAL64R32 device has 32 input pins, 32 output pins, and 256 product terms to relate the inputs to outputs. The PAL64R32 logic circuit is four times as dense as the PAL16R8 device, one of which is also included in this design exercise, because it will soon be fabricated in CMOS.

DIFFERENT STROKES FOR DIFFERENT FOLKS

Equally significant to the megaPAL, in terms of architecture, is another new PAL from MMI, the PAL20RA10 logic circuit (the RA in the designator stands for Registered Asynchronous). Each of the 10 identical macrocells of the 20RA10 features a 7474 type D flip-flop which may be asynchronously set or reset (via a product term). The clock to the macrocell register is also derived from a product term which is a somewhat radical departure from traditional PAL architectures. Other features of the 20RA10 include macrocell register bypass (this is accomplished by simultaneously asserting set and reset, which of course is an illegal condition for a 7474 register), and programable output polarity. The PAL20RA10 device is an extremely flexible and unstructured device. It is ideal for interfacing dissimilar signals, or interfacing to an asynchronous system bus. Needless to say, I have also chosen to include the PAL20RA10 device in this design exercise. MegaPAL devices are highly structured logic elements which make them ideal for synchronous logic. The

PAL20RA10 logic circuits are just the opposite, they are flexible and most effective in asynchronous environments. Because most applications have elements of structured and random logic, I am very enthusiastic about designing with megaPAL devices and PAL20RA10 logic circuits in tandem. The "trick", of course, is to partition the system so that synchronous logic is realized in the highly structured megaPAL device while the random logic is implemented by the more flexible RA PAL.

THE IBM (INVERSE BURRELL MACHINE) CHIP SET

As I am sure you have noticed, computer coupled raster graphics video systems have abounded in recent years. In just the consumer electronic sector we have witnessed the video game, the personal computer, and most recently digital television. This explosive growth, in part, has been fueled by the decreasing cost of memory, specifically RAM. RAM memory is important in modern raster video systems because it retains the luminance and chrominance information required by the display monitor to generate an image. This RAM memory is often refered to as the bit map of the video raster. A video controller is a device which facilitates a one-to-one pairing between each bit of the video RAM array and a specific location (coordinate) on the monitor's display tube. If you did investigate the Macintosh's video design, you no doubt realize the power and flexibity of PAL devices in implementing video controllers. What I propose we do together at this point, is invent a widget that does the INVERSE operation of a video controller, specifically a video digitizer/frame buffer. Video controllers typically pull information out of RAM to form an output signal known as "composite video", our unit accepts composite video as input and stores that information in a RAM array. We will call it a FRAME GRABBER because that is exactly what it does. The host environment for our FRAME GRABBER will be a personal computer. This environment will provide a debug monitor during development and allow us to "reconstruct" our digital image thru the PC graphics mode for display. The FRAME GRABBER will interface thru the I/O channel of ubiquitous IBM-PC compatible computers. Application support software to simulate "gray scale" will be writen in a higher level language (turbo Pascal), rather than Assembly language, in the hopes of being self explanatory.

Before beginning our design exercise, let us review some video principles and get familiar with the signals which we will be dealing with. If you have no idea how a television works, Milt Wilcox wrote a good application note titled A Color TV Primer for the E.E. which may be found in National Semiconductor Linear Applications Handbook. It is worth reading, as is the video section of Donald Finke's classic Electronic Engineering Handbook.

A BRIEF OVERVIEW OF VIDEO

Suppose we focus a typical consumer video camera (VHS or Beta format) on a target such as the one depicted in FIGURE ONE. The output signals of these types of video cameras were standardized several years ago by the EIA and formalized in a document known as RS-170A. This type of video signal is often referred to as "NTSC composite video" because it contains four distinct signal components which are merged together (for better or worse if you are a videophile). These four signals are luminance(brigthness) chrominance (color), audio (sound), and synchronization. This type of signal is sufficient to directly drive most monitors, and is also the basis for television broadcast as well. The output signal of most video sources (VCRs, Videodisks,etc.) that are sold in North America are also composite video, thus making them identical to this video camera example. That fact is handy to know, because it insures that our FRAME GRABBER widget will be compatible with most contemporary video sources (compatibility is a very important issue and should always be carefully considered by the designer as early as possible in a product design cycle).

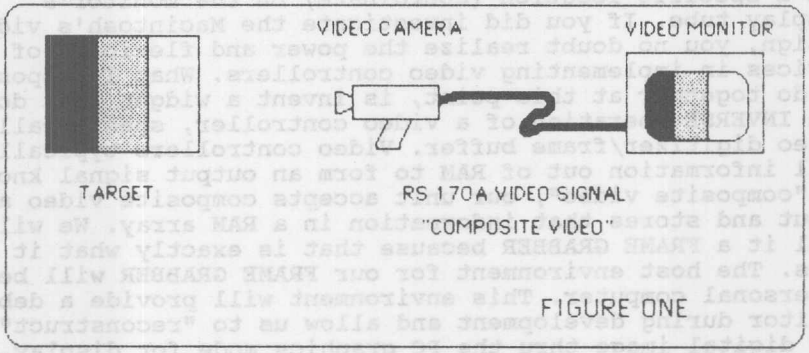


FIGURE ONE

Let's investigate the video output signal generated by the camera aimed at the "gray bar" target depicted in FIGURE ONE. The camera signal illustrated in FIGURE TWO is for the "scan" from point A to point B. The important thing for the reader to notice is that the camera output signal is proportional to the brightness (luminous intensity) of the target during the scan fron A to B. This process of scanning from right to left happens 262.5 times as the target is traversed from top to bottom. In between each scan is a time interval known as horizontal retrace (or H sync). During this time the target scan is blanked as the raster retraces its path to get into position just slightly below point A. The scan process then repeats. The horizontal retrace time interval may be seen in the camera output signal of FIGURE TWO as the "pulse" which is labeled SYNC. It takes about 64 ms for a scan and retrace to occur. A collection of 262.5 consecutive

[illegible] from the top of the target to the bottom. At this point, the scan blanks for a rather long time (the interval is known as vertical retrace and lasts for a period equal to 20 horizontal scans) as the raster moves vertically to the top of the target again. The whole field is then traced out again, however, the scans of the second field are offset by half a horizontal line. Two consecutive fields of video are known as a VIDEO FRAME. It is precisely this amount of information, a frame, which our widget will digitize and store in RAM, because this is the minimum information required to reconstruct an image. The total number of horizontal scans in one video frame is 262.5 scans/field times 2 fields/frame or 525 scans/frame (NTSC composite video is often referred to as a 525 line system). Incidentally, the reason why the two consecutive video fields are offset by half a horizontal scan is a "trick" called interlacing and it is helpful in reducing display tube flicker.

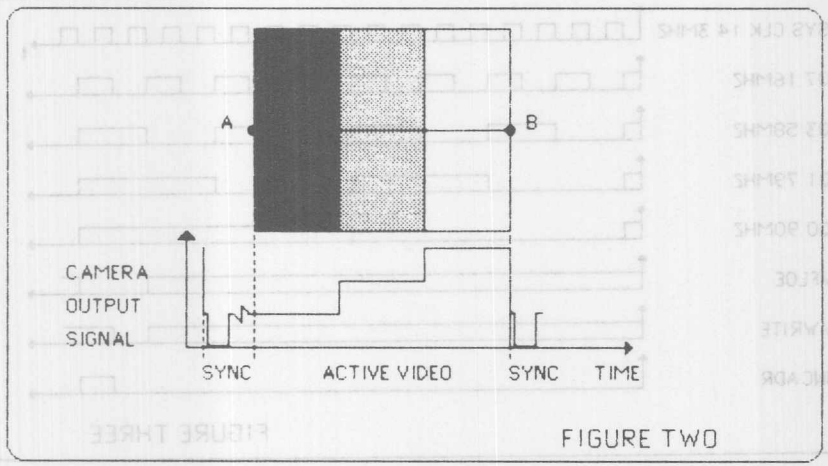


FIGURE TWO

TIMING IS EVERYTHING

In video systems, even more than in life, timing is critical. Video is a highly repetitive process and every event in that process happens in synchronous with a clock. The "clock" for composite video is usually an integer multiple (harmonic) of the NTSC "color burst" frequency. Color burst frequency is defined by the EIA to be 3.579545 MHz (a copy of the RS-170A signal standard is included as an appendix to this design exercise). The horizontal scan rate of composite video approximately 15,750 Hz (one scan period (H) = 63.556 ms). The field rate is 525/2 times the horizontal scan rate (about 60 Hz) which then makes the frame rate 30 Hz.

SOME QUICK NUMBERS

On the bus expansion slots of personal computers which are hardware compatible with the IBM-PC is a signal called OSC which has a frequency of 14.318 MHz (fourth harmonic of color burst). Since our FRAME GRABBER widget will be hosted by a PC, it is natural to use OSC as our system clock. The flash A/D conversion rate was chosen to be equal to color burst for this exercise. Some quick calculations on the FRAME GRABBER timing will reveal the following facts: If 30 Hz is the video frame rate and 14.3 MHz is the system clock, 476190 system clock periods will elapse in one video frame. Since a nineteen bit counter is required to count to 476190 , the heart of the FRAME BUFFER will indeed be a counter of this length. If the video signal is digitized to two bits of accuracy per flash conversion and the flash conversion rate is 3.58 MHz 119050 samples or 238100 bits (roughly 32k bytes of RAM) will be required to retain the information content of a video frame in memory.

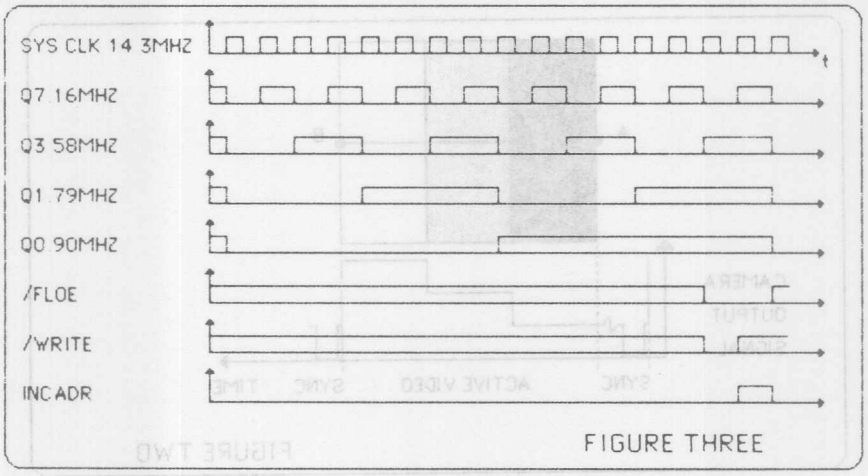


FIGURE THREE

Since the FRAME GRABBER system clock rate is 14.3 MHz and the A/D flash conversion rate is 3.58 MHz four system clock cycles (280 nsec) elapse between conversions. The memory organization is 8 bits wide and the A/D accuracy is 2 bits wide therefore 4 sucessive flash conversions will have to be "packed" into each byte of RAM. Four sucessive flashes will require 16 system clock cycles (1.12 ms) to complete. A 4-bit counter obviously increments to 16, therfore it is natural to segment our requisite 19-bit counter into a 4-bit counter (which will be implemented as the time base module of the megaPAL design specification) and a 15-bit counter (address generation module). A 15-bit counter increments to 32k which is the required memory address range to store a video frame at our particular

A/D conversion rates and resolution. The reader can crosscheck these rough numbers by remembering that a memory write occurs every 1.12 ms (16 cycles of the system clock) and a horizontal scan takes about 64 ms, therfore 58 bytes are required to store a line of video. There are 525 lines in a video frame, so the address counter will increment to 30,450 over the duration of a video frame in we allow it to "free run". The outputs of the FRAME GRABBER time base module are illustrated in FIGURE THREE, as well as the write enable signal, to the static RAMs (/WRITE), the flash A/D bus output enable (/FLOE) and address counter increment signal (INCADR). This should give some idea of the basic system timing.

CLAMP PLEASE

In most video cameras, the RS-170 output signal is usually AC coupled at the source for isolation purposes. This situation presents a small problem for the FRAME GRABBER because the input video signal is "floating", i.e., it lacks DC integrity. In order for an AC-coupled signal to be digitized, it must be "clamped" (DC restored) before it can be compared against a series of voltage thresholds. A traditional circuit to accomplish this technique of flash A/D conversion usually involves a series of linear comparators and an accurate resistor network to divide down a precision voltage reference. DC restoration is generally accomplished by "dumping" one side of a coupling capacitor with a transistor during the back porch interval of horizontal retrace. For low-resolution applications, however, a much simpler PAL-based circuit can perform the same function.

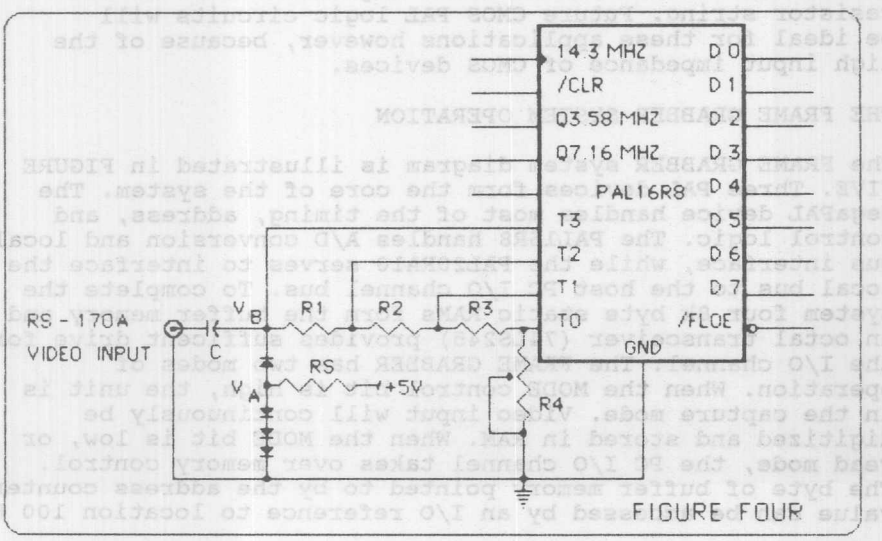
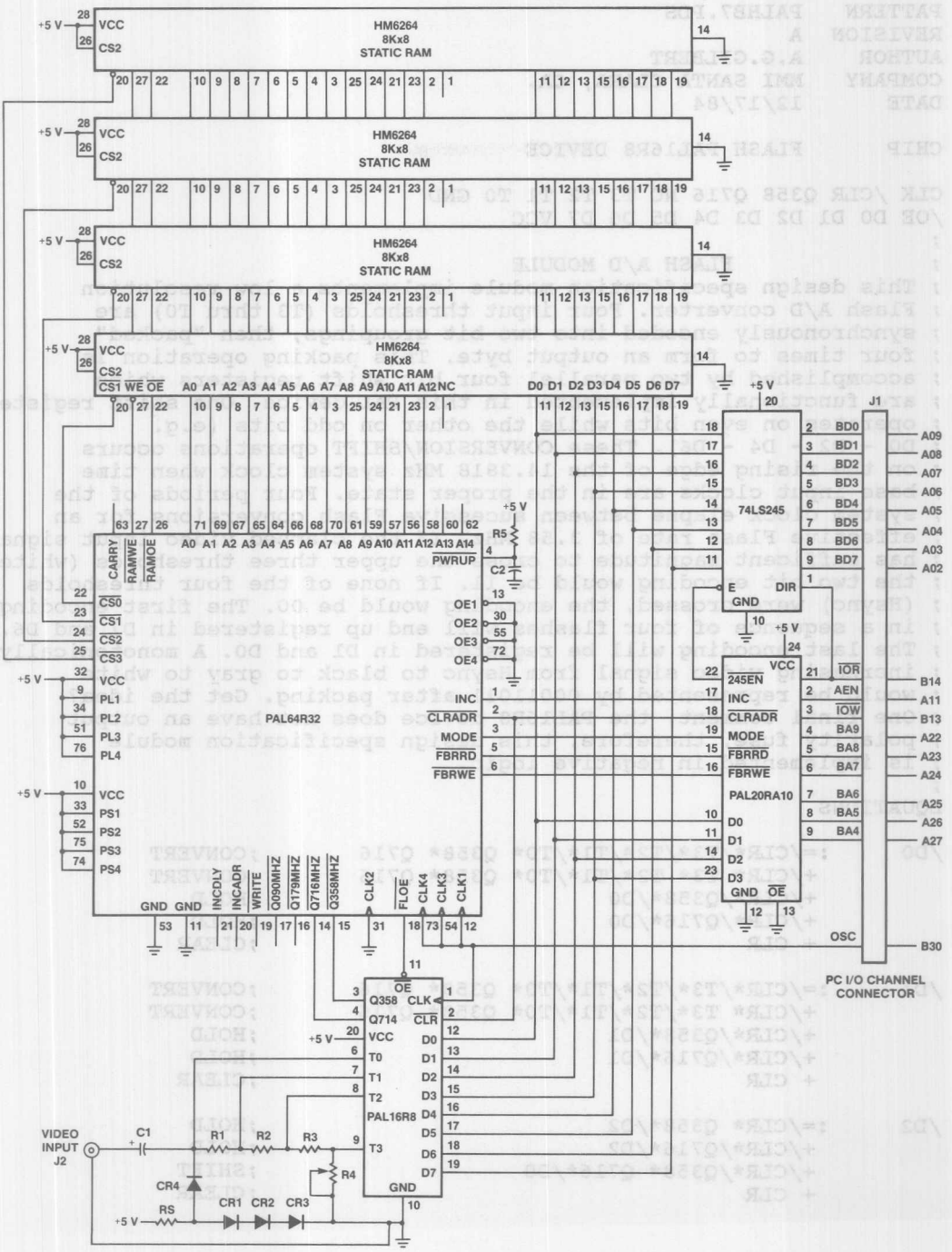


FIGURE FOUR

The FRAME GRABBER video clamp/flash converter circuit is shown in FIGURE FOUR. Let's investigate how this circuit operates from an analog point of view. The DC voltage at node A (VA) is about 2.1 volts if one assumes each of the three silicon diodes has a forward voltage drop of .7 volt when driven hard (RS must be on the order of 500 ohms). The voltage VB will then be 1.4 volts or one diode drop below VA. If the Thevenin equivalent impedance to ground at node B was infinite, the most negative part of the AC coupled video input signal (H sync) would be clamped at 1.4 volts by the action of the diodes and capacitor. In reality the circuit is not a perfect clamp because other currents do indeed enter and leave node B, but it does work reasonably well if C is large (on the order of 100 microfarads for our Z Thevenin of 5k ohms). The clamped video signal will be diminished at the input nodes T2, T1, and T0 by the ratio dictated by the voltage divider formed by R1, R2, R3, and R4. The voltages at nodes T3, T2, T1, and T0 implicitly are "compared" against the internal threshold (about 1.4 volts) of the PAL device input structure which causes a logic level discrimination or conversion. This simple flash A/D converter implemented by a PAL16R8 and a few passives performs surprisingly well if the PAL digital noise is mostly common mode with the input (avoid ground loops at all cost). This scheme may easily be extended to 3 bits of resolution which requires 8 thresholds to be encoded, if 1% resistors are used in the ladder. For higher resolution applications, the design may include comparators to front end the PAL device. Above 3 bits of resolution, a bipolar technology imposed limitation on this A/D technique occurs because the input structure of a TTL PAL device will both source and sink current depending on the voltage magnitude at the input. This leakage current obviously increases with the number of thresholds and has a cumulative negative effect in the resistor string. Future CMOS PAL logic circuits will be ideal for these applications however, because of the high input impedance of CMOS devices.

THE FRAME GRABBER SYSTEM OPERATION

The FRAME GRABBER system diagram is illustrated in FIGURE FIVE. Three PAL devices form the core of the system. The megaPAL device handles most of the timing, address, and control logic. The PAL16R8 handles A/D conversion and local bus interface, while the PAL20RA10 serves to interface the local bus to the host PC I/O channel bus. To complete the system four 8k byte static RAMs form the buffer memory and an octal transceiver (74LS245) provides sufficent drive for the I/O channel. The FRAME GRABBER has two modes of operation. When the MODE control bit is high, the unit is in the capture mode. Video input will continuously be digitized and stored in RAM. When the MODE bit is low, or read mode, the PC I/O channel takes over memory control. The byte of buffer memory pointed to by the address counter value can be accessed by an I/O reference to location 100 HEX.


HM6264
8Kx8
STATIC RAM
PAL64R32
PAL20RA10
PAL16R8
74LS245
PC I/O CHANNEL
CONNECTOR
VIDEO
INPUT
J2

```
REVISION  A
AUTHOR    A.G.GILBERT
COMPANY   MMI SANTA CLARA, CA.
DATE      12/17/84

CHIP      FLASH PAL16R8 DEVICE

CLK /CLR Q358 Q716 NC T3 T2 T1 T0 GND
/OE D0 D1 D2 D3 D4 D5 D6 D7 VCC
;
;              FLASH A/D MODULE
; This design specification module implements a low resolution
; Flash A/D converter. Four input thresholds (T3 thru T0) are
; synchronously encoded into two bit groupings, then "packed"
; four times to form an output byte. This packing operation is
; accomplished by two parallel four bit shift registers which
; are functionally implemented in this PAL device. One shift register
; operates on even bits while the other on odd bits (e.g.
; D0 - D2 - D4 - D6). These CONVERSION/SHIFT operations occurs
; on the rising edge of the 14.3818 MHz system clock when time
; base input clocks are in the proper state. Four periods of the
; system clock elapse between sucessive Flash conversions for an
; effective Flash rate of 3.58 MHz. If the clamped video input signal
; has sufficent magnitude to cross the upper three thresholds (white)
; the two bit encoding would be 11. If none of the four thresholds
; (Hsync) were crossed, the encoding would be 00. The first encoding
; in a sequence of four flashes will end up registered in D7 and D6.
; The last encoding will be registered in D1 and D0. A monotonically
; increasing video signal from Hsync to black to gray to white
; would be represented by 00011011 after packing. Get the idea?
; One final comment, the PAL16R8 device does not have an output
; polarity fuse, therefore, this design specification module
; is implemented in negative logic.
;
EQUATIONS

/D0     :=/CLR*/T3*/T2*/T1*/T0* Q358* Q716      ;CONVERT
         +/CLR* T3* T2*/T1*/T0* Q358* Q716      ;CONVERT
         +/CLR*/Q358*/D0                        ;HOLD
         +/CLR*/Q716*/D0                        ;HOLD
         + CLR                                  ;CLEAR

/D1     :=/CLR*/T3*/T2*/T1*/T0* Q358* Q716      ;CONVERT
         +/CLR* T3*/T2*/T1*/T0* Q358* Q716      ;CONVERT
         +/CLR*/Q358*/D1                        ;HOLD
         +/CLR*/Q716*/D1                        ;HOLD
         + CLR                                  ;CLEAR

/D2     :=/CLR* Q358*/D2                        ;HOLD
         +/CLR*/Q716*/D2                        ;HOLD
         +/CLR*/Q358* Q716*/D0                  ;SHIFT
         + CLR                                  ;CLEAR
```

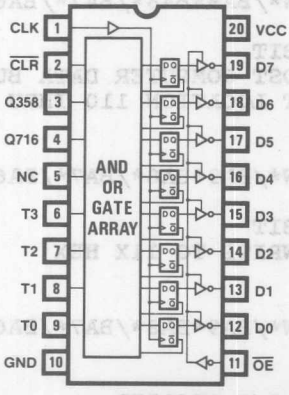
```
          +/CLR*/Q358* Q716*/D1                          ;SHIFT
          + CLR                                          ;CLEAR

/D4       :=/CLR* Q358*/D4                               ;HOLD
          +/CLR*/Q716*/D4                                ;HOLD
          +/CLR*/Q358* Q716*/D2                          ;SHIFT
          + CLR                                          ;CLEAR

/D5       :=/CLR* Q358*/D5                               ;HOLD
          +/CLR*/Q716*/D5                                ;HOLD
          +/CLR*/Q358* Q716*/D3                          ;SHIFT
          + CLR                                          ;CLEAR

/D6       :=/CLR* Q358*/D6                               ;HOLD
          +/CLR*/Q716*/D6                                ;HOLD
          +/CLR*/Q358* Q716*/D4                          ;SHIFT
          + CLR                                          ;CLEAR

/D7       :=/CLR* Q358*/D7                               ;HOLD
          +/CLR*/Q716*/D7                                ;HOLD
          +/CLR*/Q358* Q716*/D5                          ;SHIFT
          + CLR                                          ;CLEAR
;
;
; PALASM1 SOFTWARE FUNCTION TABLE DESCRIPTION
;
;                                    8  6
; CONTROL      INPUTS                5  1       OUTPUTS
;/OE CLK /CLR T3 T2 T1 T0 Q3 Q7    D7 D6 D5 D4 D3 D2 D1 D0
;
;-------------------------------------------------------
;  L   C   L  X  X  X  X  X  X    L  L  L  L  L  L  L  L    CLEAR
;  L   C   H  H  H  H  L  H  H    L  L  L  L  L  H  H      ENCODE WHITE
;  L   C   H  X  X  X  X  L  L    L  L  L  L  L  L  H  H    HOLD
;  L   C   H  X  X  X  X  L  H    L  L  L  L  H  H  H  H    SHIFT
;  L   C   H  X  X  X  X  H  L    L  L  L  L  H  H  H  H    HOLD
;  L   C   H  H  H  L  L  H  H    L  L  L  L  H  H  H  L    ENCODE GRAY
;  L   C   H  X  X  X  X  L  L    L  L  L  L  H  H  H  L    HOLD
;  L   C   H  X  X  X  X  L  H    L  L  H  H  H  L  H  L    SHIFT
;  L   C   H  X  X  X  X  H  L    L  L  H  H  H  L  H  L    HOLD
;  L   C   H  H  L  L  L  H  H    L  L  H  H  H  L  L  H    ENCODE BLACK
;  L   C   H  X  X  X  X  L  L    L  L  H  H  H  L  L  H    HOLD
;  L   C   H  X  X  X  X  L  H    H  H  H  L  L  H  L  H    SHIFT
;  L   C   H  X  X  X  X  H  L    H  H  H  L  L  H  L  H    HOLD
;  L   C   H  L  L  L  L  H  H    H  H  H  L  L  H  L  L    ENCODE HSYNC
;  L   C   H  X  X  X  X  L  L    H  H  H  L  L  H  L  L    HOLD
;  L   C   H  X  X  X  X  L  H    H  L  L  H  L  L  L  L    SHIFT
;  L   C   H  X  X  X  X  H  L    H  L  L  H  L  L  L  L    HOLD
;  H   X   X  X  X  X  X  X  X    Z  Z  Z  Z  Z  Z  Z  Z    TRISTATE
;-------------------------------------------------------
```

```
SIMULATION
;
; This "brute force" simulation specification directly
; implements the PALASM1 SOFTWARE style function table described above.
;
TRACE_ON T3 T2 T1 T0 Q358 Q716 D7 D6 D5 D4 D3 D2 D1 D0
SETF OE CLR                                          ;CLEAR
CLOCKF CLK
CHECK /D7 /D6 /D5 /D4 /D3 /D2 /D1 /D0

SETF /CLR T3 T2 T1 /T0 Q358 Q716                     ;ENCODE WHITE LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                     ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                      ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                      ;HOLD
CLOCKF CLK

SETF T3 T2 /T1 /T0 Q358 Q716                         ;ENCODE GRAY LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                     ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                      ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                      ;HOLD
CLOCKF CLK

SETF T3 /T2 /T1 /T0 Q358 Q716                        ;ENCODE BLACK LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                     ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                      ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                      ;HOLD
CLOCKF CLK

SETF /T3 /T2 /T1 /T0 Q358 Q716                       ;ENCODE HSYNC LEVEL
CLOCKF CLK
SETF /Q358 /Q716                                     ;HOLD
CLOCKF CLK
SETF /Q358 Q716                                      ;SHIFT
CLOCKF CLK
SETF Q358 /Q716                                      ;HOLD
CLOCKF CLK

SETF /OE                                             ;TRISTATE
```

```
PALH37.TRF;1                    14-FEB-1985 18:33          Page 1

Page :  1
      g  cg cg c  g cg  cg cg   cg cg cg   cg cg cg c
 T3   XXXXHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
 T2   XXXXHHHHHH HHHHHHHHHH HHHHHHHHLL LLLLLLLLLL
 T1   XXXXHHHHHH HHHHHHLLLL LLLLLLLLLL LLLLLLLLLL
 T0   XXXXLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
 Q358 XXXXHHHLLL LLLHHHHHHL LLLLLHHHHH HLLLLLLHHH
 Q716 XXXXHHHLLL HHHLLLHHHL LLHHHLLLHH HLLLHHHLLL
 D7   XXXLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLHHHH
 D6   XXXLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLHHHH
 D5   XXXLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
 D4   XXXLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHLLLL
 D3   XXXLLLLLLL LLHHHHHHHH HHHHHHHHHH HHHHHHLLLL
 D2   XXXLLLLLLL LLHHHHHHHH HHHHLLLLLL LLLLLLHHHH
 D1   XXXLLLHHHH HHHHHHHHHH HHHHHHHHHH LLLLLLLLLL
 D0   XXXLLLHHHH HHHHHHHHLL LLLLLLLLLL HHHHHHHHHH
```

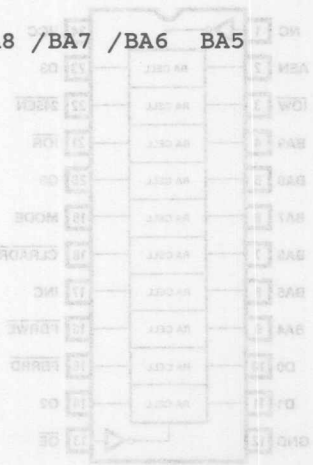


```
PALH87.TRF;1                    14-FEB-1985 18:33          Page 2

Page :  1
      g cg cg cg  cg
 T3   LLLLLLLLLL LLLL
 T2   LLLLLLLLLL LLLL
 T1   LLLLLLLLLL LLLL
 T0   LLLLLLLLLL LLLL
 Q358 HHHLLLLLLH HHHH
 Q716 HHHLLLHHHL LLLL
 D7   HHHHHHHHHH HHHZ
 D6   HHHHHHHHLL LLLZ
 D5   HHHHHHHHLL LLLZ
 D4   LLLLLLLLHH HHHZ
 D3   LLLLLLLLLL LLLZ
 D2   HHHHHHHHLL LLLZ
 D1   LLLLLLLLLL LLLZ
 D0   HHLLLLLLLL LLLZ
```



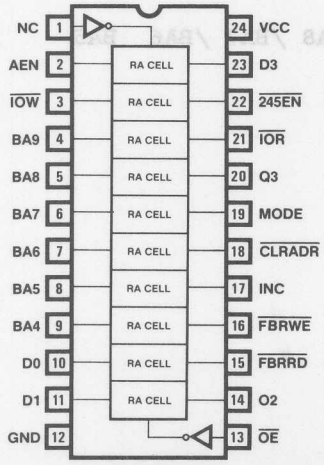
```
REVISION  A
AUTHOR    ALFIE / KELVIN
COMPANY   MMI, SANTA CLARA, CA
DATE      12/20/84

CHIP INTERFACE PAL20RA10 DEVICE

NC AEN /IOW BA9 BA8 BA7 BA6 BA5 BA4 D0 D1 GND
/OE D2 /FBRRD /FBRWE INC /CLRADR MODE Q3 /IOR /245EN D3 VCC

; This design specification module implements a nibble wide (four bit)
; control register for the FRAME GRABBER unit. The outputs of this
; module respond to the address, data, and control lines of the host
; Personal computer which must be hardware compatible with the IBM PC.
; Some of the outputs are combinatorial while others are registered for
; obvious reasons. This module is implemented in mixed logic and
; demonstrates the superior flexibity of this new PAL device.

EQUATIONS

;
;              FRAME GRABBER CONTROL REGISTER MODULE
;
; CONTROL REGISTER "INC" BIT
; OUTPUT IS DERIVED FROM HOST COMPUTER DATA BUS LINE D0
; IF AN I/O WRITE OCCURS AT LOCATION 110 THRU 11F HEX
;
INC             := D0
INC.CLKF         = /AEN*IOW*/BA9*BA8*/BA7*/BA6*/BA5*BA4

; CONTROL REGISTER "/CLRADR" BIT
; OUTPUT IS DERIVED FROM HOST COMPUTER DATA BUS LINE D1
; IF AN I/O WRITE OCCURS AT LOCATION 110 THRU 11F HEX
;
/CLRADR         := /D1
/CLRADR.CLKF     = /AEN*IOW*/BA9*BA8*/BA7*/BA6*/BA5*BA4

; CONTROL REGISTER "MODE" BIT
; OUTPUT IS DERIVED FROM HOST COMPUTER DATA BUS LINE D2
; IF AN I/O WRITE OCCURS AT LOCATION 110 THRU 11F HEX
;
MODE            := D2
MODE.CLKF        = /AEN*IOW*/BA9*BA8*/BA7*/BA6*/BA5*BA4

; UNUSED CONTROL REGISTER BIT
; ALSO RESPONDS TO AN I/O WRITE TO 11X HEX
;
Q3              := D3
Q3.CLKF          = /AEN*IOW*/BA9*BA8*/BA7*/BA6*/BA5*BA4

;
;              HOST SYSTEM I/O DECODER
;
```

```
; THIS COMBINATORIAL OUTPUT IS USED TO ENABLE THE BUS BUFFER
; 74LS245 IF THE HOST REFERENCES I/O LOCATIONS 100 THRU 11F HEX
;
245EN           = /AEN*/BA9*BA8*/BA7*/BA6*/BA5

; THIS COMBINATORIAL OUTPUT IS USED TO FILL THE FRAME BUFFER
; RAM LOCATION POINTED TO BY THE ADDRESS GENERATOR WHEN
; THE HOST COMPUTER WRITES TO I/O LOCATION 10X HEX
;
FBRWE           = /AEN*IOW*/BA9*BA8*/BA7*/BA6*/BA5*/BA4

; THIS COMBINATORIAL OUTPUT IS USED TO READ THE FRAME BUFFER
; RAM LOCATION POINTED TO BY THE ADDRESS GENERATOR WHEN
; THE HOST COMPUTER READS I/O LOCATION 10X HEX
;
FBRRD           = /AEN*IOR*/BA9*BA8*/BA7*/BA6*/BA5*/BA4


SIMULATION

TRACE_ON AEN /IOR /IOW BA9 BA8 BA7 BA6 BA5 BA4 D3 D2 D1 D0
         Q3 MODE /CLRADR INC /FBRWE /FBRRD /245EN

SETF OE /AEN /IOR IOW /BA9 BA8 /BA7 /BA6 /BA5 BA4 D3 D2 D1 D0
SETF /IOW /D0
SETF  IOW /D2                              ;CLOCK CONTROL REG
SETF /IOW /D3 /D1
SETF  IOW  D2  D0
SETF OE /AEN /IOR IOW /BA9 BA8 /BA7 /BA6 /BA5 BA4 /D3 /D2 /D1 /D0

SETF /IOR  IOW /BA4                        ;ASSERT /FBRWE

SETF  IOR /IOW /BA4                        ;ASSERT /FBRRD

SETF /IOR /IOW /BA9 BA8 /BA7 /BA6  BA5     ;ASSERT /245EN
SETF  BA6 /BA5
SETF  BA7 /BA6                             ;UNASSERT
SETF /BA8 /BA7                             ;ACTIVE ADDRESS
SETF  BA9  BA8                             ;RANGE
SETF  AEN /BA9
```



3

```
Title    :    HOST BUS INTERFACE/ Author   :    ALFIE / KELVIN
Pattern  :    PALHB6.PDS          Company  :    MMI, SANTA CLARA, CA
Revision :    A                   Date     :    12/20/84

PAL20RA10
INTERFACE
Page :  1
        g  gg gg g gggggggg
 AEN    LLLLLLLLLL LLLLLLL
/IOR    HHHHHHHHHH HLHHHHH
/IOW    LLLHLLHLLL LHHHHHH
 BA9    LLLLLLLLLL LLLLLLH
 BA8    HHHHHHHHHH HHHHHLH
 BA7    LLLLLLLLLL LLLLHLL
 BA6    LLLLLLLLLL LLLHLLL
 BA5    LLLLLLLLLL LLHLLLL
 BA4    HHHHHHHHHH LLLLLLL
 D3     HHHHHHLLLL LLLLLLL
 D2     HHHHLLLHHL LLLLLLL
 D1     HHHHHHLLLL LLLLLLL
 D0     HHHLLLLHHL LLLLLLL
 Q3     XHHHHHHHLL LLLLLLL
 MODE   XHHHLLLHHH HHHHHHH
/CLRADR XLLLLLLLHH HHHHHHH
 INC    XHHHHLLHHH HHHHHHH
/FBRWE  LHHHHHHHHH LHHHHHH
/FBRRD  LHHHHHHHHH HLHHHHH
/245EN  HLLLLLLLLL LLHHHHH
```



```
TITLE       FRAME GRABBER
PATTERN     PALHB5.PDS
REVISION    B
AUTHOR      A G GILBERT
COMPANY     MMI SANTA CLARA CA.
DATE        3/7/85

CHIP        FRAME_GRABBER  PAL64R32

INC /CLRADR MODE /PWRUP /FBRRD /FBRWE NC NC
PL1 PS1 GND CLK1 /OE1 Q716MHZ Q358MHZ Q179MHZ Q090MHZ
/FLOE /WRITE INCADR INCDLY /CS0 /CS1 /CS2 /CS3
/RAMOE /RAMWE NC NC /OE2 CLK2 VCC PS2 PL2
NC NC NC NC NC NC NC NC NC NC NC NC NC NC
NC NC PL3 PS3 GND CLK3 /OE3 A11 A10 A12 A9
A13 A8 A14 CARRY A4 A3 A5 A2 A6 A1 A7 A0 /OE4
CLK4 VCC PS4 PL4 NC NC NC NC NC NC NC NC

EQUATIONS
;
;               TIME BASE MODULE
;
;This design specification module implements a four-bit synchronous
;counter. The outputs of this counter are used internally to generate
;the proper phase of various address or data bus control signals.
;The time base counter is free running (i.e. it does not require a
;count enable signal), however, it may be reset at power up by the
;active-low assertion of the input signal /PWRUP (notice that the
;reset feature does not cost a product term, which is the more general
;case, by matching the input assertion level to desired output level).
;The system oscillator which has a frequency of 14.31818 MHz, is used
;clock the time base module.
;
Q716MHZ :=/Q716MHZ */PWRUP                                  ;TOGGLE OUTPUT
                                                ;OTHERWISE RESET  OUTPUT

Q358MHZ := Q716MHZ */Q358MHZ */PWRUP                        ;TOGGLE OUTPUT
          +/Q716MHZ * Q358MHZ */PWRUP                       ;TOGGLE OUTPUT
                                                ;OTHERWISE RESET  OUTPUT

Q179MHZ := Q716MHZ * Q358MHZ */Q179MHZ */PWRUP              ;TOGGLE OUTPUT
           +/Q716MHZ           * Q179MHZ */PWRUP            ;TOGGLE OUTPUT
           +        /Q358MHZ   * Q179MHZ */PWRUP            ;TOGGLE OUTPUT
                                                ;OTHERWISE RESET  OUTPUT

Q090MHZ := Q716MHZ * Q358MHZ * Q179MHZ */Q090MHZ */PWRUP ;TOGGLE OUTPUT
           +/Q716MHZ                   * Q090MHZ */PWRUP ;TOGGLE OUTPUT
           +        /Q358MHZ           * Q090MHZ */PWRUP ;TOGGLE OUTPUT
            +                /Q179MHZ * Q090MHZ */PWRUP ;TOGGLE OUTPUT
                                                ;OTHERWISE  RESET  OUTPUT
;
;               RAM ADDRESS GENERATOR MODULE
;
;This design specification module implements a fifteen bit synchronous
;counter. The outputs of this counter are used to address the RAM
```

```
;32K bytes. Since four 8K byte static RAMs are used to implement the
;buffer, outputs A14 and A13 are not directly connected to the memories,
;but rather are inputs to the RAM MEMORY DECODER MODULE.
;A registered look ahead carry bit (CARRY) is employed to couple
;the low and high byte of address. The address counter is enabled by
;the active high assertion of the signal INCADR. The counter may be
;reset to zero by the active low assertion of the input signal /CLRADR.
;This counter is clocked by the same 14.3 MHZ oscillator as the
;time base generator.
;
A0      :=/A0 * INCADR */CLRADR                          ;TOGGLE OUTPUT
         + A0 */INCADR */CLRADR                          ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A1      := A0 */A1 * INCADR */CLRADR                     ;TOGGLE OUTPUT
         +/A0 * A1 * INCADR */CLRADR                     ;TOGGLE OUTPUT
         +      A1 */INCADR */CLRADR                     ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A2      := A0 * A1 */A2 * INCADR */CLRADR                ;TOGGLE OUTPUT
         +/A0      * A2 * INCADR */CLRADR                ;TOGGLE OUTPUT
         +     /A1 * A2 * INCADR */CLRADR                ;TOGGLE OUTPUT
         +         A2 */INCADR */CLRADR                  ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A3      := A0 * A1 * A2 */A3 * INCADR */CLRADR           ;TOGGLE OUTPUT
         +/A0      * A3 * INCADR */CLRADR                ;TOGGLE OUTPUT
         +     /A1 * A3 * INCADR */CLRADR                ;TOGGLE OUTPUT
         +         /A2 * A3 * INCADR */CLRADR            ;TOGGLE OUTPUT
         +             A3 */INCADR */CLRADR              ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A4      := A0 * A1 * A2 * A3 */A4 * INCADR */CLRADR      ;TOGGLE OUTPUT
         +/A0               * A4 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +     /A1          * A4 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +         /A2      * A4 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +             /A3  * A4 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +                  A4 */INCADR */CLRADR         ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A5      := A0 * A1 * A2 * A3 * A4 */A5*
                               INCADR */CLRADR           ;TOGGLE OUTPUT
         +/A0               * A5 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +     /A1          * A5 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +         /A2      * A5 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +             /A3  * A5 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +             /A4  * A5 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +                  A5 */INCADR */CLRADR         ;HOLD   OUTPUT
                                                         ;OTHERWISE RESET  OUTPUT

A6      := A0 * A1 * A2 * A3 * A4 * A5 */A6*
                                 INCADR */CLRADR         ;TOGGLE OUTPUT
         +/A0               * A6 * INCADR */CLRADR       ;TOGGLE OUTPUT
         +     /A1          * A6 * INCADR */CLRADR       ;TOGGLE OUTPUT
```

```
        +          /A2      * A6 * INCADR */CLRADR      ;TOGGLE OUTPUT
        +           /A3 * A6 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +           /A4 * A6 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +           /A5 * A6 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +              A6 */INCADR */CLRADR             ;HOLD   OUTPUT
                                        ;OTHERWISE RESET  OUTPUT
A7   := A0 * A1 * A2 * A3 * A4 * A5 * A6 */A7*
                                 INCADR */CLRADR        ;TOGGLE OUTPUT
        +/A0                   * A7 * INCADR */CLRADR   ;TOGGLE OUTPUT
        +     /A1              * A7 * INCADR */CLRADR   ;TOGGLE OUTPUT
        +          /A2         * A7 * INCADR */CLRADR   ;TOGGLE OUTPUT
        +           /A3 * A7 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +           /A4 * A7 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +           /A5 * A7 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +           /A6 * A7 * INCADR */CLRADR          ;TOGGLE OUTPUT
        +              A7 */INCADR */CLRADR             ;HOLD   OUTPUT
                                        ;OTHERWISE RESET  OUTPUT
CARRY   :=/A0 * A1 * A2 * A3 * A4 * A5 * A6 * A7 *
                                 INCADR */CLRADR        ;TOGGLE CARRY
        +               CARRY    */INCADR */CLRADR     ;HOLD   CARRY
                                        ;OTHERWISE RESET  CARRY

A8      :=/A8 * CARRY * INCADR */CLRADR                 ;TOGGLE OUTPUT
        + A8 */CARRY            */CLRADR                ;HOLD   OUTPUT
        + A8          */INCADR */CLRADR                 ;HOLD   OUTPUT

A9      := A8 */A9 * CARRY * INCADR */CLRADR            ;TOGGLE OUTPUT
        +/A8 * A9 * CARRY * INCADR */CLRADR             ;TOGGLE OUTPUT
        +   A9 */CARRY          */CLRADR                ;HOLD   OUTPUT
        +   A9              */INCADR */CLRADR           ;HOLD   OUTPUT

A10     := A8 * A9 */A10 * CARRY * INCADR */CLRADR      ;TOGGLE OUTPUT
        +/A8    * A10 * CARRY * INCADR */CLRADR         ;TOGGLE OUTPUT
        +     /A9 * A10 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +         A10 */CARRY           */CLRADR        ;HOLD   OUTPUT
        +         A10             */INCADR */CLRADR     ;HOLD   OUTPUT

A11     := A8 * A9 * A10 */A11 *
                       CARRY * INCADR */CLRADR          ;TOGGLE OUTPUT
        +     /A8 * A11 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +     /A9 * A11 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +    /A10 * A11 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +         A11 */CARRY           */CLRADR        ;HOLD   OUTPUT
        +         A11             */INCADR */CLRADR     ;HOLD   OUTPUT

A12     := A8 * A9 * A10 * A11 */A12 *
                         CARRY * INCADR */CLRADR        ;TOGGLE OUTPUT
        +   /A8 * A12 * CARRY * INCADR */CLRADR         ;TOGGLE OUTPUT
        +   /A9 * A12 * CARRY * INCADR */CLRADR         ;TOGGLE OUTPUT
        +  /A10 * A12 * CARRY * INCADR */CLRADR         ;TOGGLE OUTPUT
        +  /A11 * A12 * CARRY * INCADR */CLRADR         ;TOGGLE OUTPUT
        +         A12 */CARRY           */CLRADR        ;HOLD   OUTPUT
        +         A12             */INCADR */CLRADR     ;HOLD   OUTPUT
```

```
A13     := A8 * A9 * A10 * A11 * A12 */A13 *
                          CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +    /A8 * A13 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +    /A9 * A13 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A10 * A13 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A11 * A13 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A12 * A13 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +        A13 */CARRY           */CLRADR       ;HOLD   OUTPUT
        +        A13          */INCADR */CLRADR       ;HOLD   OUTPUT

A14     := A8 * A9 * A10 * A11 * A12 * A13 */A14 *
                          CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +    /A8 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +    /A9 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A10 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A11 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A12 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +   /A13 * A14 * CARRY * INCADR */CLRADR       ;TOGGLE OUTPUT
        +        A14 */CARRY           */CLRADR       ;HOLD   OUTPUT
        +        A14          */INCADR */CLRADR       ;HOLD   OUTPUT
;
;               RAM MEMORY DECODER MODULE
;
;This design specification module implements a basic combinatorial
;2 To 4 Line Decoder. The outputs (/CS0,/CS1,/CS2,/CS3) are connected
;to the active low chip selects of the four frame buffer static RAMs.
;Since the decoder outputs are active low, it is natural to realize
;this module in negative logic. Notice that the logical sence of
;the following output variables is opposite that of their name in
;the pin list e.g. CS0 vs. /CS0 , this is key to implementing
;negative logic.
;
CS0     =/A14 */A13     ;OUTPUT ASSERTS ON BINARY COUNT OF 0 0

CS1     =/A14 * A13     ;OUTPUT ASSERTS ON BINARY COUNT OF 0 1

CS2     = A14 */A13     ;OUTPUT ASSERTS ON BINARY COUNT OF 1 0

CS3     = A14 * A13     ;OUTPUT ASSERTS ON BINARY COUNT OF 1 1
;
;               RAM MEMORY CONTROL MODULE
;
;This module generates several miscelaneous control signals which
;will be described individually. Some of these outputs are registered
;and some are combinatorial.
;
;This signal (/WRITE) is a 70 nsec negative pulse used to write the
;frame buffer RAMs when the unit is in the capture mode of operation.
;This output is derived from the time base generator and it is
;realized in negative logic.
;
WRITE   := Q716MHZ */Q358MHZ * Q179MHZ * Q090MHZ * MODE */PWRUP
```

```
;This equation implements a 2 TO 1 DATA MULTIPLEXER. The output
;signal (/RAMWE) may be thought of as the write enable line for the
;static RAMs. This output is derived indirectly from either the
;time base generator (/WRI  ) or the host personal computer control
;bus (/FBRWE). This allows the PC to fill the FRAME GRABBER by doing
;an I/O write.
;
/RAMWE  =/WRITE * MODE
        +/FBRWE */MODE

;This equation also implements a 2-TO-1 MUX, again controlled by the
;mode signal. This signal is used to enable the address counter so it
;can increment (remember that the address counter is clocked by the
;14.3 MHz system clock). The output is derived from the time base
;generator if the FRAME GRABBER is in the capture mode, or from the
;rising edge of the input signal INC if the unit is in the read mode.
;INCADR is positive polarity pulse with a duration equal to one system
;clock period which is 70 nsec.
;
INCADR  :=/Q716MHZ * Q358MHZ * Q179MHZ * Q090MHZ * MODE */PWRUP
         +/INCDLY * INC */MODE */PWRUP

;This bit of logic is used to synchronize the input signal INC with
;the 14.3 MHZ system clock. Notice that the condition of INCDLY low
;and INC high realizes a synchronous rising edge detector for the
;input signal INC (refer to the second product in the logic equation
;for INCADR).
;
INCDLY  := INC

;
;               LOCAL BUS CONTROL MODULE
;
;This design specification module generates signals which enable the
;three-state outputs of the Flash A/D and the static RAMs. The Flash
;output enable (/FLOE) is derived from the time base generator and is
;qualified by the unit being in the capture mode. The memory output
;enable (/RAMOE) is derived from the host computer control bus if
;the frame buffer is in the read mode.
;
/RAMOE  =/FBRRD */MODE + MODE

FLOE    :=/Q716MHZ * Q358MHZ * Q179MHZ * Q090MHZ * MODE */PWRUP
         + Q716MHZ */Q358MHZ * Q179MHZ * Q090MHZ * MODE */PWRUP

;This design specification was assembled on an PC compatible
;computer using the beta release of PALASM2 SOFTWARE. The simulation
;was done on a VAX using the alpha release of the simulator. Slight
;variations in syntax may occur at production release. The
;FRAME GRABBER design has not been optimized in the hopes of
;being self explanitory!   ...Alfie Gilbert
```

```
SIMULATION

TRACE_ON    CLK1 CLK2 CLK3 CLK4                    ;CLOCK SIGNALS
           /PWRUP MODE FLOE WRITE INCADR           ;CONTROL SIGNALS
            Q716MHZ Q358MHZ Q179MHZ Q090MHZ        ;TIME BASE SIGNALS
            A0 A1 A2 A3 A4 A5 A6 A7 A8 A9          ;ADDRESS SIGNALS
            A10 A11 A12

SETF OE1 OE2 OE3 OE4                               ;ENABLE OUTPUTS
     PS1 PS2 PS3 PS4                               ;UNASSERT SET
     PL1 PL2 PL3 PL4                               ;UNASSERT PRELOAD

SETF PWRUP MODE                                    ;CLEAR TIME BASE
CLOCKF CLK1 CLK3 CLK4

SETF /PWRUP                                        ;EXERCISE TIME BASE
FOR I:=1 TO 17 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
 END

SETF CLRADR                                        ;CLEAR ADDRESS COUNTER
CLOCKF CLK1 CLK3 CLK4

SETF /CLRADR                                       ;TOGGLE A1 and A0
FOR I:=1 TO 32 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
 END

SETF  A0  A1 /A2 /A3 /A4
     /A5 /A6 /A7 /A8 /A9
     /A10 /A11 /A12 /A13 /A14                      ;TOGGLE A2

FOR I:=1 TO 187 DO
 BEGIN
  CLOCKF CLK1 CLK3 CLK4
  IF I=17   THEN BEGIN SETF A2  END                ;TOGGLE A3
  IF I=17   THEN BEGIN SETF A3  END                ;TOGGLE A4
  IF I=17   THEN BEGIN SETF A4  END                ;TOGGLE A5
  IF I=17   THEN BEGIN SETF A5  END                ;TOGGLE A6
  IF I=17   THEN BEGIN SETF A6  END                ;TOGGLE A7
  IF I=17   THEN BEGIN SETF A7  END                ;TOGGLE A8
  IF I=17   THEN BEGIN SETF A8  END                ;TOGGLE A9
  IF I=17   THEN BEGIN SETF A9  END                ;TOGGLE A10
  IF I=17   THEN BEGIN SETF A10 END                ;TOGGLE A11
  IF I=17   THEN BEGIN SETF A11 END                ;TOGGLE A12
 END

SETF /A14 /A13                                     ;EXERCISE RAM DECODER
SETF /A14  A13
SETF  A14 /A13
SETF  A14  A13
```

Title : FRAME GRABBER Author : A.G.GILBERT
Pattern : PALH85.PDS Company : MMI SANTA CLARA C
Revision : A Date : 1/15/85

PAL64R32
FRAME_GRABBER
Page : 1

	gg cg c c c c c c c c c c c c cg
CLK1	XXHLLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	XXLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	XXHLLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLL
CLK4	XXHLLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLL
PWRUP	XHHHLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	XXXLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLHHHLLL
WRITE	XXXLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLHLLL
INCADR	XXXLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLHLLL
Q716MHZ	XXXHHHLLHH LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL
Q358MHZ	XXXHHHLLLL HHHLLLHHHL HHLLLLHHHH LLHHLLLLHH
Q179MHZ	XXXHHHLLLL LLLLHHHHHH HHLLLLLLLL HHHHHHHHLL
Q090MHZ	XXXHHHLLLL LLLLLLLLLL LLHHHHHHHH HHHHHHHHLL
ZD	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A1	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A2	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A3	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A4	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A5	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A6	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A7	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A8	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A9	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A10	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A11	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX
A12	XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX XXXXXXXXXX

PAL64R32
FRAME_GRABBER
Page : 2

	cg c c c c c c c c c c c c c c
CLK1	HLLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	HLLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	HLLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLHH HHLLLLLLLL
WRITE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	LLLLLLLLHH HHHHHHHHLL LLLLHHHHHH HHLLLLLLLL
Q090MHZ	LLLLLLLLLL LLLLLLHHHH HHHHHHHHHH HHLLLLLLLL
A0	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A1	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A2	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A3	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A4	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A5	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A6	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A7	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A8	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A9	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A10	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A11	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL
A12	XHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 3

	c c c c c c c c cgc c c c c c
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLLLLLLLLL HHHHLLLLLL LLLLLLLLLL
WRITE	LLLLLLLLLL LLLLLLLLLL HHLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLLLLLLL LLHHLLLLLL LLLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLLLHH HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	HHHHHHHHLL LLLLLLHHHH HHHHLLLLLL LLHHHHHHHH
Q090MHZ	LLLLLLLLHH HHHHHHHHHH HHHHLLLLLL LLLLLLLLLL
A0	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A1	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A2	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A4	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A5	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A6	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A7	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A8	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A9	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A10	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A11	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A12	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH

PAL64R32
FRAME_GRABBER
Page : 4

	c c c c c c c c cgc c c c c c
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLHHHHLLLL LLLLLLLLLL LLLLLLLLLL
WRITE	LLLLLLLLLL LLHHLLLLLL LLLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLHHLLLL LLLLLLLLLL LLLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	HHHHHHHHLL LLLLLLHHHH HHHHLLLLLL LLHHHHHHHH
Q090MHZ	HHHHHHHHHH HHHHHHLLLL LLLLLLLLLL LLHHHHHHHH
A0	HHHHHHHHHH HHHHHHLLLL LLLLLLLLLL LLLLLLLLLL
A1	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHHHHHHHHH
A2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A4	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A5	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A6	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A7	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A8	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A9	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A10	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A11	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A12	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 5

	c c c c c c c cgc c c c c c c
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLHH HHLLLLLLLL
WRITE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	HHHHHHHHLL LLLLLLHHHH HHHHLLLLLL LLHHHHHHHH
Q090MHZ	LLLLLLLLHH HHHHHHHHHH HHHHLLLLLL LLLLLLLLLL
A0	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLHHHHHHHH
A1	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHHHHHHHHH
A2	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A4	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A5	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A6	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A7	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A8	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A9	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A10	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A11	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A12	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 6

	c c c c cg c c c c c c c c c c
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLHH HHLLLLLLLL
WRITE	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	LLLLLLLLHH HHHHHHHHLL LLLLHHHHHH HHLLLLLLLL
Q090MHZ	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLHHHHHHHH
A0	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLHHHHHHHH
A1	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL HHHHHHHHHH
A2	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A4	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A5	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A6	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A7	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A8	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A9	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A10	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A11	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A12	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 7

	c cgc c c c c c c c c c cgc
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLLLLLLLLL HHHHLLLLLL LLLLLLLLLL
WRITE	LLLLLLLLLL LLLLLLLLLL HHLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLLLLLLL LLHHLLLLLL LLLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLLLHH HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	HHHHHHHHLL LLLLLLHHHH HHHHLLLLLL LLHHHHHHHH
Q090MHZ	LLLLLLLLHH HHHHHHHHHH HHHHLLLLLL LLLLLLLLLL
A0	HHHHHHHHHH HHHHHHHHHH HHHHLLLLLL LLLLLLLLLL
A1	LLLLLLLLLL LLLLLLLLLL LLLLHHHHHH HHHHHHHHHH
A2	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A4	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A5	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A6	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A7	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A8	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A9	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A10	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A11	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A12	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 8

	c c c c c c c c c c cgc c c c
CLK1	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK2	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
CLK3	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
CLK4	LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH LHLHLHLHLH
PWRUP	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
MODE	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
FLOE	LLLLLLLLLL LLHHHHLLLL LLLLLLLLLL LLLLLLLLLL
WRITE	LLLLLLLLLL LLHHLLLLLL LLLLLLLLLL LLLLLLLLLL
INCADR	LLLLLLLLLL LLLLHHLLLL LLLLLLLLLL LLLLLLLLLL
Q716MHZ	LLHHLLHHLL HHLLHHLLHH LLHHLLHHLL HHLLHHLLHH
Q358MHZ	LLLLHHHHLL LLHHHHLLLL HHHHLLLLHH HHLLLLHHHH
Q179MHZ	HHHHHHHHLL LLLLLLHHHH HHHHLLLLLL LLHHHHHHHH
Q090MHZ	HHHHHHHHHH HHHHHHLLLL LLLLLLLLLL LLHHHHHHHH
A0	LLLLLLLLLL LLLLLLHHHH HHHHHHHHHH HHHHHHHHHH
A1	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
A2	HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH HHHHHHHHHH
A3	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A4	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A5	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A6	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A7	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A8	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A9	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A10	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A11	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL
A12	LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL LLLLLLLLLL

PAL64R32
FRAME_GRABBER
Page : 10

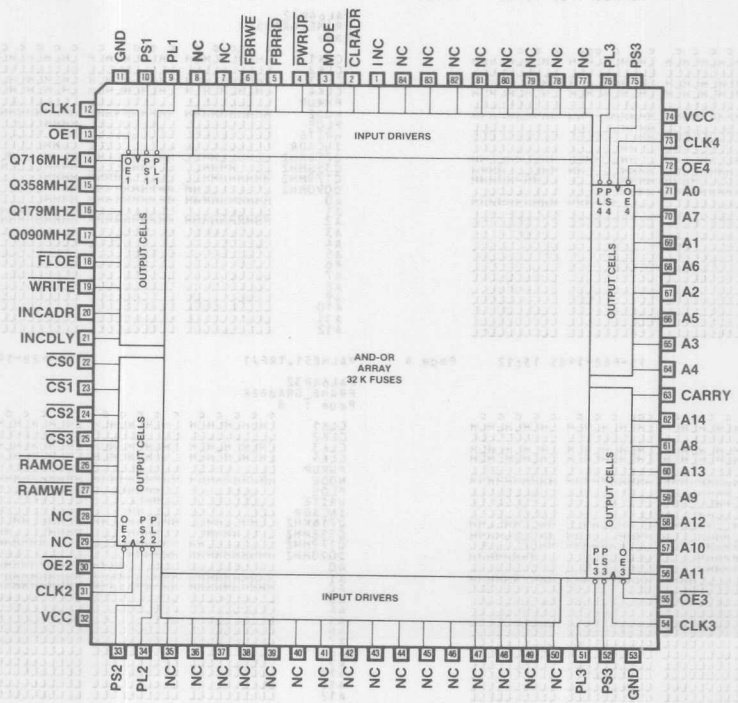
	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
CLK1	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK2	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
CLK3	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK4	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
PWRUP	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
MODE	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
FLOE	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
WRITE	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
INCADR	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
Q716MHZ	L	H	H	L	L	H	H	L	L	H	H	L	H	L	H	H
Q358MHZ	L	L	L	H	H	H	H	L	L	L	L	H	H	L	L	H
Q179MHZ	L	L	L	L	L	L	L	H	H	H	H	H	H	L	L	H
Q090MHZ	L	L	L	L	L	L	L	L	L	L	L	L	H	H	H	H
A0	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A1	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A2	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A3	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A4	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A5	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A6	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A7	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A8	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A9	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A10	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A11	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A12	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

PAL64R32
FRAME_GRABBER
Page : 11

	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
CLK1	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK2	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
CLK3	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK4	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
PWRUP	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
MODE	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
FLOE	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
WRITE	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
INCADR	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
Q716MHZ	L	H	H	L	L	H	H	L	L	H	H	L	H	L	H	H
Q358MHZ	L	L	L	H	H	H	H	L	L	L	L	H	H	L	L	H
Q179MHZ	L	L	L	L	L	L	L	H	H	H	H	H	H	L	L	H
Q090MHZ	L	L	L	L	L	L	L	L	L	L	L	L	H	H	H	H
A0	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A1	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A2	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A3	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A4	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A5	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A6	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A7	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A8	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A9	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A10	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A11	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A12	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L

PAL64R32
FRAME_GRABBER
Page : 12

	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
CLK1	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK2	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
CLK3	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
CLK4	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
PWRUP	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
MODE	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
FLOE	L	L	L	L	L	L	L	L	L	L	H	H	L	L	L	L
WRITE	L	L	L	L	L	L	L	L	L	L	H	H	L	L	L	L
INCADR	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
Q716MHZ	L	H	H	L	L	H	H	L	L	H	H	L	H	L	H	H
Q358MHZ	L	L	L	H	H	H	H	L	L	L	L	H	H	L	L	H
Q179MHZ	L	L	L	L	L	L	L	H	H	H	H	H	H	L	L	H
Q090MHZ	L	L	L	L	L	L	L	L	L	L	L	L	H	H	H	H
A0	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A1	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A2	L	L	L	L	L	L	L	L	L	L	L	L	H	H	L	L
A3	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
A4	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A5	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A6	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A7	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A8	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A9	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A10	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A11	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
A12	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L



FOOD FOR THOUGHT

The FRAME GRABBER which we just synthesized in our design exersise is actually half of a video Frame Buffer unit. Video frame buffers have become quite popular in recent years. The key elements of a video frame buffer are illustrated in FIGURE SIX. The concept behind a frame buffer is simple . A frame buffer stores the incoming video information in a RAM array for future use,and the digitized image has been stored in a memory array, it is often processed by digital signal processing techniques. These techniques may be hardware or software based. Digital signal processing implemented in hardware tend to be very fast (even real time), but expensive, while software signal processing algorithms are slower, but more cost effective in most applications. The most basic type of digital signal processing usually performed on video is image enhancement. Video signals which have been corrupted by noise are likely candidates for image enhancement. In many types of applications image enhancement is often followed by a more complex type of signal processing, known as pattern recognition. This type of signal processing is usually a software algorithm which does not execute in real time and for that reason digital image frames must be buffered in memory.

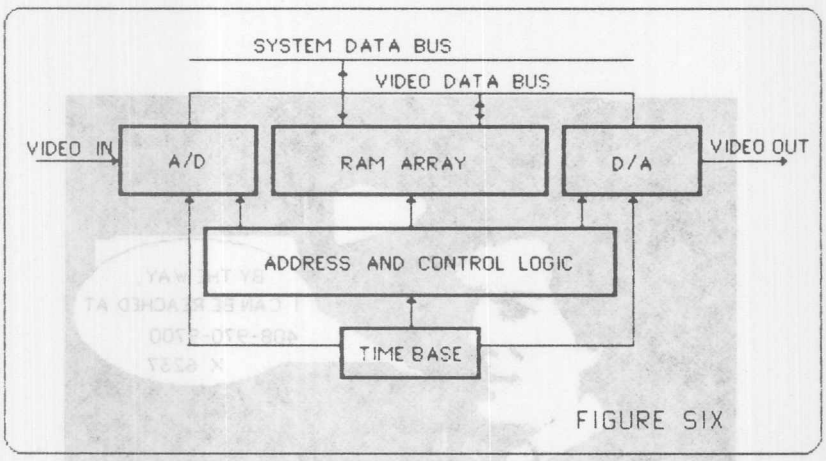


FIGURE SIX

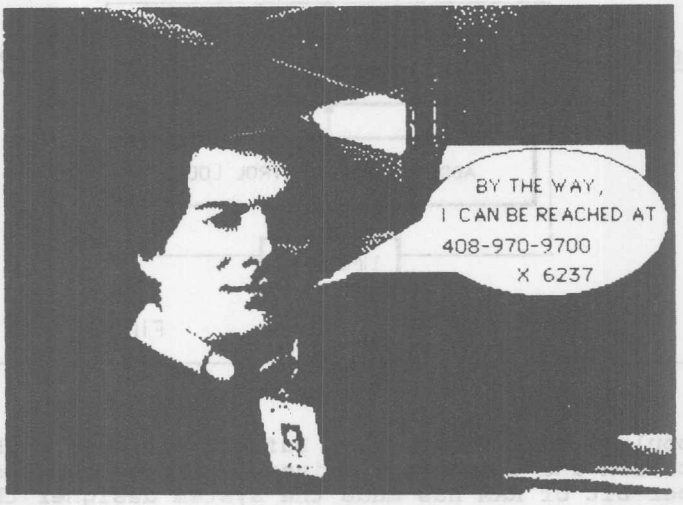
The popularity of Video Frame Buffers has been both application and technology driven. The ever decreasing cost-per-bit of RAM has made the system designer the size of memory arrays required to store video images of acceptable resolution and gray scale content affordable ASIC technology such as the megaPAL device which we just investigated, has streamlined the address, control, and timebase logic of video frame buffers as well. These technological advances have made many applications economically feasible. In the industrial sector these applications include robot control, collision

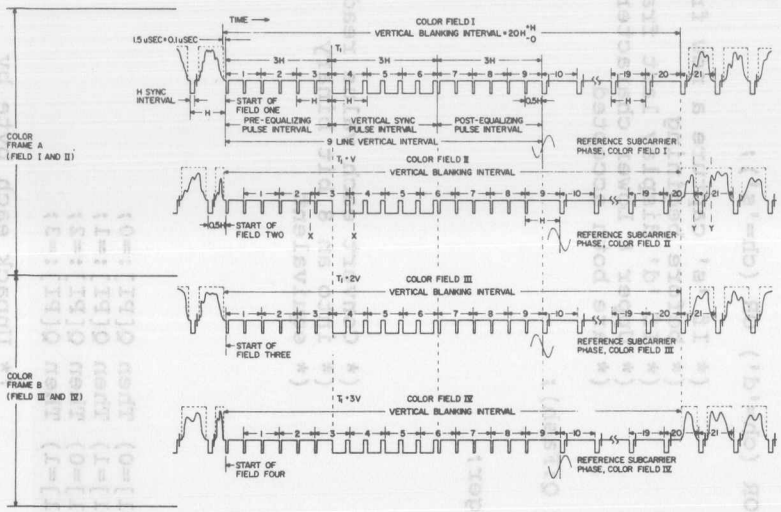
avoidance, quality control, quality assurance, incoming inspection, surveillance/security systems, and a host of others. If you are starting to get excited about the prospect of innovating something on your own, you are probably pondering what the future will be like. I think we will see vision applications abound in personal computer environments. The combination of an optical imager coupled to a personal computer is the silicon parity of a very familiar organic computer, namely, our eyes and brain. It is only a matter of time until personal computers can read and recognize for us. So my advice to designers is simple : Imagineer before you Engineer! When you get to the point of implementing your ideas, think about PAL devices, they are remarkable axels for the wheels of your mind. I hope you enjoyed reading this design exercise as much as I enjoyed creating it.

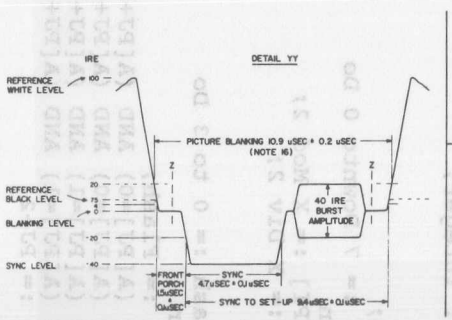
... Alfie Gilbert

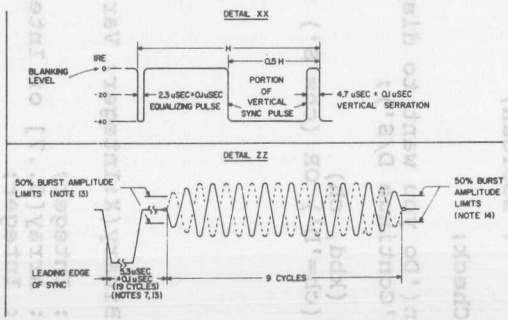
Please feel free to send comments or sugestions to

A. G. Gilbert
MMI 09-26
2175 Mission College Blvd.
Santa Clara, Ca. 95054-1592









NOTES

1. SPECIFICATIONS APPLY TO STUDIO FACILITIES. COMMON CARRIER, STUDIO TO TRANSMITTER AND TRANSMITTER CHARACTERISTICS ARE NOT INCLUDED.
2. ALL TOLERANCES AND LIMITS SHOWN IN THIS DRAWING PERMISSIBLE ONLY FOR LONG TIME VARIATIONS.
3. THE BURST FREQUENCY SHALL BE 3.579545 MHz ±10 Hz.
4. THE HORIZONTAL SCANNING FREQUENCY SHALL BE 2/455 TIMES THE BURST FREQUENCY [ONE SCAN PERIOD (H) = 63.556 μSEC].
5. THE VERTICAL SCANNING FREQUENCY SHALL BE 2/525 TIMES THE HORIZONTAL SCANNING FREQUENCY [ONE SCAN PERIOD (V) = 16,683 μSEC].
6. START OF COLOR FIELDS I AND III IS DEFINED BY A WHOLE LINE BETWEEN THE FIRST EQUALIZING PULSE AND THE PRECEDING H SYNC PULSE. START OF COLOR FIELDS II AND IV IS DEFINED BY A HALF LINE BETWEEN THE FIRST EQUALIZING PULSE AND THE PRECEDING H SYNC PULSE. COLOR FIELD I: THAT FIELD WITH POSITIVE GOING ZERO-CROSSINGS OF REFERENCE SUBCARRIER MOST NEARLY COINCIDENT WITH THE 50% AMPLITUDE POINT OF THE LEADING EDGES OF EVEN NUMBERED HORIZONTAL SYNC PULSES. REFERENCE SUBCARRIER IS A CONTINUOUS SIGNAL WITH THE SAME INSTANTANEOUS PHASE AS BURST.
7. THE ZERO-CROSSINGS OF REFERENCE SUBCARRIER SHALL BE NOMINALLY COINCIDENT WITH THE 50% POINT OF THE LEADING EDGES OF ALL HORIZONTAL SYNC PULSES, FOR THOSE CASES WHERE THE RELATIONSHIP BETWEEN SYNC AND SUBCARRIER IS CRITICAL FOR PROGRAM INTEGRATION, THE TOLERANCE ON THIS COINCIDENCE IS ±40° OF REFERENCE SUBCARRIER.
8. ALL RISE TIMES AND FALL TIMES UNLESS OTHERWISE SPECIFIED ARE TO BE 0.14 μSEC ±0.02 μSEC MEASURED FROM 10% TO 90% AMPLITUDE POINTS. ALL PULSE WIDTHS ARE MEASURED AT 50% AMPLITUDE POINTS, UNLESS OTHERWISE SPECIFIED.
9. TOLERANCE ON SYNC LEVEL, REFERENCE BLACK LEVEL (SET-UP) AND PEAK TO PEAK BURST AMPLITUDE SHALL BE ±2 IRE UNITS.
10. THE INTERVAL BEGINNING WITH LINE 17 AND EXTENDING THROUGH LINE 20 OF EACH FIELD MAY BE USED FOR TEST, CUE AND CONTROL SIGNALS.
11. EXTRANEOUS SYNCHRONOUS SIGNALS DURING BLANKING INTERVALS, INCLUDING RESIDUAL SUBCARRIER, SHALL NOT EXCEED 1 IRE UNIT. EXTRANEOUS NON-SYNCHRONOUS SIGNALS DURING BLANKING INTERVALS SHALL NOT EXCEED 0.5 IRE UNIT. ALL SPECIAL PURPOSE SIGNALS (VITS, VIR, ETC.) WHEN ADDED TO THE VERTICAL BLANKING INTERVAL ARE EXCEPTED. OVERSHOOT ON ALL PULSES DURING SYNC AND BLANKING, VERTICAL AND HORIZONTAL, SHALL NOT EXCEED 2 IRE UNITS.
12. BURST ENVELOPE RISE TIME IS 0.3 μSEC +0.2 μSEC -0.1 μSEC MEASURED BETWEEN THE 10% AND 90% AMPLITUDE POINTS. BURST IS NOT PRESENT DURING THE NINE LINE VERTICAL INTERVAL.
13. THE START OF BURST IS DEFINED BY THE ZERO-CROSSING (POSITIVE OR NEGATIVE SLOPE) THAT PRECEDES THE FIRST HALF CYCLE OF SUBCARRIER THAT IS 50% OR GREATER OF THE BURST AMPLITUDE. ITS POSITION IS NOMINALLY 19 CYCLES OF SUBCARRIER FROM THE 50% AMPLITUDE POINT OF LEADING EDGE OF SYNC. (SEE DETAIL ZZ).
14. THE END OF BURST IS DEFINED BY THE ZERO-CROSSING (POSITIVE OR NEGATIVE SLOPE) THAT FOLLOWS THE LAST HALF CYCLE OF SUBCARRIER THAT IS 50% OR GREATER OF THE BURST AMPLITUDE.
15. MONOCHROME SIGNALS SHALL BE IN ACCORDANCE WITH THIS DRAWING EXCEPT THAT BURST IS OMITTED, AND FIELDS III AND IV ARE IDENTICAL TO FIELDS I AND II RESPECTIVELY.
16. OCCASIONALLY, MEASUREMENT OF PICTURE BLANKING AT 20 IRE UNITS IS NOT POSSIBLE BECAUSE OF SCENE CONTENT AS VERIFIED ON A PICTURE MONITOR.

COLOR TELEVISION STUDIO
PICTURE LINE AMPLIFIER OUTPUT
RS 170 A
EIA TENTATIVE STANDARD

DRAWN: JWPG — NOVEMBER 8, 1977

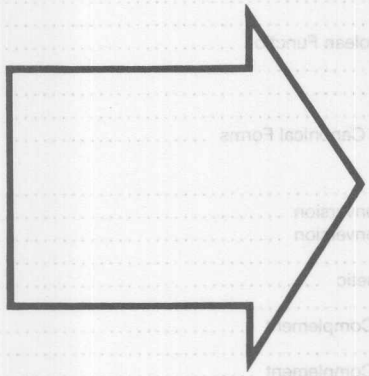
```
                                        (* Date: 3/7/85          *)
Type
  L = Array [0..239] of Integer;
  Fash = Array [0..3] of Integer;       (* Define Fash as a 4    *)
                                        (* flash encodings       *)
VAR
  X1,X2,Y1,Y2,
  X,Y,W,
  AB,C,H,I,J,K,M,N,P,Num,
  Hsync,NO,
  remain,
  P1,P2,
  Byte,
  color         : INTEGER;
  Ch            : Char;
  Line          : L;
  Q             : Fash;
  even,odd      : Boolean;

Procedure Check;                        (* Check to start the program *)
  Begin
    Writeln('Do you want to display a picture or snap one?');
    Write('Continue D/S');
    Repeat
      Read (kbd, ch)
    Until (ch='D') OR (ch='S') OR (ch='d') OR (ch='s');
  End;
                                        (* If 's' capture a new frame *)
                                        (* before begining            *)
                                        (* If 'd' display last frame  *)
                                        (* Upper & Lower characters   *)
                                        (* are both accepted          *)

Procedure Binary(X:Integer; Var Q:Fash);
  Var
    PI,PJ : Integer;
    A     : Array[0..7] of Integer;
    Flash : Integer;

  Begin
    PJ :=0;
    For PI := 7 Downto 0 Do             (* Convert each value read  *)
      Begin                             (* into an 8 bit binary     *)
        A[PI] := X Mod 2;               (* equivalent               *)
        X := X Div 2;
      End;

    For Flash := 0 to 3 Do
      Begin
        PI := Flash;
        If (A[PJ]=0) AND (A[PJ+1]=0) Then Q[PI]:=0;
        If (A[PJ]=0) AND (A[PJ+1]=1) Then Q[PI]:=1;
        If (A[PJ]=1) AND (A[PJ+1]=0) Then Q[PI]:=2;
        If (A[PJ]=1) AND (A[PJ+1]=1) Then Q[PI]:=3;
        PJ := PJ+2;
      End;                              (* Unpack each byte by      *)
```

```
  End;                                    (* converting the 8-bit    *)
                                          (* value into 4 flash values*)

Begin
  Check;                                  (* Run the check procedure  *)
  GraphMode;                              (* Graphics mode            *)
  Palette(1);
  color := 1;                             (* Use black color graphics *)
  C := 0;                                 (* Set counter to zero      *)
  I := 0;                                 (* Set all initial condition*)
  J := 1;
  N := 0;
  Hsync := 0;
  NO :=0;
  AB := 0;

  If (ch='S') OR (ch='s') Then            (* If snapping a new picture *)
    Begin                                 (* delay about 1/30 sec      *)
      Port[272] := 4;                     (* to be able to capture the *)
      Port[272] := 6;                     (* video frame after reseting*)
      For I := 1 To 11000 Do              (* the address counter and   *)
        AB := AB+1;                       (* setting the mode bit for  *)
    End;                                  (* "CAPTURE" mode            *)

  Port[272] := 4;                         (* Reset address counter     *)


  While (C<>20) AND (NO<>22000) Do
    Begin                                 (* Reset the mode bit for    *)
      Port[272] := 2;                     (* "READ" mode and increment *)
      Port[272] := 3;                     (* the address counter       *)
      X := Port[256];
      NO := NO+1;
      If X=0 Then C := C+1
      Else C := 0;
    End;

  If (NO=20000) AND (C<>20) Then
    Write('You need to adjust your light!!');

  For I := 1 To 2000 Do                   (* Disgard the 1st 2000 bytes*)
    Begin                                 (* read from the port        *)
      Port[272] := 2;
      Port[272] := 3;
      X := Port[256];
    End;

  For Y := 0 To 199 Do                    (* Start the main program    *)
    Begin                                 (* Display 200 lines of      *)
      P :=0;                              (* horizontal flashes        *)
      Hsync := 0;
      W := 255;                           (* Set the initial flashes   *)
      X1 := 0;                            (* to all '11'               *)
      H := 0;
      Even := False;
      Odd := False;
      remain := Y Mod 2;
```

```
Case remain Of
  0: Even := True;
  1: Odd  := True;
End;

While H<>1 Do                    (* Look for horizontal sync  *)
  Begin
    Port[272] := 2;
    Port[272] := 3;
    X := Port[256];
    If X=0 Then                  (* Look for the 1st byte of  *)
      Begin                      (* all 0's. X is new value   *)
        Binary(W,Q);             (* read from the port and W  *)
        For I := 0 To 3 Do       (* is te previous value. If  *)
          Line[I] := Q[I];       (* X=0 then assign W as the  *)
                                 (* first byte of the line    *)
        Binary(W,Q);             (* and X as the second byte  *)
        For I := 0 To 3 Do
          Line[I+4] := Q[I];
        H := 1;
      End
    Else                         (* Else assign X to previous *)
      W := X;                    (* value W                   *)
  End;

K := 7;

For I := 1 To 55 Do              (* At this point the first   *)
  Begin                          (* two bytes of horizontal   *)
    Port[272] := 2;              (* line are stored. Read the *)
    Port[272] := 3;              (* next 55 bytes             *)
    X := Port[256];
    Binary(X,Q);
    For J := 0 To 3 Do
      Begin
        K := K+1;
        Line[K] := Q[J];
      End;
  End;

While Hsync<>12 Do               (* Look for the first        *)
  Begin                          (* horizontal sync. Count 12 *)
    If Line[P]=0 Then            (* bytes of zeros            *)
      Hsync := Hsync+1
    Else
      Hsync := 0;
    P := P+1;
  End;

P1 := P+29;                      (* Disgard the next 29 bytes *)
P2 := P1+160;                    (* Display the next 160 bytes*)
                                 (* of data                   *)

For I := P1 To P2 Do             (* Display the flashes on    *)
  Begin                          (* line Y                    *)
    Y1 := Y;
```

```
        Y2 := Y;
        X2 := X1+2;
        If Line[I]=2 Then
          Begin
            If even Then           (* Place -X for even lines
              Plot((X1+1),Y1,color)
            Else
              Plot(X1,Y1,color);   (* Place X- for odd lines
          End;

        If Line[I]=3 Then Draw(X1,Y1,X2,Y2,color);
             X1 := X1+2;
      End;
  End;
gotoxy(1,25);
Write('Press ESC to stop.');
Read(Kbd, Ch);
If Ch=#27 Then
TextMode;
End.
```


IMPORTANT
NOTES



Contents Section 4

1.1 The Language of Logic

Although you may not be aware of it, you are already an expert at forming, simplifying and comprehending Boolean equations and expressions. Boolean algebra, in its most common application, is concerned with the truth or falsity of statements; and any time you describe what circumstances would make something true or false, you have made a Boolean equation.

For example, suppose A is true only if B and C are true. These three letters may represent anything you like — A may be whether or not you may become president, B may be whether or not you are elected, and C may be whether or not you are a citizen of the U.S.A. You may become president only if you are elected and you are a citizen of the United States. If we wrote that statement in equation form, it might look like this:

A = B*C

where the * is a shorthand notation for the word 'and.' A, B and C are all Boolean variables, since they represent some value which may be either true or false. You either are a citizen of the United States, or you are not — there is no in between. Examining the relationship between these three variables, we find that:

1) if you are elected and you are citizen then you may become president;
2) if you are elected but you are not a citizen then you cannot become president;
3) if you are not elected, but you are a citizen, you still can't become president, and;
4) if you are neither elected nor a citizen, then you definately cannot become president.

This same relationship, which may be expressed in terms of an English sentence of a Boolean expression may also be represented by a table of all the possibilities, called a truth table. If we let '1' stand for true, and '0' stand for false, we can make the following table:

B	C	A
0	0	0
0	1	0
1	0	0
1	1	1

Table 1-1

The table above is a standard way of expressing logical relationships. Our truth table lists the possibilities one-by-one. If B and C are false, then A will be false, If B is true and C is false, then A will still be false. If B is false, and C is true then A will still be false. However, if B and C are both true, then A will be true.

1.2 AND, OR and NOT

The fact is that every time you have an equation of the form:

A = B*C

you will have a truth table of the form in section 1.2 because the table and the word 'and' are just two ways of expressing the same relationship between two Boolean variables.

true. This equation could be written:

A = B+C

Do not confuse the '+' with the addition sign of arithmetic; in Boolean algebra, it is shorthand notation for the word 'or'. A truth table for this equation would be:

B	C	A
0	0	0
0	1	1
1	0	1
1	1	1

This table expresses a different relationship between the variables than AND does; AND requires that both of its operands be true for the expression to be true. OR only requires that one of its operands be true for the expression to be true. From the table above, we can see that:

1) if both B and C are false, then A is false:
2) if B is false, and C is true, then A is true:
3) if B is true and C is false, then A is true and:
4) if both B and C are true, then A is true.

Finally, let's look at the operator 'not'. If A equals not B, then the value of A is the inverse of B. This equation would be:

A = /B

Again, the '/' should not be mistaken for the division sign of arithmetic. It is a shorthand notation for the Boolean operator, 'not'. The truth table for this equation would be:

B	A
0	1
1	0

which is to say that:

1) if B is false, then A is true and:
2) if B is true then A is false.

1.3 Precedence

In arithmetic, the multiplication sign is always evaluated before the addition sign. For example:

3+4x7

is 31, not 49. Similarly, the AND sign is always evaluated before the OR sign. Another way to say this is that AND has a higher precedence than OR.

Of course, in arithmetic, the precedence of operators may be changed with parentheses. If you wish the expression:

3+4x7

to be evaluated as 49, then you should write it as:

(3+4)x7

The parentheses enclose a subexpression that should be evaluated before the expression as a whole can be evaluated.

Of the three Boolean operators we have seen so far, NOT has the highest precedence, then AND, then OR.

1.4 Associativity and Commutativity

Both the AND and OR operators have the property of associativity (in fact, all Boolean operators have this property, except for NOT). The property of associativity says that in an expression with more than one operator of the same kind, it does not matter which you evaluate first. In terms of equations, this would be:

$$B*(C*D) = (B*C)*D$$

or

$$B+(C+D) = (B+C)+D$$

All Boolean operators (except for NOT) are also commutative. This means that the order in which the operands appear is not important. In equations, that would be

$$B*C = C*B$$

or

$$B+C = C+B$$

1.5 Postulates and Theorems

In 1854, the mathematician and philosopher George Boole published his book, 'An Investigation of The Laws of Thought', in which he demonstrated how classical logic could be defined with algebraic terminology and operations. Then, in 1938, C. E. Shannon published his paper "A Symbolic Analysis of Relay and Switching Circuits", which demonstrated a Boolean algebra of two values called "switching algebra", which could be used to represent the properties of bistable electric switching circuits. A minimal set of formal postulates is needed in order to define this Boolean algebra. Here we will define Boolean algebra to be an algebra defined over the set B, where B = (False, True) and over the operators AND (*), OR (+) and NOT (/), such that:

1) All operators are closed (which means that it is impossible to create a Boolean expression that has a value other than True or False),
2) Postulates 1 through 4 in Table 1-6 are true, and
3) NOT is an operator which, when applied to a Boolean variable, x, creates its complement such that, if x = True then /x = False, and if x = False then /x = True.

Given this basic set of rules, it is possible to derive any of the theorems in Table 1-6, For example:

Theorem 1a) x+x = x

x+x	= (x+x) = True	by Postulate 1b
	= (x+x)(x+/x)	by Postulate 2a
	= x+(x*/x)	by Postulate 4b
	= x+False	by Postulate 2b
	= x	by Postulate 1a

1.5.1 Duality

One of the most important properties of Boolean algebra is the *duality principle*. This principle states that any algebraic expression that may be deduced from the postulates of Boolean algebra has a dual which is also true. The dual of an expression is obtained by replacing all Trues with Falses, all Falses with Trues, all ANDs with ORs, and all ORs with ANDs. For example:

Theorem 2a x+True = True

has the dual:

x*False = False

which is also theorem 2b. All postulates and theorems listed in Table 1-6 are listed as pairs of duals. Of course, any of these theorems could also be derived without using the duality principle. For example:

Theorem 2b x*False = False

x*False	= False+(x+False)	by Postulate 1a
	= (x*/x)+(x+False)	by Postulate 2b
	= x*(/x+False)	by Postulate 4a
	= x*/x	by Postulate 1a
	= False	by Postulate 2b

1.5.2 Using Truth Tables

Finally, theorems may be demonstrated with truth tables. A theorem always holds true if it holds true for all cases; and since two variables can only have two values each, there are only four possible cases, so it is reasonable to look at a theorem on a case-by-case basis. For example, we can prove Theorem 5a with the following truth table:

x	y	/(x+y)	(/x*/y)
F	F	T	T
F	T	F	F
T	F	F	F
T	T	F	F

Table 1-7

It can be seen from Table 1-7 that, in every case, /(x+y) is equal to (/x*/y).

1.5.3 Complement of a Boolean Function

A *Boolean expression* is some mixture of Boolean variables and operators that has a value. For example:

x+y*z*/a

is a Boolean expression. A *Boolean function* is a statement in which two expressions are equated. For example:

a = b*c
/(c*d) = /c+/d

are Boolean functions. (The difference is the presence of an equal sign. It is worth noting that equals, or equivalence is also a Boolean function because two expressions are either equal or they aren't. However, in this book we will attempt to present only true equations, so the Boolean values of an equals sign may be ignored in functions.)

So far, we have talked about a Boolean expression's value as True or False. More frequently, these values are written as 1 and 0, with 1 standing for True, and 0 standing for False. From now on, we will also adopt this standard.

The complement of an expression may be written easily by placing the NOT operator in front of the enclosed expression:

/(x+y*z*/a)

but it is also possible to complement a function. The *complement* of a function is obtained by complementing both sides of an equation. For example, given the equation:

/a = b*c+1

the complement would be:

$$/*(/a) = /(b*c+1)$$

which could be simplified:

a = /(b*c+1)	by Theorem 3
a = /(b*c)*/1	by Theorem 5a
a = /(b*c)*0	def. of complement
a = 0	by Theorem 2b

Note the differences between obtaining the complement of a function, and obtaining the dual of a function. The complement is obtained by complementing the entire expression on both sides of the equation, and manipulating it from there with the given postulates and theorems. The dual of a function is obtained by replacing all 1's with 0's, all 0's with 1's, all ANDs with ORs, and all ORs with ANDs.

In fact, the easiest way in which to obtain the complement of a function is by taking the dual of the function and complementing each individual variable (called a *literal*). For example, the complement of:

$$F = (x+/y)*[w*(x+z)]$$

can be found by

1) taking the dual:

Dual: F = (x*/y)+[w+(x*z)] and,

2) complementing each literal:

Complementing: /F = (/x*y)+[/w+(/x*/z)]

1.6 Algebraic Simplification

A *literal* is a complemented (/x) or uncomplemented (x) variable. A *term* is a subexpression, often enclosed in parentheses. The equation:

$$F = (x+/y)*/x$$

has three literals and two terms. *Simplifying* a Boolean equation is an attempt to minimize the number of literals or the number of terms in an equation. Unfortunately, in many situations, one can only be minimized at the expense of the other, so it is important to decide from the outset whether you are minimizing literals or terms. Literals can be minimized by repeated applications of the postulates and theorems of Boolean algebra (Table 1-6), but there is no algorithm; it is a trial and error process. For example, the equation:

$$F = (x*/z) + [(x+y)*/z]$$

may be simplified through the following steps:

F = (x*/z)+[(x+Y)*/z]	
= (/z*x)+[/z*(x+Y)]	Postulate 3b
= /z*[x+(x+y)]	Postulate 4a
= /z*[(x+x)+y]	Theorem 4a
= /z*(x+y)	Theorem 1a

The equation is now simplified because there are no postulates or theorems, which, when applied, will serve to further reduce the number of literals.

1.6.1 Sum of Products and Product of Sums

When an equation is in the form:

$$F = (a*b)+(c*/a)+e$$

for example, it is said to be in *sum of products* form. This is because the equation is composed of a number of product terms (ANDs) that are summed (ORed) together. The subexpression result of two operands ANDed together is referred to as a product because of the resemblance of the AND operator to the multiplication operator of arithmetic; the result of OR is referred to as a sum because of the resemblance of the OR operator to the addition operator of arithmetic.

When an equation is in the form:

$$F = (a+b)*(a+/b)$$

for example, it is said to be in *product of sums* form, because it is composed of a number of sum terms (OR) that are ANDed together. Both sum of products and product of sums forms are called *standard form*.

1.6.2 Canonical Forms

If an equation has three variables that are complemented or uncomplemented, then there are a limited number of ways in which these variables can be ANDed or ORed together. Referring to Table 1-9, under the column 'Minterms', and the subcolumn 'Terms', there are seven different ways in which three variables could be ANDed together. Each combination has been given a name — the letter 'm' and a number. For example, the expression:

$$(/x*y*z) \text{ is } m_3,$$

while the expression:

$$(x*y*/z) \text{ is } m_7.$$

Using these shorthand notations for expressions, we can refer to the equation:

$$F = (/x*y*/z)+(x*/y*/z)+(x*/y*z)$$

as:

$$F = m_2+m_4+m_5$$

which is much more compact. When an equation is expressed in terms of these named AND subexpressions, or *minterms* it is said to be in sum of minterms form.

Similarly, there are seven ways in which three variables may be ORed together, each variable being primed or unprimed. A named OR subexpression is called a *maxterm*. The equation:

$$F = (x+y+z)*(x+/y+/z)*(/x+/y+/z)$$

could also be written as:

$$F = m_0*m_3*m_7$$

since each OR subexpression has been given a name consisting of an 'M' and a number. (See the column 'Maxterms' in Table 1-9.) An equation expressed in this way is said to be written in *product of maxterms* form. Both sum of minterms and product of maxterms forms are called *canonical forms*.

In many equations, not every variable is represented in every term, but it is still possible to write them in canonical form. A little algebraic manipulation will produce the missing terms that are needed. For example, the equation:

$$F = (x*y*z)+(/x*y)$$

is missing a 'z' in its second term. In order to write this equation in sum of minterms form, we must first take the following steps:

F = (x*y*z)+(/x*y*1) Postulate 1b
= (x*y*z)+[/x*y*(z+/z)] Postulate 2a
= (x*y*z)+(/x*y*z)+(/x*y*/z) Postulate 4a
= m_7+m_3+m_2

To create a missing variable in a maxterm, use the duals of the postulates used above. To create more than one missing variable, expand the equation as many times as is needed by following the steps above.

1.6.3 Conversion Between Canonical Forms

Canonical forms do not only exist because they are more compact; using canonical forms, it is possible to write any equation expressed in sum of products in terms of product of sums.

Given the equation:

F = (/a*b*/c)+(a*/b*c)+(a*b*c)
= m_2+m_5+m_7

we can take its complement by forming an equation from all the minterms that are NOT present in the equation:

/F = m_0+m_1+m_3+m_4+m_6
= (/a*/b*/c)+(/a*/b*c)+(/a*b*c)+(a*/b*/c)+(a*b*/c)

Finally, using the dual/complement method, we can take the complement again. Of course, by Theorem 3 (Table 1-9), anything that is complemented twice returns to its original value:

F = (a+b+c)*(a+b+/c)*(a+/b+/c)*(/a+b+c)*(/a+b+/c)

We have now expressed function F, originally in sum of products, in product of sums form. Any Boolean equation can be written in either form.

An even quicker way of doing this conversion is to write a product of maxterms equations using the maxterm numbers which did not appear in the original equation. For example, in our equation F, written in sum of minterms form, we used the numbers 2, 5 and 7. In our product of maxterms form, we would use the maxterms 0, 1, 3, 4 and 6.

F = m_0+m_1+m_3+m_4+m_6
= (a+b+c)*(a+b+/c)*(a+/b+/c)*(/a+b+c)*(/a+/b+c)

This works because each maxterm is the dual of the minterm that has the same number.

Of course, any equation written in the canonical forms can likely be simplified; so after converting from standard form to canonical form, then converting from one canonical form to another, you may wish to simplify your equation.

Postulate 1	(a) x+False = x (b) x*True = x
Postulate 2	(a) x+/x = True (b) x*/x = False
Postulate 3	(a) x+y = y+x (b) x*y = y*x
Postulate 4	(a) x*(y+z) = (x*y)+(x*z) (b) x+(y*z) = (x+y)*(x+z)
Theorem 1	(a) x+x = x (b) x*x = x
Theorem 2	(a) x+True = True (b) x*False = False
Theorem 3	/(x) = x
Theorem 4	(a) x+(y+z) = (x+y)+z (b) x*(y*z) = (x*y)*z
Theorem 5	(a) /(x+y) = /x*/y (b) /(x*y) = /x+/y
Theorem 6	(a) x+(x*y) = x (b) x*(x+y) = x

Table 1-6. Postulates and Theorems of Boolean Algebra

X	Y	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
			*		X		Y	:+:	+			/Y		/X			

Table 1-7. Boolean Operators

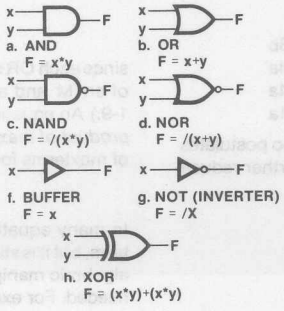


Table 1-10. Logic Gates

2.0 Binary Systems

Binary numbers utilize a base 2 number system that can only be in one of two logical states: a "0" or a "1". This number system is used in current digital computer systems because the outputs of most switching circuits can only be in one of the two logical states. Also, when transistor circuits are operating in one of two modes only, greater reliability can be obtained.

2.1 Base Conversion

Normally, decimal (base 10) numbers are written using a positional notation. In other words, the value of the number is determined by multiplying each digit to an appropriate power of 10 which is dependent on its relative position to the decimal point.

Example 2.1

$$714.02 = 7x10^2+1x10^1+4x10^0+0x10^{-1}+2x10^{-2}$$

2.1.1 Base 2 to Base 10 Conversion

Similarly, binary (base 2) numbers are also position-dependent relative to the binary point; each binary digit is multiplied by an appropriate power of 2 in order to obtain the decimal equivalent. The following example shows the conversion from a base 2 number to a base 10 number.

Example 2.2

$$\begin{aligned} 101.01_2 &= 1x2^2+0x2^1+1x2^0+0x2^{-1}+1x2^{-2} \\ &= 4+0+1+0+1/4 \\ &= 5.25_{10} \end{aligned}$$

Notice that the binary point separates the positive and the negative powers of 2. This is similar to the case of the decimal point separating the positive and negative powers of 10.

2.1.2 Base 10 to Base 2 Conversion

Converting a base 10 integer to a base 2 integer requires utilizing the division method. To explain, let N represent the base 10 integer. Divide this integer, N, by 2 since base 2 is desired. As a result, there should be a quotient, Q_0, and a remainder, R_0. Then divide the previous quotient, Q_0, by 2 again and continue this process until the final quotient equals zero. The desired binary digits are the remainders resulting from each division step; the least significant bit starts with R_0.

Example 2.3

Converts 61_{10} to binary:

61/2 = 30	remainder = 1	LSB
30/2 = 15	remainder = 0	
15/2 = 7	remainder = 1	
7/2 = 3	remainder = 1	
3/2 = 1	remainder = 1	
1/2 = 0	remainder = 1	MSB

$61_{10} = 111101_2$

Converting decimal fractions to binary requires successive multiplications by 2. Let F be a decimal fraction. Multiply this number F by 2 and obtain an integer and a fraction result. Take this integer and multiply once again by 2. Continue this process until it terminates or until a sufficient number of digits has been reached. The desired digits are the integer parts that were obtained at each multiplication step. The most significant digit is obtained first.

Example 2.4

Convert 0.375_{10} to binary

0.375	0.750	0.500	$I_0 = 0$	MSB
x 2	x 2	x 2	$I_1 = 1$	
0.750	1.500	1.000	$I_2 = 1$	LSB

$0.375_{10} = 0.011$

Note that if this procedure doesn't terminate, then the result must be a repeating fraction.

2.1.3 Base 2 to Base 8

To convert binary to *octal* (base 8) or vice versa is very simple and can be done by inspection. Each octal digit corresponds to three binary digits since it can be in one of eight states (0 to 7). Therefore, the binary number should be divided into groups of three starting from the binary point. Each group on both sides of the binary point is replaced by an octal digit representation.

Example 2.5

$$\begin{aligned} 101110.011_2 &= 101\ \ 110\ .\ 011 \\ &= 5\ \ 6\ .\ 3_8 \end{aligned}$$

Similarly, binary to *hexadecimal* (base 16) and vice versa can also be done easily. This time, instead of three, the binary number is broken up into groups of four. The reason is because a hexadecimal digit can assume one of sixteen states (0 to 9, A, B, C, E and F). Again starting from the binary point, each group is replaced by its hexadecimal equivalent.

Example 2.6

$$\begin{aligned} 11100101.0011_2 &= 1110\ \ 0101\ .\ 0011 \\ &= E\ \ 5\ .\ 3_{16} \end{aligned}$$

2.2 Simplicity of Binary Arithmetic

Due to the design of logic networks, it is much easier to do binary than decimal arithmetic in digital systems. Although binary arithmetic is implemented in about the same manner, the addition tables are much easier. Fortunately, numerical subtractions may be performed by addition operations between numbers. This property is of little use in the decimal system. However, much can be gained if used in the binary system. This is mainly due to the fact that in a binary system, complements of numbers are easily implemented, and the same hardware can be used for addition and subtraction operations. This allows for considerable savings in terms of system hardware design.

2.2.1 1's Complement

To find the 1's complement of a binary number is easily done by inverting each digit (0 or 1) up to the most significant digit specified.

Example 2.7

The 1's complement of:

01011.1101 = 10100.0010

2.2.2 Subtraction with 1's Complement

To subtract two positive binary numbers X and Y, (X-Y), the following procedures should be used:

1. Take the 1's complement of Y and add it to X.
2. Check results for overflow carry:
   a. If there is an overflow carry, add it to the least significant digit of the result.
   b. If there is no overflow carry, the result is negative. Then, complement this result and place a minus sign in front.

Example 2.8

a) 1010.11 - 1000.01 = ?

```
                1010.11
              + 0111.10  ← 1's complement of 1000.01
overflow 1      0010.01
              +        1  ← add overflow carry
                0010.10  → answer
```

b) 1001.10 - 1100.11 = ?

```
                1001.10
              + 0011.00  ← 1's complement of 1100.11
no overflow     1100.10  → -0011.01
```

Since there is no overflow carry, take the 1's complement of 1100.10 and add a negative sign in front of it:
Answer is -0011.01

2.2.3 2's Complement

The most widely used numbering manipulation technique in current digital computers is the 2's complement method. This method is easily implemented with any decent computer instruction set. Using the same hardware for addition and subtraction in 2's complement makes system design simpler and can lead to savings in cost.

To find the 2's complement of a binary number requires the following:

1. Take the logical complement by inverting each digit of the binary number.
2. Add 1 to the least significant digit.

Example 2.9

The 2's complement of 001100.01 is:

```
step (1)    110011.10  ← logical complement of 001100.01
step (2) +         1
            110011.11  → answer
```

This technique can also be done by visual inspection. Start with the least significant digit of the number and visually scan to the left. Leave all digits unchanged until the first "1" is encountered. Then invert all the remaining digits to the left. Note that the binary point has no effect on this procedure.

2.2.4 Subtraction with 2's Complement

The steps for subtracting two binary numbers X and Y, (X-Y), are as follows:

1. Add X to the 2's complement of Y.
2. Check result for overflow carry:
   a. If there is an overflow carry, then throw it out. The result now represents (X-Y).
   b. If there is no overflow carry, the number is negative. Take the 2's complement of the result and place a negative sign in front of it.

Example 2.10

a) 1110.11 - 1011.10 = ?

```
                1110.11
              + 0100.10  ← 2's complement of 1011.10
overflow carry
throw out 1     0011.01  → +0011.01
```

b) 0001.11 - 1000.10 = ?

```
                0001.11
              + 0111.10  ← 2's complement of 1000.10
no overflow
carry           1001.01  → -0110.11
```

Since there is no overflow, this number is negative. Therefore, take the 2's complement of 1001.01 and add a minus sign in front: Answer is -0110.11

3.0 Karnaugh Map

3.1 Karnaugh Map Technique

There exists a technique that allows the logic designer to minimize Sum of Product terms by utilizing *Karnaugh maps*. The Karnaugh map (referred to as *K-map*) graphically displays the implicants (*minterm*) in any Sum of Product expression. It is derived directly from the truth table of this expression. K-maps are very useful for minimizing three, four, five and even six variable functions, but it gets too complicated beyond six. For expressions with more than six variables, the numerical manipulation should be done on a computer that uses the Quine-McCluskey method. This technique will not be discussed in this book.

3.1.1 K-Map Reading Procedure

Each minterm cell in the K-map has a value of "1" as determined by the truth table. Circle those single minterm cells that will combine with its adjacent cells to form larger groups of 1, 2, 4, 8, etc. If each single minterm cell is grouped individually, the map reading process should yield the original Sum of Product expression.

However, if two minterm cells are grouped together, at least one variable is dropped. This is because the theorem X*Y+X*/Y = X has been executed once. If a group of four adjacent minterm cells have been combined, then the theorem has been executed twice and two variables are dropped. Thus, a group of eight adjacent cells result in three variables being dropped. Therefore, the main objective is to minimize the number of minterm cell groupings while maximizing the number of minterm cells in each grouping. By minimizing the number of cell groupings, the number of inputs to the OR function is reduced. On the other hand, by maximizing the number of cells in each grouping, the number of inputs to the AND function is reduced.

3.1.2 K-Map Matrix Labels

In labeling the K-map matrix, the following rule should be followed.

Top to bottom or left to right:

Two-variable	Three-variable	Four-variable
00	000	Add a '0' MSB and use the three-variable chart for the first half. For the second half, add a '1' MSB and repeat the same chart in reverse order.
01	001	
11	011	
10	010	
	110	
	111	
	101	
	100	

Notice that the number of variables shown above is referring to one axis only (X or Y). However, this technique may be used for any number of variables that may be desired on each axis. For any axis greater than one variable, the second-half is a mirror image of the first-half with the MSB equal to a '1'. This can be seen above when comparing the three-variable list to the two-variable list.

3.1.3 K-Map Examples

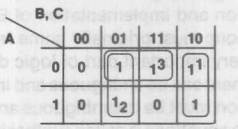
Examples of three- and four-variables K-maps are shown below. The corresponding truth tables for the examples are also shown to illustrate the derivation of the K-maps.

Example 3.1

Three-Variables K-map:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Truth Table

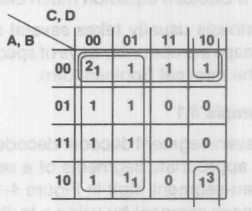


Karnaugh Map

product term 1 = B*/C
product term 2 = /B*C
product term 3 = /A*C
(SOP form)
F = B*/C+/B*C+/A*C

Example 3.2

Four-Variables K-map:



Karnaugh Map

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Truth Table

product term 1 = /C
product term 2 = /A*/B
product term 3 = /B*C*/D
(SOP form)
F = /C+(/A*/B)+(/B*C*/D)

4

4.0 Combinational Logic

4.1 Logic Design Introduction

Logic design is a combination of analysis, synthesis, minimization and implementation of Boolean functions. Boolean functions must originally come from worded statements. This is a very important part of logic design because the worded statement can be ambiguous and imprecise, while the Boolean equation must be unambiguous and exact. The conversion of words to equations is called synthesis. Engineers must be careful when synthesizing a problem because many times the originator of a problem is not a technical person. It is the responsibility of the logic designer to review the synthesis of the problem with the originator to make sure the solution is suitable.

4.2 Combinational Design

Combinational logic is a network whose output is solely dependent upon its inputs. It has no feedback loops or memory elements.

The first step in combinational design is to analyze the problem and then define it in an exact manner. This will make synthesizing a Boolean equation much easier.

Synthesis usually takes several steps. Using truth tables and K-maps are common ways of specifying a problem and putting it in the minimal Boolean form.

Example 4.1

A seven-segment decoder decodes a BCD number and turns on the appropriate segments of a seven-segment digit. Given the seven-segment digit in Figure 4-1 develop a minimal equation for each segment by using a truth table and K-maps.

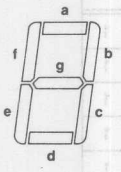
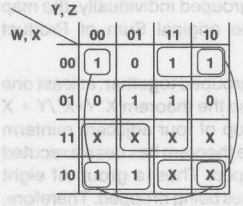


Figure 4-1. Seven-Segment Decoder

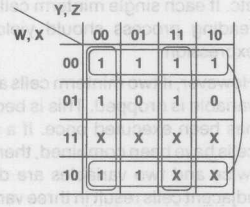
Forming a truth table from Figure 4-1 is done by writing all '10' possible inputs down, then determining which segments should be activated for each input. For example, segment 'a' is activated whenever; 2, 3, 5, 7, 8, 9 or 0 is input to the decoder. Once the table is formed, a K-map can be made for each segment. The K-maps are used to derive a Sum of Products logic equation for each segment.

K-maps are an excellent way of forming equations when three to six variables are involved in a problem. Two standard algebraic forms of the function can be derived — the standard Sum of Products — (minterm expansion) and the standard Product of Sums (maxterm expansion). A network of AND and OR gates is derived directly from either form.

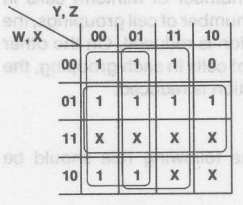
DECIMAL	INPUTS				OUTPUTS						
	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1



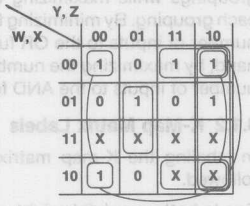
a = /X*/Z+W*/Y+X*Z+/X*Y



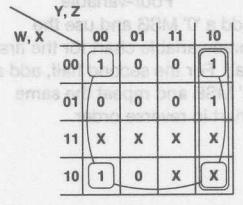
b = /X+/Y*/Z+Y*Z



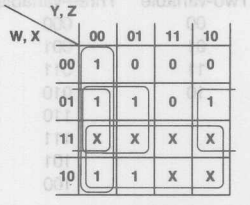
c = /Y+X+Z



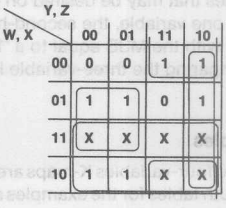
d = /X*/Z+/X*Y+Y*/Z+X*/Y*Z



e = /X*/Z+Y*/Z



f = W+X*/Z+X*/Y+/Y*/Z



g = W+X*/Y+/X*Y+Y*/Z

ROW	A	B	C	MINTERMS	MAXTERMS
0	0	0	0	/A*/B*/C = m0	A+B+C = M0
1	0	0	1	/A*/B*C = m1	A+B+/C = M1
2	0	1	0	/A*B*/C = m2	A+/B+C = M2
3	0	1	1	/A*B*C = m3	A+/B+/C = M3
4	1	0	0	A*/B*/C = m4	/A+B+C = M4
5	1	0	1	A*/B*C = m5	/A+B+/C = M5
6	1	1	0	A*B*/C = m6	/A+/B+C = M6
7	1	1	1	A*B*C = m7	/A+/B+/C = M7

Table 4-1. Minterm and Maxterm Expansions

Each minterm is a product term, so a Sum of Products expression may be written with minterms.

Example 4.2

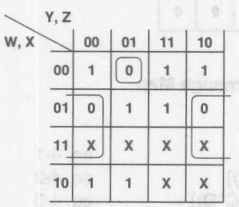
Rewrite the Sum of Products equations from example 4.1 in minterm form.

a = Σm (0, 2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 15)
b = Σm (0, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 15)
c = Σm (0, 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15)
d = Σm (0, 2, 3, 5, 6, 8, 10, 11, 13)
e = Σm (0, 2, 6, 8, 10, 14)
f = Σm (0, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15)
g = Σm (2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15)

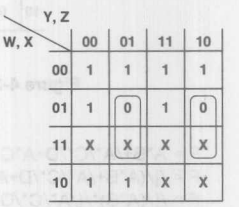
Each maxterm is a sum of variables. It is derived by solving a K-map for the *0-terms* instead of the *1-terms*. Maxterms are used in a Product of Sums solution.

Example 4.3

Rework the first two K-maps from Example 4.1 to get a maxterm solution.



/a = X*/Z+/W*X*/Y*Z
a = (/X+Z)*(W+X+Y+/Z)
a = ΣM(1,4,6,12,14)



/b = X*/Y+X*/Z
b = (/X+Y)*(/X+Z)
b = ΣM(5,6,13,14)

Given either algebraic form, it is a simple matter of converting to the other form, or the inverse of either form.

GIVEN FORM	DESIRED FORM			
	Minterm expansion of F	Maxterm expansion of F	Minterm expansion of F	Maxterm expansion of F
Minterm expansion of F	—	Maxterm numbers are those numbers not on the minterm list for F	List minterms not present in F	Maxterm numbers are the same as minterm numbers of F
Maxterm expansion of F	Minterm numbers are those numbers not on the maxterm list for F	—	Minterm numbers are the same as maxterm numbers of F	List maxterms not present in F

Table 4-2. Conversion of Forms Table

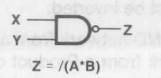
4.3 NAND Gates and NOR Gates

A set of logic operators is said to be functionally complete if any Boolean function can be expressed in terms of this set of operations. The set; AND, OR and NOT, is functionally complete. The NAND and the NOR gates are each functionally complete by themselves. Therefore they are called *Universal* gates.

Conversion of AND and OR networks to NAND networks is carried out by starting with a minimal sum of products expression and then applying the theorem; F = /(/F). This equation should then be solved using De Morgan's theorem.

NAND

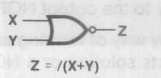
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



Z = /(A*B)

NOR

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0



Z = /(X+Y)

Table 4-3. Truth Tables

4

Example 4-3

From the K-map in Figure 4-2, find equations for an AND-OR, NAND-NAND, OR-NAND and NOR-OR networks.

A, B \ C, D	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	1	1	1	1
10	0	0	0	0

Figure 4-2. Karnaugh Map

F = A*B+/A*/C*/D+A*C*D eq. 4-1
F = /[/(A*B+/A*/C*/D+A*C*D)] eq. 4-2
F = /[/(A*B)*(/A*/C*/D)*/(A*C*D)] eq. 4-3
F = /[(/A+/B)*(A+C+D)*(/A+/C+/D)] eq. 4-4
F = /(/A+/B)+/(A+C+D)+(/A+/C+/D) eq. 4-5

Equations 4-2 thru 4-5 are AND-OR, NAND-NAND, OR-NAND and NOR-OR networks, respectively.

In order to get a network of NOR gates we must start with the minimum Product of Sums form of F.

Example 4-4

From the K-map in Figure 4-2 find equations for OR-AND, NOR-NOR, AND-NOR and NAND-AND networks.

/F = /A*/C*D+/A*C*/D+A*/B eq. 4-6
F = A+C+/D*A+/C+D*/A+B eq. 4-7
F = /[/(A+C+/D*A+/C+D*/A+B)] eq. 4-8
F = /[/(A+C+/D)+/(A+/C+D)+/(/A+B)] eq. 4-9
F = /(/A*/C*D+/A*C*/D+A*/B) eq. 4-10
F = /(/A*/C*D)*/(/A*C*/D)*/(A*/B) eq. 4-11

Equations 4-7, 4-9, 4-10 and 4-11 are OR-AND, NOR-NOR, AND-NOR and NAND-AND networks, respectively.

NAND-NAND and NOR-NOR networks are very common in industry because both the NAND and NOR gates are universal gates. Thus, these gates are made in great quantities, making them more available for designers.

A NAND-NAND network is made from a Sum of Products solution. The AND and OR gates of the SOP solution are replaced by NAND gates with all the interconnections staying the same. Variables that are input directly to the output gate must be inverted.

A NOR-NOR network is made from a Product of Sums solution. The OR and AND gates are replaced by NOR gates with all interconnections staying the same. Any variables that are input directly to the output NOR gate must be inverted.

An easy way of forming either a NAND network from a Sum of Products solution or a NOR network from a Product of Sums solution is to place two inversion bubbles in series between the two levels as demonstrated in Figure 4-3.

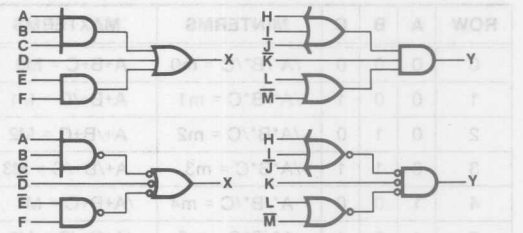


Figure 4-3. Network Conversion

4.4 Multiplexers

Multiplexers are circuits which select one of 2^n input lines using n selector lines. For example, an eight-input multiplexer selects one of 2^3 input lines using three select lines.

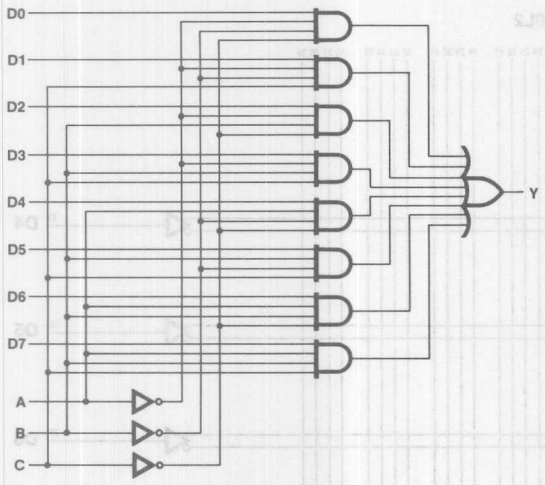
Example 4.5

Design an 8:1 multiplexer in SOP form by using a truth table.

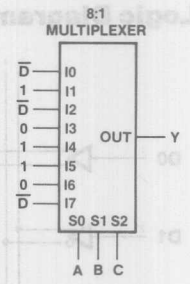
SELECT			MULTIPLEXER INPUTS								OUTPUT
A	B	C	D0	D1	D2	D3	D4	D5	D6	D7	Y
0	0	0	0	X	X	X	X	X	X	X	0
0	0	0	1	X	X	X	X	X	X	X	1
0	0	1	X	0	X	X	X	X	X	X	0
0	0	1	X	1	X	X	X	X	X	X	1
0	1	0	X	X	0	X	X	X	X	X	0
0	1	0	X	X	1	X	X	X	X	X	1
0	1	1	X	X	X	0	X	X	X	X	0
0	1	1	X	X	X	1	X	X	X	X	1
1	0	0	X	X	X	X	0	X	X	X	0
1	0	0	X	X	X	X	1	X	X	X	1
1	0	1	X	X	X	X	X	0	X	X	0
1	0	1	X	X	X	X	X	1	X	X	1
1	1	0	X	X	X	X	X	X	0	X	0
1	1	0	X	X	X	X	X	X	1	X	1
1	1	1	X	X	X	X	X	X	X	0	0
1	1	1	X	X	X	X	X	X	X	1	1

Truth Table for 8:1 Multiplexer

As can be seen from the truth table A, B and C select one of the eight multiplexer inputs to appear on the output, Y. If A, B and C = 011, then the D3 AND gate will be enabled while all the other AND gates will be disabled. This allows D3 to be 'ORed' with seven zeros and thus end up on the output Y.



A	B	C	
0	0	0	I0 = /D
0	0	1	I1 = 1
0	1	0	I2 = /D
0	1	1	I3 = 0
1	0	0	I4 = 1
1	0	1	I5 = 1
1	1	0	I6 = 0
1	1	1	I7 = /D



Example 4.6

Design a dual 8:1 mux with the appropriate PAL device.

When selecting a PAL device several things must be considered. Will the design need registers, how many inputs and outputs are there, are the outputs active high or active low? For a Dual 8:1 mux the select lines will be shared but the eight data inputs to each mux are independent. Thus, we need nineteen inputs and two outputs for the design. This narrows our choices down to one PAL device, the PAL20L2. The output of the PAL20L2 is active low but this causes no problems because an active high output will result by simply inverting all the data inputs.

Multiplexers have been widely used as logic devices as well as selector circuits. A 4:1 mux can be used to realize any three-variable function. An 8:1 mux can realize any four-variable function.

Example 4.7

Solve the K-map in Figure 4-4 and build the circuit with an 8:1 multiplexer.

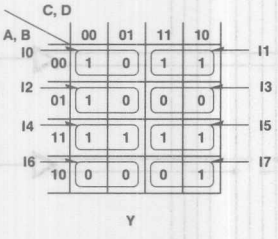


Figure 4-4. Eight One-Variable Karnaugh Maps

A, B and C are used as control inputs to the multiplexer, this leaves D as the only real variable in the problem. The 16-square K-map can thus be broken up into eight one-variable K-maps. Each map is solved for one of the eight data inputs to the 8:1 multiplexer.

4.5 Decoders

On a multiplexer with n address lines, one of the 2^n inputs is selected to be output. On a decoder with n address lines, one of the 2^n output lines is forced either high or low, depending on the design of the decoder. Table 4-4 shows a truth table for a 3-to-8 decoder.

SELECT LINES			OUTPUT LINES							
A	B	C	f	g	h	i	j	k	l	m
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Table 4-4. Decoder Truth Table

A decoder will have as many outputs as there are possible binary input combinations. It can be seen from Table 4-4 that only one output can be equal to 1 at any one time. The outputted 1 represents the minterm combination that was input to the decoder. It can also be noticed from Table 4-4 that there is not a combination of inputs that will give all 0's on the outputs. Many designs need this ability. It can be added simply by putting an enable line in all of the output AND gates. The logic design and block diagram for the 3-bit decoder in Table 4-4 appears in Figure 4-5.

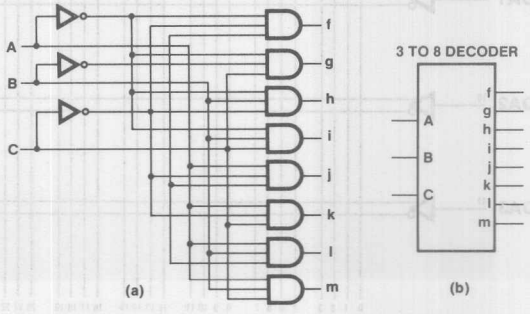
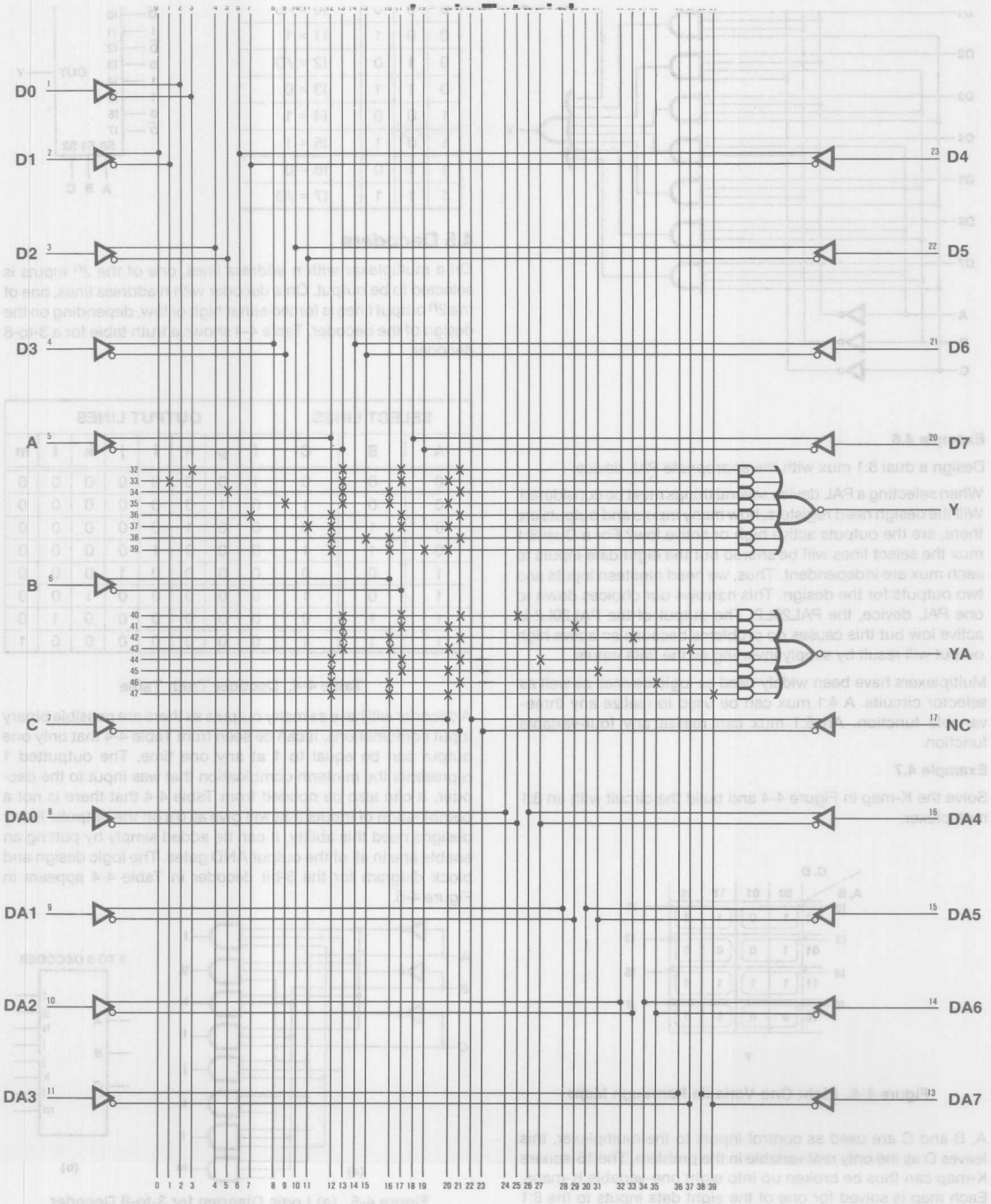


Figure 4-5. (a) Logic Diagram for 3-to-8 Decoder (b) Block Diagram for 3-to-8 Decoder


D0
D1
D2
D3
A
B
C
DA0
DA1
DA2
DA3
D4
D5
D6
D7
Y
YA
NC
DA4
DA5
DA6
DA7

A magnitude comparator is a combinational circuit that compares two numbers, then outputs one of three signals; A > B, A = B or A<B.

Example 4.8

Design a 3-bit magnitude comparator in a Sum of Products form, then fit it into an appropriate PAL device.

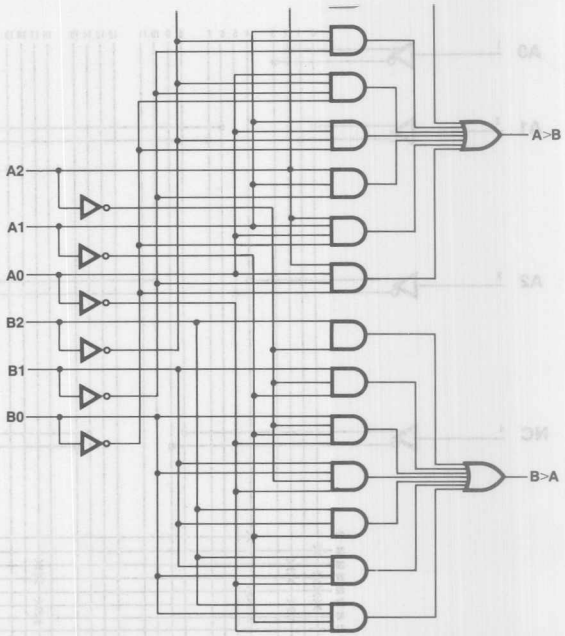
A2, A1, A0 \ B2, B1, B0	000	001	011	010	110	111	101	100
000	0	0	0	0	0	0	0	0
001	1	0	0	0	0	0	0	0
011	1	1	0	1	0	0	0	0
010	1	1	0	0	0	0	0	0
110	1	1	1	1	0	0	1	1
111	1	1	1	1	1	0	1	1
101	1	1	1	1	0	0	0	1
100	1	1	1	1	0	0	0	0

A>B = A2*/B2
+ A1*/B2*/B1
+ A0*/B2*/B1*/B0
+ A1*A0*/B2*/B0
+ A2*A1*/B1
+ A2*A1*A0*/B0
+ A2*A0*/B1*/B0

A2, A1, A0 \ B2, B1, B0	000	001	011	010	110	111	101	100
000	0	1	1	1	1	1	1	1
001	0	0	1	1	1	1	1	1
011	0	0	0	0	1	1	1	1
010	0	0	1	0	1	1	1	1
110	0	0	0	0	0	1	0	0
111	0	0	0	0	0	0	0	0
101	0	0	0	0	1	1	0	0
100	0	0	0	0	1	1	1	0

B>A = /A2*B2
+ /A2*/A1*B1
+ /A2*/A1*/A0*B0
+ /A2*/A0*B1*B0
+ /A1*B2*B1
+ /A0*B2*B1*B0
+ /A1*/A0*B2*B0

The six-variable K-maps are used to produce Sum of Product equations for A > B and B > A. These equations are then used to form the two-level logic diagram of the 3-bit magnitude comparator.



The logic diagram of the 3-bit comparator shows that there are six inputs and two outputs in the circuit. Each output is derived from seven product terms. The PAL16H2 fits the design best. This PAL device has more inputs than are needed, but it is the smallest PAL device with enough product terms to realize the circuit. A NOR gate external to the PAL device can be used to get the result A = B. The outputs of the PAL device will be the inputs to the NOR gate, when both inputs equal '0', A equals B.

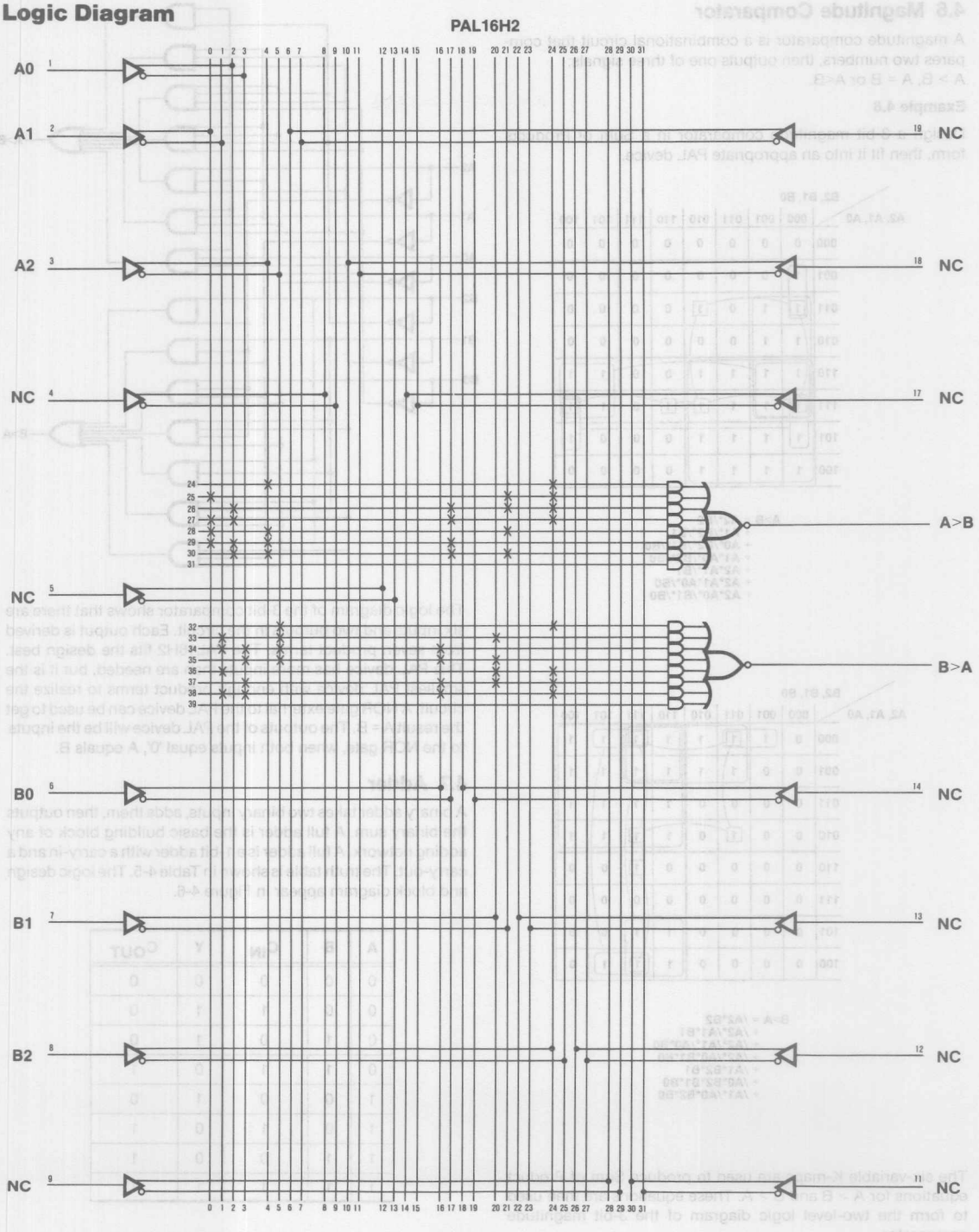
4.7 Adder

A binary adder takes two binary inputs, adds them, then outputs the binary sum. A full adder is the basic building block of any adding network. A full adder is a 1-bit adder with a carry-in and a carry-out. The truth table is shown in Table 4-5. The logic design and block diagram appear in Figure 4-6.

A	B	C_{IN}	Y	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

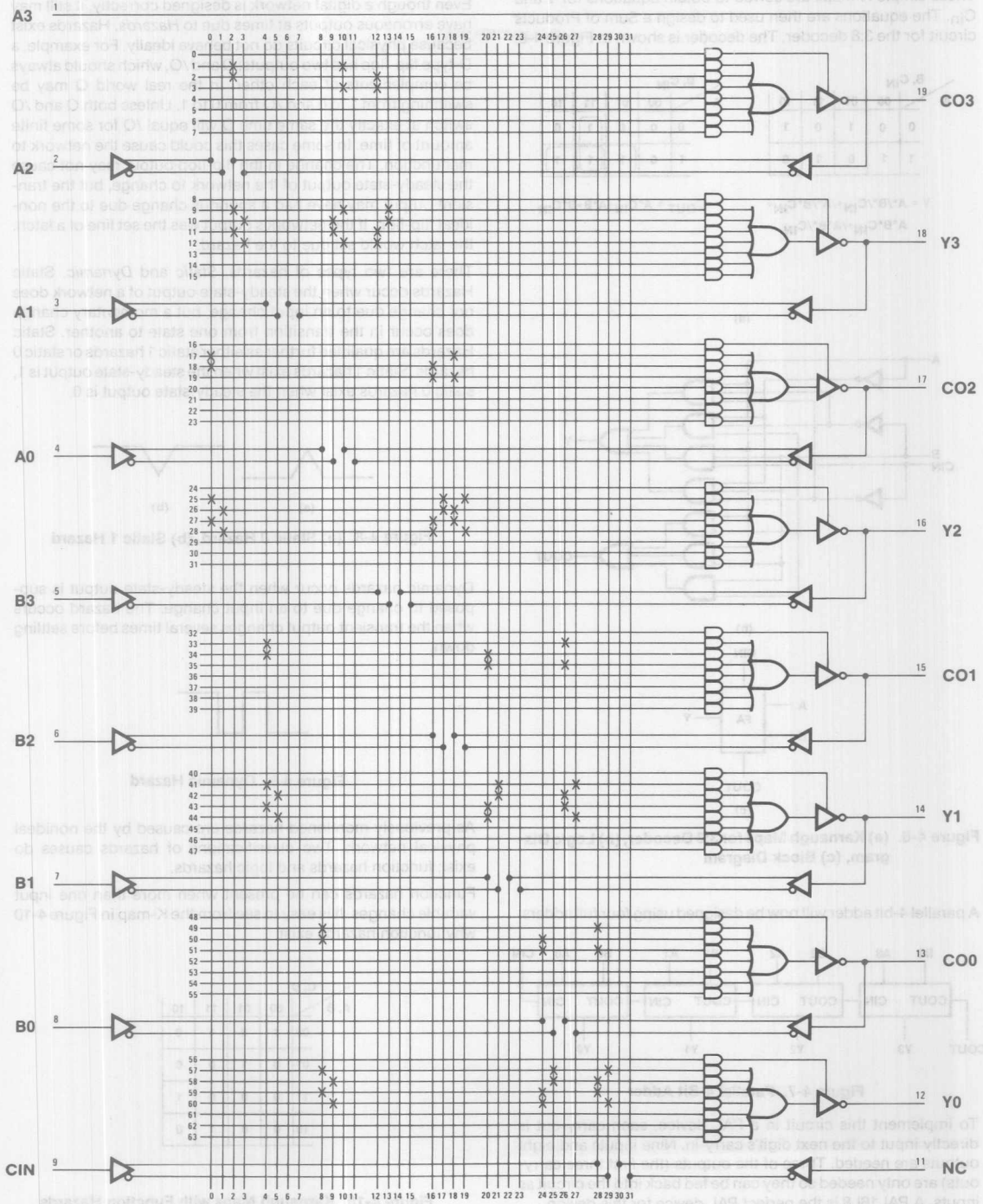
Table 4-5. Truth Table for Full Adder

Logic Diagram



Logic Diagram

PAL16L8



The truth table is used to form K-maps for the outputs Y and C_{in}. These simple K-maps are solved to obtain equations for Y and C_{in}. The equations are then used to design a Sum of Products circuit for the 3:8 decoder. The decoder is shown in Figure 4-6.

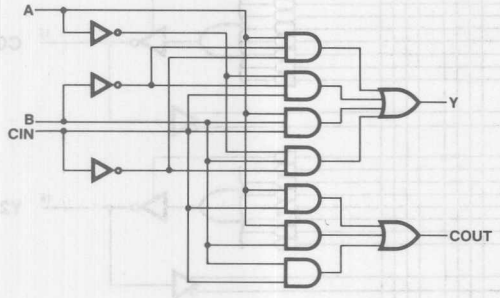
A \ B, C_{IN}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$Y = A^*/B^*/C_{IN}{}^+/A^*/B^*C_{IN}{}^+ A^*B^*C_{IN}{}^+/A^*B^*/C_{IN}$

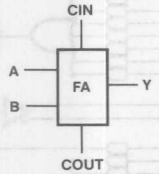
A \ B, C_{IN}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$C_{OUT} = A^*C_{IN}{}^+A^*B^+B^*C_{IN}$

(a)



(b)



(c)

Figure 4-6. (a) Karnaugh Maps for 3:8 Decoder, (b) Logic Diagram, (c) Block Diagram

A parallel 4-bit adder will now be designed using four full adders.

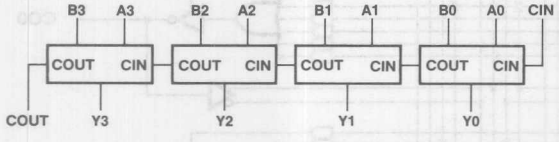


Figure 4-7. Parallel 4-Bit Adder

To implement this circuit in a PAL device, each carry-out is directly input to the next digit's carry-in. Nine inputs and eight outputs are needed. Three of the outputs (the first three carry-outs) are only needed so they can be fed back into the circuit as inputs. A PAL16L8 is the perfect PAL device for this design.

4.8 Hazards

Even though a digital network is designed correctly, it still may have erroneous outputs at times due to *Hazards*. Hazards exist because physical circuits do not behave ideally. For example, a D-type flip-flop has two outputs; Q and /Q, which should always be complements of each other. In the real world Q may be switching from 1 to 0 and /Q from 0 to 1. Unless both Q and /Q switch at exactly the same time Q will equal /Q for some finite amount of time. In some cases this could cause the network to malfunction. The change in the flip-flop output may not cause the steady-state output of the network to change, but the transient output may have had a spurious change due to the non-ideal flip-flop. If the network's output was the set line of a latch, the latch would set due to the hazard.

There are two types of hazards, *Static* and *Dynamic*. Static Hazards occur when the steady-state output of a network does not change due to an input change, but a momentary change does occur in the transition from one state to another. Static Hazards are qualified further as either static 1 hazards or static 0 hazards. Static 1 hazards exist when the steady-state output is 1, static 0 hazards exist when the steady-state output is 0.

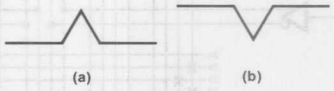


Figure 4-8. (a) Static 0 Hazard, (b) Static 1 Hazard

Dynamic hazards occur when the steady-state output is supposed to change due to an input change. The hazard occurs when the transient output changes several times before settling down.

Figure 4-9. Dynamic Hazard

As previously mentioned hazards are caused by the nonideal physical network. Two classifications of hazards causes do exist; function hazards and logic hazards.

Function hazards can be present when more than one input variable changes. It is easy to see from the K-map in Figure 4-10 why function hazards exist.

A, B \ C, D	00	01	11	10
00	1	0	1	0
01	0	1	0	0
11	0	0	0	1
10	0	0	1	0

Figure 4-10. Karnaugh Maps with Function Hazards

A function static 1 hazard is present when the input variables A, B, C and D go from < 0000 > to < 0101 >. If both B and D changed simultaneously no temporary erroneous pulse would appear on the output; however in the real world either B or D would change first. The transient output would have gone to 0 from being momentarily in state < 0100 > or < 0001 >. Looking at the K-map, it is easy to see function static hazards and function dynamic hazards.

The easiest way to avoid function hazards is by restricting input changes to one variable at a time. This method is not always possible though, because the inputs may not be under your control.

Logic hazards exist because of the way a function is realized. Logic hazards can exist even if input changes are rescticted to one variable at a time.

A K-map is a very good way of locating logic hazards. When trying to locate the static 0 and static 1 logic hazards on a K-map it is only necessary to map the 1-sets or the *0-sets*.

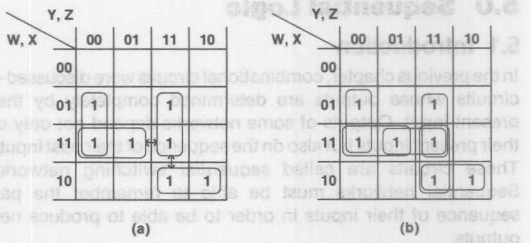


Figure 4-11. (a) Karnaugh Map with Two Logic Hazards (b) Karnaugh Map with No Logic Hazards

A 1-set is a product expression derived from a grouping of 1's on the K-map. If there are two adjacent input states that produce a 1 on the output, but are not covered by the same 1-term, a static logic hazard exists. Logic hazards may be eliminated by redesigning the circuit so adjacent input states that produce ones are covered by the same 1-term.

4

5.1 Introduction

In the previous chapter, combinational circuits were discussed — circuits whose outputs are determined completely by their present input. Outputs of some networks depend not only on their present inputs but also on the sequence of their past inputs. These circuits are called *sequential* switching networks. Sequential networks must be able to remember the past sequence of their inputs in order to be able to produce new outputs.

5.2 Latches

In order for a sequential circuit to remember the previous inputs, it must retain those values in some memory elements. The most basic memory element is called a *latch*. Latches are memory devices with one or more inputs that effect their outputs.

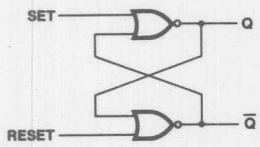
One of the most important properties of the latches is that any change to the input of the latch will appear at the output and the new output will be delayed only by the propagation delay of the gates between inputs and the outputs. All the latches have this transparency property and usually are referred to as transparent latches. A latch constructed of NOR gates is shown in Figure 5-1a.

5.2.1 RS Latch

The circuit in Figure 5-1a is called a *SET-RESET* or an RS latch. There are two input lines to an RS latch, which are used to control the state of the latch. The rules for this type of latch are:

1. If SET = RESET = 0, then the latch remains in the same state and output does not change.
2. A '1' on the SET and '0' on the RESET line will make the latch SET to '1'.
3. A '0' on the SET line and a '1' on the RESET line causes the latch RESET to '0' state.
4. If SET = RESET = 1, then Q and /Q will be '0' at the same time which is meaningless. When designing with an RS latch, we should remember that SET = RESET = 1 is forbidden.

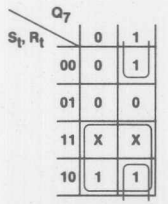
Following an RS latch circuit, its *state table, characteristic equation* and waveforms are shown.



(a)

S_t	R_t	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

(b)



$Q_{t+1} = S_t + Q_t{}^* / R_t$
(where, $S_t{}^* R_t = 0$)

(c)

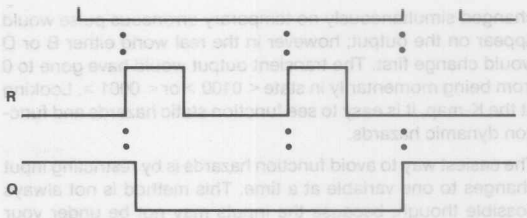
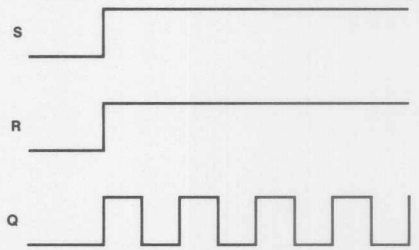
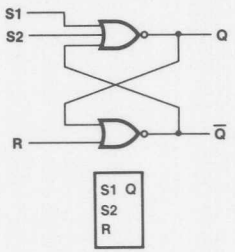


Figure 5-1. RS Latch (a) Logic Circuit (b) State Table (c) Karnaugh Map (d) Waveforms

Let's examine the RS latch circuit when SET = RESET = 1. In this case the output, Q, will toggle for a long period of time, and it is unpredictable to know when the circuit will be out of this state. The following waveforms examine this case:



Latches can have more than two inputs. In Figure 5-2, we examine a latch circuit that has two SET terms instead of one. The latch circuit and the transition table are shown.



(a)

S1	S2	R	Q_t	Q_{t+1}
0	0	0	X	NC
1	X	0	X	1
X	1	0	X	1
0	0	1	X	0
1	X	1	X	0
X	1	1	X	0

(b)

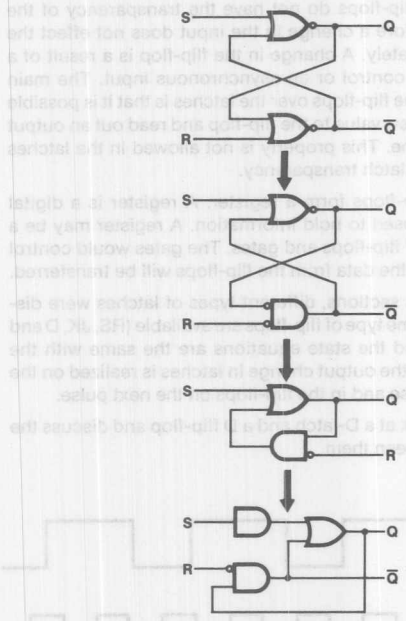
Figure 5-2. RS Latch (a) Latch Circuit (b) State Table

The RS latch configuration shown in Figure 5-1a, can be modified considering that an OR gate with an inverted output is equivalent to an AND gate with inverted inputs.

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

The RS latch schematic can be changed, so that it uses the above equivalence:

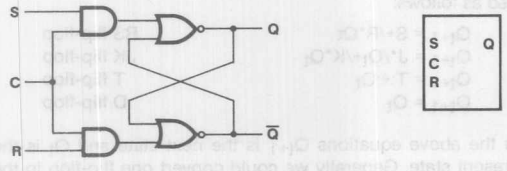


This configuration for realizing a latch is very useful, because it is in sum-of-product form.

In some applications using latches, it is desired for the input data to be effective only when another signal — usually referred to as a control signal — is active. For these applications, the RS latch could be modified as shown in Figure 5-3.

It is apparent from Figure 5-3 that only when the control signal (C) is active, the values of the SET and RESET are effective. So when C = 0, the changes in the SET-RESET terms would not have any effect on the output.

A thorough examination of Figure 5-3b will show that a change in the input of the latch does not effect the output simultaneously, and there is a short delay for this change to appear on the output. This delay is caused because of the propagation delays of the gates between inputs and outputs.



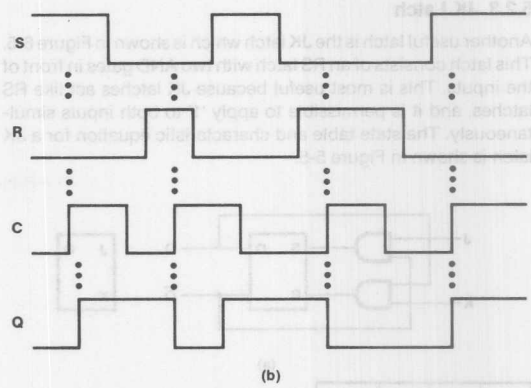
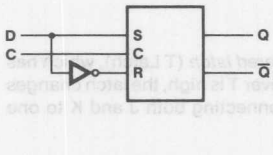


Figure 5-3. RS Latch with Control (a) Latch Circuit (b) Waveforms

5.2.2 D Latch

Other kinds of latches are used in sequential circuits. One of the most popular latches is called a *delay latch* — D latch. An RS latch is modified to a D-latch by inserting an inverter between S and R, and assigning S to D input term. The D latch will take the value of its input and transfer it to the output. The advantage of the D latch over the RS latch is that in the former only one input is needed and there is no forbidden state. The only disadvantage of the D latch is that it does not have a "no change" state. This state could be reached by inserting a control signal, C, as an input to the latch (Figure 5-4).



D_t	Q_{t+1}
0	0
1	1

$Q_{t+1} = D_t$

(b)

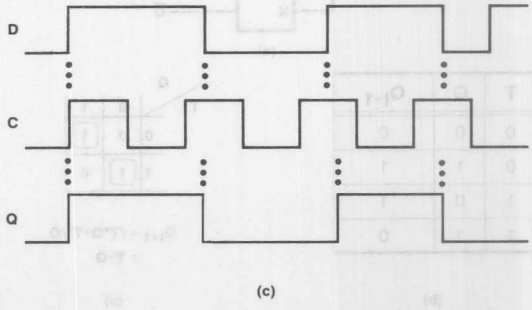
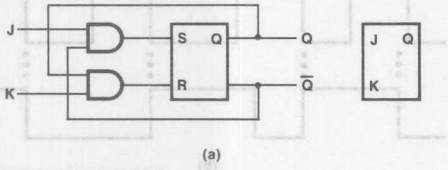


Figure 5-4. D Latch (a) Logic Circuit (b) State Table (c) Waveforms

5.2.3 JK Latch

Another useful latch is the JK latch which is shown in Figure 6-5. This latch consists of an RS latch with two AND gates in front of the inputs. This is most useful because JK latches act like RS latches, and it is permissible to apply '1' to both inputs simultaneously. The state table and characteristic equation for a JK latch is shown in Figure 5-5.



(a)

St	Rt	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(b)

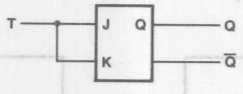
J, K \ Q	0	1
00	0	1
01	0	0
11	1	0
10	1	1

$Q_{t+1} = J^*/Q+/K^*Q$

Figure 5-5. JK Latch (a) Logic Circuit (b) State Table and Characteristic Equation

5.2.4 T Latch

Another type of latch is a *triggered latch* (T Latch), which has only one input called T. Whenever T is high, the latch changes state. T latch is realized by connecting both J and K to one input, T.



(a)

T	Q	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

(b)

T \ Q	0	1
0	0	1
1	1	0

$Q_{t+1} = /T^*Q+T^*/Q$
$= T+Q$

(c)

Figure 5-6. T Latch (a) Logic Circuit (b) State Table and Characteristic Equation

5.3 Flip-Flops

A *flip-flop* is also a bistable device — a circuit with only two stable states. There is one main difference between flip-flops and latches. Flip-flops do not have the transparency of the latches. Therefore a change in the input does not effect the output immediately. A change in the flip-flop is a result of a change to the control or an asynchronous input. The main advantage of the flip-flops over the latches is that it is possible to "read in" a new value to the flip-flop and read out an output at the same time. This property is not allowed in the latches because of the latch transparency.

A group of flip-flops form a register. A register is a digital device that is used to hold information. A register may be a combination of flip-flops and gates. The gates would control how and when the data from the flip-flops will be transferred.

In the previous sections, different types of latches were discussed. The same type of flip-flops are available (RS, JK, D and T flip-flops) and the state equations are the same with the difference that the output change in latches is realized on the same clock pulse and in the flip-flops on the next pulse.

Let's take a look at a D-latch and a D flip-flop and discuss the difference between them.

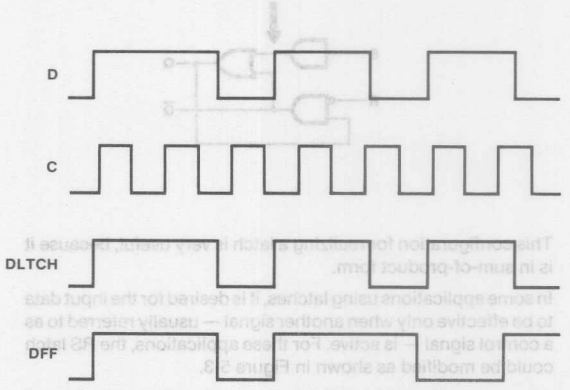


Figure 5-7. Comparison of a D-Latch and a D Flip-Flop

Examining the above waveforms, we could notice that the D latch and D flip-flop act the same except when the D input changes while C = 1. Because in the flip-flop at the edge of each clock pulse the input is seen and the flip-flop maintains the value till the next clock edge. But in latches the output follows the input while C = 1.

5.3.1 Characteristic Equations

The characteristic equations for various flip-flops are summarized as follows:

$Q_{t+1} = S+/R^*Qt$	RS flip-flop
$Q_{t+1} = J^*/Q_t+/K^*Q_t$	JK flip-flop
$Q_{t+1} = T:+:Q_t$	T flip-flop
$Q_{t+1} = Q_t$	D flip-flop

In the above equations Q_{t+1} is the next state and Q_t is the present state. Generally we could convert one flip-flop to the other by inserting some gates in front of the RS flip-flop.

5.4 Designing Sequential Circuits

As it was stated earlier, the states of a sequential circuit depend not only on the present states of its inputs, but also on the past history of them. A sequential circuit is constructed of flip-flops and gates. The gates construct the combinational part of a sequential circuit, and we could have any number of flip-flops that are needed. A general block diagram of a sequential circuit is shown in Figure 5-8.

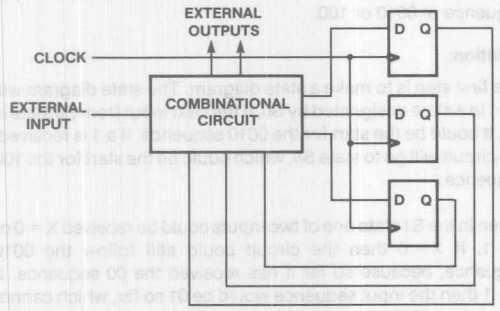


Figure 5-8. Sequential Circuit Block Diagram

The combinational section receives external binary inputs and feeds information to the flip-flops. The flip-flops have a feedback path to the combinational circuit. The circuit has external outputs. At the rising edge of each clock pulse the information from the combinational circuit is read into the flip-flops and the new outputs are generated. The outputs do not change until the next clock pulse.

A general block diagram of a sequential circuit has been reviewed. Now let's look at a more specific circuit. In Figure 6-9, an example of a sequential circuit is shown.

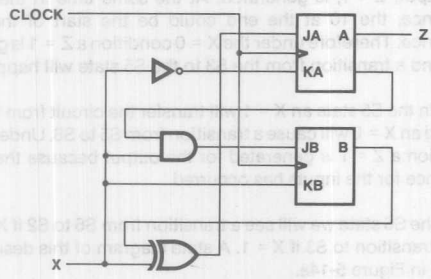


Figure 5-9. Example of a Sequential Circuit

This circuit consists of two JK flip-flops, an inverter, an AND gate and an XOR gate. It has an external binary input, X, and an external ouput, Z.

In this section, we try to familiarize ourselves with the analysis and synthesis of sequential circuits. Synthesis will be covered first, because it would make the analysis understanding easier.

5.4.1 Transition Tables

The states of a sequential circuit are determined by its inputs, the outputs and states of the flip-flops. In order to examine these states, we should determine the input equations to the flip-flops. Let's look at Figure 5-9. Using the characteristic equation for the JK flip-flop, the input equaiton for Figure 5-9 will be examined:

$J_A = X:+:B \qquad J_B = X^*/A \qquad Z = A$

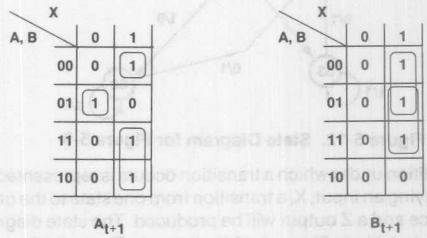
$K_A = /X \qquad K_B = X$

The next state equations are:

$A_{t+1} = J_A{}^*/Q+/K_A{}^*Q = (X:+:B)^*/A+X^*A$
$= (X^*/B+/X^*B)^*/A+X^*A$
$= /A^*/B^*X+/A^*B^*/X+B^*/X$

$B_{t+1} = J_B{}^*/Q+/K_B{}^*Q = /A^*/B^*X+B^*X$

the above are called the *state equations.* The corresponding K-maps are:



Using these maps, the *transition tables* for Figure 5-9 are derived. There are only four different combinations that A and B could have. The next state values are derived using these four possible combinations.

The present state is the state of the flip-flop before the clock pulse, the next state is the state of flip-flop after the clock pulse has been applied. The present output, Z, is the output of the sequential circuit after the clock pulse. As mentioned before, the circuit can have four states: AB = 00, 01, 11 and 10. At this point, we try to cover transitions for one input state. Suppose the circuit is in the 00 state. If an X = 0 input is applied, the next state will be 00. If an X = 1 is applied, the next state will be 11. the output for both cases is Z = 0.

PRESENT STATE	NEXT STATE A_{t+1} B_{t+1}		PRESENT OUTPUT
AB	X = 0	X = 1	Z
00	00	11	0
01	10	01	0
11	00	11	1
10	00	10	1

Figure 5-10. Transition Table for Figure 5-9

5.4.2 State Tables and State Diagrams

Examining the transition table, we notice that AB has four combinations. We could assign letters to these four combinations: $S_0 = 00, S_1 = 01, S_2 = 11$ and $S_3 = 10$. Using these assignments the transition table could be modified to the *state table* shown in Figure 5-11.

PRESENT STATE	NEXT STATE		PRESENT OUTPUT Z
	X = 0	X = 1	
S0	S0	S2	0
S1	S3	S1	0
S2	S0	S2	1
S3	S0	S3	1

Figure 5-11. State Table for Figure 5-9

A state diagram can be derived using the state table. A state diagram could show transitions between the states when a specific input is applied. Each state is represented by a circle and the transition between the state is shown by arrows.

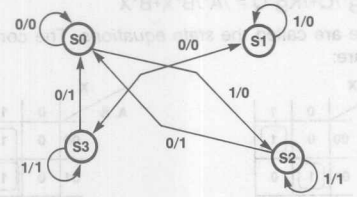


Figure 5-12. State Diagram for Figure 5-9

The condition under which a transition occurs is represented by X/Z. Applying an input, X, a transition from one state to the other takes place and a Z output will be produced. The state diagram for the state table in Figure 5-11 is shown in Figure 5-12.

We have gone through a complete analysis procedure for the previous example. This process could be summarized as follows:

1. Using a given network, determine the input equations.
2. Derive the next state equations, using the flip-flop characteristic equations:
   $Q_{t+1} = D$ D flip-flop
   $Q_{t+1} = T:+:Q$ T flip-flop
   $Q_{t+1} = J^*/Q+/K^*Q$ JK flip-flop
   $Q_{t+1} = S+/R^*Q$ RS flip-flop
3. Derive the corresponding K-maps, and transition tables.
4. Assigning states to the variables, make the state table.
5. From the state table, draw the state diagram.

The analysis of a sequential circuit could be easier to understand, because the same steps could be taken in reverse order. A flow chart of the procedure is shown below.

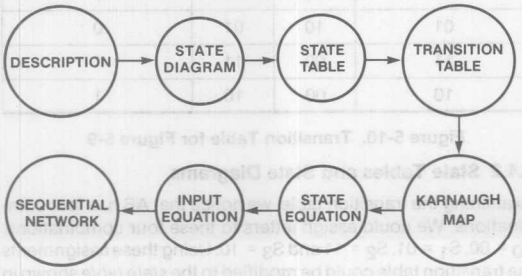


Figure 5-13. Flow Chart of Sequential Circuit Analysis

5.4.3 Design Examples

In this section, the procedure for designing sequential circuits will be shown in more detail by analyzing some design problems. The design steps will follow the flow table shown in the previous section.

Example 5.1:

Design a clocked sequential circuit which receives an input, X, and will produce an output, Z = 1, after it has received an input sequence of 0010 or 100.

Solution:

The first step is to make a state diagram. The state diagram will start in a state designated by S0, if the next input from S0 state is a 0, it could be the start for the 0010 sequence. If a 1 is received, the circuit will go to state S4, which could be the start for the 100 sequence.

When in the S1 state one of two inputs could be received X = 0 or X = 1. If X = 0 then the circuit could still follow the 0010 sequence, because so far it has received the 00 sequence. If X = 1 then the input sequence would be 01 so far, which cannot follow the 0010 any more but it could be a start for the 100 sequence, therefore under this condition the circuit would have a transition from state S1 to S4.

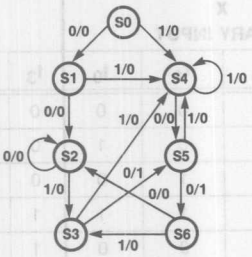
While in the S4 state, if an X = 1 is received it would stay in the same state, because a sequence of 11 could still follow the 100 pattern. If X = 0 is received the 100 pattern could continue, because 10 follows the 100 pattern.

When in the S2 state if an X = 0 is received, it will stay in the S2 state and if X = 1 then it will have a transition from the S2 to S3 state. While in the S3 state, we will have a transition to state S4 if X = 1, because the input sequence would be 0011 which cannot follow the 0010 sequence any more, but the 1 at the end could be a beginning for the 100 sequence which is the same state as the S4. If X = 0 is received then the 0010 sequence is complete and an output, Z = 1, is generated. At the same time in the 0010 sequence, the 10 at the end could be the start of the 100 sequence. Therefore under the X = 0 condition a Z = 1 is generated and a transition from the S3 to the S5 state will happen.

While in the S5 state an X = 1 will transfer the circuit from S5 to S4, and an X = 0 will cause a transition from S5 to S6. Under this condition a Z = 1 is generated for the output because the 100 sequence for the inputs has occurred.

From the S6 state we will see a transition from S6 to S2 if X = 0, and a transition to S3 if X = 1. A state diagram of this design is shown in Figure 5-14a.

By analyzing the state diagram, a state table is generated (Figure 5-14b). A careful look at the state table will show that the S2 and the S6 states are equivalent, because under X = 0 they both have a transition to state S2 with Z = 0, and under X = 1 they would transfer to S3 with Z = 0. So the state table could be summarized to Figure 5-14c. After summarizing the state table, there will be a total of six states left, therefore at least three variables will be needed for the state assignments ($2^3 = 8$). In this case only six assignments will be used. The state variables are designated by A, B and C. The state assignments are illustrated in Figure 5-14d.



(a)

PRESENT STATE	NEXT STATE	
	X = 0	X = 1
S0	S1,0	S4,0
S1	S2,0	S4,0
S2	S2,0	S3,0
S3	S5,1	S4,0
S4	S5,0	S4,0
S5	S6,1	S4,0
S6	S2,0	S3,0

(b)

PRESENT STATE	NEXT STATE	
	X = 0	X = 1
S0	S1,0	S4,0
S1	S2,0	S4,0
S2	S2,0	S3,0
S3	S5,1	S4,0
S4	S5,0	S4,0
S5	S2,1	S3,0

(c)

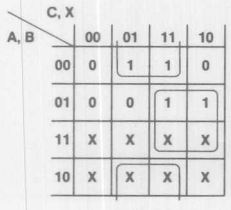
J K
0 → 0 0 X
0 → 1 1 X
1 → 0 X 1
1 → 1 X 0

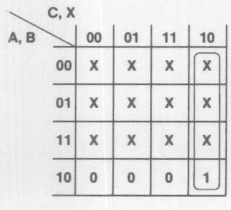
$(ABC)_t$	$(ABC)_{t+1}$		Z	
	X = 0	X = 1	X = 0	X = 1
000	001	100	0	0
001	010	100	0	0
010	010	011	0	0
011	101	100	1	0
100	101	100	0	0
101	010	100	1	0

(d)

Figure 5-14. Example 5.1 (a) State Diagram (b, c) State Tables (d) Transition Table

From the Assignment table, the K-maps for each state are derived. In this design the circuit is realized by JK flip-flops, therefore two K-maps are needed for each state variable (shown in Figure 5-15).





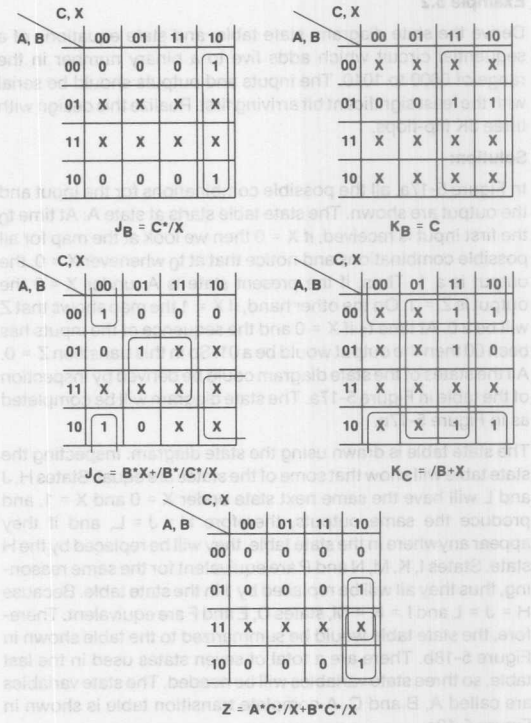


Figure 5-15. Karnaugh Maps for Example 5.1

State equations are derived using the K-maps. Using the state equations, the circuit diagram for the design would be obtained as shown in Figure 5-16. The state equations are summarized as follows.

$J_A = /B^*X+B^*C$ $J_B = C^*/X$ $J_C = B^*X+A^*/X$

$K_A = /X^*C$ $K_B = C$ $K_C = /B+X$

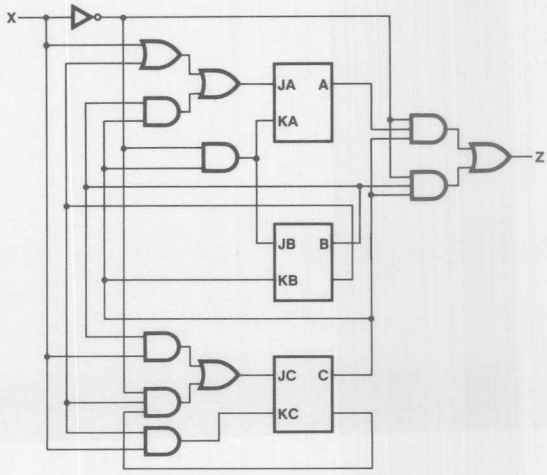


Figure 5-16. Circuit Diagram for Example 5.1

Example 5.2

Derive the state diagram, state table, and state equations of a sequential circuit which adds five to a binary number in the range of 0000 to 1010. The inputs and outputs should be serial with the least significant bit arriving first. Realize this design with three JK flip-flops.

Solution:

In Figure 5-17a, all the possible combinations for the input and the output are shown. The state table starts at state A. At time t_0 the first input is received, if X = 0 then we look at the map for all possible combinations and notice that at t_0 whenever X = 0, the output is a 1. Thus, if the present state is A, under X = 0 the output is Z = 1. On the other hand, if X = 1 the map shows that Z will be a 0. At time t_1 if X = 0 and the sequence of the inputs has been 00 then the output would be a 01. So in this transition Z = 0. All the states of the state diagram could be derived by inspection of the table in Figure 5-17a. The state diagram will be completed as in Figure 5-17b.

The state table is drawn using the state diagram. Inspecting the state table will show that some of the states are equal. States H, J and L will have the same next state under X = 0 and X = 1, and produce the same outputs, therefore H = J = L, and if they appear anywhere in the state table, they will be replaced by the H state. States I, K, M, N and P are equivalent for the same reasoning, thus they all will be replaced by 1 in the state table. Because H = J = L and I = K = M, states D, E and F are equivalent. Therefore, the state table would be summarized to the table shown in Figure 5-18b. There are a total of seven states used in the last table, so three state variables will be needed. The state variables are called A, B and C. A complete transition table is shown in Figure 5-18c.

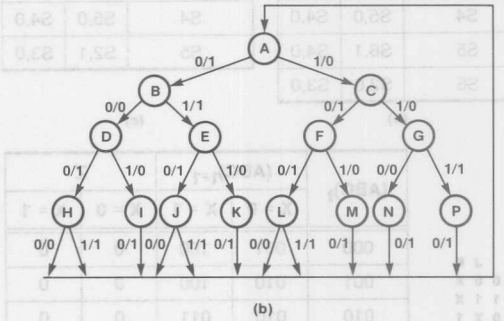
The K-maps are drawn, using the state table. The state equations are calculated using the K-maps.

JA = B*X+B*C KA = B
JB = /A*C+A*/C KB = A+/C*X+C*/X
JC = /A*/X KC = A+B*X

Z = B*/X+A*X+/A*/C*X+/B*C*X

X BINARY INPUT				Z OUTPUT			
t_3	t_2	t_1	t_0	t_3	t_2	t_1	t_0
0	0	0	0	0	1	0	1
0	0	0	1	0	1	1	0
0	0	1	0	0	1	1	1
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	1	1

(a)



(b)

Figure 5-17. Example 5.2 (a) Truth Table (b) State Diagram

PRESENT STATE	NEXT STATE		Z	
	X = 0	X = 1	X = 0	X = 1
A	B	A	1	0
B	D	E	0	1
C	F	G	1	0
D	H	I	1	0
E	J	K	1	0
F	L	M	1	0
G	N	P	0	1
H	A	A	0	1
I	A	—	1	—
J	A	A	0	1
K	A	—	1	—
L	A	A	0	1
M	A	—	1	—
N	A	—	1	—
P	A	—	1	—

(a)

PRESENT STATE	NEXT STATE		Z	
	X = 0	X = 1	X = 0	X = 1
A	B	A	1	0
B	D	D	0	1
C	F	G	1	0
D	H	I	1	0
G	I	I	0	1
H	A	A	0	1
I	A	—	1	-

(b)

PRESENT STATE	NEXT STATE		Z	
	X = 0	X = 1	X = 0	X = 1
000	001	000	1	0
001	011	011	0	1
010	011	100	1	0
011	101	110	1	0
100	110	110	0	1
101	000	000	0	1
110	000	—	1	—

(c)

Figure 5-18. Example 5.2 (a) State Table (b) Reduced State Table (c) Transition Table

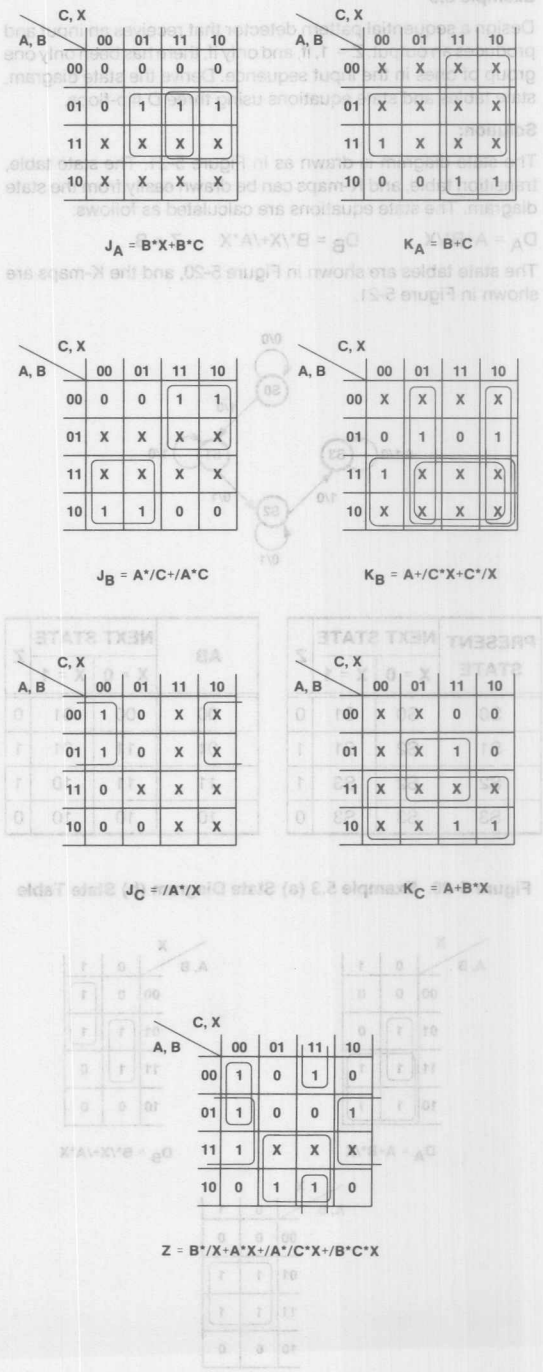


Figure 5-19. Karnaugh Maps for Example 5.2

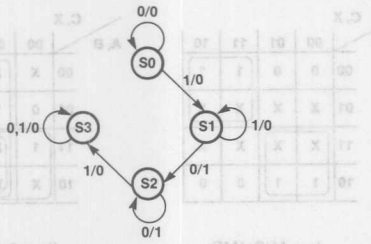
Design a sequential pattern detector that receives an input and produces an output, Z = 1, if, and only if, there has been only one group of ones in the input sequence. Derive the state diagram, state tables and state equations using three D flip-flops.

Solution:

The state diagram is drawn as in Figure 5-21. The state table, transition table, and K-maps can be drawn easily from the state diagram. The state equations are calculated as follows:

$D_A = A+B^*/X \qquad D_B = B^*/X+/A^*X \qquad Z = B$

The state tables are shown in Figure 5-20, and the K-maps are shown in Figure 5-21.



PRESENT STATE	NEXT STATE		Z
	X = 0	X = 1	
S0	S0	S1	0
S1	S2	S1	1
S2	S2	S3	1
S3	S3	S3	0

AB	NEXT STATE		Z
	X = 0	X = 1	
00	00	01	0
01	11	01	1
11	11	10	1
10	10	10	0

Figure 5-20. Example 5.3 (a) State Diagram (b) State Table

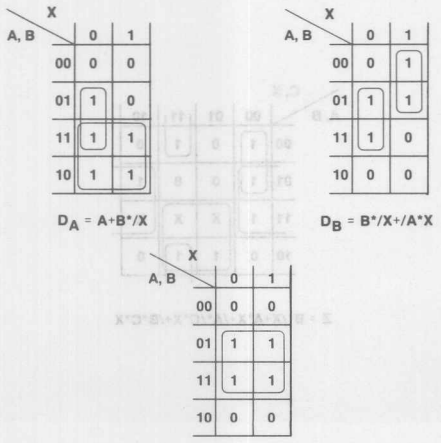


Figure 5-21. Karnaugh Maps for Example 5.3

Counters are among the most commonly used sequential circuits. In the following sections, they are covered in detail.

A register that goes through a predetermined state upon receiving an input pulse is called a counter. Counters are one of the simplest sequential circuits, and are found in almost all equipment containing digital logic. According to what sequence of logic a counter follows, we will have different types of counters: BCD, binary, etc.

The binary counter is one of the simplest counters. An n-bit binary counter is a register with n flip-flops and associated combinational logic that follows the sequence of n-bits from $0\text{-}2^{n-1}$.

Example 5.4

Design a 3-bit up/down binary counter. There are three outputs from the binary counter: DA, DB and DC. The input to the counter is X, the counter will increment if X=1, and decrement if X = 0. Design this circuit using three D flip-flops.

Solution:

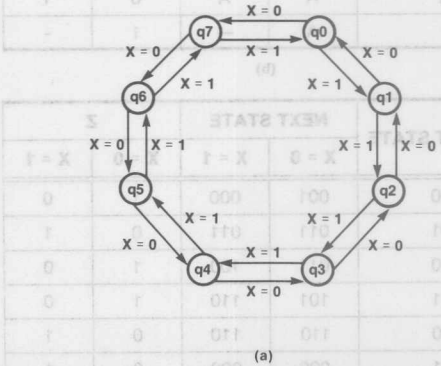
This circuit would have eight different states, because the 3-bit counter goes through eight different states. The states are called q0, q1, q2, q3, q4, q5, q6 and q7. If X = 1 each state will have a transition to its next higher state. For example q0 will go to q1, q1 will go to q2 and so on. If X = 0 then each state will change to its previous state. For example, q7 will go to q6, q5 to q4 and so on.

The state diagram of this design is shown in Figure 5-22a. The state, and transition tables which are derived from the diagram are shown in Figure 5-22b. The K-maps and the state variables are shown in Figure 5-22c. The state equations are calculated using the K-maps. The equations are summarized as follows:

$D_A = /A^*/B^*/C^*/X+/A^*B^*C^*X+A^*B^*/C^*X+A^*/B^*X+A^*C^*/X$

$D_B = /B^*/C^*/X+/B^*C^*X+B^*/C^*X+B^*C^*/X$

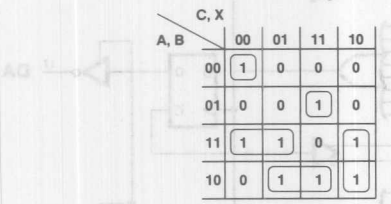
$D_C = /C$



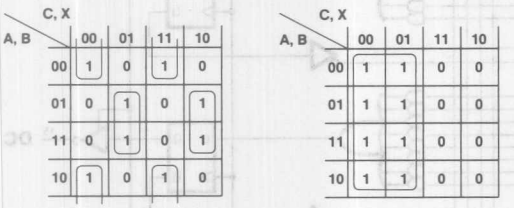
PRESENT STATE	NEXT STATE	
	X = 0	X = 1
q0	q7	q1
q1	q0	q2
q2	q1	q3
q3	q2	q4
q4	q3	q5
q5	q4	q6
q6	q5	q7
q7	q6	q0

PRESENT ABC	NEXT STATE	
	X = 0	X = 1
000	111	001
001	000	010
010	001	011
011	010	100
100	011	101
101	100	110
110	101	111
111	110	000

(b)



D_A = /A*/B*/C*/X+/A*B*C*X
+ A*B*/C+A*/B*X+A*C*/X



D_B = /B*/C*/X+/B*C*X
+ B*C*/X

D_C = /C

(c)

Figure 5-22. 3-Bit Up/Down Counter (a) State Diagram (b) State Tables (c) Karnaugh Maps

Example 5.5

Implement the 3-bit up/down counter of the Example 5.4 using the PAL device most suitable for the design.

Solution:

The 3-bit binary counter designed in the previous example will require three flip-flops. The PAL device for this design will require at least three flip-flops. Since there is no PAL device available that has only three flip-flops, the best PAL device will be the PAL16R4 because it has four flip-flops and contains the AND-OR gates needed to realize the combinational circuit of the counter. The schematic of the 3-bit up/down counter using the PAL16R4 is shown in Figure 5-23.

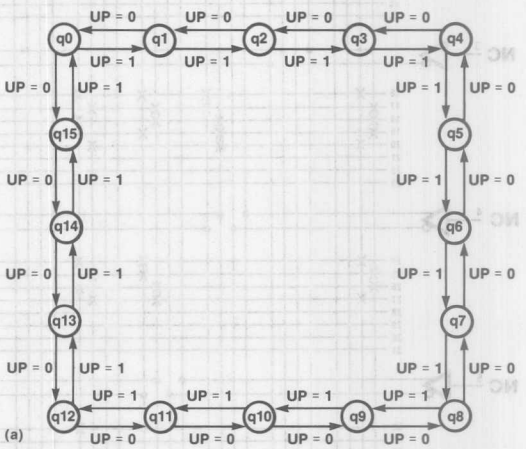
Design of a binary counter could be complicated by adding new features to it. A counter can be made to count up or down, load new values in, or clear the present state of the counter and reset to 0's. These features are discussed in the following example.

Example 5.6:

Design a 4-bit up/down counter that receives an input, X, if X = 1 the counter counts up and if X = 0 it counts down. The load feature loads new values into the output registers. The clear operation clears the output registers and resets them to all lows (zeros). Design this counter using D-type flip-flops and realize the circuit with the proper PAL device.

Solution:

First we design a 4-bit up/down binary counter, then we design the new function into it. A binary counter could have sixteen different possible states. The procedure of making the state diagram, state table, and K-maps is similar to the design of the 3-bit counter. The corresponding drawings are shown in Figures 5-24 and 5-25.



(a)

PRESENT STATE	NEXT STATE	
	X = 0	X = 1
q0	q15	q1
q1	q14	q2
q2	q13	q3
q3	q12	q4
q4	q11	q5
q5	q10	q6
q6	q9	q7
q7	q8	q8
q8	q7	q9
q9	q6	q10
q10	q5	q11
q11	q4	q12
q12	q3	q13
q13	q2	q14
q14	q1	q15
q15	q0	q0

PRESENT STATE	NEXT STATE	
	X = 0	X = 1
0000	1111	0001
0001	1110	0010
0010	1101	0011
0011	1100	0100
0100	1011	0101
0101	1010	0110
0110	1001	0111
0111	1000	1000
1000	0111	1001
1001	0110	1010
1010	0101	1011
1011	0100	1100
1100	0011	1101
1101	0010	1110
1110	0001	1111
1111	0000	0000

(b)

Figure 5-23. 5-Bit Counter (a) State Diagram (b) State Tables

Logic Diagram

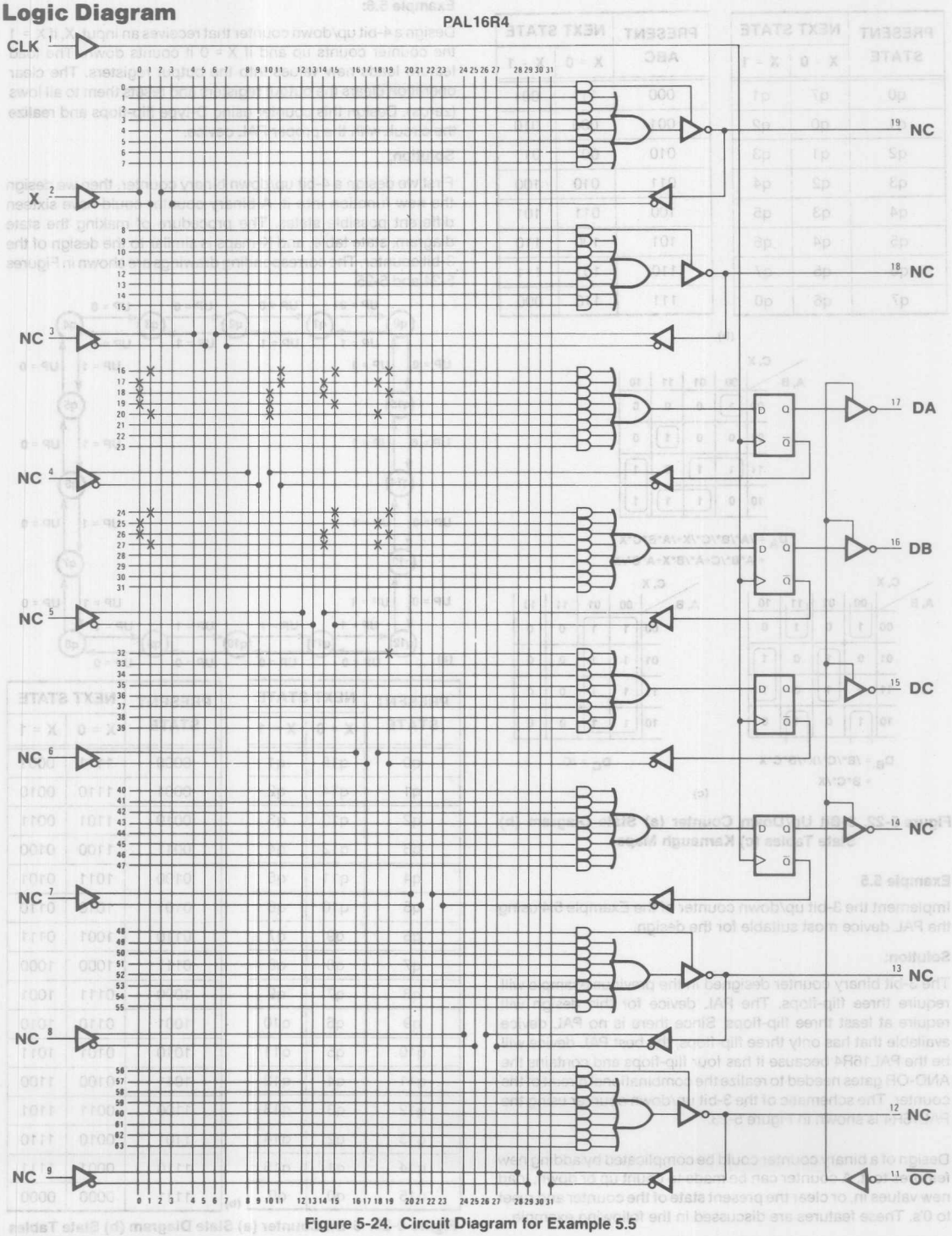


Figure 5-24. Circuit Diagram for Example 5.5

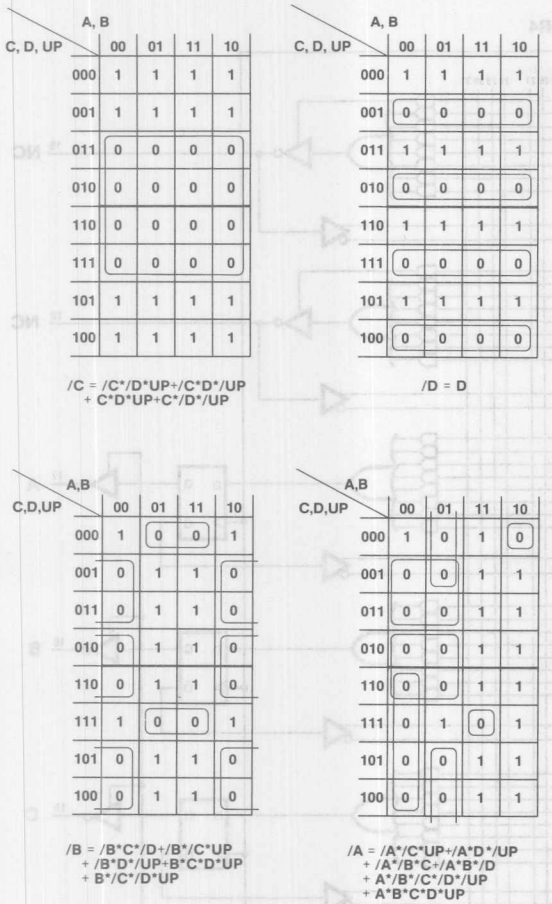


Figure 5-25. Karnaugh Maps for 4-Bit Counter

As you might have noticed, the equations are derived for the low outputs because many PAL devices have only low outputs. The equations derived so far are written just for an up/down counter. To implement the clear function, the signal should be ANDed with the other terms in the equations for A, B, C and D, and a CLEAR term should be ORed with each equation. The LOAD function should be ORed with all the terms to make them lows and another term should be added to each equation that consists of LOAD* (new value). The modified and final equations are listed below:

```
/A = /LOAD*/CLR*/A*  /C*   UP
   + /LOAD*/CLR*/A*      D*/UP
   + /LOAD*/CLR*/A*/B* C
   + /LOAD*/CLR*/A* B*   /D
   + /LOAD*/CLR* A*/B*/C*/D*/UP
   + /LOAD*/CLR* A* B* C* D* UP
   +  LOAD*/CLR* AI
   + CLR

/B = /LOAD*/CLR*/B* C*/D
   + /LOAD*/CLR*/B*/C*   UP
   + /LOAD*/CLR*/B*    D*/UP
   + /LOAD*/CLR* B* C* D* UP
   + /LOAD*/CLR* B*/C*/D*/UP
   +  LOAD*/CLR*/BI
   + CLR

/C = /LOAD*/CLR*/C*/D* UP
   + /LOAD*/CLR*/C* D*/UP
   + /LOAD*/CLR* C* D* UP
   + /LOAD*/CLR* C*/D*/UP
   +  LOAD*/CLR*/CI
   + CLR

/D = /LOAD*/CLR* D
   +  LOAD*/CLR*/DI
   + CLR
```

Using the above equations and PAL16R4, the schematic for the 4-bit counter could be generated. This design is shown in Figure 5-26.

As we try to design bigger counters, the design of them using the state tables and K-maps gets more diffucult. In the design of the 4-bit counter, K-maps were used. If we try to design a counter bigger than this summarizing the equations will be very tough to do. Therefore, we try to find a general solution for solving the counter design problems. Let's try to write the equation for the most significant bit (MSB) of an n-bit binary counter (Q_n).

Let's look at the case where the counter is counting up. The new value of Q_n will depend on the carry-in from bit Q_{n-1} into Q_n. If all least significant bits (LSBs) are high when we count up, we will have a carry-in from Q_{n-1} into Q_n.

$$C_{IN} = Q_{n-1}{}^*Q_{n-2}{}^*...Q1{}^*Q0{}^*UP$$

Now let's look at the following table:

UP	Qn	CARRY INTO Qn	NEW Qn
H	L	L	L
H	L	H	H
H	H	L	H
H	H	H	L

Carry-in Table

Examining the above table, it is easily concluded that:

$Q_n := Q_n$:+: carry-into Q_n
$Q_n := Q_n$:+:($Q_{n-1}{}^*Q_{n-2}{}^*...Q0{}^*UP$)

Now that we have calculated the equation for the count-up case, let's look at the count down. Carry-in from Q_{n-1} into Q_n will be high if all the LSBs are low and we are counting up.

Borrow in Qn := /Qn-1*/Qn-2*.../Q0*/UP

Let's look at the following table:

UP	Qn	CARRY INTO Qn	NEW Qn
L	L	L	L
L	L	H	H
L	H	L	H
L	H	H	L

Borrow-in Table

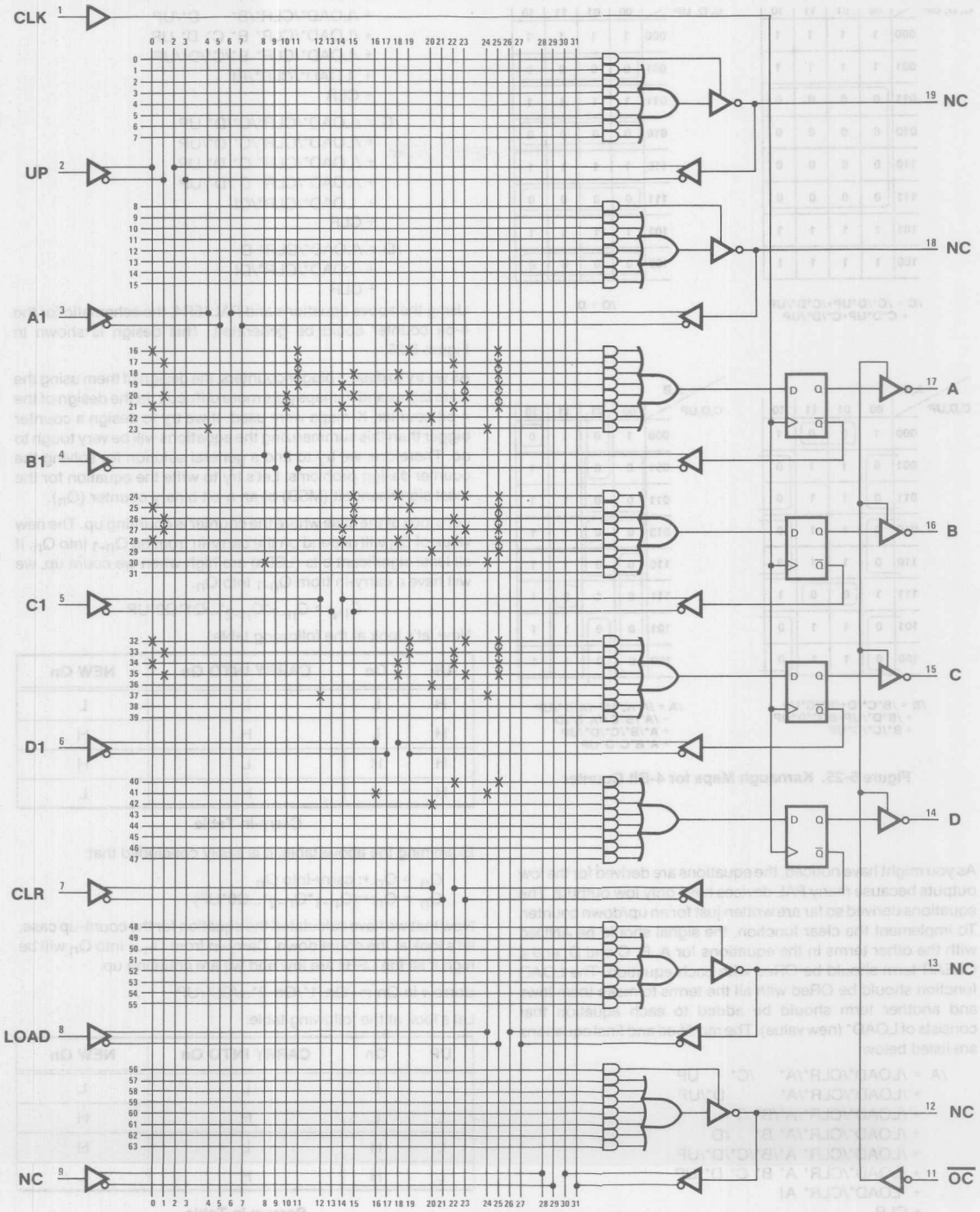


Figure 5-26. PAL Device Schematic for 4-Bit Counter

So:

$Q_n = Q_n$:+: Carry-into Q_n
$Q_n = Q_n$:+:(/Q_{n-1}*/Q_{n-2}*.../Q1*/Q0*/UP)

Therefore:

$Q_n = Q_n$:+:(Q_{n-1}*Q_{n-2}*...Q1*Q0*UP)
+ Q_n:+:(/Q_{n-1}*/Q_{n-2}*.../Q1*/Q0/*UP)

Let's try to summarize the above equaiton. For simplicity, let's give shorter names to different parts of the above equation.

A = Q_n
B = Q_{n-1}*Q_{n-2}*...Q1*Q0*UP
C = /Q_{n-1}*/Q_{n-2}*.../Q1*/Q0*/UP

Therefore the equation is expressed as:

A = A:+:B
+ A:+:C

B and C are exclusive from each other, so the above equation could be summarized to:

A = A:+:B+C

then:

$Q_n = Q_n$:+:(Q_{n-1}*Q_{n-2}*...Q1*Q0*UP) Eq.5.1
+ (/Q_{n-1}*/Q_{n-2}*.../Q1*/Q0*/UP)

Using the solution discussed above, let's try to solve a design problem.

Example 5.7:

Design an n-bit counter that can count up, count down, SET and LOAD new values into the counter. SET overrides LOAD, count and hold. LOAD overrides count. Count is conditional on carry in, otherwise it holds.

Solution:

The above operations are exercised in the function table and summarized in the operations table.

SET	LD	CIN	UP	D	Q	OPERATION
H	X	X	X	X	X	Set all HIGH
L	L	X	X	D	D	Load D
L	H	H	X	X	Q	Hold Q
L	H	L	L	X	Q plus 1	Increment
L	L	L	H	X	Q minus 1	Decrement

Now using the operations table, we could start writing the equations for our counter. We will try to generate a general equation that can be used for any bit in the counter. If we take the nth bit, we have D_n as input and Q_n as output. There are four operations that can happen for any given bit: LOAD, HOLD, SET and count up or count down. We load the new value in the counter's register if SET is OFF (/SET = H). So Q_n is replaced by new D_n value if (/SET = H, LOAD = H), the expression that will allow loading the new value will be /SET*LOAD*/D_n. In order to be able to HOLD the Q_n value, we should have the following conditions: (SET = L, LOAD = L, C_{IN} = L). So the expression for holding the old value is:

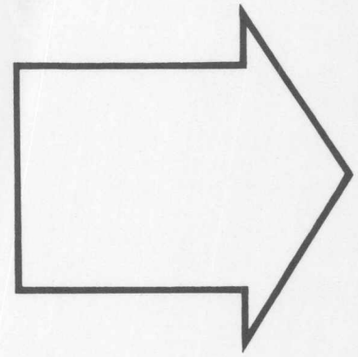
/SET*/LOAD*/C_{IN}*Q_n

There are two more functions for the counter: count up and count down. These two functions have been calculated in Eq. 6.1. Using this equation and the calculation for the HOLD and LOAD cases, the final equation for the nth bit is calculated.

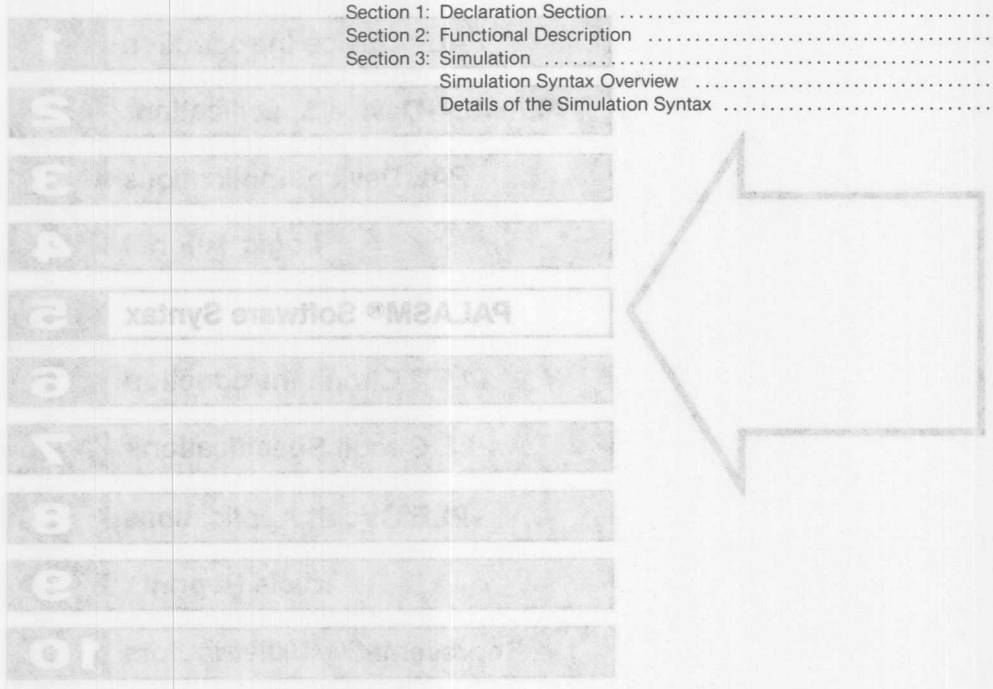
Q_n = /SET*LOAD*D_n Eq. 5.2
+ /SET*/LOAD*/Q_n
:+: /SET*/LOAD*C_{IN}*UP*Q0*Q1*...Q_{n-1}
+ /SET*/LOAD*C_{IN}*/UP*/Q0*/Q1*.../Q_{n-1}

Using the above general equation, any large counter could be designed.


IMPORTANT
NOTES



Contents Section 5



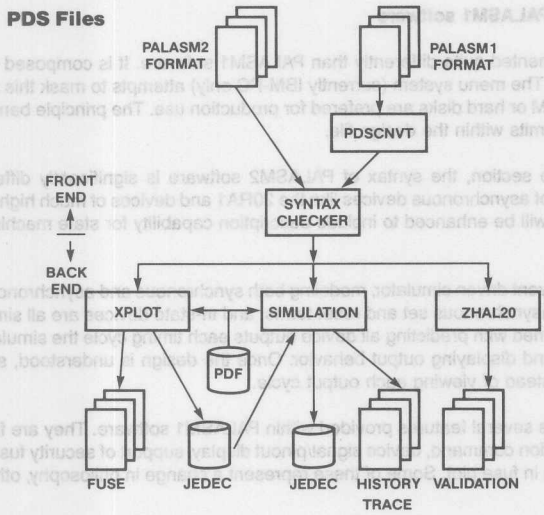
PALASM2 Software

Introduction

PALASM2 software is a high-level language to describe implementation and simulation of logic designs. For every logic application a PAL device Design Specification (PDS) can be created using PALASM2 software syntax. This Pal device Design Specification consists of a set of functional equations and simulation commands. The functional equations describe how the logic is to be implemented on PAL device and the simulation commands describe how the logic should behave after implementation. The detail of the syntax of the language will be described in later sections.

The PAL device Design Specification is processed by the PALASM2 software compiler at the front end and an intermediate file is created. This intermediate file is then processed by various modules at the back end to accomplish various functions. At present MMI has developed an XPLOT generator which can generate the fuse patterns and the standard JEDEC file from PDS which can be downloaded to various programmers to program a PAL device.

Also available is a SIMULATOR package which will exercise each simulation command in the PDS against the functional description to check for the functional validity of the design. The output of the SIMULATOR is a set of vectors which can be physically exercised against the programmed part. (At date of publication the SIMULATOR is an ALPNA version, check with your local FAE for current availability.)



PALASM2 Software Flow Chart

Key Features:

—Assembles PAL device Design Specifications.
—Simulates and verifies the logic behavior.
—Generates PAL device fuse patterns in JEDEC format.
—Reports errors in syntax, assembly and simulation.

—New structured logic description syntax models more complex PAL devices and permits compaction of logic descriptions.
—String substitution allows concise mnemonic names for long frequently used logic expressions.
—Powerful functional simulator accurately models asynchronous and mixed PAL architectures.
—Structured high level language for describing test procedures, instead of long truth tables.

PALASM2 software supports PAL device assembly and simulation using these PAL devices:

20-pin Devices:

10H8	10L8	10P8	
12H6	12L6	12P6	
14H4	14L4	14P4	
16H2	16L2	16P2	
16L8	16P8	16C1	
16R4,6,8	16RP4,6,8		ZHAL20

24-pin Devices:

6L16	8L14		
12L10	12P10	14L8	14P8
16L6	16P6	18L4	18P4
20P2	20L2	20C1	
20L8,10	20X4,8,10	20R4,6,8	
20RA1	20S1	20P8E	20RS4,8,10

MegaPAL:

32R16	64R32

NOTE: PALASM2 software does NOT support 16A4 and 16X4 PAL devices.

Functional differences from PALASM1 software

PALASM2 software is implemented quite differently than PALASM1 software. It is composed of several interacting programs coupled by disk files. The menu system (currently IBM-PC only) attempts to mask this somewhat, but floppy based files slow interaction. RAM or hard disks are prefered for production use. The principle benefit of the reorganization is the freedom from fixed limits within the design file.

As mentioned, in the feature section, the syntax of PALASM2 software is significantly different from PALASM1 sofstware. It allows description of asynchronous devices like the 20RA1 and devices of much higher complexity, like the MegaPAL devices. The syntax will be enhanced to include description capability for state machines and bus oriented logic elements.

The logic simulator is a true event driven simulator, modeling both synchronous and asynchronous events accurately. Cross-coupled logic functions, asynchronous set and reset terms, and tri-state devices are all simulated correctly. The user need not be initially concerned with predicting all device outputs each timing cycle the simulator may be operated as a logic analyzer, capturing and displaying output behavior. Once the design is understood, specific output values may be sampled selectively, instead of viewing each output cycle.

PALASM2 software also omits several features provided within PALASM1 software. They are fault coverage prediction for test vectors, documentation command, device signal/pinout display, support of security fuse and printing of logic equations for each product term in fuse plot. Some of these represent a change in philosophy, others will be provide in later versions of the program.

Structure of PAL Device Design Specifications

This section gives an overview of the PAL device Design Specification (PDS). It explains the three sections of the PDS with regard to the layout of each section, the information to be provided in each section, and what function it serves in the design process. The detailed syntax and contents of each section will be discussed in the succeeding sections. Examples will be used to illustrate correct syntax along with common misconceptions.

1 : Declaration Section
- —Customer Profile Information
- —Pin list Information

2 : Functional Description
- —Reserved Words
- —Basic Operation
- —String Substitution
- —Combinatorial Equations
- —Registered Equations
- —Equations for special functions.
- —Boolean operators

3 : Simulation Description
- —Simulation Directives
- —Structured Control Constructs

5

This section is used to record information about the customer necessary for documentation and future reference the type of PAL device used, the signal names given to each pin of the device, and some predefined states and strings to be used in the design. This section must come before all the other sections of the PDS. Any signal name that is going to be used in the equations has to be declared first in this section. There is a customer profile information section for documentation and future reference purposes.

Comments may be inserted freely, and begin with a semi-colon character—";". Any of the items described may be in lower or upper case all items are case-folded before processing. Line length maximum is 13 characters all beyond that are ignored. All control characters (including tabs) are treated as a single space. The maximum length of any legal statement or logic equation is 100 tokens (a token is either a pin name, a simulation command, a keyword or a logical operator).

The declaration section has two parts. They are:

—Customer Profile Information.
—Pin list information.

Example

Customer Profile	TITLE	This is an example to illustrate the syntax
	PATTERN	ABC1234—MMI
	REVISION	000-ABC1234
	AUTHOR	Imtiyaz Bengali
	COMPANY	Monolithic Memories Inc.
	DATE	Dec. 5, 1984
	CHIP	example—only 20RS10
Pin List	CLK ONE	/TWO THREE SET RESET NC NC NC WRITE READ GND
	/OE	OUT1 OUT2 NC NC NC /ACK /MMI NC NC MEMADR VCC

Customer Profile Information SYNTAX

;	keyword	: optional data
	TITLE	<title of the design>
	PATTERN	<pattern identification>
	REVISION	<revision identification>
	AUTHOR	<designer's name>
	COMPANY	<company name>
	DATE	<date of creation>

NOTE:

—The keywords should appear in the format described in the table above one to a line. You may put optional data on the lines following the keywords it will be retained to annotate your fuseplot and simulation files. Warning messages will be generated if the keywords are omitted you may still generate fuseplot and simulation results.

—Each item can be any alphanumeric text including spaces, tabs and underscores. Only the first 2 characters are used additional characters present are ignored. Do not use any of these special characters:

!@#$%^&*()-=+{}[]":;'<>?,.

(The software does not specifically check for usage of these characters within these lines.)

CHIP <chip name> <PAL type> <pin list>

CHIP

CHIP is the keyword which is neccessary to start the pin list information.

chip name

<chip name> can be any alphanumeric word (except a reserved word or a PAL type) and is a required identifier. It should follow the same conventions for syntax as a pin name.

The <chip name> name has to be provided before the <PAL type> field is specified.

PAL type

<PAL type> is the part number of any supported PAL device manufactured by MMI.

All PAL devices with different speed/power options are given the same generic name.

For example PAL16R8, PAL16R8A, PAL16R8A2, PAL16R8A4, and PAL16R8B all have the same generic name PAL16R8.

Pin list

<pin list> is a list of signal names to be assigned to the pins of the device.

The length of signal names cannot exceed 14 characters. At least one of these characters must be a letter the remainder may be letters, numbers, or underscores.

For example: 1, 2, 3, ... is an illegal pin list, but p1, p2, p3, ... or 1p, 2p, 3p, ... are legal pin lists.

The signal can be specified as active-low or active-high. Active low signals are preceded by (/A is an active-low signal).

Signal names are separated by spaces or commas.

Special pins of the device (like the power and ground pins) are assigned special names. The power pin is assigned VCC and the ground pin is assigned GND. These names should come at the appropriate places in the pin list. For example in PAL20R pin number 2 is VCC and pin number 1 is GND. If any pin is not used, it must be specified as NC (noconnect).

Pins are listed in the order expected for DIP (dual inline package), reguardless of whether the user is planning to eventually program DIP, LCC or Chip Carrier devices. Any pin reording for other packages must be done by the programmer or other special fixture. The PAL64R3 device pinout is specified for 84 pin package.

Symbols not to be used in defining signal names in pin list:

ANY NON ALPHA NUMERIC CHARACTERS !@#$%^&*()-=+{}[]":;'<>?,. (/ can be used for active low signals)

Only numbers, letters and the underscore character may be used to create a pin name.
Do not use reserved words or PAL device types as signal name.

Section 2: Functional Description

All the implementation details of an application are given in this part of the PDS using boolean equations. The information provided in this section is used to generate the locations of the fuses to be blown during programming of the part. Depending on the type of outputs, one can use a combinatorial equation identified by '= or a registered equation identified by ':='. Also many outputs, such as the PAL20RA10, have special programmable functions associated with them functional equations can be used with these. The following lists different ways of specifying a function:

SYNTAX

```
EQUATIONS   ;  keyword marking the begining of ; functional description.
<signal>          =  Function ( <signal>, <operator>)
<signal>          :=  Function (<signal>,<operator>)
<signal>.<sfunc>  =  Function (<signal>,<operator>)

<signal>    is the name of a pin from the pin list
<operator>  /, *, +, :+:
<sfunc> is a special function associated with the output signal.
```

Certain PAL devices such as the PAL20RA1 have programmable functions for registers the Clock function (CLKF), the Set function (SETF), the Reset function (RSTF) and the Tristate function (TRST). These functions are represented by special equations using the keyword of the special function to suffex the signal name, for example:

```
out.CLKF = A * B
```

This means that product term A B controls the Clock function of the output OUT.

Reserved Words

```
TITLE    PATTERN    REVISION    AUTHOR    COMPANY    DATE
CHIP     EQUATIONS  SIMULATION  STRING DO
SETF     RSTF   CLKF    TRST    PRLD    CLOCKF    GENERATE    WHILE    FOR    IF
THEN     ELSE   BEGIN   END     VCC     GND     NC
```

Note: All PAL types are also reserved words.

Basic Operators

The basic operators are defined to perform INVERT, AND, OR, and EXCLUSIVE-OR operations. These basic operators can be used to describe any logic function on the right side of the equation.

/ —INVERT operator
This operation is used whenever a signal has to be inverted. It preceeds the signal to be inverted. For example: /A means "not a"

* —AND operator
This operation is used when ANDing two or more boolean variables. The operation of ANDing of all the signals result in a product term. For example A /B C means "A AND (NOT B) AND C"

\+ —OR operator
This operator is used when ORing two or more product terms and/or signals. For example: A + /C means "A OR (NOT C)"

:+: —EXCLUSIVE-OR operator
This operator is used when EXCLUSIVE-ORing two or more product terms and/or signals. For example: A :+: E means "A EXCLUSIVE-OR E"

With these basic logical operators, one can write logic equations for almost any application. The most useful form of the equations will be in the sum-of-products form it is the only currently supported form. True logic expansion and simplification will be supported in future versions of the PALASM software system. The following sections will illustrate the use of all these operators, except for the XOR. Operator precedence is in the order listed /,*,+,:+

String Substitution

In many applications it is possible that a certain expression or part of it is repeated many times. To eliminate repetition of typing the same text again, one can declare that text with a shorter word using string substitution. Then instead of the full text one can use the word used to identify the text. String substitution is textual replacement and the compiler does not try to find any logical meaning to it. Hence the user should be very specific in what he wants to substitute.

STRING <string name> '<text to be substituted>

STRING is a keyword and every string substitution should start with the key word STRING.

<string name> is any user defined name of up to I alphanumeric characters. The name has to be unique. This means that the name should not be a reserved word, one of signal names defined in the pin list, or one of the string names used elsewhere.

<text to be substituted> is any legal expression and should be specified within single quotes. The length of the text to be substituted is not limited to one line it may be made up of several lines. There is no fixed length to any single string memory is allocated as necessary to store each. Each part of it must follow the syntax rules of the pin name identifiers and be delimited by blanks or tabs.

The compiler does one to one substitution of the string name with the text to be substituted. It does not try to find the meaning of an expression after substitution.

Currently a maximum of 2 unique strings may be used within any design file.

```
Ex. 1:  STRING LOAD      '  LD * /CIN '
        STRING CARRY     '  /LD * /SET * /SET * CUP '
        STRING INPUT     '  A1 + /A2 + A3 '
```

One can also use previously defined string names in the string declaration. For example

```
STRING OUTPUT   'LOAD * CARRY'
```

The user has to be very careful when using substitution for the final meaning. For example consider Ex I.

If in the equation section there is an occurrence of INPUT then an expression like /AI /A A will result after substitution and not /(AI /A A3) as the user most probably would expect. If the later meaning is what the user wants, the string definition should be:

```
STRING INPUT ' (A1 + /A2 + A3) '
```

(This logic expansion is currently not supported in PALASM2)

NOTE:

Once again I want to stress that the user has to be careful of the logical correctness of the final expression after substitution.

Each substitution string should be separated from other characters by at least one blank or tab wherever they are used. For example

```
A1 := LOAD * /A3 + A1
```

not:

```
A1 :=LOAD* /A3 + A1
```

All substitution strings must be declared after the pin list and before the functional description.

Comments are NOT allowed within the single quotes defining the string.

This section will discuss the syntax of combinatorial equations.

```
output = <product term> + <product term> +...
<product term> ===>> <signal> *...
<signal> *. <signal> ===>> /identifier    identifier.
identifier ===>> declared pin name
```

The combinatorial equation is identified by the operator '='. The signal on the left side of the '= sign is the output for which the equation is described. This output signal can be active-high (output) or active-low (/output).

For PAL devices with programmable polarity, the polarity fuse is blown or left intact according to the polarities given to the left side of the equation and those used in the pin list when they are the same, the fuse is blown when they differ the fuse is left intact. For example:

```
CHIP POLARITY—EXAMPLE PAL16P8 A B C D /E /F nc nc nc GND
Y /Z W /V NC NC NC NC NC VCC
EQUATIONS
    Y = A * B + /C * D
   /Z = E * F + /F * /E
   /W = E
    V = /f
```

In this example, equations for outputs for Y and Z have the same polarity as they are described in the pin list. In programmable-polarity parts (PAL16P8), the polarity fuse is blown. If this is an active-low part (PAL16L8), then it is an error.

The outputs W and V have polarity reversed from that specified in the pin list. The polarity fuse is left intact.

The programmable-polarity feature allows the user to describe the function in either active-low or active-high state. The user does not have to transform the function using De Morgan's theorem.

Registered Equation

This section will discuss the syntax of registered equations. These equations are described for outputs with a a register. For example: Each output of PAL16R8 is a registered output.

```
output := <product term> + <produc term> + ...
<product term> ===>> <signal> *...
<signal> * . <signal> ===>> /identifier    identifier.
```

The registered equation is identified by the operator ':='.

The signal on the left side of the ':= sign is the output for which the equation is described.

The clock to the register is the special clock pin depending on the PAL type, (example PAL16R has pin number as the clock pin), or the clock is generated by a special product term described in a CLKF functional equation on the 20RA10.

The transition at the output of the register takes place on the rising edge of the clock.

This output signal can be active-high (output) or active-low (/output).

For programmable polarity parts, the polarity fuse is blown or left intact according to the polarity given to the left side of the equation and that used in the pin list when they are the same, the fuse is blown, when they differ the fuse is left intact. For example:

```
CHIP POLARITY—EXAMPLE PAL16RP8
CLK A B C D /E /F nc nc GND
Y /Z W /V NC NC NC NC NC VCC
EQUATIONS

   Y = A * B + /C * D
  /Z = E * F + /F * /E
  /W = E
   V = /f
```

In this example, equations for outputs for Y and Z have the same polarity as they are specified in the pin list. In programmable polarity parts (like PAL16RP8), the polarity fuse is blown. If this is an active low part (PAL16R8), then this is an error.

The outputs W and V have polarity reversed from that specified in the pin list. The polarity fuse is left intact.

The programmable polarity feature allows the user to describe the function in either active-low or active-high state. The user does not have to transform the function using De Morgan's theorem.

Functional Equations

In some PAL devices, the outputs have special functions which are controlled by a programmable product term. For example in PAL16L8, the tri-state function is controlled by a product term.

In the PAL20RA1 device, every registered output has four programmable functions, which are:

1) programmable clock function
2) programmable set function
3) programmable reset
4) programmable tri-state

In order to describe the functions to be implemented in the PAL device, functional equations are used. Order of appearance in a PAL device design specification is not significant for functional equations. These functional equations have the following syntax.

```
output.sfunc = <product term>
```

The left side of the equation identifies the function for the output defined by the right side of the equation. The following keywords are to be used to identify the functions:

CLKF	for programmable clock function.
SETF	for programmable set function.
RSTF	for programmable reset function.
TRST	for programmable tri-state function.

The following default conditions apply:

CLKF —the default condition is that all the fuses are left intact i.e. it is connected to GND.

SETF —If the output is defined as combinatorial, default value is VCC.
If the output is defined as registered, default value is GND.

RSTF —If the output is defined as combinatorial, the default value is VCC.
If the output is defined as registered, the default value is GND.

TRST —The default value is VCC.

Here are some details of these functions and their default conditions.

Default for SETF and RSTF

In PAL20RA10, it is always possible to bypass the register by having the SET and RESET product terms high. There are two ways of doing this. One way is to be explicit as follows:

OUT := A + /B + D * E ; Output defined as registered OUT.SETF = VCC OUT.RSTF = VCC OUT.CLKF = GND The other way is to be implicit as follows:

```
OUT = A + /B + D * E      ; Output defined as combinatorial
```

In the implicit case, the program XPLOT will take care of the default conditions for SETF, RSTF and CLKF

Default for registered output

In some cases, the user might not want to use the SET and RESET functions. Again, he can be explicit as follows:

```
OUT := A + /B
OUT.SETF = GND
OUT.RSTF = GND
OUT.CLKF = CLK
```

or he can be implicit as follows:

```
OUT := A + /B
OUT.CLKF = CLK
```

The program XPLOT will take care of the default conditions and program the appropriate fuses.

Default for TRST

The default for the tri-state function is VCC. This can be specified explicitly as follows:

```
OUT := A + B
OUT.CLKF = CLK
OUT.TRST = VCC.
```

or implicity by not specifying the tri-state function, the XPLOT program will blow all the fuses.

Default for CLKF

If the clock function is not defined, all the fuses are left intact.

If the output is defined as a registered output, the CLKF MUST BE DEFINED. Otherwise, XPLOT will give an error. You can define it as GND if you do not want to use it but it must be defined.

If the output is defined as combinatorial, then it is redundant to define the CLKF. XPLOT will give an error if it is defined.

Polarity

The user must remember that the polarity fuse is located in front of the register and hence effects the inversion of the data path. The data path is the output of the OR gate through polarity fuse and into the register. It does not effect the set or reset function of the output.

If no output equation is defined, the polarity fuse is left intact.

Section 3: Simulation

Simulation is a very important part of any design cycle. After the user has defined the logic in terms of equations, it is neccessary to be able to verify that the equations do implement the required function. The basic feature of any simulation is the ability to give a set of inputs to the design and be able to check the outputs for correctness. This ability to give values to inputs and observe the outputs is crucial to any simulation language.

Logic Simulation can be described using

—SETF, CLOCKF, TRACE—ON, TRACE—OFF, and CHECK commands.

—FOR loop to iterate a set of commands a fixed number of times.

—WHILE <condition> DO loop also to iterate a set of commands till the condition is false.

—IF <condition> THEN ELSE for conditional branching.

PALASM2 software has an 'Event Driven Simulator supporting all the different PAL device architectures, both asynchronous and synchronous. The program is so designed that internal events generated by asynchronous/synchronous feedbacks and external events generated by the user are simulated in a very realistic way. Oscillatory conditions are also detected and reported to the user. Conflict in the expected and the actual value of any signal is an error which is detected by the simulator and reported to the user. The simulation continues from that point using the actual value of the signal.

The PALASM2 software language has simulation commands which are 'English like words thereby making the simulation specification very natural to read and understand. There are facilities for iterative looping, conditional branching, setting of signals, checking of signal values, and selective observation of signals. All these commands will be explained in the following paragraphs.

All the simulation results are stored in two files, a history file (<filename>.HST) and a trace file (<filename>.TRF). The history file has the values of all the signals from the start of simulation to the end. The trace file has the values of the signals mentioned in the TRACE—ON statement and only up to the next TRACE—OFF statement.

The simulation results are organized in a horizontal format resembling a timing diagram. Each page contains 4 vectors. a maximum of 51 vectors are allowed with this release of the simulator. Corresponding to each SETF and CLOCKF statement in the simulation a 'g or 'c appears on the horizontal axis in the result files. A CLOCKF statement causes the clock to go L to H to L. The 'c appears over the final L. This helps the user to identify the vector corresponding to SETF or CLOCKF statement.

The basic philosophy of the simulation language is to make it easy for the designer to describe the function in a natural way so that it is easy to comprehend the behavior of the design from the simulation specification. PALASM simulation language is divided into two sections Directives and Structured Control. The simulation directives provide commands to establish circuit inputs, clock waveforms, check for circuit outputs and capture time response waveforms as desired by the user.

The simulation section is introduced by the keyword SIMULATION.

Simulation Directives—Syntax:

```
SETF            <signal list>
CLOCKF          <clock signals>
CHECK           <signal list>
TRACE—ON        <signal list>
TRACE—OFF
```

Structured Control constructs—Syntax:

```
FOR I:= <lower limit> TO <upper limit> DO
    BEGIN          <statements>        END

WHILE <condition> DO
    BEGIN          <statements>        END

IF          <condition> THEN
    BEGIN          <statements>        END
  ELSE
    BEGIN          <statements>        END
```

The structured control constructs are used to build up sequences of operations that repeat or are modified as a result of particular logic values or conditions. They provide the basic looping and decision branching of structured high-level programming languages. <condition> is a boolean expression or a mathematical equality. The condition is true if the boolean expression is asserted or the mathematical equality is satisfied.

Details of the Simulation Syntax:

Each of the individual statements will be described, along with some related items necessary for their proper usage.

SETF statement

```
SETF <signal list>
Ex. SETF A /OE B /RESET /D0 D1 D2
```

The signal is set high (H) if it is not preceded by otherwise it is set low (L). In the above example A B D and D are all set to H and OE, RESET, and D are all set to L.

The signal should only be set if a change is wanted from the previous value. The simulator always remembers the last value of all the signals. At the start of simulation all signals are assumed to have don't care value (X).

Every time a SETF statement is executed, a vector is generated and all the equations that are effected are evaluated. Any internally generated events are also detected and evaluated. Depending upon the activity, many more vectors can be generated by a single SETF statement because of feedbacks and asynchronous events. The simulator continues this till the system stabilizes, that is, until there are no more changes in the output signals or no events are generated. If the system fails to stabilise after 1 iterations, then an oscillatory condition is detected and the simulation halts.

CLOCKF statement

CLOCKF <list of clock signals>
Ex. CLOCKF CLK1 CLK2

The CLOCKF statement has the list of clock signals (dedicated clock pins) to which a clock pulse is to be applied. Only the clock pins of the device can be used in the CLOCKF statement, any other pin is an illegal signal for CLOCKF statement.

Each CLOCKF statement corresponds to a pulse going from low to high to low. Thus two vectors are generated in the proccess and during the positive edge transition, the new value of the registers which are clocked is transfered to the output. No action takes place for the registers that are not clocked.

At every CLOCKF statement, internally generated events and asynchronous events are detected and if present, more vectors are generated. The operation of CLOCKF is similair to that of SETF statement except that it goes through a pulse rather than a level.

CHECK statement

CHECK <signal list>
EX. CHECK Q /Q /Q

This is a facility provided to the user to keep track of the simulation results. The signals in the check statement are the output signals which the user wants to check. In the above example, the user wants to check if Q is high and Q and Q are low. Again a signal without is to be checked high and the signal with is to be checked for low.

Whenever a CHECK statement is executed, the simulator compares the actual value and the expected value of a particular signal. If they are equal then no action is taken. Otherwise, an error is reported and the simulator continues assuming the actual value. The error is reported by . in the vector at the place of the error and also the vector number. The history and trace files will contain the . at the particular location.

This is a powerful statement that should be used at important points in your simulation for debugging your design.

TRACE—ON statement

TRACE—ON <signal list>
Ex. TRACE—ON /OE SET RESET D0 D1 D2 D3 /Q0 /Q1 /Q2

This statement contains the list of signals which have to be listed in the trace file. The signals will be listed in the same order and with the same polarity as present in the TRACE—ON statement. This list of signals will be active until the next TRACE—OFF statement or untill the end of the simulation specification. New signals can be traced on after the TRACE—OFF statement.

This statement helps the user to group the signals more naturally for debugging purposes. For example, all control signals can be grouped together, then all the data signals can be grouped together, and then all the output signals can be grouped together. This makes the observation of the results in the trace file very easy.

TRACE—OFF statement

TRACE—OFF

This statement traces off all the signals mentioned in the latest TRACE—ON statement. After this statement, no more results are added to the trace file until the next TRACE—ON statement is executed. Thus all the results between the current TRACE—OFF statement and the next TRACE—ON statement are not displayed in the trace file.

This feature helps to break up the results in different time frames which are critical for debugging purposes, rather than having unwanted results. It should be remembered that the history file contains all the information from the start of the simulation to the end of simulation. The signals are in the same order and of the same polarity as mentioned in the pin list of the CHIP statement.

FOR loop

```
FOR <index var> := <lower limit> TO <upper limit> DO
    BEGIN
        <statements>
        ...
    END
Ex. FOR J:= 3 to 8 DO BEGIN
      SETF A /B CLOCKF clk
      IF J= 5 THEN BEGIN CHECK Q0 END
      ELSE BEGIN CHECK /Q0 END
    END
```

The FOR loop provides a repetitive execution of statements which is very powerful. Many statements can be embedded in a FOR loop even another FOR statement with a different indexing variable. Using this statement one can generate many vectors by just increasing the limits of the for loop.

The <lower limit> should be less than or equal to the upper limit. All the limit values should be greater than or equal to zero. You can not use negative values for the limits. The loop is not executed if the conditions expressed in the limits are equal.

IF THEN ELSE

```
IF <cond> THEN          or          IF >cond< THEN
   BEGIN                               BEGIN
     <Statements>                        <Statements>
   END                                 END
 ELSE
   BEGIN
     <Statements>
   END
```

There are two variations of this statement. In the first usage, there is an ELSE clause and in the second usage there is no ELSE clause. If the <condition> is true the THEN clause is exectuted otherwise the ELSE clause is executed. If there is no ELSE clause, then the simulation executes the next statement after the IF statement. Condition expressions can not contain nested parenthesis.

The <condition> can be any mathematical equality;
(=, >, <, >=, <=, <>), for example:

```
IF (I<2) THEN
```

The <condition> also be any boolean expression, for example:

```
IF (DRDY * /CLR) THEN
```

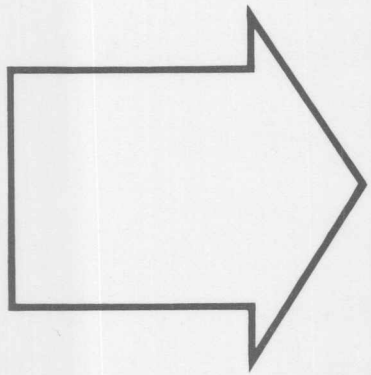
In the first example the condition of I less than is checked and in the second example the expression (DRDY /CLR) is evaluated and if it is true then the condition is true.

WHILE loop:

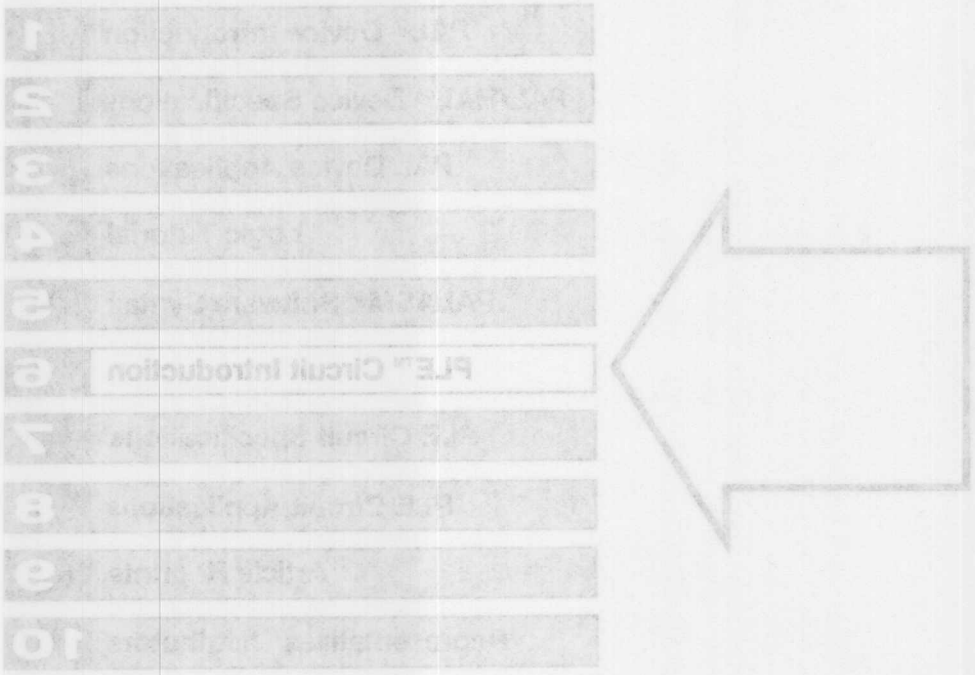
```
WHILE <condition> DO
   BEGIN      <statements>      END
```

The WHILE loop provides a repetetive execution of statements which may be controled by evaluation of logic conditions present within the PAL device. Many statements can be embedded in a WHILE loop including even other looping constructs. The WHILE loop is used to itterate a set of commands until the condition is false.

The <condition> can be any boolean expression of logic signals.



Contents Section 6



An Introduction to Programmable Logic Elements

A logic function, whether combinatorial or sequential, may be represented in Sum of Product (SOP) form by using De Morgan's law and Boolean Algebra. Any complex multi-level logic function can easily be reduced to a two-level AND-OR configuration. This property of logic functions lends a very regular character, making it possible to implement them in a structured methodical way. The uniform AND-OR array-like architecture of Programmable Logic Devices was conceived for a clean and efficient implementation of these functions, as shown in Figure 1.

Either or both of the arrays can be programmable, constituting three distinct families of devices as shown in Figure 2.

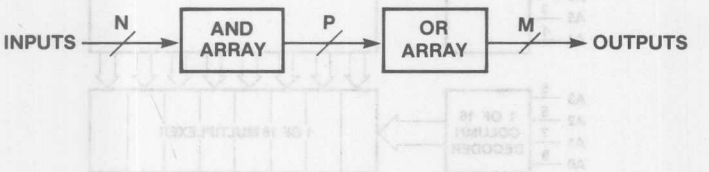


Figure 1. Structure of Programmable Logic Devices

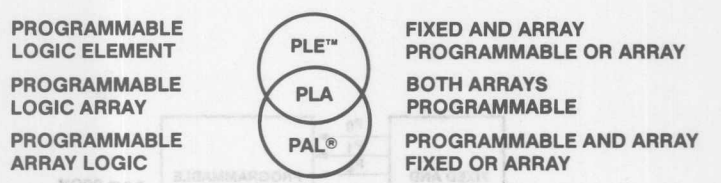


Figure 2. Structural Differences Between PLE, PAL and PLA Devices

PAL®, HAL® and PALASM® are registered trademarks of Monolithic Memories.
PLE™ is a trademark of Monolithic Memories.

PLE Device Architecture

The array like structure of a PROM lends itself naturally to being viewed as a two-level AND-OR logic circuit. The inputs to the PROM are fully decoded into all possible combinations in the fixed AND plane. Each combination (product term) is fuse connected to each output in the programmable OR plane.

For a PLE device, a product term is equivalent to an AND gate equal in size to the number of inputs. Each output is equivalent to an OR gate connected to all the AND gates. Programming a fuse then implies breaking a connection between an AND gate and OR gate.

Thus a PROM has a convenient structure for implementing combinatorial logic when a large number of input combinations are required, or a large number of product terms per output is desired. Registered PROMs are ideally suited for implementing complex sequential machines which contain a large number of variables in the state equations.

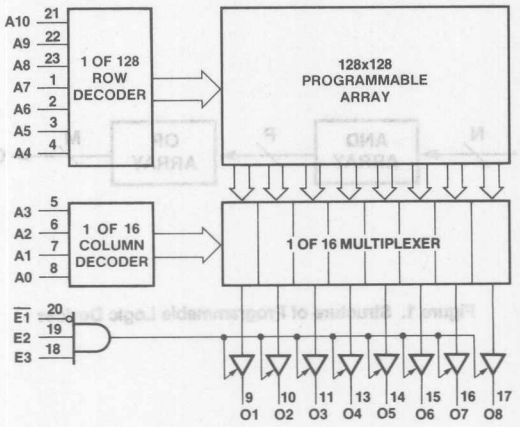


Figure 3. General Block Diagram of a 2Kx8 PROM

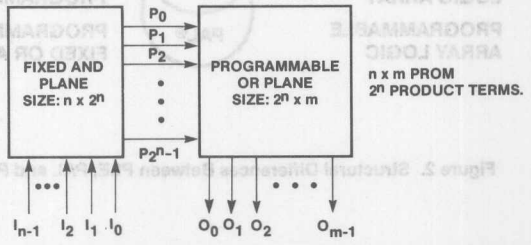


Figure 4. Block Diagram of a PROM Viewed as a PLE Device

In terms of a Karnaugh map, each minterm in the map corresponds to one product term in the PLE array. Two or more adjacent minterms cannot be be combined to generate a prime implicant, or eliminate a logic hazard. For example, the following Karnaugh map will generate the function f = ab + ab. The minimized function f = a as indicated by the dotted prime implicant can not be implemented. The PLE device does not contain a product term with fewer than all its inputs present.

The absence of prime implicants in a PLE array may cause logic hazards which may be unavoidable in asynchronous control systems. However these hazards are masked out in sychronous control systems by the registers, and are largely irrelevant in data path applications where only the final steady state results are looked at. Indeed, most applications of PLE devices are in synchronous control systems to replace random logic. In the data paths, they are used to generate complex functions like ALU operations, high-speed multiplication, Pseudo Random Number sequences, Error Detection codes, etc.

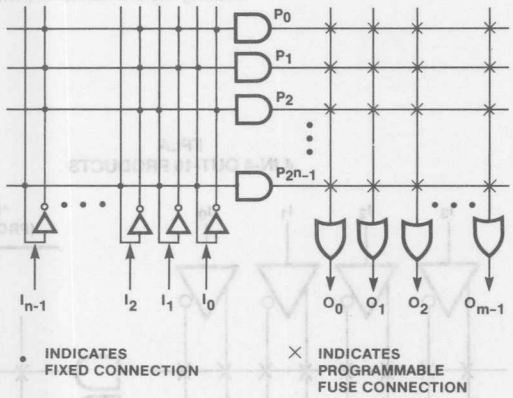


Figure 5. Equivalent Logic Circuit. A Virgin PLE Device Has All the Fuses Intact

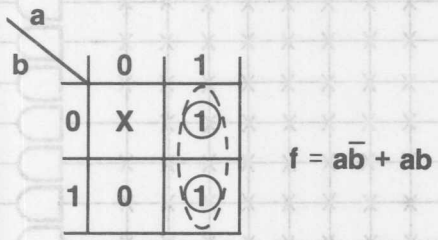


Figure 6. An Arbitrary Two-Variable Karnaugh Map for a PLE Device

PLAs have the most general architecture with both arrays programmable. For N inputs, M outputs, and P product terms in a PLA, the AND array contains 2 x N x P programmable crosspoint connections. All possible combinations of the inputs, taken together, or in groups, or even a single input, can have a product term in the AND array. The inputs not desired in a product term are disconnected, by removing the corresponding crosspoint connection. In field programmable PLAs, this corresponds to electrically blowing a fusible link connection. These PLAs also usually contain less product terms than the maximum possible 2^n, to conserve chip area.

outputs, but may be disconnected by blowing that fuse connection. This architecture is used for implementing logic functions with a large number of product terms of varying sizes, or a large number of product terms per output.

Field-programmable PLAs are generally not very-high performance. They are slower in speed compared to PAL and PLE devices. A given signal must pass through two large programmable arrays, which increases the capacitance on the signal, and increases the delay. For most applications, a large number of crosspoints in one or another array are usually left intact, making the architectural flexibility redundant.

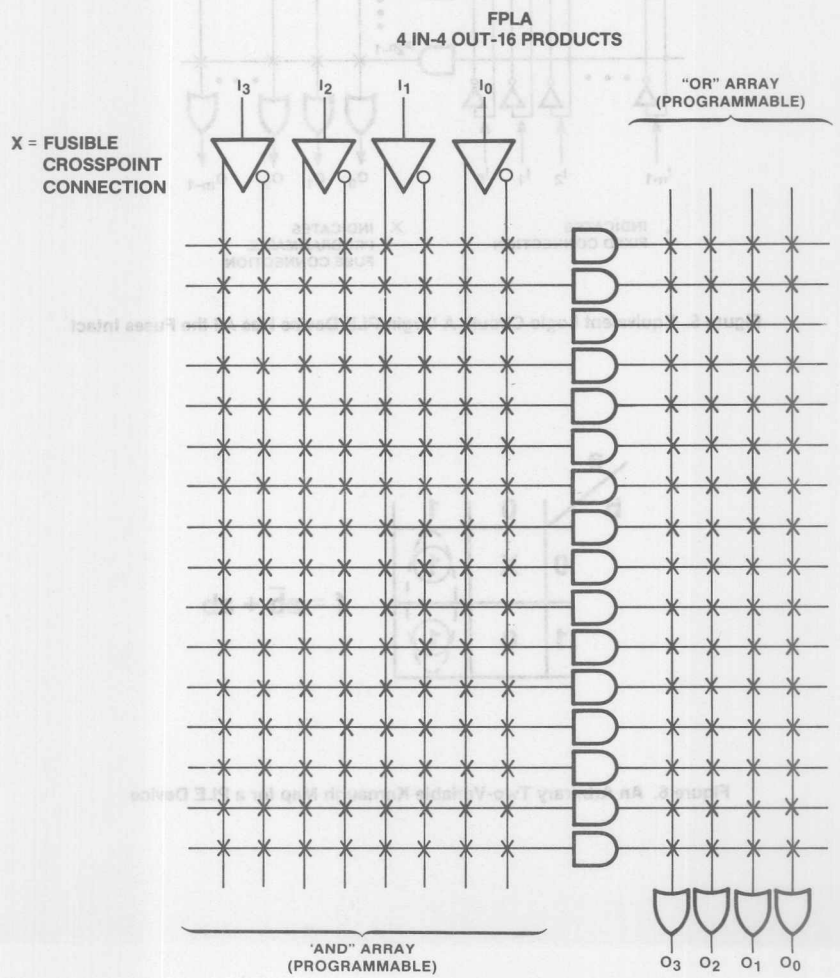


Figure 7. Structure of FPLA

PAL® Architecture

PAL devices are the most useful and efficient of the fuse programmable logic family. First developed and patented by Monolithic Memories in 1976, the PAL devices have become well known for their friendliness in system configurations. The programmable AND array and fixed OR array eliminate most of the redundancies associated with PLAs. The PAL AND array is logically identical to a PLA AND array, with the only difference that PAL devices have fewer product terms. In the fixed OR array, each product term is connected to only one output or OR gate, which eliminates product term sharing of PLAs. The PAL device configuration has permitted several architectural innovations, making the PAL circuit family of devices extremely useful for implementing all kinds of logic functions. PAL device features include outputs with/without registers that are internally fed back to the AND array, special XOR gates in the OR array, arithmetic carry generate gates in the feedback path in the AND array, programmable I/O pins, and programmable output polarity. New generation PAL devices also have the feature of product term sharing where a product term can be attached to one of two adjacent sum terms.

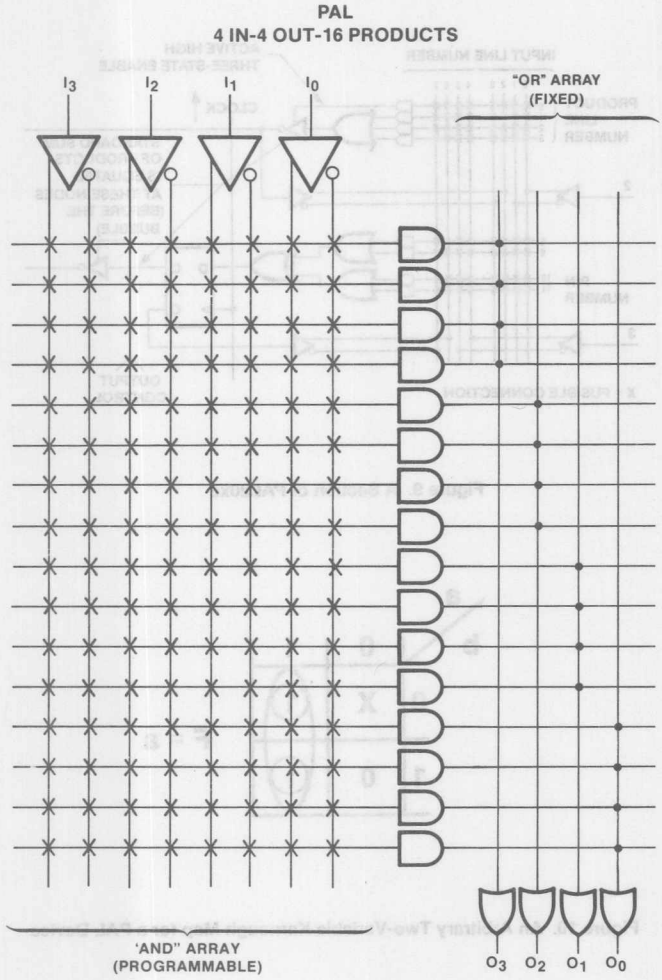


Figure 8. Structure of PAL Device

As the PAL AND array is programmable, logic functions can be minimized and logic hazards removed, by combining adjacent minterms in a Karnaugh map. This group of minterms or implicants is implemented as a product term. Thus PAL device outputs can be designed to be glitch-free, and ideal for implementing control logic. Figure 10 illustrates the absence of hazards and race conditions in a PAL device.

The limitation of PAL device is that they have a restricted number of product terms per output, and fewer product terms in general. Certain logic functions containing a large number of product terms would require a large number of PAL devices to implement them, which increases the propagation delay and the chip count. For these applications, PLE devices are ideal.

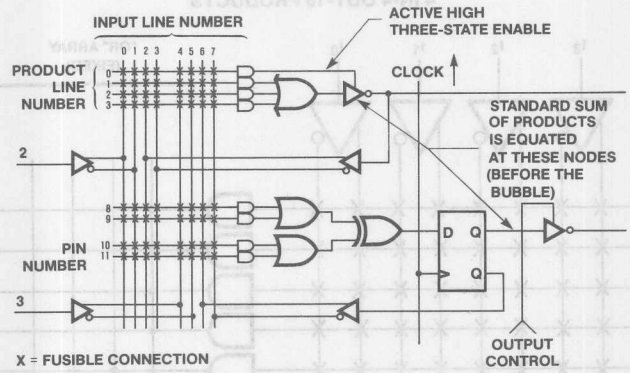


Figure 9. A Section of PAL20x8

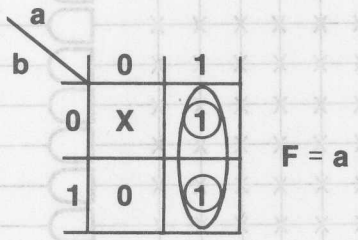


Figure 10. An Arbitrary Two-Variable Karnaugh Map for a PAL Device

PLE and PAL Devices

PLE devices bridge the gap between the flexibility of PLAs, and the product term restrictions in PAL devices. Those applications for which PAL devices are not suitable, PLE devices take over. Where a PAL device typically has a large number of inputs, and a small number of product terms, the PLE devices have a restricted number of inputs and a large number of product terms. Also, it has a large number of product terms per output with full product term sharing, whereas PAL devices have a restricted number of product terms per output with no product term sharing. Thus PAL and PLE devices complement each other both structurally and functionally.

PLE Device Features

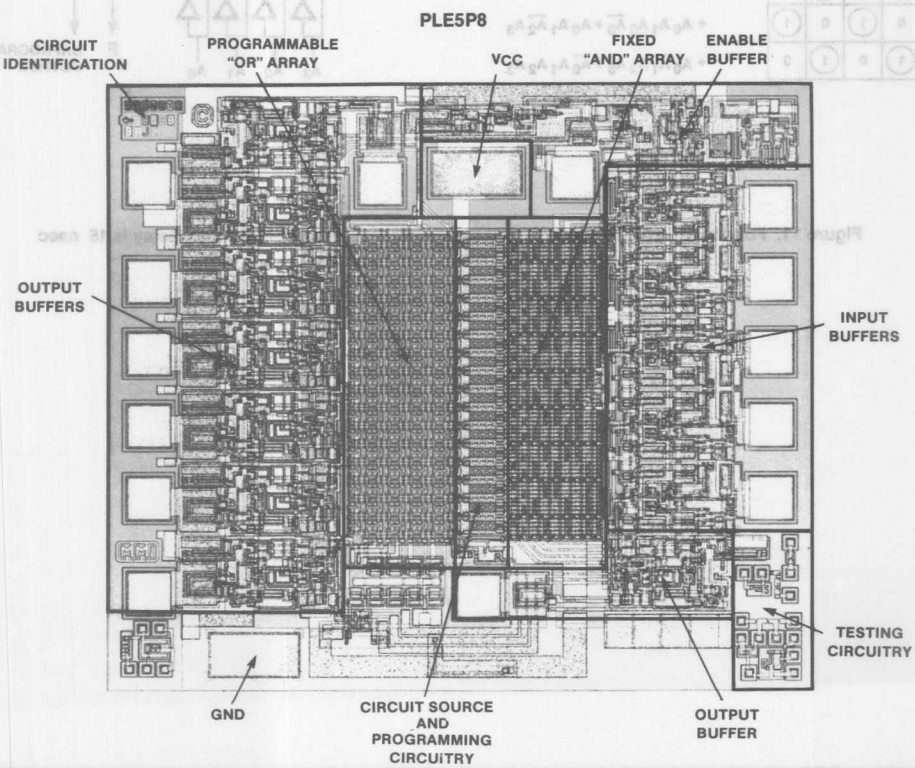
- **2^n product terms per output are available with n inputs each.**
- **Programmable output polarity.**
- **Programmable initialization in registered PLE devices.**
- **High-level functional specification of PLE systems in terms of logic equations.**
- **Input-to-output delays comparable to discrete logic gates (typically less than 15 ns).**
- **PNP inputs provide high impedance.**
- **Monolithic Memories' TiW fusible-link technology and advanced self-aligned washed-emitter bipolar process guarantees a programming yield greater than 98%.**

PLE Circuit Family

The PLE circuit family of devices is an extension of Monolithic Memories' TiW PROM family. The entire line of TiW PROMs including the Registered parts are available in PLE circuit configurations. High speed and diagnostic versions in military and commercial temperature ranges are available.

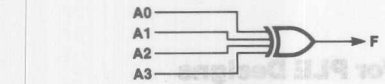
CAD Tool for PLE Designs

PLEASM software is a PLE Assembler written in FORTRAN 77 and available on most mini- and microcomputer systems. It enables specifying PLE Circuit data in terms of logic equations (like those in Figure 14), which are assembled into a fuse-pattern format compatible with commercially available PROM programmers. PLEASM software also generates a truth table from the logic equations which is later used as test vectors for logic simulation and verification. PLEASM allows complete design customization and documentation of PLE systems in a simple high-level functional language.



Control Path Example

A common application of PLE devices in the control path of a synchronous system, is to replace random logic, or customize logic functions. An n-input exclusive OR function is quite commonly required in comparator and adder circuits. It contains 2^{n-1} product terms, which increases exponentially with n. Therefore, it is very efficient to implement large XOR functions in PLE devices. Figure 11 shows the implementation of a 4-input XOR in a PLE circuit.



KARNAUGH MAP

A_2, A_3 \ A_0, A_1	00	01	11	10
00	0	(1)	0	(1)
01	(1)	0	(1)	0
11	0	(1)	0	(1)
10	(1)	0	(1)	0

PLEASM EQUATIONS:

$$F = A_0 \oplus A_1 \oplus A_2 \oplus A_3$$

$$= A_0\overline{A_1}\overline{A_2}\overline{A_3} + \overline{A_0}A_1\overline{A_2}\overline{A_3}$$

$$+ \overline{A_0}\overline{A_1}A_2\overline{A_3} + \overline{A_0}\overline{A_1}\overline{A_2}A_3$$

$$+ A_0A_1A_2\overline{A_3} + A_0A_1\overline{A_2}A_3$$

$$+ A_0\overline{A_1}A_2A_3 + \overline{A_0}A_1A_2A_3$$

SECTION OF PLE5P8/A

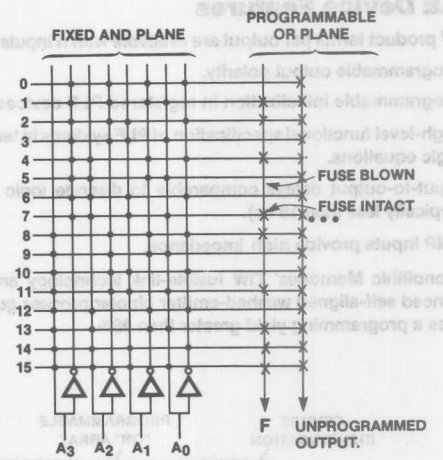


Figure 11. Four-Input XOR Function Implemented in a PLE Device. Maximum Delay is 15 nsec

Data Path Example

In the data path, a registered PLE device can be used to implement complex functions, like a Pseudo Random Number (PRN) Generator. RPN sequences are useful in encoding and decoding of information in signal processing and communication systems. They are used for data encryption in secure communication links, and error detection and correction codes in data communication systems. PRN sequences are also utilized as test vectors for circuit simulation, as signal modulators in radar range-finding systems, and as reference white noise in many signal processing applications. Figure 12 illustrates a typical mechanism for generating PRN sequences.

The advantage of using PLE devices for implementing PRN sequences is that any polynomial can be quickly customized in it. In data encryption systems where the code is frequently changed for protection from unauthorized access, PLE devices can be used to generate a new code each time, or several codes can be implemented in the same device. An example of a PRN generator implemented in a Registered PLE device is shown in Figure 13.

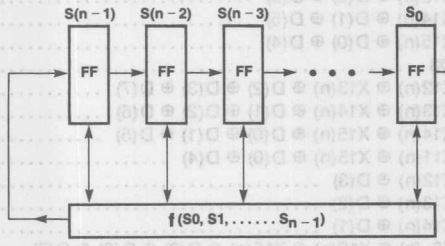


Figure 12. An n-Stage Linear Feedback Shift Register (LFSR)

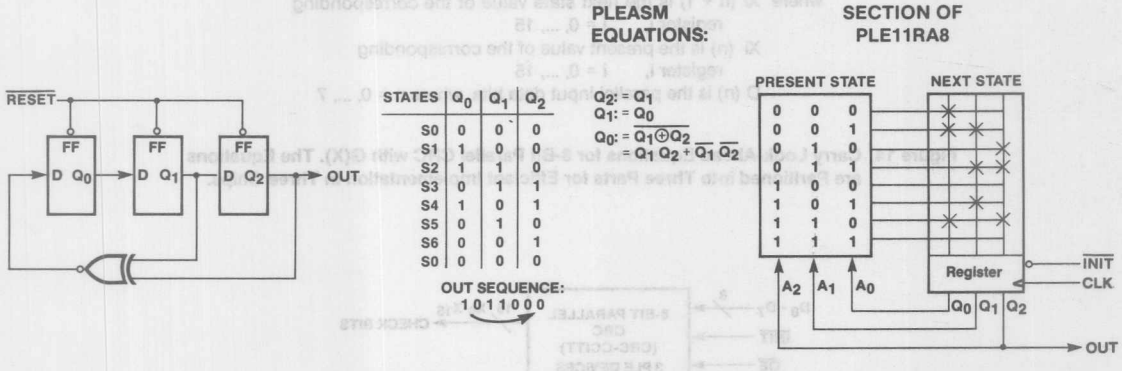


Figure 13. A Three-Stage Pseudo Random Number Generator Implemented in a Registered PLE Device

Parallel CRC in Registered PLE Devices

Implementing a high-speed M-bit Parallel Cyclic Redundancy Check (CRC) code in a registered PLE circuit is almost trivial. Once the M-bit carry look-ahead equations are determined, PLEASM software is used to assemble these equations into a fuse pattern for the Registered PLE device.

The speed of operation of parallel CRC implemented in registered PLE devices will remain the same for any generator polynomial and M. Increasing complexity of the carry look-ahead equations only increases the number of devices required to implement them. It does not decrease the speed of operation.

To illustrate with a practical example, Figure 14 shows the 8-bit carry look-ahead equations for an 8-bit Parallel implementation of the following generator polynomial:

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

also called the CRC-CCITT standard. These equations are derived in Application Note AN-125, where an implementation in four PAL devices is also shown with a maximum delay of 90 nsec. Figure 16 shows an implementation in three 24-pin registered PLE devices and one SSI part. The maximum delay is 50 nsec.

X0 (n + 1) := X8 (n) ⊕ X12(n) ⊕ D(3) ⊕ D(7) chip1
X1 (n + 1) := X9 (n) ⊕ X13(n) ⊕ D(2) ⊕ D(6) chip2
X2 (n + 1) := X10(n) ⊕ X14(n) ⊕ D(1) ⊕ D(5) chip3
X3 (n + 1) := X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3
X4 (n + 1) := X12(n) ⊕ D3 ..... chip1
X5 (n + 1) := X8 (n) ⊕ X12(n) ⊕ X13(n) ⊕ D(2) ⊕ D(3) ⊕ D(7) chip1
X6 (n + 1) := X9 (n) ⊕ X13(n) ⊕ X14(n) ⊕ D(1) ⊕ D(2) ⊕ D(6) chip2
X7 (n + 1) := X10(n) ⊕ X14(n) ⊕ X15(n) ⊕ D(0) ⊕ D(1) ⊕ D(5) chip3
X8 (n + 1) := X0 (n) ⊕ X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3
X9 (n + 1) := X1 (n) ⊕ X12(n) ⊕ D(3) chip1
X10(n + 1) := X2 (n) ⊕ X13(n) ⊕ D(2) chip2
X11(n + 1) := X3 (n) ⊕ X14(n) ⊕ D(1) chip2
X12(n + 1) := X4 (n) ⊕ X8 (n) ⊕ X12(n) ⊕ X15(n) ⊕ D(0) ⊕ D(3) ⊕ D(7) chip1
X13(n + 1) := X5 (n) ⊕ X9 (n) ⊕ X13(n) ⊕ D(2) ⊕ D(6) chip2
X14(n + 1) := X6 (n) ⊕ X10(n) ⊕ X14(n) ⊕ D(1) ⊕ D(5) chip3
X15(n + 1) := X7 (n) ⊕ X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3

where Xi (n + 1) is the next state value of the corresponding register i, i = 0, ..., 15
Xi (n) is the present value of the corresponding register i, i = 0, ..., 15
D (n) is the parallel input data bits, where n = 0, ..., 7

Figure 14. Carry Look-Ahead Equations for 8-Bit Parallel CRC with G(X). The Equations are Paritioned into Three Parts for Efficient Implementation in Three Chips.

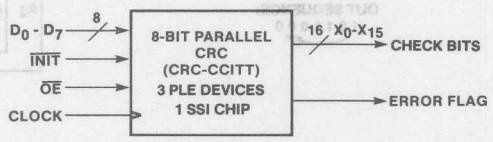


Figure 15. Block Diagram of 8-bit Parallel CRC

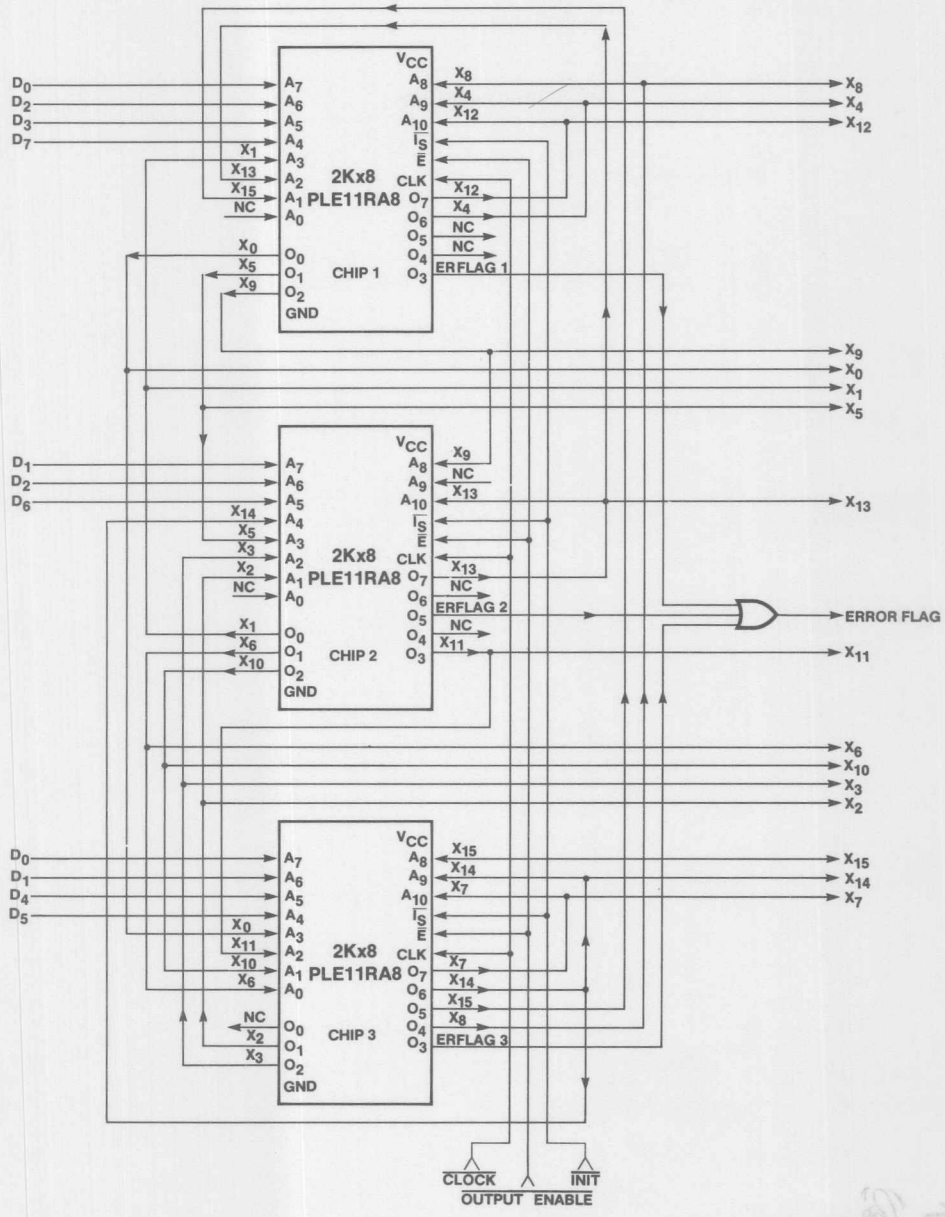
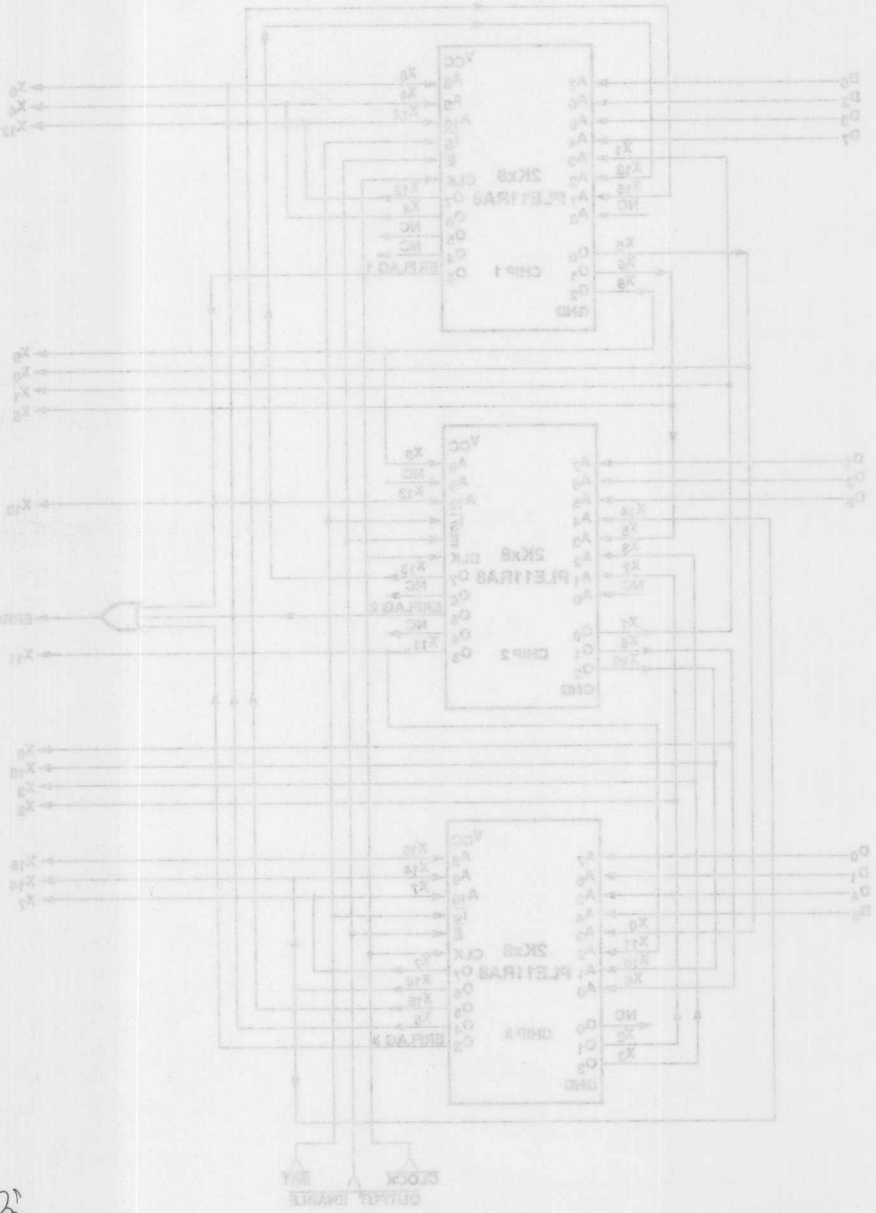
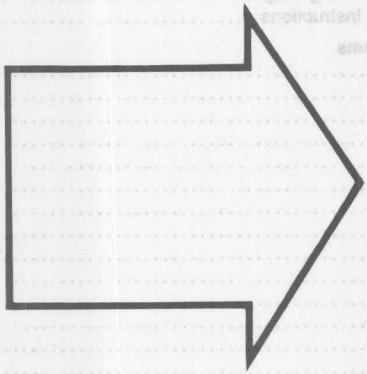


Figure 16. Diagram Showing How to Connect Three Registered PLE Devices to Implement 8-Bit Parallel CRC. The Error Flag is Valid on the Next Clock Pulse After All the Data Has Been Clocked In.
Error Flag = $X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8 + X_9 + X_{10} + X_{11} + X_{12} + X_{13} + X_{14} + X_{15}$




PLE
IMPORTANT NOTES



Contents Section 7

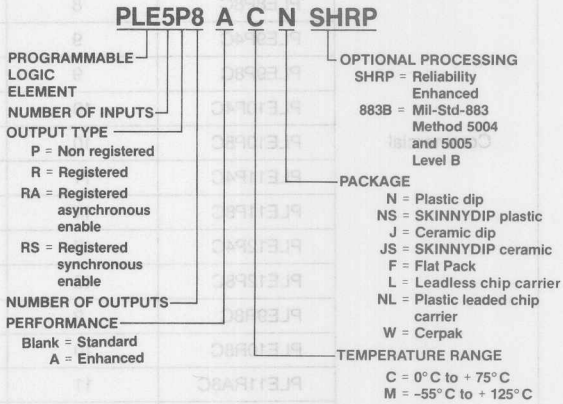
PLE Device to PROM Cross Reference

TEMPERATURE RANGE	PLE NUMBER	INPUTS	OUTPUTS	OUTPUT TYPE	ARRAY SIZE	PROM NUMBER
Commercial	PLE5P8C	5	8	Three-State	32x8	63S081
	PLE5P8AC	5	8	Three-State	32x8	63S081A
	PLE8P4C	8	4	Three-State	256x4	63S141A
	PLE8P8C	8	8	Three-State	256x8	63S281A
	PLE9P4C	9	4	Three-State	512x4	63S241A
	PLE9P8C	9	8	Three-State	512x8	63S481A
	PLE10P4C	10	4	Three-State	1024x4	63S441A
	PLE10P8C	10	8	Three-State	1024x8	63S881A
	PLE11P4C	11	4	Three-State	2048x4	63S841A
	PLE11P8C	11	8	Three-State	2048x8	63S1681A
	PLE12P4C	12	4	Three-State	4096x4	63S1641A
	PLE12P8C	12	8	Three-State	4096x8	63S3281A
	PLE9R8C	9	8	Register	512x8	63RA481A
	PLE10R8C	10	8	Register	1024x8	63RS881A
	PLE11RA8C	11	8	Register	2048x8	63RA1681A
	PLE11RS8C	11	8	Register	2048x8	63RS1681A
Military	PLE5P8M	5	8	Three-State	32x8	53S081
	PLE8P4M	8	4	Three-State	256x4	53S141A
	PLE8P8M	8	8	Three-State	256x8	53S281A
	PLE9P4M	9	4	Three-State	512x4	53S241A
	PLE9P8M	9	8	Three-State	512x8	53S481A
	PLE10P4M	10	4	Three-State	1024x4	53S441A
	PLE10P8M	10	8	Three-State	1024x8	53S881A
	PLE11P4M	11	4	Three-State	2048x4	53S841A
	PLE11P8M	11	8	Three-State	2048x8	53S1681A
	PLE12P4M	12	4	Three-State	4096x4	53S1641A
	PLE12P8M	12	8	Three-State	4096x4	53S3281A
	PLE9R8M	9	8	Register	512x8	53RA481A
	PLE10R8M	10	8	Register	1024x8	53RS881A
	PLE11RA8M	11	8	Register	2048x8	53RA1681A
	PLE11RS8M	11	8	Register	2048x8	53RS1681A

Features/Benefits

- Programmable replacement for conventional TTL logic
- Reduces IC inventories and simplifies their control
- Expedites and simplifies prototyping and board layout
- Saves space with 0.3 inch SKINNYDIP® packages
- Programmed on standard PROM programmers
- Test and simulation made simple with PLEASM software
- Low-current PNP inputs
- Three-state outputs
- Reliable TiW fuses guarantee >98% programming yield

Ordering Information



PLE Device Selection Guide

PART NUMBER	INPUTS	OUTPUTS	PRODUCT TERMS	OUTPUT REGISTERS	t_{PD}(ns) MAX*
PLE5P8	5	8	32		25
PLE5P8A	5	8	32		15
PLE8P4	8	4	256		30
PLE8P8	8	8	256		28
PLE9P4	9	4	512		35
PLE9P8	9	8	512		30
PLE10P4	10	4	1024		35
PLE10P8	10	8	1024		35
PLE11P4	11	4	2048		35
PLE11P8	11	8	2048		35
PLE12P4	12	4	4096		35
PLE12P8	12	8	4096		40
PLE9R8	9	8	512	8	15
PLE10R8	10	8	1024	8	15
PLE11RA8	11	8	2048	8	15
PLE11RS8	11	8	2048	8	15

* Clock to output time for registered outputs.

Note: Commercial limits specified.

SKINNYDIP® is a registered trademark of Monolithic Memories.
PLE™, PLEASM™ and IdeaLogic™ are trademarks of Monolithic Memories.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2374
TWX: 910-338-2376
Monolithic Memories MMI

PLE Device Means Programmable Logic Element

Joining the world of IdeaLogic™ is a new generation of high-speed PROMs which the designer can use as logic elements. The combination of PLE devices with PAL devices can greatly enhance system speed while providing almost unlimited design freedom.

Basically, a PLE circuit is ideal when a large number of product terms is required; on the other hand, a PAL circuit is best suited for situations when many inputs are needed.

The PLE circuit transfer function is the familiar OR of products. Like PAL circuits, a PLE circuit has a single array of fusible links. Unlike PAL circuits, PLE circuits have a programmable OR array driven by a fixed AND array (a PAL device is a programmed AND array driving a fixed OR array).

PRODUCT TERMS AND INPUT LINES

	PLE	PAL
Product Terms	32 to 4096	1 to 16
Input Lines	5 to 12	6 to 64

The PLE device family features common electrical parameters and programming algorithms, low-current PNP inputs, full Schottky clamping and three-state outputs.

The entire PLE device family is programmed on conventional PROM programmers with the appropriate personality cards and socket adapters.

Registered PLE Devices

The registered PLE devices have on-chip "D" type registers, versatile output enable control through synchronous and asynchronous enable inputs and flexible start-up sequencing through programmable initialization.

Data is transferred into the output registers on the rising edge of the clock. Provided that the asynchronous ($\overline{E}$) and synchronous ($\overline{ES}$) enables are Low, the data will appear at the outputs. Prior to the positive clock edge, register data are not affected by changes in addressing or synchronous enable inputs.

Data control is made flexible with synchronous and asynchronous enable inputs. Outputs may be set to the high-impedance state at any time by setting $\overline{E}$ to a High or if $\overline{ES}$ is High when the rising clock edge occurs. When V_{CC} power is first applied the synchronous enable flip-flop will be in the set condition causing the outputs to be in the high-impedance state.

A flexible initialization feature allows start-up and time-out sequencing with 1:16 programmable words to be loaded into the output registers. With the synchronous INITALIZE ($\overline{IS}$) pin Low, one of the 16 initialize words, addressed through pins 5, 6, 7 and 8 will be set in the output registers independent of all other input pins. The unprogrammed state of $\overline{IS}$ words are Low, presenting a CLEAR with $\overline{IS}$ pin Low. With all $\overline{IS}$ column words (A3-A0) programmed to the same pattern, the $\overline{IS}$ function will be independent of both row and column addressing and may be used as a single pin control. With all $\overline{IS}$ words programmed High a PRESET function is performed.

PLE9R8 has asynchronous PRESET and CLEAR functions. With the chip enabled, a Low on the $\overline{PR}$ input will cause all outputs to be set to the High state. When the $\overline{CLR}$ input is set Low the output registers are reset and all outputs will be set to the Low state. The $\overline{PR}$ and $\overline{CLR}$ functions are common to all output registers and independent of all other data input states.

7

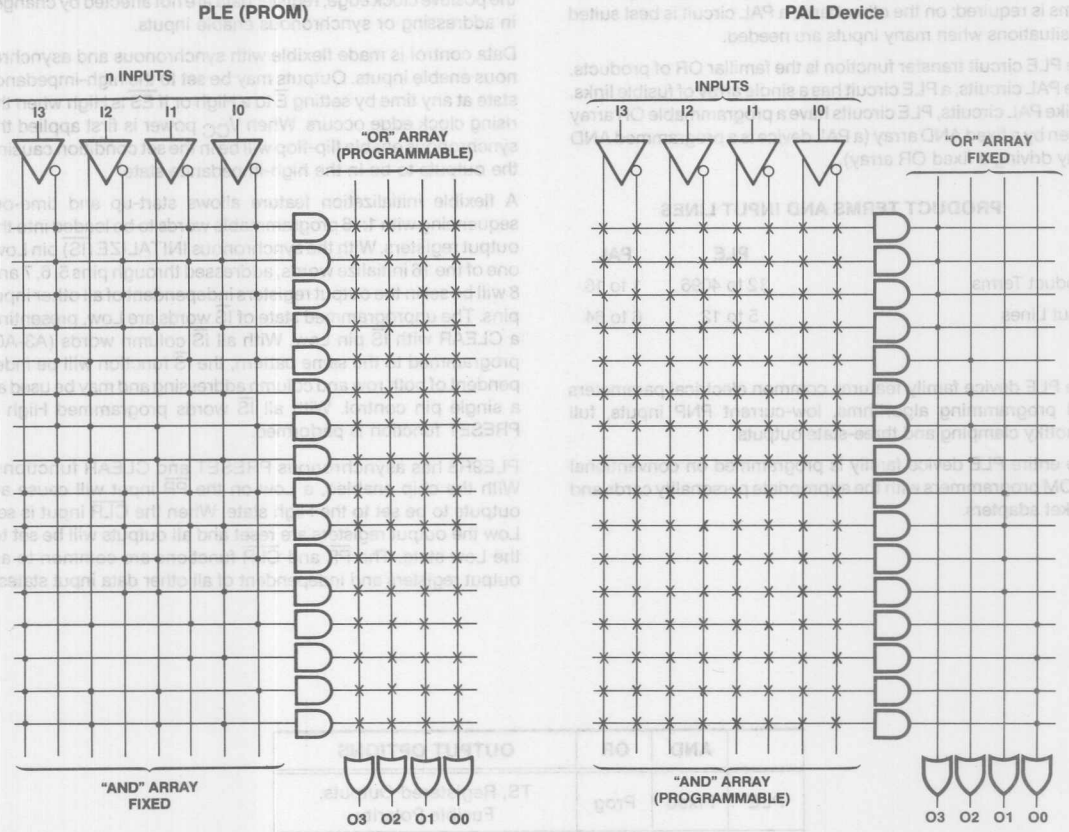
	AND	OR	OUTPUT OPTIONS
PLE	Fixed	Prog	TS, Registered Outputs, Fusible Polarity
FPLA	Prog	Prog	TS, OC, Fusible Polarity
FPGA	Prog	Prog	TS, OC, Fusible Polarity
FPLS	Prog	Prog	TS, Registered Feedback I/O
PAL	Prog	Fixed	TS, Registered Feedback I/O Fusible Polarity

PLEASM Software

Software that makes programmable logic easy.

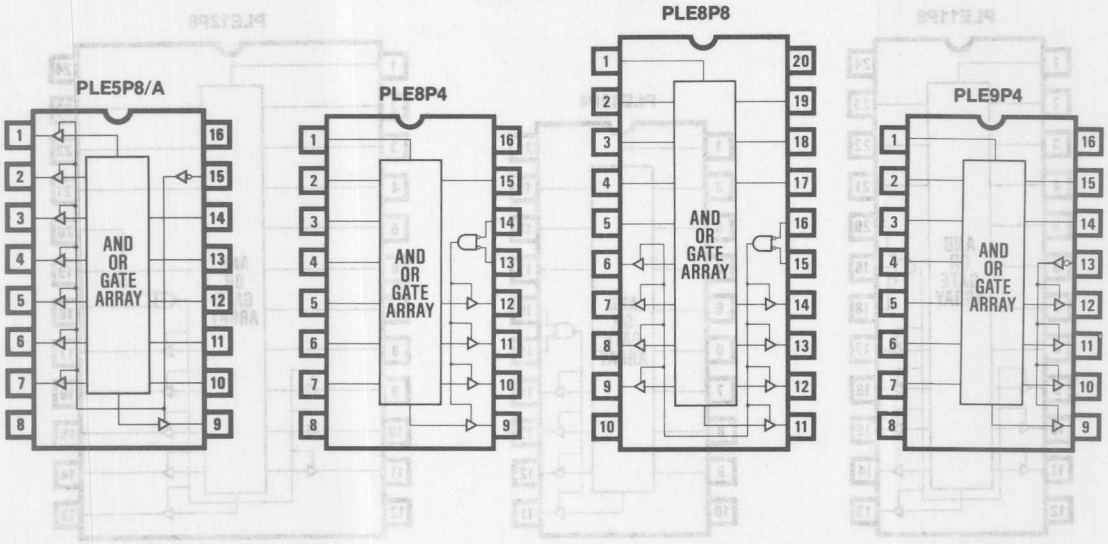
Monolithic Memories has developed a software tool to assist in designing and programming PROMs as PLE devices. This package called "PLEASM" software (PLE Assembler) is available for several computers including the VAX/VMS and IBM PC/DOS. PLEASM software converts design equations (Boolean and arithmetic) into truth tables and formats compatible with PROM programmers. A simulator is also provided to test a design using a Function Table before actually programming the PLE chip.

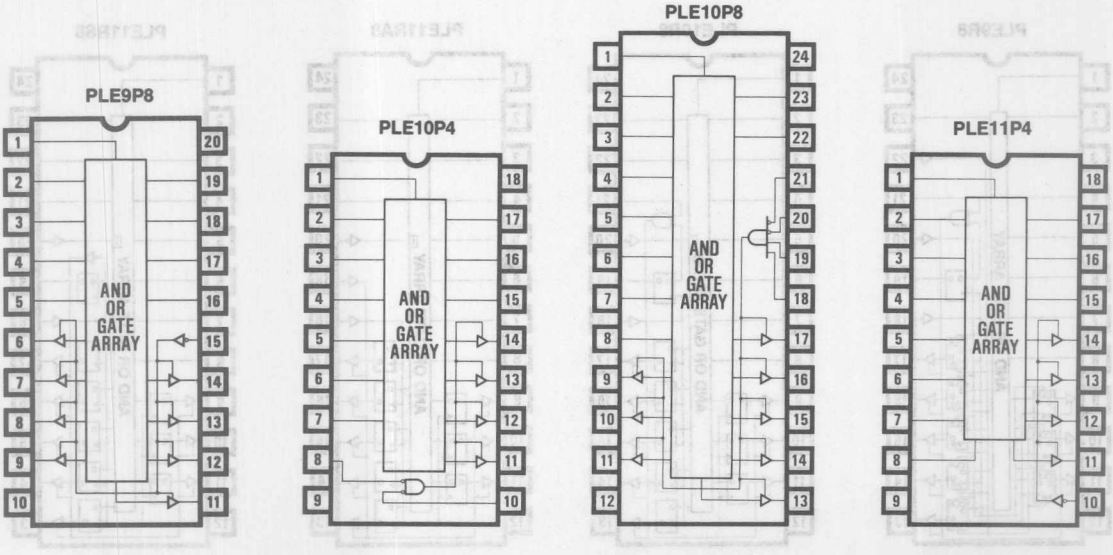
PLEASM software may be requested through the Monolithic Memories IdeaLogic Exchange.



Note: • = Hardwired connection
X = Programmable fuse with a diode

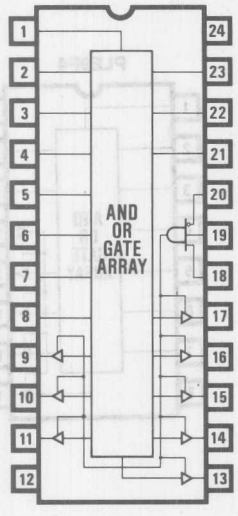
Logic Symbols



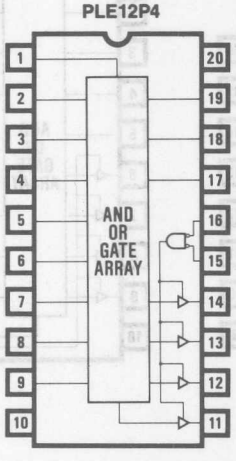


Logic Symbols

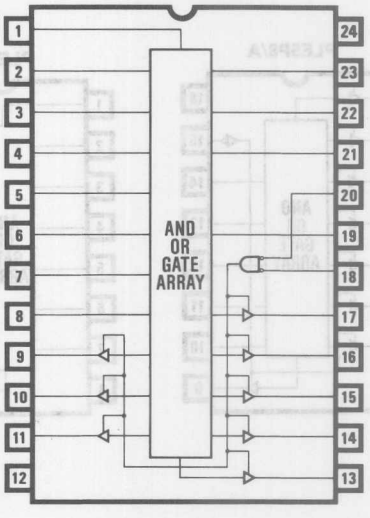
PLE11P8



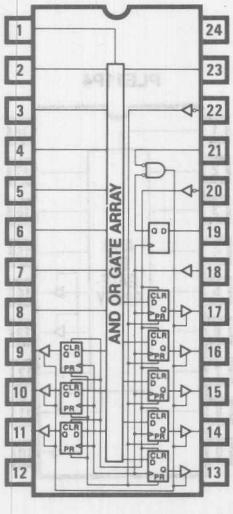
PLE12P4



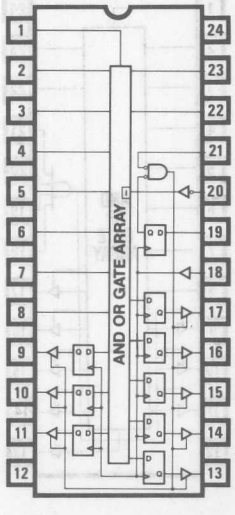
PLE12P8



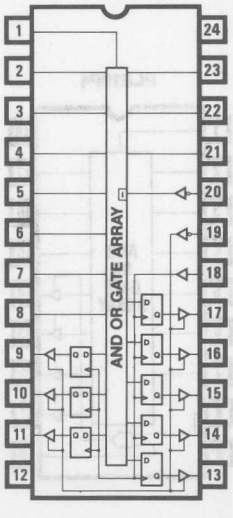
PLE9R8



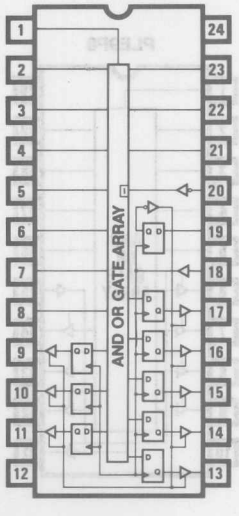
PLE10R8



PLE11RA8



PLE11RS8



Absolute Maximum Ratings

	Operating	Programming
Supply voltage V_{CC}	-0.5 V to 7 V	12 V
Input voltage	-1.5 V to 7 V	7 V
Off-state output voltage	-0.5 V to 5.5 V	12 V
Storage temperature	-65° to +150°C	

Operating Conditions

SYMBOL	PARAMETER	COMMERCIAL MIN	COMMERCIAL TYP*	COMMERCIAL MAX	MILITARY MIN	MILITARY TYP*	MILITARY MAX	UNIT
V_{CC}	Supply voltage	4.75	5	5.25	4.5	5	5.5	V
T_A	Operating free-air temperature	0	25	75	-55	25	125	°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITION			MIN	TYP†	MAX	UNIT
V_{IL}	Low-level input voltage	Guaranteed input logical low voltage for all inputs**					0.8	V
V_{IH}	High-level input voltage	Guaranteed input logical high voltage for all inputs**			2.0			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = -18 mA			-0.8	-1.5	V
				9R8,10R8,11RA/RS			-1.2	
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.4 V			-0.02	-0.25	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = V_{CC}				40	μA
V_{OL}	Low-level output voltage	V_{CC} = MIN	I_{OL} = 16 mA	Com		0.3	45	V
				11P8,12P8,9R8, 10R8,11RA/RS8,Mil		0.3	0.5	
V_{OH}	High-level output voltage	V_{CC} = MIN	Com I_{OH} = -3.2 mA		2.4	2.9		V
			Mil I_{OH} = -2 mA					
I_{OZL}	Off-state output current	V_{CC} = MAX	V_O = 0.4 V				-40	μA
I_{OZH}			V_O = 2.4 V				40	
I_{OS}	Output short-circuit current*	V_{CC} = 5 V	V_O = 0 V		-20	-50	-90	mA
I_{CC}	Supply current	V_{CC} = MAX, All inputs TTL; all outputs open	5P8			90	125	mA
			5P8A			90	125	
			8PA			80	130	
			8P8			90	140	
			9P4			90	130	
			9P8			104	155	
			10P4			95	140	
			10P8			92	170	
			11P4			110	150	
			11P8			135	185	
			12P4			130	175	
			12P8			150	190	
			9R8			130	180	
			10R8			130	180	
			11RA8			140	185	
			11RS8			140	185	

† Vertical at 5.0 V V_{CC} and 25° C T_A.

* Not more than one output should be shorted at a time and duration of the short circuit should not exceed one second.

** VIL and VIM limits are absolute with respect to the device ground pin(s) and include all overshoots due to test equipment noise.

Switching Characteristics Over Military Operating Conditions

DEVICE TYPE	t_{PD} (ns) PROPAGATION DELAY MAX	t_{PZX} AND t_{PXZ} (ns) INPUT TO OUTPUT ENABLE/DISABLE TIME MAX
5P8M	35	30
8P4M	40	30
8P8M	40	30
9P4M	45	30
9P8M	40	30
10P4M	50	30
10P8M	45	30
11P4M	50	30
11P8M	50	30
12P4M	50	30
12P8M	50	35

Switching Characteristics Over Commercial Operating Conditions

DEVICE TYPE	t_{PD} (ns) PROPAGATION DELAY MAX	t_{PZX} AND t_{PXZ} (ns) INPUT TO OUTPUT ENABLE/DISABLE TIME MAX
5P8AC	15	20
5P8C	25	20
8P4C	30	20
8P8C	28	25
9P4C	35	20
9P8C	30	25
10P4C	35	25
10P8C	35	25
11P4C	35	25
11P8C	35	25
12P4C	35	25
12P8C	40	30

Operating Conditions

SYMBOL	PARAMETER	COMMERCIAL MIN	COMMERCIAL TYP*	COMMERCIAL MAX	MILITARY MIN	MILITARY TYP*	MILITARY MAX	UNIT
t_w	Width of clock (High or Low)	20	10		20	10		ns
t_{prw} t_{clrw}	Width of preset or clear (Low) to Output (High or Low)	20	10		20	10		ns
t_{prr} t_{clrr}	Recovery from preset or clear (Low) to clock High	20	11		25	11		ns
t_{su}	Setup time from input to clock	30	22		35	22		ns
$t_s(\overline{ES})$	Setup time from $\overline{ES}$ to clock	10	7		15	7		ns
t_h	Hold time from input to clock	0	-5		0	-5		ns
t_h $(\overline{ES})$	Hold time from $\overline{ES}$ to clock	5	-3		5	-3		ns

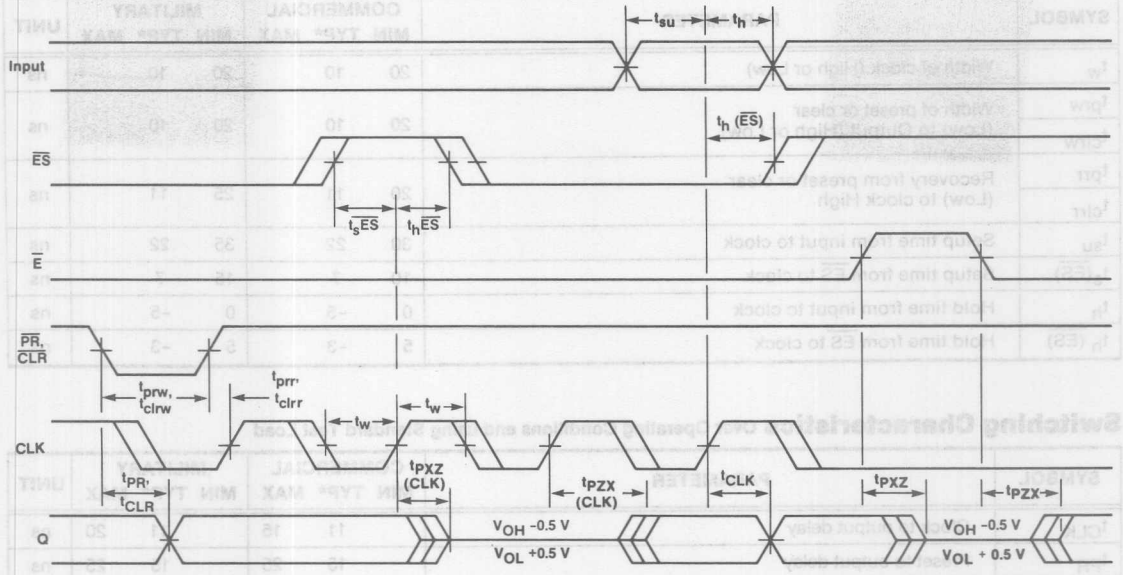
Switching Characteristics Over Operating Conditions and using Standard Test Load

SYMBOL	PARAMETER	COMMERCIAL MIN	COMMERCIAL TYP*	COMMERCIAL MAX	MILITARY MIN	MILITARY TYP*	MILITARY MAX	UNIT
t_{CLK}	Clock to output delay		11	15		11	20	ns
t_{PR}	Preset to output delay		15	25		15	25	ns
t_{CLR}	Clear to output delay		18	25		18	35	ns
t_{PZX} (CLK)	Clock to output enable time		14	25		14	30	ns
t_{PXZ} (CLK)	Clock to output disable time		14	25		14	30	ns
t_{PZX}	Input to output enable time		10	20		10	25	ns
t_{PXZ}	Input to output disable time		10	20		10	25	ns

* Typical at 5.0 V V_{CC} and 25°C T_A.

7

Definition of Waveforms



NOTES:
1. Input pulse amplitude 0 V to 3.0 V.
2. Input rise and fall times 2-5 ns from 0.8 V to 2.0 V.
3. Input access measured at the 1.5 V level.
4. Switch S_1 is closed. C_L = 30 pF and outputs measured at 1.5 V level for all tests except t_{PXZ} and t_{PZX}.
5. t_{PZX} and $t_{PZX(CLK)}$ are measured at the 1.5 V output level with C_L = 30 pF. S_1 is open for high impedance to "1" test and closed for high impedance to "0" test.

   t_{PXZ} and $t_{PXZ(CLK)}$ are tested with C_L = 5 pF. S_1 is open for "1" to high impedance test, measured at V_{OH}-0.5 V output level; S_1 is closed for "0" to high impedance test measured at V_{OL}+0.5 V output level.

Operating Conditions

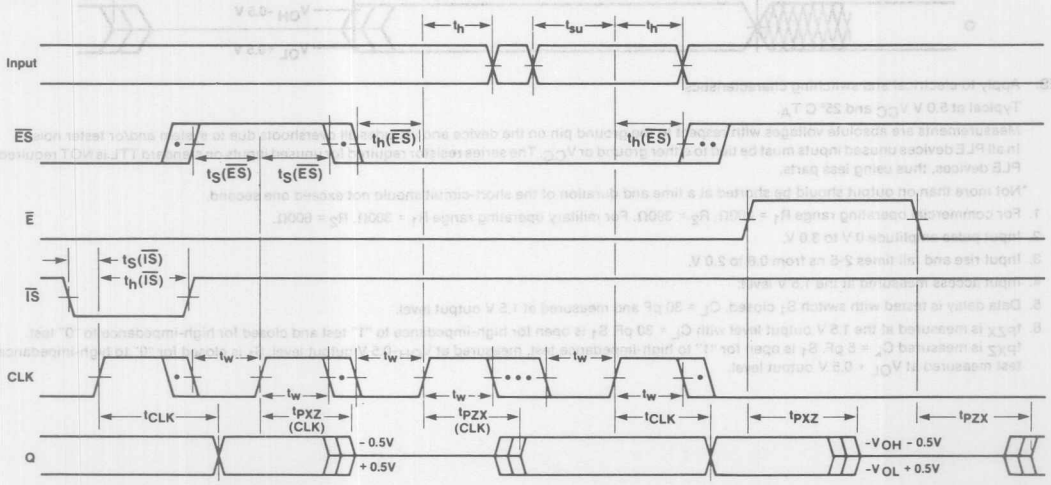
SYMBOL	PARAMETER	COMMERCIAL MIN	COMMERCIAL TYP*	COMMERCIAL MAX	MILITARY MIN	MILITARY TYP*	MILITARY MAX	UNIT
t_w	Width of clock (High or Low)	20	10		20	10		ns
t_{su}	Setup time from input to clock (10R8)	30	25		40	25		ns
t_{su}	Setup time from input to clock (11RA8, 11RS8)	35	28		40	28		ns
t_s ($\overline{ES}$)	Setup time from $\overline{ES}$ to clock	15	7		15	7		ns
t_s ($\overline{IS}$)	Setup time from $\overline{IS}$ to clock	25	20		30	20		ns
t_h	Hold time input to clock	0	-5		0	-5		ns
t_h ($\overline{ES}$)	Hold time ($\overline{ES}$)	5	-3		5	-3		ns
t_h ($\overline{IS}$)	Hold time ($\overline{IS}$)	0	-5		0	-5		ns

Switching Characteristics Over Operating Conditions and using Standard Test Load

SYMBOL	PARAMETER	COMMERCIAL MIN	COMMERCIAL TYP*	COMMERCIAL MAX	MILITARY MIN	MILITARY TYP*	MILITARY MAX	UNIT
t_{CLK}	Clock to output delay		10	15		10	20	ns
t_{PZX} (CLK)	Clock to output enable time		17	25		17	30	ns
t_{PXZ} (CLK)	Clock to output disable time		17	25		17	30	ns
t_{PZX}	Input to output enable time		17	25		17	30	ns
t_{PXZ}	Input to output disable time		17	25		17	30	ns

* Typical at 5.0 V V_{CC} and 25°C T_A.

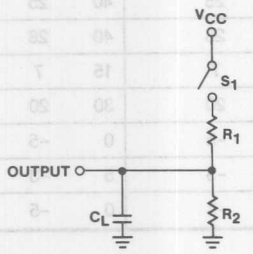
Definition of Waveforms



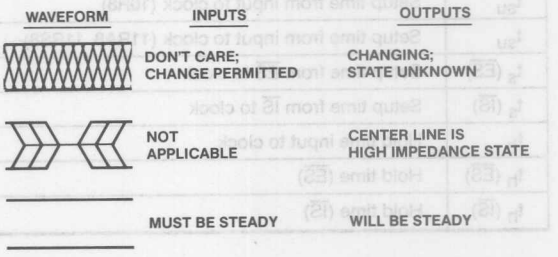
NOTES:
1. Input pulse amplitude 0 V to 3.0 V.
2. Input rise and fall times 2-5 ns from 0.8 V to 2.0 V.
3. Input access measured at the 1.5 V level.
4. Switch S_1 is closed. C_L = 30 pF and outputs measured at 1.5 V level for all tests except t_{PZX} and t_{PXZ}.
5. t_{PZX} and $t_{PZX(CLK)}$ are measured at the 1.5 V output level with C_L = 30 pF. S_1 is open for high impedance to "1" test and closed for high impedance to "0" test.

   t_{PXZ} and $t_{PXZ(CLK)}$ are tested with C_L = 5 pF. S_1 is open for "1" to high impedance test, measured at V_{OH}-0.5 V output level; S_1 is closed for "0" to high impedance test measured at V_{OL}+0.5 V output level.

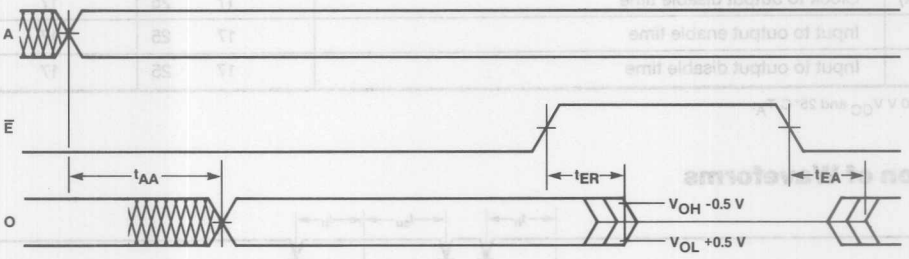
Switching Test Load



Definition of Timing Diagram



Definition of Waveforms



NOTES: Apply to electrical and switching characteristics

Typical at 5.0 V V_{CC} and 25° C T_A.

Measurements are absolute voltages with respect to the ground pin on the device and includes all overshoots due to system and/or tester noise. In all PLE devices unused inputs must be tied to either ground or V_{CC}. The series resistor required for unused inputs on standard TTL is NOT required for PLE devices, thus using less parts.

*Not more than on output should be shorted at a time and duration of the short-circuit should not exceed one second.

1. For commercial operating range R_1 = 200Ω, R_2 = 390Ω. For military operating range R_1 = 300Ω, R_2 = 600Ω.
2. Input pulse amplitude 0 V to 3.0 V.
3. Input rise and fall times 2-5 ns from 0.8 to 2.0 V.
4. Input access measured at the 1.5 V level.
5. Data delay is tested with switch S_1 closed. C_L = 30 pF and measured at 1.5 V output level.
6. t_{PZX} is measured at the 1.5 V output level with C_L = 30 pF. S_1 is open for high-impedance to "1" test and closed for high-impedance to "0" test. t_{PXZ} is measured C_L = 5 pF. S_1 is open for "1" to high-impedance test, measured at V_{OH}-0.5 V output level; S_1 is closed for "0" to high-impedance test measured at V_{OL} + 0.5 V output level.

PLE™ Device Family Programming Instructions

Device Description

All of the members of the PLE device family are manufactured with all outputs LOW in all storage locations. To produce a HIGH as a particular word, a Titanium-Tungsten Fusible-Link must be changed from a low resistance to a high resistance. This procedure is called programming.

Programming Description

To program a particular bit normal TTL levels are applied to all inputs. Programming occurs when:

1. V_{CC} is raised to an elevated level.
2. The output to be programmed is raised to an elevated level.
3. The device is enabled.

In order to avoid misprogramming the PLE device only one output at a time is to be programmed. Outputs not being programmed should be connected to V_{CC} via 5 KΩ resistors.

Unless specified, Inputs should be at VIL.

Programming Sequence

The sequence of programming conditions is critical and must occur in the following order:

1. Select the appropriate address with chip disabled
2. Increase V_{CC} to programming voltage
3. Increase appropriate output voltage to programming voltage
4. Enable chip for programming pulse width
5. Decrease V_{OUT} and V_{CC} to normal levels

Programming Timing

In order to insure the proper sequence, a delay of 100 ns or greater must be allowed between steps. The enabling pulse must not occur less than 100 ns after the output voltage reaches programming level. The rise time of the voltage on V_{CC} and the output must be between 1 and 10 V/μs.

Verification

After each programming pulse verification of the programmed bit should be made with both low and high V_{CC}. The loading of the output is not critical and any loading within the DC specifications of the part is satisfactory.

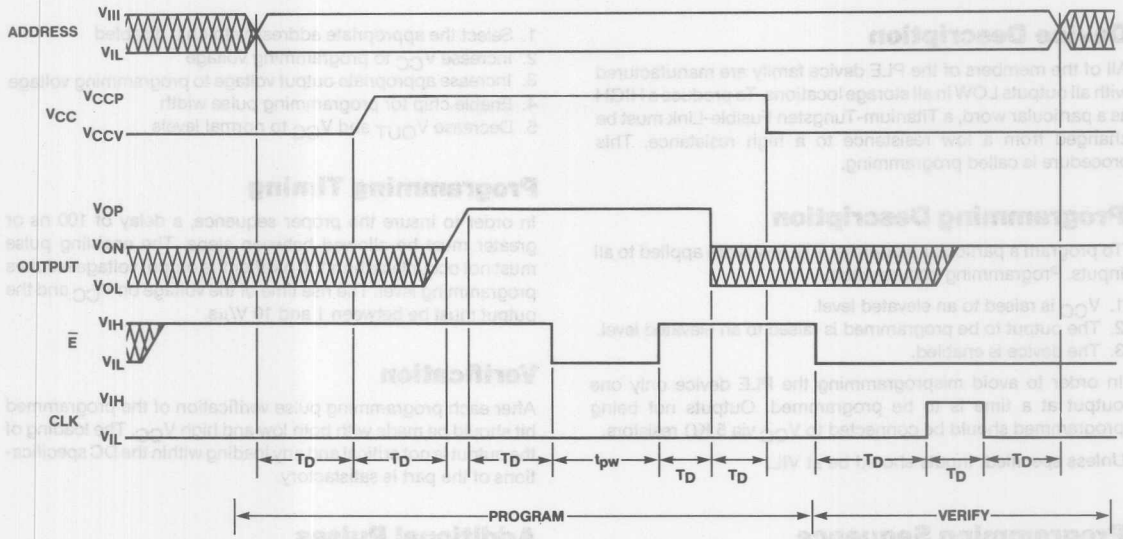
Additional Pulses

Up to 10 programming pulses should be applied until verification indicates that the bit has programmed. Following verification, apply five additional programming pulses to the bit being programmed.

7

Programming Parameters Do not test these parameters or you may program the device

SYMBOL	PARAMETER	RECOMMENDED MIN	RECOMMENDED VALUE	RECOMMENDED MAX	UNIT
V_{CCP}	Required V_{CC} for programming	11.5	11.75	12.0	V
V_{OP}	Required output voltage for programming	10.5	11.0	11.5	V
t_R	Rise time of V_{CC} or V_{OUT}	1.0	5.0	10.0	V/μS
I_{CCP}	Current limit of V_{CCP} supply	800	1200		mA
I_{OP}	Current limit of V_{OP} supply	15	20		mA
t_{PW}	Programming pulse width (enabled)	9	10	11	μS
V_{CC}	Low V_{CC} for verification	4.2	4.3	4.4	V
V_{CC}	High V_{CC} for verification	5.8	6.0	6.2	V
MDC	Maximum duty cycle of V_{CCP}		25	25	%
t_D	Delay time between programming steps	100	120		ns
V_{IL}	Input low level	0	0	0.5	V
V_{IH}	Input high level	2.4	3.0	5.5	V



Programming Equipment and Software Suppliers

Monolithic Memories' PLE devices are designed and tested to give a programming yield greater than 98%. If your programming yield is lower, check your programmer. It may not be properly calibrated.

Programming is final manufacturing — it must be quality-controlled. Equipment must be calibrated as a regular routine, ideally under the actual conditions of use. Each time a new board or a new programming module is inserted, the whole system should be checked. Both timing and voltages must meet published specifications for the device.

Remember — The best PLE devices available can be made unreliable by improper programming techniques.

PROGRAMMERS

Data I/O Corp.
10525 Willows Rd. N.E.
Redmond, WA 98073-9746
(800) 426-1045

Kontron Electronics, Inc.
630 Price Ave.

Digelec Inc.
586 Weddell Dr.
Suite 1,
Sunnyvale, CA 94089
(408) 745-0722

Stag Microsystems Inc.
528-5 Weddell Dr.
Sunnyvale, CA 94089
(408) 745-1991

Varix Corp.
1210 E. Campbell Rd. No. 100

SOFTWARE

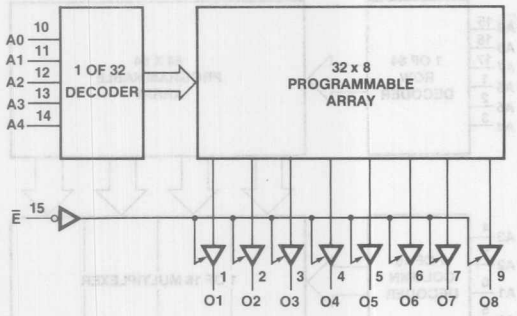
PLEASM
Monolithic Memories
IdeaLogic Exchange
2175 Mission College Blvd. M/S 09-07
Santa Clara, CA 95054
(408) 970-9700 x. 6085

Redmond, WA 98073-9746
(800) 426-1045

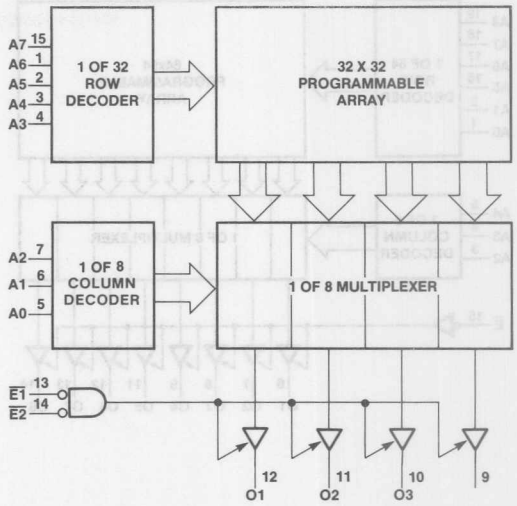
CUPL
Assisted Technology
2381 Zanker Rd. No. 150
San Jose, CA 95131
(408) 942-8787

Block Diagrams

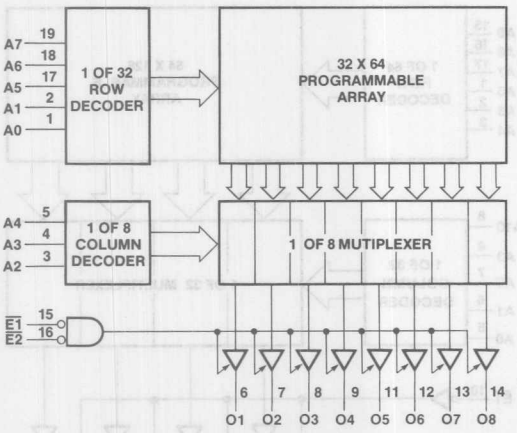
PLE5P8/A



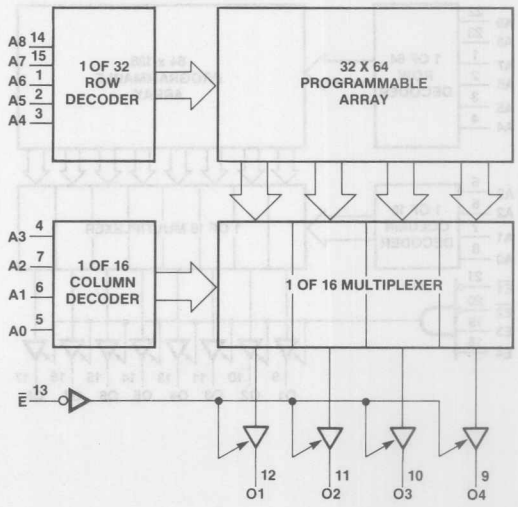
PLE8P4



PLE8P8



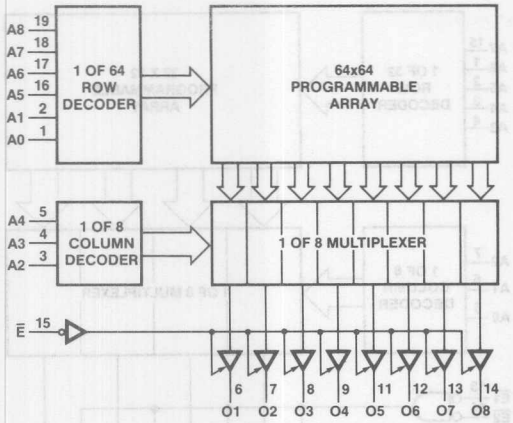
PLE9P4



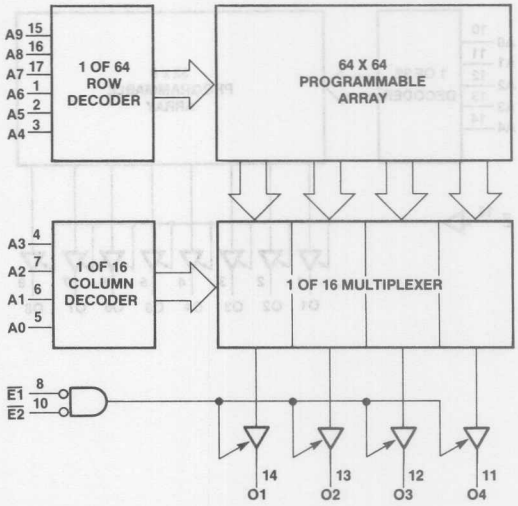
7

Block Diagrams

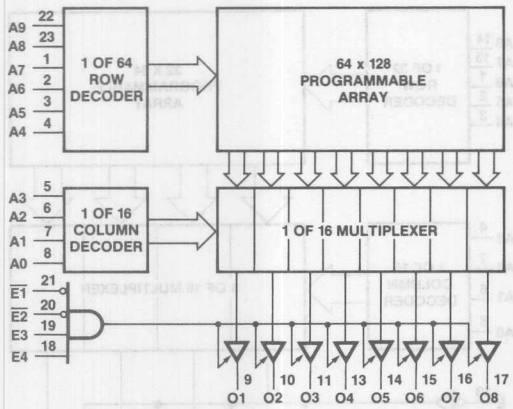
PLE9P8



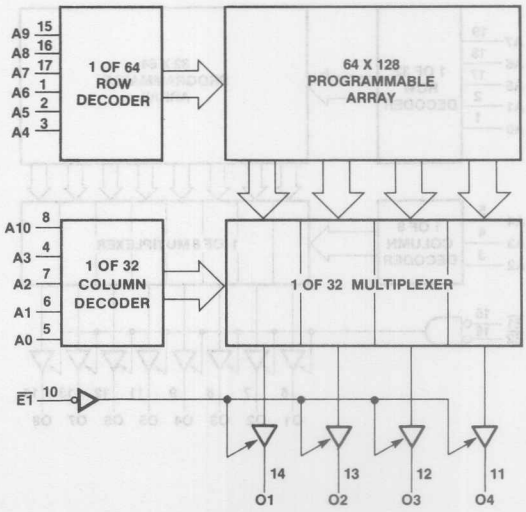
PLE10P4



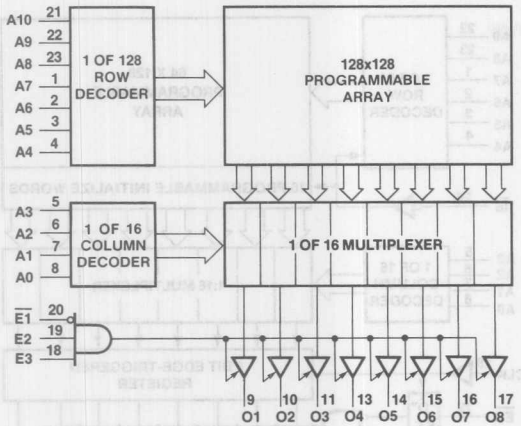
PLE10P8



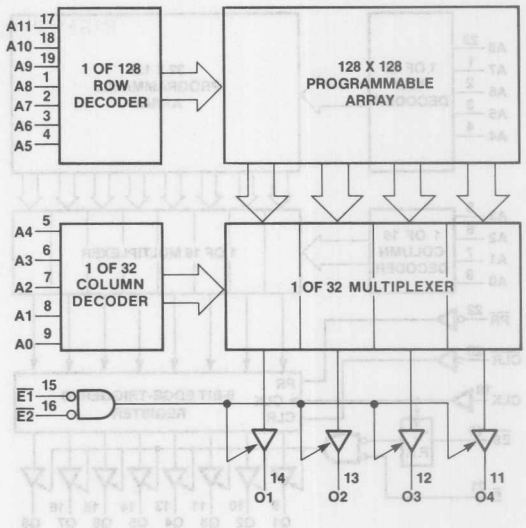
PLE11P4



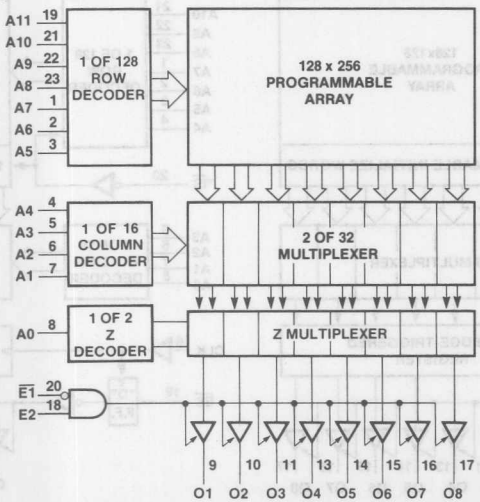
PLE11P8



PLE12P4



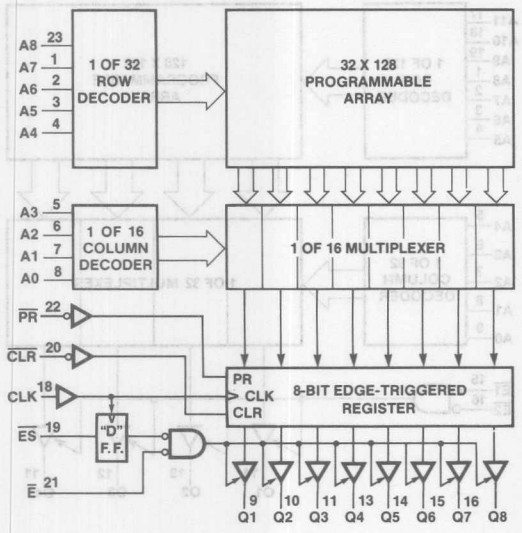
PLE12P8



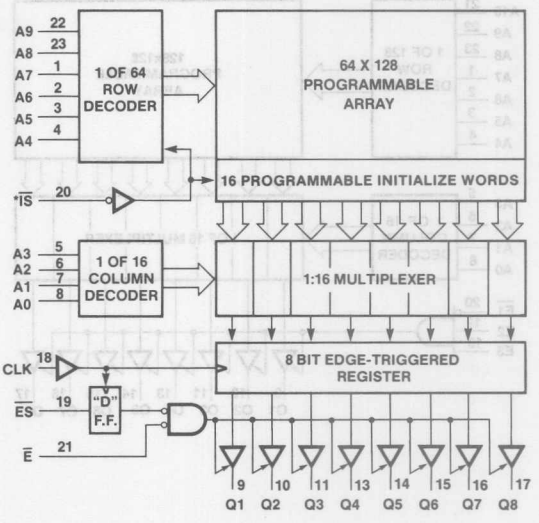
7

Block Diagrams

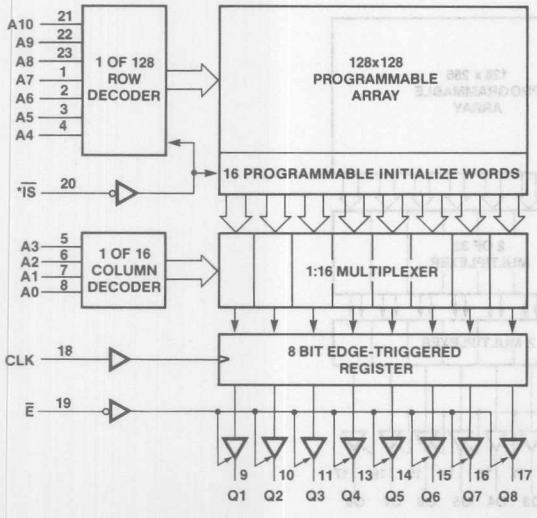
PLE9R8



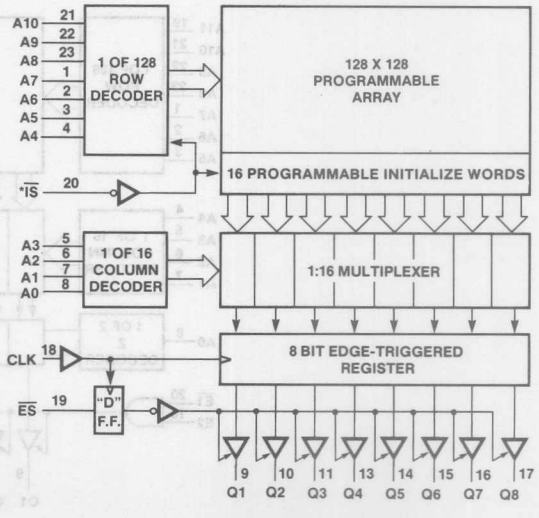
PLE10R8



PLE11RA8



PLE11RS8



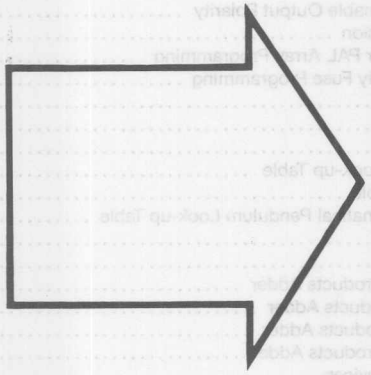
*$\overline{IS}$ selects 1:16 programmable initialization words.

Monolithic Memories PLE Device Programmer Reference Chart

Source and Location	Data I/O 10525 Willow Rd. N.E. Redmond, WA 98073	Kontron Electronics 630 Price Ave. Redwood City, CA 94063	Stag Microsystems 528 Weddell Dr., Suite 1 Sunnyvale, CA 94089	Digelec 586-1 Weddell Dr. Sunnyvale, CA 94089	Varix - Suite 100 1210 E. Campbell Rd. Richardson, TX 75081
Programmer Model(s)	Model 19/29 Model 22	Model MPP-80S	Model PPX Model PP17	UP803	OMNI
MMI Generic Bipolar PROM Personality Module	UniPak Rev 08 UniPak II Rev 05 (Not all PLEs are supported by earlier UniPak revisions)	MOD4		FAM Mod. No. 12	
Socket Adapter(s) and Device Code					
PLE5P8/ PLE5P8A	F18 P02 Model 22A- Adapter 351A-064	SA3	AM110-2 Code 21	DA No. 2 Pinout 1A Switch 0-6	63S081
PLE8P4	F18 P01 Model 22A- Adapter 351A-064	SA4-2	AM130-2 Code 21	DA No. 2 Pinout 1B Switch 0-6	63S141
PLE9P4	F18 P03 Model 22A- Adapter 351A-064	SA4-1	AM130-3 Code 21	DA No. 1 Pinout 1D Switch 2-14	63S241
PLE8P8	F18 P08				
PLE10P4	F18 P05 Model 22A- Adapter 351A-064	SA4	AM140-2 Code 21	DA No. 3 Pinout 1E Switch 0-6	63S441
PLE9P8	F18 P09				
PLE9R8	F18 P65† Model 22A- Adapter 351A-074	SA31-2	†	Pinout 1H† Switch 5-14	†
PLE10P8	F18 P16		MFR29 CODE 32 Model PPZ		
PLE11P4	F18 P06 Model 22A- Adapter 351A-064	SA4-4	AM140-3 Code 21	DA No. Pinout 1L Switch 5-14	63S841
PLE10R8	F18 P86† Model 22A- Adapter 351A-074 (300 mil pkg)		†	DA No. 64† Switch 0-12	†
PLE12P4	F18 P53 Model 22A- Adapter 351A-064	SA20	AM120-6 Code 21	DA No. 70† Switch 4-12	63S1641
PLE11RA8 PLE11RS8	F18 PA3	†	†	†	†
PLE11P8	F18 P21	SA5-4	AM100-5 Code 21	†	63S1681
PLE12P8	F18 P63	†	†	DA No. 64 Pinout 47 Switch 0-4	†

† — Contact manufacturer for availability and programming information


IMPORTANT
NOTES



Contents Section 8

Random Logic Replacement

Random Logic Replacement

PROMs, as logic elements, have been providing solutions as replacements of random logic. This is the concept of PROM as a Programmable Logic Element (PLE) device.

The usages of PLE devices include simple multiplexer/demultiplexer/encoder/decoder, control signal generators, data communications support like CRC, and arithmetic elements like ALUs, multipliers, sine and inverse look-up tables, and applications in signal processing.

The advantages of PLE devices over SSI/MSI logic devices are the flexibility of design and the fast turnaround time which non-programmable devices cannot offer. For example, if a decoder is used to select between memory pages and I/O ports, once a design is done, it will be fixed — it not easy to find a part to be put just in the same place without modification of PC board layout in case the designer wants to expand the memory or to increase the I/O. For a PLE device, what is needed is to program another PLE device and place it in the same socket where the old part was placed. In addition, it can allow designers to define their logic functions in a component.

The AND-OR planar structure of the PLE circuit array lends itself naturally to being viewed as a two-level logic circuit. The fixed AND plane contains all possible combinations of the literals of its inputs. Each combination (product term) is fuse-connected to each output in the programmable OR plane.

A common PLE device application in the control path is to customize logic functions. An n input exclusive OR function is quite commonly required in comparator and adder circuits. It contains 2^{n-1} product terms, which becomes quite large for large values of n. Therefore, it is very convenient to implement large XOR functions in PLE devices.

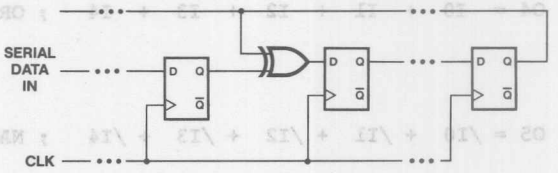
The PLE logic circuit implementation of a 4-input XOR is shown below.

ab \ cd	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Although it seems that XOR functions may be replaced by SSIs, in most applications, the XOR functions will not be alone by themselves, PLE circuits can provide the flexibility of adding in additional functions without using additional packages.

In the data path, a PLE device can be used to implement complex functions such as a Pseudo Random Number (PRN) Generator. Random number sequences are useful in encoding and decoding of information in signal processing and communications systems. They are used for data encryption, image quantization, waveform synchronization, and white noise generation, etc.

There are many techniques for generating PRN sequences. The most common technique, however, is to use 'n' stages of linear shift registers with feedback through a logic function. The function f is an arbitrary function chosen for a specific application. A most general linear function is an 'm' input XOR ($m \leq n$).



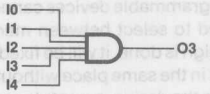
There are a number of examples in the following session which shows how a PLE device can be used to replace SSI/MSI logic devices using PLEASM software.

8

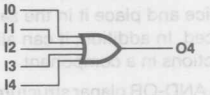
```
PLE5P8                                    PLE CIRCUIT DESIGN SPECIFICATION
P5000                                     VINCENT COLI 10/03/83
BASIC GATES
MMI SANTA CLARA, CALIFORNIA
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8

O1 =  I0                                  ; BUFFER
```


```
O2 = /I0                                  ; INVERTER
```

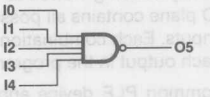

```
O3 =  I0  *  I1  *  I2  *  I3  *  I4      ; AND GATE
```



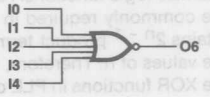

```
O4 =  I0  +  I1  +  I2  +  I3  +  I4      ; OR GATE
```



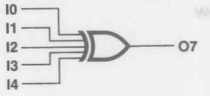

```
O5 = /I0  + /I1  + /I2  + /I3  + /I4      ; NAND GATE
```



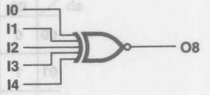

```
O6 = /I0  * /I1  * /I2  * /I3  * /I4      ; NOR GATE
```




```
O7 =  I0 :+: I1 :+: I2 :+: I3 :+: I4      ; EXCLUSIVE OR GATE
```




```
O8 =  I0 :*: I1 :*: I2 :*: I3 :*: I4      ; EXCLUSIVE NOR GATE
```



BASIC GATES (cont'd)

FUNCTION TABLE

```
I0 I1 I2 I3 I4 O1 O2 O3 O4 O5 O6 O7 O8

;INPUT    - -  OUTPUTS  FROM  BASIC  GATES  - -
;01234    BUF  INV  AND  OR  NAND  NOR  XOR  XNOR   COMMENTS
-------------------------------------------------------------------
 LLLLL     L    H    L   L    H     H    L    L     ALL ZEROS
 HHHHH     H    L    H   H    L     L    H    H     ALL ONES
 HLHLH     H    L    L   H    H     L    H    H     ODD CHECKERBOARD
 LHLHL     L    H    L   H    H     L    L    L     EVEN CHECKERBOARD
-------------------------------------------------------------------
```

DESCRIPTION

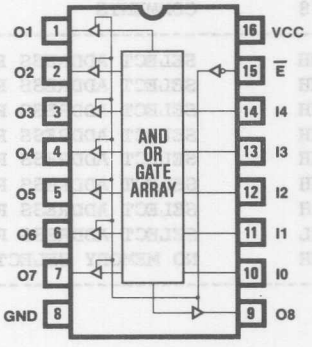
THIS EXAMPLE ILLUSTRATES THE USE OF PLE DEVICES TO IMPLEMENT THE BASIC GATES: BUFFER, INVERTER, AND GATE, OR GATE, NAND GATE, NOR GATE, EXCLUSIVE OR GATE, AND EXCLUSIVE NOR GATE.

NOTE ALSO THAT THREE-STATE OUTPUTS ARE PROVIDED WITH ONE ACTIVE LOW OUTPUT ENABLE CONTROL (/E).

PLEASM SOFTWARE GENERATES THE PROM TRUTH TABLE FROM THE LOGIC EQUATIONS AND SIMULATES THE FUNCTION TABLE IN THE LOGIC EQUATIONS.

BASIC GATES

PLE5P8



8

P5001 ULRIK MUELLER 05/01/84

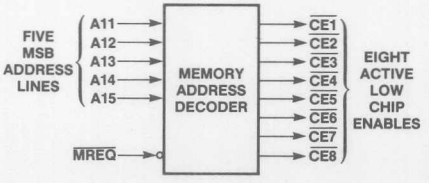
MEMORY ADDRESS DECODER

MMI SANTA CLARA, CALIFORNIA

```
.ADD A11 A12 A13 A14 A15 /MREQ
.DAT /CE1 /CE2 /CE3 /CE4 /CE5 /CE6 /CE7 /CE8

CE1 = /A11*/A12*/A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 0K-2K

CE2 =  A11*/A12*/A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 2K-4K

CE3 = /A11* A12*/A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 4K-6K

CE4 =  A11* A12*/A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 6K-8K

CE5 = /A11*/A12* A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 8K-10K

CE6 =  A11*/A12* A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 10K-12K

CE7 = /A11* A12* A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 12K-14K

CE8 =  A11* A12* A13*/A14*/A15* MREQ      ; SELECTS ADDRESS RANGE 14K-16K
```

FUNCTION TABLE

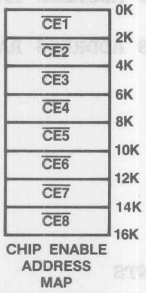
```
A11 A12 A13 A14 A15 /MREQ /CE1 /CE2 /CE3 /CE4 /CE5 /CE6 /CE7 /CE8

; ADD LINES
;   11111              CHIP ENABLES
;   12345      /MREQ     12345678        COMMENTS
-------------------------------------------------------------------------
    LLLLL        L       LHHHHHHH        SELECT ADDRESS RANGE 0-2K
    HLLLL        L       HLHHHHHH        SELECT ADDRESS RANGE 2K-4K
    LHLLL        L       HHLHHHHH        SELECT ADDRESS RANGE 4K-6K
    HHLLL        L       HHHLHHHH        SELECT ADDRESS RANGE 6K-8K
    LLHLL        L       HHHHLHHH        SELECT ADDRESS RANGE 8K-10K
    HLHLL        L       HHHHHLHH        SELECT ADDRESS RANGE 10K-12K
    LHHLL        L       HHHHHHLH        SELECT ADDRESS RANGE 12K-14K
    HHHLL        L       HHHHHHHL        SELECT ADDRESS RANGE 14K-16K
    XXXXX        H       HHHHHHHH        NO MEMORY SELECT (/MREQ=H)
-------------------------------------------------------------------------
```



MEMORY ADDRESS DECODER (cont'd)

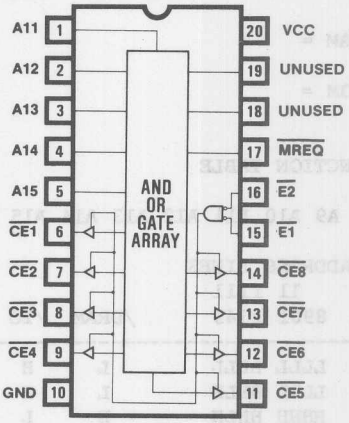
DESCRIPTION

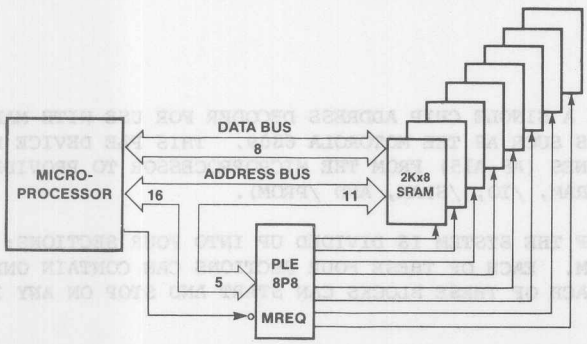
THIS PLE8P8 PROVIDES A SINGLE CHIP ADDRESS DECODER FOR USE WITH MANY POPULAR 8-BIT MICROPROCESSORS SUCH AS THE Z80 AND 8080. THE FIVE MSB ADDRESS LINES (A11-A15) AND THE MEMORY REQUEST LINE (/MREQ) FROM THE Z80 MICROPROCESSOR ARE DECODED TO PRODUCE EIGHT ACTIVE LOW CHIP ENABLES (/CE1-/CE8) TO SELECT A RANGE OF 2K BYTES FROM A BANK OF EIGHT 2Kx8 STATIC RAMS. THIS BANK OF STATIC RAMS WILL OCCUPY THE LOWEST 16K BYTES OF ADDRESS SPACE LEAVING THE UPPER 48K BYTE SPACE AVAILABLE FOR OTHER MEMORIES AND I/O. THE PLE8P8 HAS THREE ADDITIONAL INPUTS WHICH CAN BE RESERVED FOR FUTURE SYSTEM EXPANSION.



MEMORY ADDRESS DECODER

PLE8P8

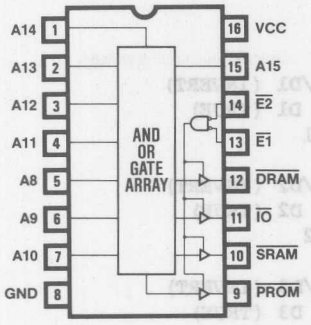




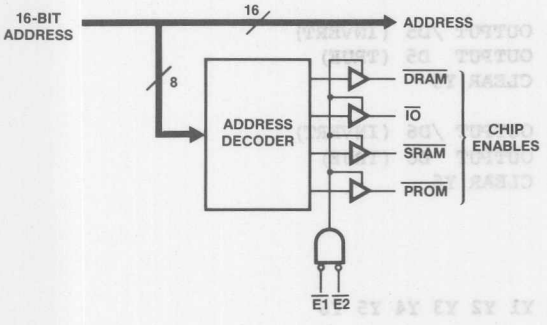
8

```
PLE8P4                                          PLE CIRCUIT DESIGN SPECIFICATION
P5029                                                    VINCENT COLI 10/13/84
6809 ADDRESS DECODER
MMI SANTA CLARA, CALIFORNIA
.ADD A8 A9 A10 A11 A12 A13 A14 A15
.DAT /DRAM /IO /SRAM /PROM

DRAM = /A8*          A12* A13*/A14* A15  ; SELECTS ADDRESS RANGE 0000-BEFF
     +    /A9*       A12* A13*/A14* A15
     +        /A10*   A12* A13*/A14* A15
     +             /A11* A12* A13*/A14* A15
     +                  /A12*     /A14
     +                       /A13*/A14
     +                                /A15

IO   =  A8* A9* A10* A11* A12* A13*/A14* A15  ; SELECTS ADDRESS RANGE BF00-BFFF

SRAM =                       /A13* A14* A15  ; SELECTS ADDRESS RANGE C000-DFFF

PROM =                        A13* A14* A15  ; SELECTS ADDRESS RANGE E000-FFFF


FUNCTION TABLE

A8 A9 A10 A11 A12 A13 A14 A15 /DRAM /IO /SRAM /PROM

; ADDRESS LINES
;     11 1111
;   8901 2345       /DRAM  /IO  /SRAM  /PROM      COMMENTS
-------------------------------------------------------------------------------
    LLLL LLLL         L     H     H      H        00XX HEX SELECTS DRAMS
    LLLL HHLH         L     H     H      H        B0XX HEX SELECTS DRAMS
    HHHH HHLH         H     L     H      H        BFXX HEX SELECTS I/O PORTS
    LLLL LLHH         H     H     L      H        C0XX HEX SELECTS SRAM
    LLLL HLHH         H     H     L      H        D0XX HEX SELECTS SRAM
    LLLL LHHH         H     H     H      L        E0XX HEX SELECTS PROM
    HHHH HHHH         H     H     H      L        FFXX HEX SELECTS PROM
-------------------------------------------------------------------------------


DESCRIPTION

THIS PLE8P4 PROVIDES A SINGLE CHIP ADDRESS DECODER FOR USE WITH MANY POPULAR
8-BIT MICROPROCESSORS SUCH AS THE MOTOROLA 6809.  THIS PLE DEVICE DECODES THE
EIGHT MSB ADDRESS LINES (A8-A15) FROM THE MICROPROCESSOR TO PROVIDE FOUR ACTIVE
LOW CHIP ENABLES (/DRAM, /IO, /SRAM, AND /PROM).

THE 64K MEMORY MAP OF THE SYSTEM IS DIVIDED UP INTO FOUR SECTIONS:  DRAM, IO
PORTS, SRAM, AND PROM.  EACH OF THESE FOUR SECTIONS CAN CONTAIN ONE OR MORE
BLOCKS OF MEMORY.  EACH OF THESE BLOCKS CAN START AND STOP ON ANY 256 BIT
BOUNDARY
```

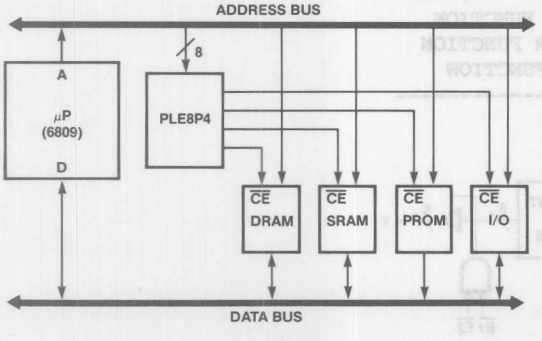
6809 Address Decoder
PLE8P4



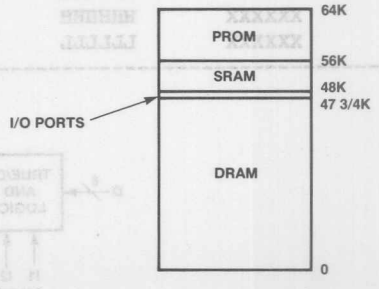
Block Diagram



System Diagram

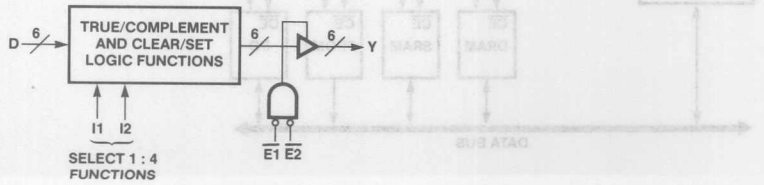


Memory Map



8

```
PLE8P8                                          PLE CIRCUIT DESIGN SPECIFICATION
P5002                                           JOEL ROSENBERG 10/26/83
6-BIT TRUE/COMPLEMENT AND CLEAR/SET LOGIC FUNCTIONS
MMI SANTA CLARA, CALIFORNIA
.ADD I1 I2 D1 D2 D3 D4 D5 D6
.DAT Y1 Y2 Y3 Y4 Y5 Y6


Y1 = /I1*/I2*/D1        ; OUTPUT /D1 (INVERT)
   + /I1* I2* D1        ; OUTPUT  D1 (TRUE)
   +  I1*/I2            ; CLEAR Y1

Y2 = /I1*/I2*/D2        ; OUTPUT /D2 (INVERT)
   + /I1* I2* D2        ; OUTPUT  D2 (TRUE)
   +  I1*/I2            ; CLEAR Y2

Y3 = /I1*/I2*/D3        ; OUTPUT /D3 (INVERT)
   + /I1* I2* D3        ; OUTPUT  D3 (TRUE)
   +  I1*/I2            ; CLEAR Y3

Y4 = /I1*/I2*/D4        ; OUTPUT /D4 (INVERT)
   + /I1* I2* D4        ; OUTPUT  D4 (TRUE)
   +  I1*/I2            ; CLEAR Y4

Y5 = /I1*/I2*/D5        ; OUTPUT /D5 (INVERT)
   + /I1* I2* D5        ; OUTPUT  D5 (TRUE)
   +  I1*/I2            ; CLEAR Y5

Y6 = /I1*/I2*/D6        ; OUTPUT /D6 (INVERT)
   + /I1* I2* D6        ; OUTPUT  D6 (TRUE)
   +  I1*/I2            ; CLEAR Y6


FUNCTION TABLE

I1 I2 D1 D2 D3 D4 D5 D6 Y1 Y2 Y3 Y4 Y5 Y6

;CONTROL     INPUT  D     OUTPUT Y
; LINES       123456       123456       COMMENTS
-------------------------------------------------------
   LL         LHLHLH       HLHLHL       INVERT FUNCTION
   LH         LHLHLH       LHLHLH       TRUE FUNCTION
   HL         XXXXXX       HHHHHH       CLEAR FUNCTION
   HH         XXXXXX       LLLLLL       SET FUNCTION
-------------------------------------------------------
```



TRUE/COMPLEMENT AND CLEAR/SET (cont'd)

DESCRIPTION

THIS PLE8P8 IS A 6-BIT TRUE/COMPLEMENT AND CLEAR/SET LOGIC FUNCTIONS. THE CONTROL LINES (I1 AND I2) SELECT ONE OF FOUR LOGIC FUNCTIONS FOR THE 6-BIT INPUT DATA (D1-D6) AND THE 6-BIT OUTPUT FUNCTION (Y1-Y6).

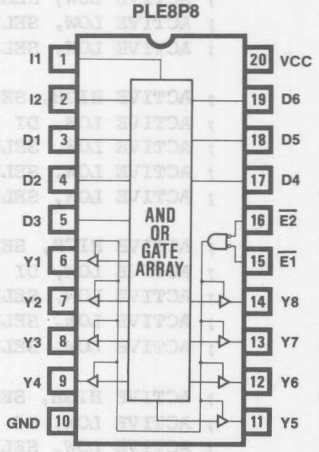
WHEN I1 IS FALSE (I1=LOW) THE FUNCTION IS INVERT IF I2 IS FALSE (I2=LOW) OR TRUE IF I2 IS TRUE (I2=HIGH).

WHEN I1 IS TRUE (I1=HIGH) THE FUNCTION IS CLEAR IF I2 IS FALSE (I2=LOW) OR SET IF I2 IS TRUE (I2=HIGH).

THE PLE8P8 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE LOW OUTPUT ENABLE CONTROLS (/E1 AND /E2).

I1	I2	D1-D6	Y1-Y6	FUNCTION
L	L	D	/D	INVERT
L	H	D	D	TRUE
H	L	X	H	CLEAR
H	H	X	L	SET

6-BIT TRUE/COMPLEMENT ZERO/ONE LOGIC FUNCTIONS



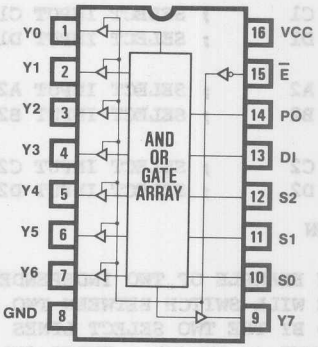
8

```
PLE5P8                                  PLE CIRCUIT DESIGN SPECIFICATION
P5003                                                  FRANK LEE 04/15/84
EXPANDABLE 3-TO-8 DEMULTIPLEXER
MMI SANTA CLARA, CALIFORNIA
.ADD S0 S1 S2 DI PO
.DAT Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7


Y0 =  PO *  DI * /S2 * /S1 * /S0        ; ACTIVE HIGH, SELECT 0
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       *  S2                    ; ACTIVE LOW, SELECT 4-7
   + /PO             *  S1              ; ACTIVE LOW, SELECT 2,3,6,7
   + /PO                   *  S0        ; ACTIVE LOW, SELECT 1,3,5,7

Y1 =  PO *  DI * /S2 * /S1 *  S0        ; ACTIVE HIGH, SELECT 1
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       *  S2                    ; ACTIVE LOW, SELECT 4,5,6,7
   + /PO             *  S1              ; ACTIVE LOW, SELECT 2,3,6,7
   + /PO                   * /S0        ; ACTIVE LOW, SELECT 0,2,4,6

Y2 =  PO *  DI * /S2 *  S1 * /S0        ; ACTIVE HIGH, SELECT 2
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       *  S2                    ; ACTIVE LOW, SELECT 4-7
   + /PO             * /S1              ; ACTIVE LOW, SELECT 0,1,4,5
   + /PO                   *  S0        ; ACTIVE LOW, SELECT 1,3,5,7

Y3 =  PO *  DI * /S2 *  S1 *  S0        ; ACTIVE HIGH, SELECT 3
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       *  S2                    ; ACTIVE LOW, SELECT 4-7
   + /PO             * /S1              ; ACTIVE LOW, SELECT 0,1,4,5
   + /PO                   * /S0        ; ACTIVE LOW, SELECT 0,2,4,6

Y4 =  PO *  DI *  S2 * /S1 * /S0        ; ACTIVE HIGH, SELECT 4
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       * /S2                    ; ACTIVE LOW, SELECT 0-3
   + /PO             *  S1              ; ACTIVE LOW, SELECT 2,3,6,7
   + /PO                   *  S0        ; ACTIVE LOW, SELECT 1,3,5,7

Y5 =  PO *  DI *  S2 * /S1 *  S0        ; ACTIVE HIGH, SELECT 5
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       * /S2                    ; ACTIVE LOW, SELECT 0-3
   + /PO             *  S1              ; ACTIVE LOW, SELECT 2,3,6,7
   + /PO                   * /S0        ; ACTIVE LOW, SELECT 0,2,4,6

Y6 =  PO *  DI *  S2 *  S1 * /S0        ; ACTIVE HIGH, SELECT 6
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       * /S2                    ; ACTIVE LOW, SELECT 0-3
   + /PO             * /S1              ; ACTIVE LOW, SELECT 0,1,4,5
   + /PO                   *  S0        ; ACTIVE LOW, SELECT 1,3,5,7

Y7 =  PO *  DI *  S2 *  S1 *  S0        ; ACTIVE HIGH, SELECT 7
   + /PO *  DI                          ; ACTIVE LOW, DI INACTIVE
   + /PO       * /S2                    ; ACTIVE LOW, SELECT 0-3
   + /PO             * /S1              ; ACTIVE LOW, SELECT 0,1,4,5
   + /PO                   * /S0        ; ACTIVE LOW, SELECT 0,2,4,6
```

```
PO DI S2 S1 S0 Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0

;       SSS    YYYYYYYY
;PO DI  210    76543210      COMMENTS
-----------------------------------------
 H  L   XXX    LLLLLLLL      DATA INPUT = 0
 H  H   LLL    LLLLLLLH      SELECT OUTPUT 0
 H  H   LLH    LLLLLLHL      SELECT OUTPUT 1
 H  H   LHL    LLLLLHLL      SELECT OUTPUT 2
 H  H   LHH    LLLLHLLL      SELECT OUTPUT 3
 H  H   HLL    LLLHLLLL      SELECT OUTPUT 4
 H  H   HLH    LLHLLLLL      SELECT OUTPUT 5
 H  H   HHL    LHLLLLLL      SELECT OUTPUT 6
 H  H   HHH    HLLLLLLL      SELECT OUTPUT 7
 L  H   XXX    HHHHHHHH      DATA INPUT = 0
 L  L   LLL    HHHHHHHL      SELECT OUTPUT 0
 L  L   LLH    HHHHHHLH      SELECT OUTPUT 1
 L  L   LHL    HHHHHLHH      SELECT OUTPUT 2
 L  L   LHH    HHHHLHHH      SELECT OUTPUT 3
 L  L   HLL    HHHLHHHH      SELECT OUTPUT 4
 L  L   HLH    HHLHHHHH      SELECT OUTPUT 5
 L  L   HHL    HLHHHHHH      SELECT OUTPUT 6
 L  L   HHH    LHHHHHHH      SELECT OUTPUT 7
-----------------------------------------
```

EXPANDABLE
3-TO-8 DEMULTIPLEXER

PLE5P8



DESCRIPTION

THIS PLE5P8 IMPLEMENTS AN EXPANDABLE 3-TO-8 DEMULTIPLEXER. THE DEVICE DEMULTIPLEXES THREE SELECT INPUT SIGNALS (S2-S0) INTO EIGHT OUTPUTS (Y7-Y0) USING THE INPUT DI WITH POLARITY SELECT PO. SINCE THE DEVICE HAS THREE-STATE OUTPUTS, IT CAN BE EXPANDED USING THE ACTIVE LOW ENABLE PIN (/E).

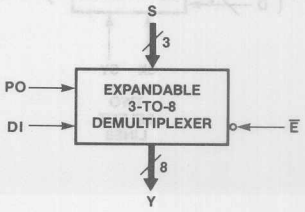
8

PIN ASSIGNMENTS:

1. PO — HIGH INDICATES OUTPUT IS ACTIVE HIGH. LOW INDICATES OUTPUT IS ACTIVE LOW.
2. DI — DATA INPUT (DEMULTIPLEXING INPUT). ACTIVE LOW IF PO IS LOW.
3. S2-S0 — SELECT PINS. S2 IS THE MOST SIGNIFICANT BIT. ACTIVE HIGH REGARDLESS OF PO.
4. Y7-Y0 — OUTPUTS. CAN BE ACTIVE HIGH OR ACTIVE LOW DEPENDING ON PO. ACTIVE LOW IF PO IS LOW.

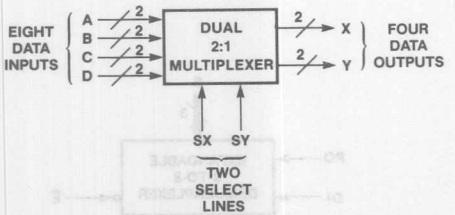
OPERATIONS TABLE:

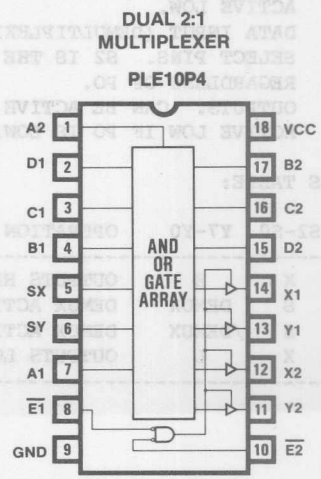
PO	DI	S2-S0	Y7-Y0	OPERATION
L	H	X	H	OUTPUTS HIGH
H	H	S	DEMUX	DEMUX ACTIVE HIGH
L	L	S	/DEMUX	DEMUX ACTIVE LOW
H	L	X	L	OUTPUTS LOW



```
DUAL 2:1 MULTIPLEXER
MMI SANTA CLARA, CALIFORNIA
.ADD SX SY A1 B1 C1 D1 A2 B2 C2 D2
.DAT X1 Y1 X2 Y2


X1 = /SX* A1       ; SELECT INPUT A1
   +  SX* B1       ; SELECT INPUT B1

Y1 = /SY* C1       ; SELECT INPUT C1
   +  SY* D1       ; SELECT INPUT D1

X2 = /SX* A2       ; SELECT INPUT A2
   +  SX* B2       ; SELECT INPUT B2

Y2 = /SY* C2       ; SELECT INPUT C2
   +  SY* D2       ; SELECT INPUT D2

DESCRIPTION

THIS IS AN EXAMPLE OF TWO INDEPENDENT 2-TO-1 MULTIPLEXERS USING A PLE10P4.
THE DEVICE WILL SWITCH BETWEEN TWO PAIRS OF 2-BIT INPUTS (A, B AND C, D), AS
DETERMINED BY THE TWO SELECT LINES (SX, SY), FOR OUTPUT THROUGH TWO PAIRS OF
2-BIT OUTPUTS (X AND Y).  THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH TWO
ACTIVE LOW ENABLE PINS (/E1 AND /E2).  THE FUNCTIONS OF THE DEVICE ARE
SUMMARIZED IN THE TABLE BELOW:
```

SELECT LINES		INPUT A, B				INPUT C, D					OUTPUT X, Y				
S	S	A	A	B	B	C	C	D	D	!	X	Y	X	Y	
X	Y	1	2	1	2	1	2	1	2	!	1	1	2	2	FUNCTION
L	L	A1	A2	X	X	C1	C2	X	X	!	A1	C1	A2	C2	SELECT A, C
L	H	A1	A2	X	X	X	X	D1	D2	!	A1	D1	A2	D2	SELECT A, D
H	L	X	X	B1	B2	C1	C2	X	X	!	B1	B2	C1	C2	SELECT B, C
H	H	X	X	B1	B2	X	X	D1	D2	!	B1	D1	B2	D2	SELECT B, D





```
PLE10P4                          PLE CIRCUIT DESIGN SPECIFICATION
P5005                                          S. HORIKO 04/29/84
QUAD 2:1 MULTIPLEXER WITH POLARITY CONTROL
MMI JAPAN
.ADD SEL POL A0 A1 A2 A3 B0 B1 B2 B3
.DAT Y0 Y1 Y2 Y3

Y0 = /SEL*/POL*/A0                ; SELECT INPUT /A0 (COMP)
   + /SEL* POL* A0                ; SELECT INPUT  A0 (TRUE)
   +  SEL*/POL*/B0                ; SELECT INPUT /B0 (COMP)
   +  SEL* POL* B0                ; SELECT INPUT  B0 (TRUE)

Y1 = /SEL*/POL*/A1                ; SELECT INPUT /A1 (COMP)
   + /SEL* POL* A1                ; SELECT INPUT  A1 (TRUE)
   +  SEL*/POL*/B1                ; SELECT INPUT /B1 (COMP)
   +  SEL* POL* B1                ; SELECT INPUT  B1 (TRUE)

Y2 = /SEL*/POL*/A2                ; SELECT INPUT /A2 (COMP)
   + /SEL* POL* A2                ; SELECT INPUT  A2 (TRUE)
   +  SEL*/POL*/B2                ; SELECT INPUT /B2 (COMP)
   +  SEL* POL* B2                ; SELECT INPUT  B2 (TRUE)

Y3 = /SEL*/POL*/A3                ; SELECT INPUT /A3 (COMP)
   + /SEL* POL* A3                ; SELECT INPUT  A3 (TRUE)
   +  SEL*/POL*/B3                ; SELECT INPUT /B3 (COMP)
   +  SEL* POL* B3                ; SELECT INPUT  B3 (TRUE)

FUNCTION TABLE

SEL POL A0 A1 A2 A3 B0 B1 B2 B3 Y0 Y1 Y2 Y3

; SELECT    AAAA BBBB    YYYY
;SEL  POL   0123 0123    0123      COMMENTS
-------------------------------------------------------------
  L    L    LLLL XXXX    HHHH      SELECT COMP INPUT /A=00
  L    L    LHLH XXXX    HLHL      SELECT COMP INPUT /A=05
  L    L    HLHL XXXX    LHLH      SELECT COMP INPUT /A=10
  L    L    HHHH XXXX    LLLL      SELECT COMP INPUT /A=15
  L    H    LLLL XXXX    LLLL      SELECT TRUE INPUT  A=00
  L    H    LHLH XXXX    LHLH      SELECT TRUE INPUT  A=05
  L    H    HLHL XXXX    HLHL      SELECT TRUE INPUT  A=10
  L    H    HHHH XXXX    HHHH      SELECT TRUE INPUT  A=15
  H    L    XXXX LLLL    HHHH      SELECT COMP INPUT /B=00
  H    L    XXXX LHLH    HLHL      SELECT COMP INPUT /B=05
  H    L    XXXX HLHL    LHLH      SELECT COMP INPUT /B=10
  H    L    XXXX HHHH    LLLL      SELECT COMP INPUT /B=15
  H    H    XXXX LLLL    LLLL      SELECT TRUE INPUT  B=00
  H    H    XXXX HLHL    HLHL      SELECT TRUE INPUT  B=05
  H    H    XXXX LHLH    LHLH      SELECT TRUE INPUT  B=10
  H    H    XXXX HHHH    HHHH      SELECT TRUE INPUT  B=15
-------------------------------------------------------------
```

QUAD 2:1 MULTIPLEXER WITH POLARITY CONTROL (cont'd)

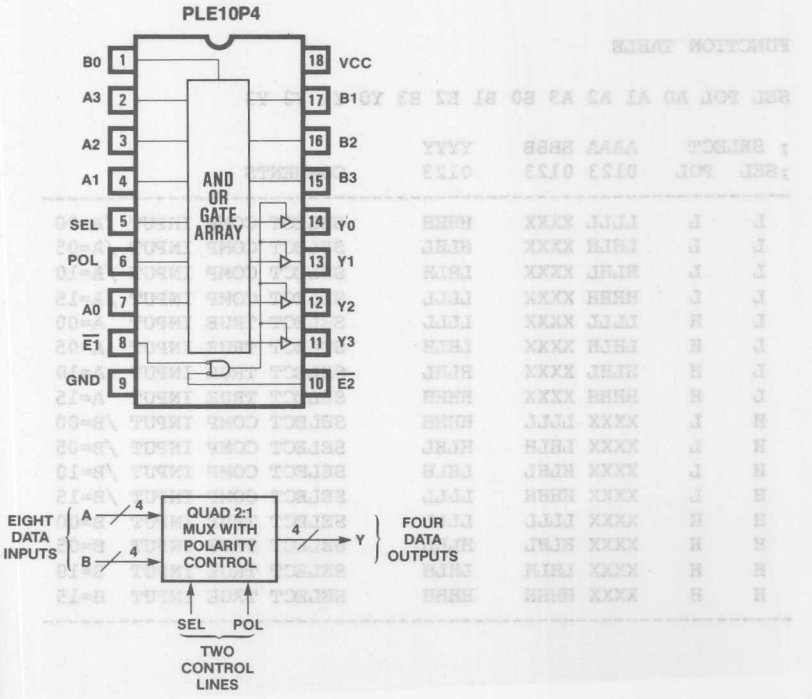
DESCRIPTION

THIS IS AN EXAMPLE OF A QUAD 2:1 MULTIPLEXER WITH POLARITY CONTROL IMPLEMENTED IN A PLE10P4. THE DEVICE SELECTS BETWEEN TWO 4-BIT INPUTS (A1-A4 AND B1-B4) WHICH ARE DIRECTED TO ONE 4-BIT OUTPUT (Y1-Y4) AS DETERMINED BY ONE INPUT SELECT LINE (SEL) AND POLARITY CONTROL (POL). WHEN POLARITY IS TRUE (POL=HIGH), THE TRUE OF THE INPUT SIGNAL IS SELECTED. WHEN POLARITY IS FALSE (POL=LOW), THE COMPLEMENT OF THE INPUT SIGNAL IS SELECTED.

THE PLE10P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE LOW ENABLE PINS (/E1 AND /E2). THE FUNCTION IS SUMMARIZED BELOW:

SEL	POL	A1-A4	B1-B4	Y1-Y4
L	H	A	X	A
L	L	A	X	/A
H	H	X	B	B
H	L	X	B	/B

QUAD 2:1 MULTIPLEXER WITH POLARITY CONTROL



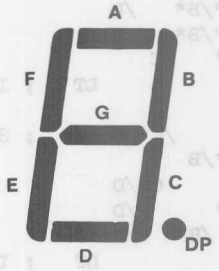
```
PLE5P8                                  PLE CIRCUIT DESIGN SPECIFICATION
P5006                                                ULRIK MUELLER 04/29/84
HEXADECIMAL TO SEVEN SEGMENT DECODER
MMI SANTA CLARA, CALIFORNIA
.ADD A B C D LT
.DAT /OA /OB /OC /OD /OE /OF /OG /DP

OA =     B*   /D      ; SEGMENT A
   +     B* C
   + /A*   /C*/D
   +  A*    C*/D
   + /A*       D
   +    /B*/C* D
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT A

OB =       /C*/D      ; SEGMENT B
   +  A* B*   /D
   + /A*/B*   /D
   +  A*/B*    D
   + /A*   /C
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT B

OC =       /C* D      ; SEGMENT C
   +  A*/B
   +       C*/D
   +  A*     /D
   +    /B*  /D
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT C

OD = /A*/B*/C         ; SEGMENT D
   +    /B*    D
   +  A*/B* C
   +  A* B*/C
   + /A* B* C
   + /A* B*  /D
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT D

OE = /A*   /C         ; SEGMENT E
   +        C* D
   + /A* B
   +  A* B*    D
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT E

OF = /A*/B            ; SEGMENT F
   +    /B* C*/D
   +       /C* D
   +     B*    D
   + /A* B* C
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT F

OG =     B*/C         ; SEGMENT G
   + /A* B
   +       /C* D
   +  A*       D
   +    /B* C*/D
   +             LT    ; IF LT=H MAKE BLANK TEST ON SEGMENT G

DP = LT               ; TURNS DP ON ONLY WHEN LT=H
```

DESCRIPTION

THIS EXAMPLE ILLUSTRATES THE USE OF A PLE5P8 AS A HEXADECIMAL TO SEVEN SEGMENT DECODER. THE DEVICE DECODES A 4-BIT BINARY INPUT (D,C,B,A) INTO THE SEVEN SEGMENT OUTPUTS NEEDED TO DRIVE AN LED DISPLAY. NOTE THAT THIS DESIGN IS AN IMPROVEMENT FROM THE 74LS47 SINCE ALL 16 HEXADECIMAL DIGITS (0-F) CAN BE DISPLAYED. A LAMP TEST IS PROVIDED TO ILLUMINATE ALL SEVEN SEGMENTS AND THE DECIMAL POINT (IF DP IS CONNECTED) BY BRINGING LAMP TEST HIGH (LT=HIGH) REGARDLESS OF THE OTHER BINARY INPUTS. THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH ONE ACTIVE LOW ENABLE PIN (/E).

INPUT DIGIT	LT	INPUT D C B A	SEGMENT ON	OUTPUT DISPLAY
0	L	L L L L	ABCDEF	0
1	L	L L L H	BC	1
2	L	L L H L	ABDEG	2
3	L	L L H H	ABCDG	3
4	L	L H L L	BCDFG	4
5	L	L H L H	ACDFG	5
6	L	L H H L	ACDEFG	6
7	L	L H H H	ABC	7
8	L	H L L L	ABCDEFG	8
9	L	H L L H	ABCFG	9
A	L	H L H L	ABCEFG	A
B	L	H L H H	CDEFG	b
C	L	H H L L	ADEF	C
D	L	H H L H	BCDEG	d
E	L	H H H L	ADEFG	E
F	L	H H H H	AEFG	F
X	H	X X X X	ABCDEFG	8 *

SEGMENT IDENTIFICATION

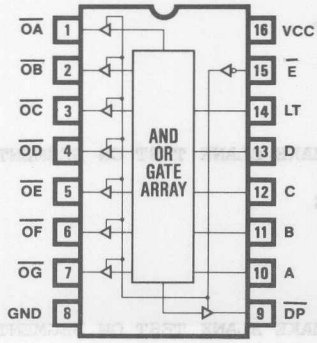


CHARACTER SET

* BLANK TEST OF DISPLAY

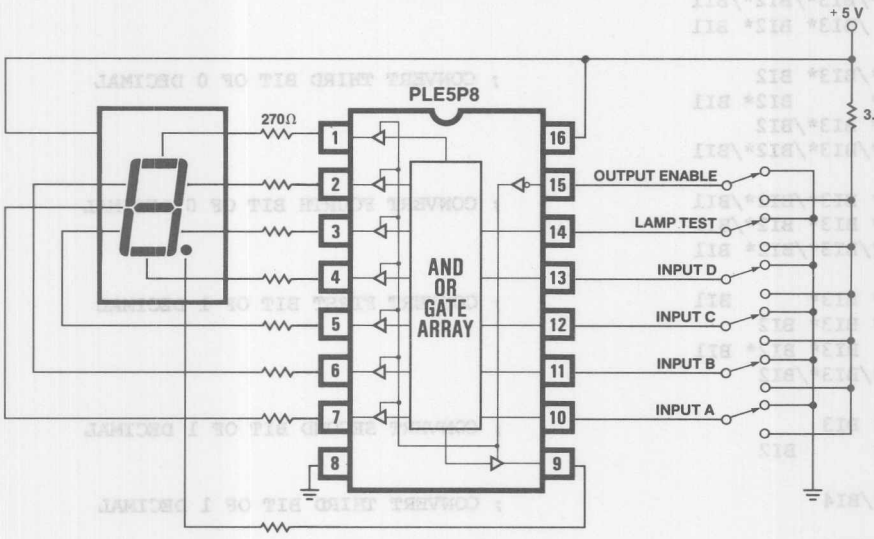
HEXADECIMAL TO SEVEN-SEGMENT DECODER

PLE5P8



HEXADECIMAL TO SEVEN-SEGMENT DECODER (cont'd)

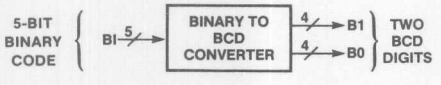
HEXADECIMAL TO SEVEN-SEGMENT DECODER



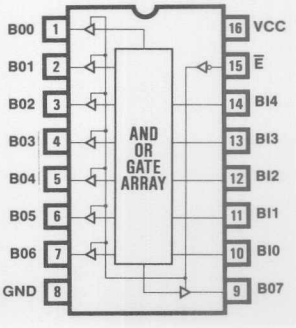
8

```
PLE5P8                              PLE CIRCUIT DESIGN SPECIFICATION
P5007                                          VINCENT COLI 02/03/82
5-BIT BINARY TO BCD CONVERTER
MMI SANTA CLARA, CALIFORNIA
.ADD BI0 BI1 BI2 BI3 BI4
.DAT B00 B01 B02 B03 B10 B11 B12 B13


B00 =  BI0                          ; CONVERT FIRST BIT OF 0 DECIMAL (LSB)

B01 = /BI4*/BI3*      BI1           ; CONVERT SECOND BIT OF 0 DECIMAL
    + /BI4* BI3* BI2*/BI1
    +  BI4* BI3*/BI2* BI1
    +  BI4*/BI3*/BI2*/BI1
    +      /BI3* BI2* BI1

B02 = /BI4*/BI3* BI2                ; CONVERT THIRD BIT OF 0 DECIMAL
    + /BI4*      BI2* BI1
    +  BI4* BI3*/BI2
    +  BI4*/BI3*/BI2*/BI1

B03 = /BI4* BI3*/BI2*/BI1           ; CONVERT FOURTH BIT OF 0 DECIMAL
    +  BI4* BI3* BI2*/BI1
    +  BI4*/BI3*/BI2* BI1

B10 = /BI4* BI3*      BI1           ; CONVERT FIRST BIT OF 1 DECIMAL
    + /BI4* BI3* BI2
    +       BI3* BI2* BI1
    +  BI4*/BI3*/BI2

B11 =  BI4* BI3                     ; CONVERT SECOND BIT OF 1 DECIMAL
    +  BI4*      BI2

B12 =  BI4*/BI4                     ; CONVERT THIRD BIT OF 1 DECIMAL

B13 =  BI4*/BI4                     ; CONVERT FOURTH BIT OF 1 DECIMAL (MSB)
```

5-BIT BINARY TO BCD CONVERTER



PLE5P8



FUNCTION TABLE

BI4 BI3 BI2 BI1 BI0 B13 B12 B11 B10 B03 B02 B01 B00

;ADDRESS ;BINARY ;43 210	----DATA---- BCD 1 3210	BCD 0 3210	DESCRIPTION (DECIMAL VALUE)
LL LLL	LLLL	LLLL	0
LL LLH	LLLL	LLLH	1
LL LHH	LLLL	LLHH	3
LL LHL	LLLL	LLHL	2
LL HHL	LLLL	LHHL	6
LL HHH	LLLL	LHHH	7
LL HLH	LLLL	LHLH	5
LL HLL	LLLL	LHLL	4
;			
LH LLL	LLLL	HLLL	8
LH LLH	LLLL	HLLH	9
LH LHH	LLLH	LLLH	11
LH LHL	LLLH	LLLL	10
LH HHL	LLLH	LHLL	14
LH HHH	LLLH	LHLH	15
LH HLH	LLLH	LLHH	13
LH HLL	LLLH	LLHL	12
;			
HL LLL	LLLH	LHHL	16
HL LLH	LLLH	LHHH	17
HL LHH	LLLH	HLLH	19
HL LHL	LLLH	HLLL	18
HL HHL	LLHL	LLHL	22
HL HHH	LLHL	LLHH	23
HL HLH	LLHL	LLLH	21
HL HLL	LLHL	LLLL	20
;			
HH LLL	LLHL	LHLL	24
HH LLH	LLHL	LHLH	25
HH LHH	LLHL	LHHH	27
HH LHL	LLHL	LHHL	26
HH HHL	LLHH	LLLL	30
HH HHH	LLHH	LLLH	31
HH HLH	LLHL	HLLH	29
HH HLL	LLHL	HLLL	28

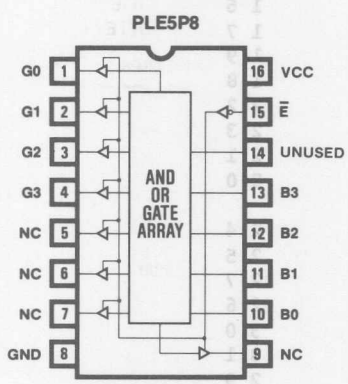
DESCRIPTION

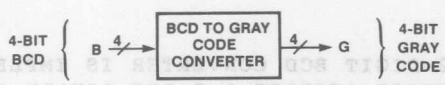
THIS 5-BIT BINARY TO 2-DIGIT BCD CONVERTER IS IMPLEMENTED IN A PLE5P8 LOGIC CIRCUIT. THE DEVICE ACCEPTS A 5-BIT BINARY INPUT (BI) AND CONVERTS THIS INTO TWO 4-BIT BINARY CODED DECIMAL (BCD) OUTPUTS (B1 AND B0).

THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH ONE ACTIVE LOW ENABLE PIN (/E).

```
PLE5P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5008                                    VINCENT COLI 10/16/81
4-BIT BCD TO GRAY CODE CONVERTER
MMI SANTA CLARA, CALIFORNIA
.ADD B0 B1 B2 B3
.DAT G0 G1 G2 G3


G0 = B0 :+: B1                          ; CONVERT G0 (LSB)

G1 = B1 :+: B2                          ; CONVERT G1

G2 = B2 :+: B3                          ; CONVERT G2

G3 = B3                                 ; CONVERT G3 (MSB)


DESCRIPTION

THIS PLE5P8 WILL CONVERT A 4-BIT BCD INPUT (B3-B0) INTO A 4-BIT
GRAY CODE REPRESENTATION (G3-G0) FOR OUTPUT.
```

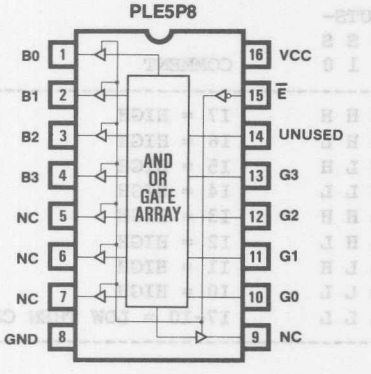
4-BIT BCD TO GRAY CODE CONVERTER

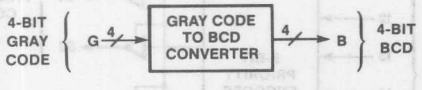




```
PLE5P8                    PLE CIRCUIT DESIGN SPECIFICATION
P5009                                VINCENT COLI 03/16/84
4-BIT GRAY CODE TO BCD CONVERTER
MMI SANTA CLARA, CALIFORNIA
.ADD G0 G1 G2 G3
.DAT B0 B1 B2 B3


B0 = G0 :+: G1 :+: G2 :+: G3      ; CONVERT B0 (LSB)

B1 = G1 :+: G2 :+: G3             ; CONVERT B1

B2 = G2 :+: G3                    ; CONVERT B2

B3 = G3                           ; CONVERT B3 (MSB)


DESCRIPTION

THIS PLE5P8 WILL CONVERT A 4-BIT GRAY CODE INPUT (G3-G0) INTO A 4-BIT
BINARY REPRESENTATION (B3-B0) FOR OUTPUT.
```

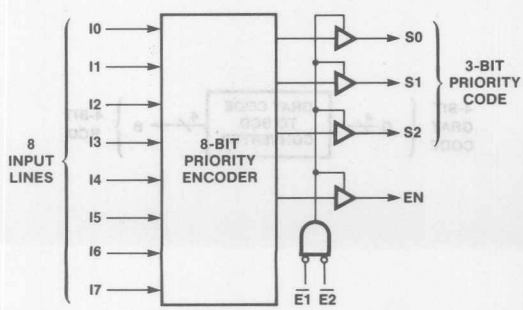
4-BIT GRAY CODE TO BCD CONVERTER





8

```
P5010                                   FRANK LEE/ULRIK MUELLER 05/14/84
8-BIT PRIORITY ENCODER
MMI SANTA CLARA, CALIFORNIA
.ADD I0 I1 I2 I3 I4 I5 I6 I7
.DAT S0 S1 S2 EN


S0 =  I7                                ; I7-I0 = 1XXXXXXX
    + /I6* I5                           ; I7-I0 = X01XXXXX
    + /I6*/I4* I3                       ; I7-I0 = X0X01XXX
    + /I6*/I4*/I2* I1                   ; I7-I0 = X0X0X01X

S1 =  I7                                ; I7-I0 = 1XXXXXXX
    +  I6                               ; I7-I0 = X1XXXXXX
    + /I5*/I4* I3                       ; I7-I0 = XX001XXX
    + /I5*/I4* I2                       ; I7-I0 = XX00X1XX

S2 =  I7                                ; I7-I0 = 1XXXXXXX
    +  I6                               ; I7-I0 = X1XXXXXX
    +  I5                               ; I7-I0 = XX1XXXXX
    +  I4                               ; I7-I0 = XXX1XXXX

EN = /I0*/I1*/I2*/I3*/I4*/I5*/I6*/I7    ; ALL LOWS ENABLE NEXT PRIORITY ENCODER


FUNCTION TABLE

I7 I6 I5 I4 I3 I2 I1 I0 EN S2 S1 S0

; --INPUT LINES--      -OUTPUTS-
; I  I  I  I  I  I  I  I   E   S  S  S
; 7  6  5  4  3  2  1  0   N   2  1  0     COMMENT
-------------------------------------------------------------------
  H  X  X  X  X  X  X  X   L   H  H  H     I7 = HIGH
  L  H  X  X  X  X  X  X   L   H  H  L     I6 = HIGH
  L  L  H  X  X  X  X  X   L   H  L  H     I5 = HIGH
  L  L  L  H  X  X  X  X   L   H  L  L     I4 = HIGH
  L  L  L  L  H  X  X  X   L   L  H  H     I3 = HIGH
  L  L  L  L  L  H  X  X   L   L  H  L     I2 = HIGH
  L  L  L  L  L  L  H  X   L   L  L  H     I1 = HIGH
  L  L  L  L  L  L  L  H   L   L  L  L     I0 = HIGH
  L  L  L  L  L  L  L  L   H   L  L  L     I7-I0 = LOW THEN CARRY OUT
-------------------------------------------------------------------
```



8-BIT PRIORITY ENCODER (cont'd)

DESCRIPTION

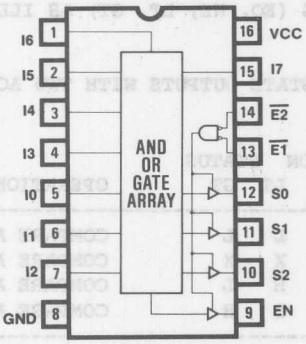
THIS 8-BIT PRIORITY ENCODER SCANS FOR THE FIRST HIGH INPUT LINE (I7-I0) FROM I7 (WHICH HAS THE HIGHEST PRIORITY) TO I0 (WHICH HAS THE LOWEST PRIORITY). IT WILL GENERATE A BINARY ENCODED OUTPUT (S2-S0) WHICH WILL POINT TO THE HIGHEST PRIORITY INPUT WHICH IS AT A HIGH STATE.

IF NO INPUT LINES ARE HIGH (I7-I0=LOW), THEN THE BINARY ENCODED OUTPUTS WILL BE ZERO (S2-S0=LOW) AND THE ENABLE OUTPUT WILL BE HIGH (EN=HIGH) INDICATING A CARRY OUT TO THE NEXT PRIORITY ENCODER. THE OUTPUT ENABLE WILL BE LOW (EN=LOW) IF ANY OF THE INPUT LINES ARE HIGH.

THE PLE8P4 ALSO HAS THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

8-BIT PRIORITY ENCODER

PLE8P4



```
PLE8P4                                    PLE CIRCUIT DESIGN SPECIFICATION
P5011                                                ULRIK MUELLER 04/01/83
4-BIT MAGNITUDE COMPARATOR
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 A2 A3 B0 B1 B2 B3
.DAT EQ NE LT GT

EQ =  A3:*:B3 *  A2:*:B2 *  A1:*:B1 *  A0:*:B0 ; A = B

NE =  A3:+:B3 +  A2:+:B2 +  A1:+:B1 +  A0:+:B0 ; A NOT = B

LT = /A3 * B3                                   ; A3 LT B3
   +  A3:*:B3 * /A2 * B2                        ; A2 LT B2
   +  A3:*:B3 *  A2:*:B2 * /A1 * B1             ; A1 LT B1
   +  A3:*:B3 *  A2:*:B2 *  A1:*:B1 * /A0 * B0 ; A0 LT B0

GT =  A3 */B3                                   ; A3 GT B3
   +  A3:*:B3 *  A2 */B2                        ; A2 GT B2
   +  A3:*:B3 *  A2:*:B2 *  A1 */B1             ; A1 GT B1
   +  A3:*:B3 *  A2:*:B2 *  A1:*:B1 *  A0 */B0 ; A0 GT B0
```

DESCRIPTION

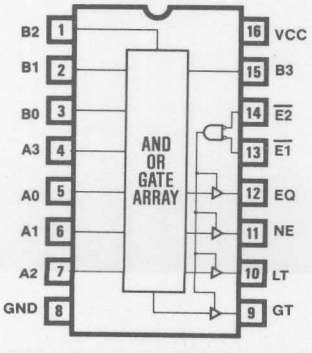
THIS PLE8P4 COMPARES TWO 4-BIT NUMBERS (A3-A0 AND B3-B0) TO ESTABLISH IF THEY ARE EQUAL (A = B THEN EQ=H), NOT EQUAL (A NOT = B THEN NE=H), LESS THAN (A LT B THEN LT=H), OR GREATER THAN (A GT B THEN GT=H) AND REPORTS THE COMPARISON STATUS ON THE OUTPUTS (EQ, NE, LT, GT) AS ILLUSTRATED IN THE OPERATIONS TABLE BELOW.

THE PLE8P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

INPUT NUMBERS A3-A0	B3-B0	COMPARISON STATUS EQ	NE	LT	GT	OPERATION
A	= B	H	L	L	L	COMPARE A EQUAL TO B
A NOT	= B	L	H	X	X	COMPARE A NOT EQUAL TO B
A	LT B	L	H	H	L	COMPARE A LESS THAN B
A	GT B	L	H	L	H	COMPARE A GREATER THAN B

4-BIT MAGNITUDE COMPARATOR

PLE8P4



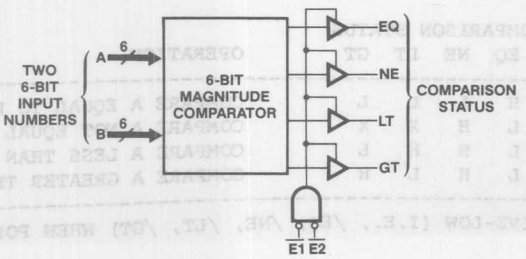
```
PLE12P4                                    PLE CIRCUIT DESIGN SPECIFICATION
P5012                                                VINCENT COLI 10/16/83
6-BIT MAGNITUDE COMPARATOR
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 A2 A3 A4 A5 B0 B1 B2 B3 B4 B5
.DAT EQ NE LT GT

EQ =  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2:*:B2 *  A1:*:B1 *  A0:*:B0 ; A = B

NE =  A5:+:B5 +  A4:+:B4 +  A3:+:B3 +  A2:+:B2 +  A1:+:B1 +  A0:+:B0 ; A NOT= B

LT = /A5 * B5                                                          ; A5 LT B5
   +  A5:*:B5 * /A4 * B4                                               ; A4 LT B4
   +  A5:*:B5 *  A4:*:B4 * /A3 * B3                                    ; A3 LT B3
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 * /A2 * B2                         ; A2 LT B2
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2:*:B2 * /A1 * B1              ; A1 LT B1
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2:*:B2 *  A1:*:B1 * /A0 * B0  ; A0 LT B0

GT =  A5 */B5                                                          ; A5 GT B5
   +  A5:*:B5 *  A4 */B4                                               ; A4 GT B4
   +  A5:*:B5 *  A4:*:B4 *  A3 */B3                                    ; A3 GT B3
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2 */B2                         ; A2 GT B2
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2:*:B2 *  A1 */B1              ; A1 GT B1
   +  A5:*:B5 *  A4:*:B4 *  A3:*:B3 *  A2:*:B2 *  A1:*:B1 *  A0 */B0  ; A0 GT B0
```

DESCRIPTION

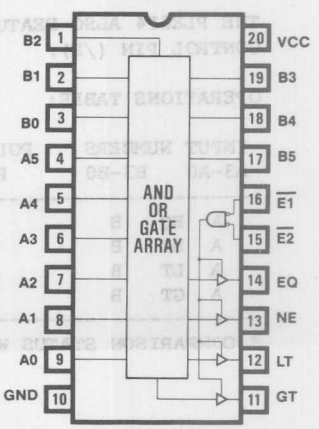
THIS PLE12P4 COMPARES TWO 6-BIT NUMBERS (A5-A0 AND B5-B0) TO ESTABLISH IF THEY ARE EQUAL (A = B THEN EQ=H), NOT EQUAL (A NOT = B THEN NE=H), LESS THAN (A LT B THEN LT=H), OR GREATER THAN (A GT B THEN GT=H) AND REPORTS THE COMPARISON STATUS ON THE OUTPUTS (EQ, NE, LT, GT) AS ILLUSTRATED IN THE OPERATIONS TABLE BELOW.

THE PLE12P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

INPUT NUMBERS A5-A0	INPUT NUMBERS B5-B0	COMPARISON STATUS EQ	NE	LT	GT	OPERATION
A	= B	H	L	L	L	COMPARE A EQUAL TO B
A NOT	= B	L	H	X	X	COMPARE A NOT EQUAL TO B
A	LT B	L	H	H	L	COMPARE A LESS THAN B
A	GT B	L	H	L	H	COMPARE A GREATER THAN B



6-BIT MAGNITUDE COMPARATOR
PLE12P4



8

```
PLE9P4                                  PLE CIRCUIT DESIGN SPECIFICATION
P5011A                                                  COLI/MUELLER 09/09/84
4-BIT MAGNITUDE COMPARATOR WITH POLARITY CONTROL
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 A2 A3 B0 B1 B2 B3 POL
.DAT EQ NE LT GT

EQ =  A3:*:B3* POL *  A2:*:B2* POL *  A1:*:B1* POL *  A0:*:B0* POL  ; A  EQ B
   +  A3:+:B3*/POL *  A2:+:B2*/POL *  A1:+:B1*/POL *  A0:+:B0*/POL  ; A /EQ B

NE =  A3:+:B3* POL +  A2:+:B2* POL +  A1:+:B1* POL +  A0:+:B0* POL  ; A  NE B
   +  A3:*:B3*/POL +  A2:*:B2*/POL +  A1:*:B1*/POL +  A0:*:B0*/POL  ; A /NE B

LT = /A3 * B3* POL                                                   ; A3  LT B3
   +  A3:*:B3* POL * /A2 * B2* POL                                   ; A2  LT B2
   +  A3:*:B3* POL *  A2:*:B2* POL * /A1 * B1* POL                   ; A1  LT B1
   +  A3:*:B3* POL *  A2:*:B2* POL *  A1:*:B1* POL * /A0 * B0* POL   ; A0  LT B0
   +  A3 */B3*/POL                                                   ; A3 /LT B3
   +  A3:*:B3*/POL *  A2 */B2*/POL                                   ; A2 /LT B2
   +  A3:*:B3*/POL *  A2:*:B2*/POL *  A1 */B1*/POL                   ; A1 /LT B1
   +  A3:*:B3*/POL *  A2:*:B2*/POL *  A1:*:B1*/POL *  A0 */B0*/POL   ; A0 /LT B0
   +  A3:*:B3*/POL +  A2:*:B2*/POL +  A1:*:B1*/POL +  A0:*:B0*/POL   ; A  /LT B

GT =  A3 */B3* POL                                                   ; A3  GT B3
   +  A3:*:B3* POL *  A2 */B2* POL                                   ; A2  GT B2
   +  A3:*:B3* POL *  A2:*:B2* POL *  A1 */B1* POL                   ; A1  GT B1
   +  A3:*:B3* POL *  A2:*:B2* POL *  A1:*:B1* POL *  A0 */B0* POL   ; A0  GT B0
   + /A3 * B3*/POL                                                   ; A3 /GT B3
   +  A3:*:B3*/POL * /A2 * B2*/POL                                   ; A2 /GT B2
   +  A3:*:B3*/POL *  A2:*:B2*/POL * /A1 * B1*/POL                   ; A1 /GT B1
   +  A3:*:B3*/POL *  A2:*:B2*/POL *  A1:*:B1*/POL * /A0 * B0*/POL   ; A0 /GT B0
   +  A3:*:B3*/POL +  A2:*:B2*/POL +  A1:*:B1*/POL +  A0:*:B0*/POL   ; A  /GT B
```

DESCRIPTION

THIS PLE9P4 COMPARES TWO 4-BIT NUMBERS (A3-A0 AND B3-B0) TO ESTABLISH IF THEY ARE EQUAL (A EQ B), NOT EQUAL (A NE B), LESS THAN (A LT B), OR GREATER THAN (A GT B). THE COMPARISON STATUS IS REPORTED WITH ACTIVE-HIGH POLARITY (EQ, NE, LT, GT) WHEN THE POLARITY CONTROL INPUT IS TRUE (POL=H) AND WITH ACTIVE-LOW POLARITY (/EQ, /NE, /LT, /GT) WHEN THE POLARITY CONTROL INPUT IS FALSE (POL=L).

THE PLE8P4 ALSO FEATURES THREE-STATE OUTPUTS WITH ONE ACTIVE-LOW OUTPUT ENABLE CONTROL PIN (/E).

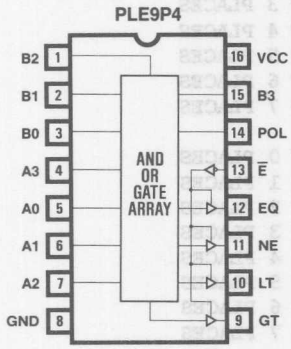
OPERATIONS TABLE:

INPUT NUMBERS A3-A0		B3-B0	POLARITY POL *	COMPARISON STATUS EQ	NE	LT	GT	OPERATION
A	EQ	B	H	H	L	L	L	COMPARE A EQUAL TO B
A	NE	B	H	L	H	X	X	COMPARE A NOT EQUAL TO B
A	LT	B	H	L	H	H	L	COMPARE A LESS THAN B
A	GT	B	H	L	H	L	H	COMPARE A GREATER THAN B

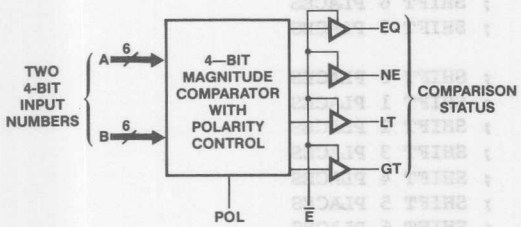
* COMPARISON STATUS WILL BE ACTIVE-LOW (I.E., /EQ, /NE, /LT, /GT) WHEN POL=L.

4-Bit Magnitude Comparator
with Polarity Control

PLE9P4



Block Diagram



8

```
PLE11P8                                          PLE CIRCUIT DESIGN PROBLEM
P5013                                                VINCENT COLI 06/12/84
8-BIT BARREL SHIFTER
MMI SANTA CLARA, CALIFORNIA
.ADD D0 D1 D2 D3 D4 D5 D6 D7 S0 S1 S2
.DAT O0 O1 O2 O3 O4 O5 O6 O7


O0 = /S0*/S1*/S2* D0       ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D1       ; SHIFT 1 PLACES
   + /S0* S1*/S2* D2       ; SHIFT 2 PLACES
   +  S0* S1*/S2* D3       ; SHIFT 3 PLACES
   + /S0*/S1* S2* D4       ; SHIFT 4 PLACES
   +  S0*/S1* S2* D5       ; SHIFT 5 PLACES
   + /S0* S1* S2* D6       ; SHIFT 6 PLACES
   +  S0* S1* S2* D7       ; SHIFT 7 PLACES

O1 = /S0*/S1*/S2* D1       ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D2       ; SHIFT 1 PLACES
   + /S0* S1*/S2* D3       ; SHIFT 2 PLACES
   +  S0* S1*/S2* D4       ; SHIFT 3 PLACES
   + /S0*/S1* S2* D5       ; SHIFT 4 PLACES
   +  S0*/S1* S2* D6       ; SHIFT 5 PLACES
   + /S0* S1* S2* D7       ; SHIFT 6 PLACES
   +  S0* S1* S2* D0       ; SHIFT 7 PLACES

O2 = /S0*/S1*/S2* D2       ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D3       ; SHIFT 1 PLACES
   + /S0* S1*/S2* D4       ; SHIFT 2 PLACES
   +  S0* S1*/S2* D5       ; SHIFT 3 PLACES
   + /S0*/S1* S2* D6       ; SHIFT 4 PLACES
   +  S0*/S1* S2* D7       ; SHIFT 5 PLACES
   + /S0* S1* S2* D0       ; SHIFT 6 PLACES
   +  S0* S1* S2* D1       ; SHIFT 7 PLACES

O3 = /S0*/S1*/S2* D3       ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D4       ; SHIFT 1 PLACES
   + /S0* S1*/S2* D5       ; SHIFT 2 PLACES
   +  S0* S1*/S2* D6       ; SHIFT 3 PLACES
   + /S0*/S1* S2* D7       ; SHIFT 4 PLACES
   +  S0*/S1* S2* D0       ; SHIFT 5 PLACES
   + /S0* S1* S2* D1       ; SHIFT 6 PLACES
   +  S0* S1* S2* D2       ; SHIFT 7 PLACES

O4 = /S0*/S1*/S2* D4       ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D5       ; SHIFT 1 PLACES
   + /S0* S1*/S2* D6       ; SHIFT 2 PLACES
   +  S0* S1*/S2* D7       ; SHIFT 3 PLACES
   + /S0*/S1* S2* D0       ; SHIFT 4 PLACES
   +  S0*/S1* S2* D1       ; SHIFT 5 PLACES
   + /S0* S1* S2* D2       ; SHIFT 6 PLACES
   +  S0* S1* S2* D3       ; SHIFT 7 PLACES
```

8-BIT BARREL SHIFTER (cont'd)

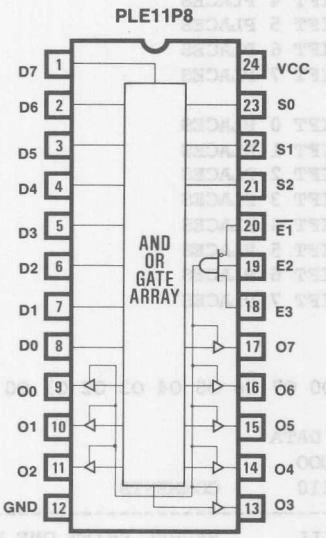
DESCRIPTION

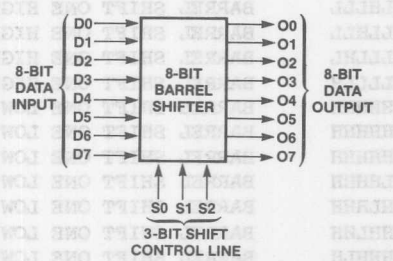
THE 8-BIT BARREL SHIFTER, IMPLEMENTED IN A PLE11P8, ROTATES EIGHT BITS OF DATA (D7-D0) A NUMBER OF LOCATIONS INTO THE OUTPUTS (O7-O0) AS SPECIFIED BY THE 3-BIT BINARY ENCODED SHIFT CONTROL LINE (S2-S0). THE THREE-STATE OUTPUTS ARE IN A HIGH-Z STATE WHEN ANY ONE OF THE TWO OUTPUT ENABLE PINS (/E1 OR /E1) ARE HIGH.

A POSSIBLE UPGRADE VERSION OF THIS DESIGN IMPLEMENTED IN A PLE12P8 COULD INCLUDE A DIRECTION CONTROL LINE. THIS CONTROL LINE PERMITS THE 8-BIT BARREL SHIFTER TO ROTATE DATA IN EITHER DIRECTION (LEFT OR RIGHT).

8-BIT BARREL SHIFTER

PLE11P8





8

8-BIT BARREL SHIFTER (cont'd)

```
O5 = /S0*/S1*/S2* D5      ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D6      ; SHIFT 1 PLACES
   + /S0* S1*/S2* D7      ; SHIFT 2 PLACES
   +  S0* S1*/S2* D0      ; SHIFT 3 PLACES
   + /S0*/S1* S2* D1      ; SHIFT 4 PLACES
   +  S0*/S1* S2* D2      ; SHIFT 5 PLACES
   + /S0* S1* S2* D3      ; SHIFT 6 PLACES
   +  S0* S1* S2* D4      ; SHIFT 7 PLACES

O6 = /S0*/S1*/S2* D6      ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D7      ; SHIFT 1 PLACES
   + /S0* S1*/S2* D0      ; SHIFT 2 PLACES
   +  S0* S1*/S2* D1      ; SHIFT 3 PLACES
   + /S0*/S1* S2* D2      ; SHIFT 4 PLACES
   +  S0*/S1* S2* D3      ; SHIFT 5 PLACES
   + /S0* S1* S2* D4      ; SHIFT 6 PLACES
   +  S0* S1* S2* D5      ; SHIFT 7 PLACES

O7 = /S0*/S1*/S2* D7      ; SHIFT 0 PLACES
   +  S0*/S1*/S2* D0      ; SHIFT 1 PLACES
   + /S0* S1*/S2* D1      ; SHIFT 2 PLACES
   +  S0* S1*/S2* D2      ; SHIFT 3 PLACES
   + /S0*/S1* S2* D3      ; SHIFT 4 PLACES
   +  S0*/S1* S2* D4      ; SHIFT 5 PLACES
   + /S0* S1* S2* D5      ; SHIFT 6 PLACES
   +  S0* S1* S2* D6      ; SHIFT 7 PLACES
```

FUNCTION TABLE

S2 S1 S0 D7 D6 D5 D4 D3 D2 D1 D0 O7 O6 O5 O4 O3 O2 O1 O0

;SHIFT ; SSS ; 210	INPUT DATA DDDDDDDD 76543210	OUTPUT DATA OOOOOOOO 76543210	COMMENTS
LLL	HLLLLLLL	HLLLLLLL	BARREL SHIFT ONE HIGH 0 PLACES
LLH	HLLLLLLL	LHLLLLLL	BARREL SHIFT ONE HIGH 1 PLACES
LHL	HLLLLLLL	LLHLLLLL	BARREL SHIFT ONE HIGH 2 PLACES
LHH	HLLLLLLL	LLLHLLLL	BARREL SHIFT ONE HIGH 3 PLACES
HLL	HLLLLLLL	LLLLHLLL	BARREL SHIFT ONE HIGH 4 PLACES
HLH	HLLLLLLL	LLLLLHLL	BARREL SHIFT ONE HIGH 5 PLACES
HHL	HLLLLLLL	LLLLLLHL	BARREL SHIFT ONE HIGH 6 PLACES
HHH	HLLLLLLL	LLLLLLLH	BARREL SHIFT ONE HIGH 7 PLACES
LLL	LHHHHHHH	LHHHHHHH	BARREL SHIFT ONE LOW 0 PLACES
LLH	LHHHHHHH	HLHHHHHH	BARREL SHIFT ONE LOW 1 PLACES
LHL	LHHHHHHH	HHLHHHHH	BARREL SHIFT ONE LOW 2 PLACES
LHH	LHHHHHHH	HHHLHHHH	BARREL SHIFT ONE LOW 3 PLACES
HLL	LHHHHHHH	HHHHLHHH	BARREL SHIFT ONE LOW 4 PLACES
HLH	LHHHHHHH	HHHHHLHH	BARREL SHIFT ONE LOW 5 PLACES
HHL	LHHHHHHH	HHHHHHLH	BARREL SHIFT ONE LOW 6 PLACES
HHH	LHHHHHHH	HHHHHHHL	BARREL SHIFT ONE LOW 7 PLACES

```
PLE11P4                          PLE CIRCUIT DESIGN SPECIFICATION
P5014                                              CHRIS JAY 05/30/84
4-BIT RIGHT SHIFTER WITH PROGRAMMABLE OUTPUT POLARITY
MMI LTD., FARNBOROUGH, U.K.
.ADD S0 S1 INV D0 D1 D2 D3 D4 D5 D6 /EN
.DAT O0 O1 O2 O3


O0 =  D0*/S0*/S1*/INV* EN      ; SELECT INPUT  D0
   + /D0*/S0*/S1* INV* EN      ; SELECT INPUT /D0
   +  D1* S0*/S1*/INV* EN      ; SELECT INPUT  D1
   + /D1* S0*/S1* INV* EN      ; SELECT INPUT /D1
   +  D2*/S0* S1*/INV* EN      ; SELECT INPUT  D2
   + /D2*/S0* S1* INV* EN      ; SELECT INPUT /D2
   +  D3* S0* S1*/INV* EN      ; SELECT INPUT  D3
   + /D3* S0* S1* INV* EN      ; SELECT INPUT /D3

O1 =  D1*/S0*/S1*/INV* EN      ; SELECT INPUT  D1
   + /D1*/S0*/S1* INV* EN      ; SELECT INPUT /D1
   +  D2* S0*/S1*/INV* EN      ; SELECT INPUT  D2
   + /D2* S0*/S1* INV* EN      ; SELECT INPUT /D2
   +  D3*/S0* S1*/INV* EN      ; SELECT INPUT  D3
   + /D3*/S0* S1* INV* EN      ; SELECT INPUT /D3
   +  D4* S0* S1*/INV* EN      ; SELECT INPUT  D4
   + /D4* S0* S1* INV* EN      ; SELECT INPUT /D4

O2 =  D2*/S0*/S1*/INV* EN      ; SELECT INPUT  D2
   + /D2*/S0*/S1* INV* EN      ; SELECT INPUT /D2
   +  D3* S0*/S1*/INV* EN      ; SELECT INPUT  D3
   + /D3* S0*/S1* INV* EN      ; SELECT INPUT /D3
   +  D4*/S0* S1*/INV* EN      ; SELECT INPUT  D4
   + /D4*/S0* S1* INV* EN      ; SELECT INPUT /D4
   +  D5* S0* S1*/INV* EN      ; SELECT INPUT  D5
   + /D5* S0* S1* INV* EN      ; SELECT INPUT /D5

O3 =  D3*/S0*/S1*/INV* EN      ; SELECT INPUT  D3
   + /D3*/S0*/S1* INV* EN      ; SELECT INPUT /D3
   +  D4* S0*/S1*/INV* EN      ; SELECT INPUT  D4
   + /D4* S0*/S1* INV* EN      ; SELECT INPUT /D4
   +  D5*/S0* S1*/INV* EN      ; SELECT INPUT  D5
   + /D5*/S0* S1* INV* EN      ; SELECT INPUT /D5
   +  D6* S0* S1*/INV* EN      ; SELECT INPUT  D6
   + /D6* S0* S1* INV* EN      ; SELECT INPUT /D6
```

4-BIT RIGHT SHIFTER WITH PROGRAMMABLE OUTPUT POLARITY (cont'd)

FUNCTION TABLE

```
/EN S1 S0 INV D6 D5 D4 D3 D2 D1 D0 O3 O2 O1 O0
```

;-CONTROL-				-INPUT DATA-							OUTPUTS				
;/			I								O	O	O	O	
;E	S	S	N	D	D	D	D	D	D	D	3	2	1	0	COMMENTS
;N	1	0	V	6	5	4	3	2	1	0					
H	X	X	X	X	X	X	X	X	X	X	L	L	L	L	TEST ENABLE, OUTPUTS GO LOW
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	SHIFT COUNT = 0, TRUE POLARITY
L	L	H	L	L	L	L	H	H	H	H	L	H	H	H	SHIFT COUNT = 1, TRUE POLARITY
L	H	L	L	L	L	L	H	H	H	H	L	L	H	H	SHIFT COUNT = 2, TRUE POLARITY
L	H	H	L	L	L	L	H	H	H	H	L	L	L	H	SHIFT COUNT = 3, TRUE POLARITY
L	L	L	H	L	L	L	H	H	H	H	L	L	L	L	SHIFT COUNT = 0, COMP POLARITY
L	L	H	H	L	L	L	H	H	H	H	H	L	L	L	SHIFT COUNT = 1, COMP POLARITY
L	H	L	H	L	L	L	H	H	H	H	H	H	L	L	SHIFT COUNT = 2, COMP POLARITY
L	H	H	H	L	L	L	H	H	H	H	H	H	H	L	SHIFT COUNT = 3, COMP POLARITY

4-BIT RIGHT SHIFTER WITH PROGRAMMABLE OUTPUT POLARITY (cont'd)

DESCRIPTION

THIS PLE11P4 IMPLEMENTS A 4-BIT RIGHT SHIFTER WITH PROGRAMMABLE OUTPUT POLARITY. THE SHIFTER CAN RIGHT SHIFT SEVEN BITS OF DATA, FOUR BITS AT A TIME. THE SEVEN DATA INPUTS (D6-D0) ARE SHIFTED 0, 1, 2, OR 3 LOCATIONS AS DETERMINED BY THE 2-BIT SHIFT CONTROL LINE (S1-S0). THE SHIFTED DATA IS THEN DIRECTED TO THE FOUR OUTPUTS (O3-O0).

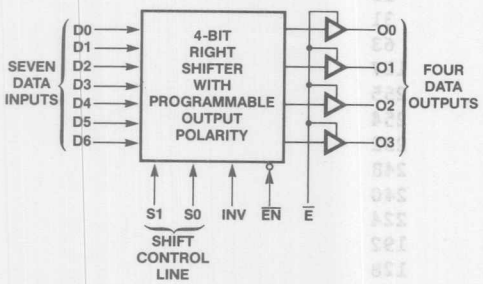
THE OUTPUT DATA IS NONINVERTED (O=D) WHEN INV=L AND INVERTED (O=/D) WHEN INV=H. THE OUTPUTS ARE FORCED LOW (O=L) WHEN /EN=H REGARDLESS OF OTHER INPUTS. THE PLE11P4 ALSO FEATURES THREE-STATE OUTPUTS WITH ONE ACTIVE LOW OUTPUT ENABLE (/E).

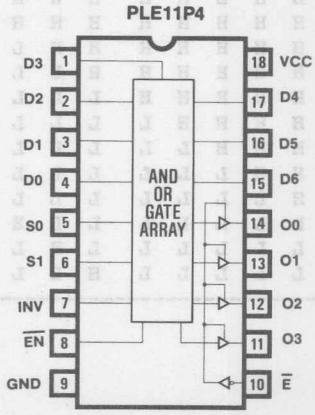
A POSSIBLE UPGRADE VERSION OF THIS DESIGN IMPLEMENTED IN A PLE12P4 COULD INCLUDE A DIRECTION CONTROL LINE. THIS CONTROL LINE PERMITS THE 4-BIT RIGHT SHIFTER TO SHIFT DATA IN EITHER DIRECTION (LEFT OR RIGHT).

OPERATIONS TABLE:

/EN	INV	S1-S0	D6-D0	O3-O0	OPERATION
H	X	X	X	L	DISABLE OUTPUTS LOW
L	L	N	D	SHIFT(D)	SHIFT NONINVERTED DATA "N" PLACES
L	H	N	D	SHIFT(/D)	SHIFT INVERTED DATA "N" PLACES

4-BIT RIGHT SHIFTER WITH PROGRAMMABLE OUTPUT POLARITY



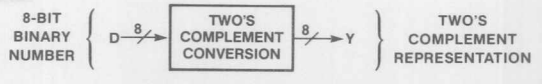


8

```
PLE8P8                                  PLE CIRCUIT DESIGN SPECIFICATION
P5015                                                  MIKE VOGEL 11/28/83
8-BIT TWO'S COMPLEMENT CONVERSION
MMI BREA, CALIFORNIA
.ADD D0 D1 D2 D3 D4 D5 D6 D7
.DAT Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7

Y0 = D0                                                ; CONVERT 1ST BIT (LSB)

Y1 = D1 :+: D0                                         ; CONVERT 2ND BIT

Y2 = D2 :+: D0 + D1                                    ; CONVERT 3RD BIT

Y3 = D3 :+: D0 + D1 + D2                               ; CONVERT 4TH BIT

Y4 = D4 :+: D0 + D1 + D2 + D3                          ; CONVERT 5TH BIT

Y5 = D5 :+: D0 + D1 + D2 + D3 + D4                     ; CONVERT 6TH BIT

Y6 = D6 :+: D0 + D1 + D2 + D3 + D4 + D5                ; CONVERT 7TH BIT

Y7 = D7 :+: D0 + D1 + D2 + D3 + D4 + D5 + D6           ; CONVERT 8TH BIT (MSB)
```

FUNCTION TABLE

D7	D6	D5	D4	D3	D2	D1	D0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	;DECIMAL
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	0
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	1
L	L	L	L	L	L	H	H	H	H	H	H	H	H	L	H	3
L	L	L	L	L	H	H	H	H	H	H	H	H	L	L	H	7
L	L	L	L	H	H	H	H	H	H	H	H	L	L	L	H	15
L	L	L	H	H	H	H	H	H	H	H	L	L	L	L	H	31
L	L	H	H	H	H	H	H	H	H	L	L	L	L	L	H	63
L	H	H	H	H	H	H	H	H	L	L	L	L	L	L	H	127
H	H	H	H	H	H	H	H	L	L	L	L	L	L	L	L	255
H	H	H	H	H	H	H	L	L	L	L	L	L	L	H	L	254
H	H	H	H	H	H	L	L	L	L	L	L	L	H	L	L	252
H	H	H	H	H	L	L	L	L	L	L	L	H	L	L	L	248
H	H	H	H	L	L	L	L	L	L	L	H	L	L	L	L	240
H	H	H	L	L	L	L	L	L	L	H	L	L	L	L	L	224
H	H	L	L	L	L	L	L	L	H	L	L	L	L	L	L	192
H	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L	128



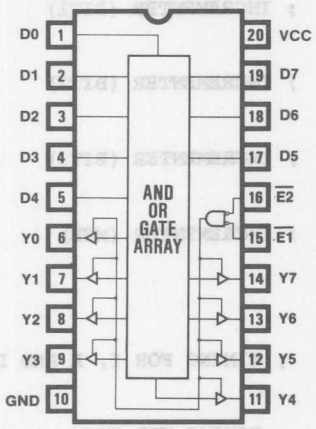
8-BIT TWO'S COMPLEMENT CONVERSION (cont'd)

DESCRIPTION

THIS PLE8P8 CONVERTS AN 8-BIT BINARY NUMBER (D7-D0) INTO TWO'S COMPLEMENT REPRESENTATION (Y7-Y0) WHERE D7 AND Y7 ARE THE MSB AND D0 AND Y0 ARE THE LSB. TWO'S COMPLEMENT REPRESENTATION IS USED IN SIGNED ARITHMETIC SYSTEMS.

8-BIT TWO'S COMPLEMENT CONVERSION

PLE8P8

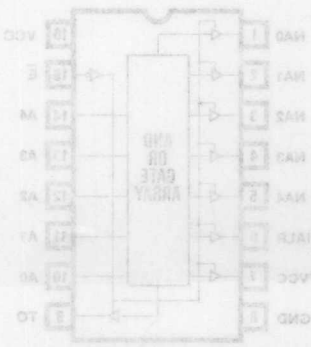


```
PLE5P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5027                                          S. HORIKO 11/29/83
A PORTION OF TIMING GENERATOR FOR PAL ARRAY PROGRAMMING
MMI JAPAN
.ADD A0 A1 A2 A3 A4
.DAT NA0 NA1 NA2 NA3 NA4 TIALR TVCC TO

; NEXT ADDRESS GENERATOR

NA0  =  /A0                          ; INCREMENTER (LSB)

NA1  =   A0                          ; INCREMENTER (BIT1)
    :+:  A1

NA2  =   A2                          ; INCREMENTER (BIT2)
    :+:  A0* A1

NA3  =   A3                          ; INCREMENTER (BIT3)
    :+:  A0* A1* A2

NA4  =   A4                          ; INCREMENTER (MSB)
    :+:  A0* A1* A2* A3

; TIMING WAVEFORMS

TIALR = /A4*/A3                       ; TIMING FOR I, A AND L/R
      + /A4*    /A2*/A1

TVCC  = /A4*/A3* A2                   ; TIMING FOR VCC
      + /A4* A3*/A2*/A1

TO    = /A4* A3*/A2*/A1*/A0           ; TIMING FOR O
      + /A4*/A3* A2* A1
      + /A4*/A3* A2*     A0
```

TIMING GENERATOR FOR PAL PROGRAMMING (cont'd)

FUNCTION TABLE

```
A4 A3 A2 A1 A0 NA4 NA3 NA2 NA1 NA0 TIALR TVCC TO

;          NNNNN
;AAAAA      AAAAA    TIMING WAVEFORMS
;43210      43210    TIALR  TVCC   TO      ; ## ;    COMMENTS
-------------------------------------------------------------------------
 LLLLL      LLLLH      H      L      L      ; 01 ;    ASSERT TIALR
 LLLLH      LLLHL      H      L      L      ; 02 ;
 LLLHL      LLLHH      H      L      L      ; 03 ;
 LLLHH      LLHLL      H      L      L      ; 04 ;
 LLHLL      LLHLH      H      H      L      ; 05 ;    ASSERT TVCC
 LLHLH      LLHHL      H      H      H      ; 06 ;    ASSERT TO
 LLHHL      LLHHH      H      H      H      ; 07 ;
 LLHHH      LHLLL      H      H      H      ; 08 ;
 LHLLL      LHLLH      H      H      H      ; 09 ;
 LHLLH      LHLHL      H      H      L      ; 10 ;    CLEAR TO
 LHLHL      LHLHH      L      L      L      ; 11 ;    CLEAR TIALR & TVCC
 LHLHH      LHHLL      L      L      L      ; 12 ;
 LHHLL      LHHLH      L      L      L      ; 13 ;
 LHHLH      LHHHL      L      L      L      ; 14 ;
 LHHHL      LHHHH      L      L      L      ; 15 ;
 LHHHH      HLLLL      L      L      L      ; 16 ;
 HLLLL      HLLLH      L      L      L      ; 17 ;
 HLLLH      HLLHL      L      L      L      ; 18 ;
 HLLHL      HLLHH      L      L      L      ; 19 ;
 HLLHH      HLHLL      L      L      L      ; 20 ;
 HLHLL      HLHLH      L      L      L      ; 21 ;
 HLHLH      HLHHL      L      L      L      ; 22 ;
 HLHHL      HLHHH      L      L      L      ; 23 ;
 HLHHH      HHLLL      L      L      L      ; 24 ;
 HHLLL      HHLLH      L      L      L      ; 25 ;
 HHLLH      HHLHL      L      L      L      ; 26 ;
 HHLHL      HHLHH      L      L      L      ; 27 ;
 HHLHH      HHHLL      L      L      L      ; 28 ;
 HHHLL      HHHLH      L      L      L      ; 29 ;
 HHHLH      HHHHL      L      L      L      ; 30 ;
 HHHHL      HHHHH      L      L      L      ; 31 ;
 HHHHH      LLLLL      L      L      L      ; 32 ;
-------------------------------------------------------------------------
```



8

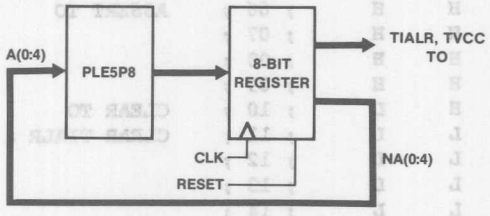
TIMING GENERATOR FOR PAL PROGRAMMING (cont'd)

DESCRIPTION

THIS LOGIC SPECIFICATION IS A TIMING SIGNAL GENERATOR TO BE USED FOR ARRAY PROGRAMMING OF PAL DEVICES. A PLE5P8 FOLLOWED BY AN 8-BIT REGISTER ARE USED TO IMPLEMENT THIS FUNCTION.

THE PLE CONTAINS BOTH 5-BIT NEXT ADDRESS AND 3-BIT WAVEFORMS. TIALR OUTPUT IS A TIMING WAVEFORM FOR I, A, AND L/R SIGNALS, AND TVCC AND TO OUTPUTS ARE USED FOR VCC AND O SIGNALS, RESPECTIVELY.

THE SCHEMATIC IS AS FOLLOWS:

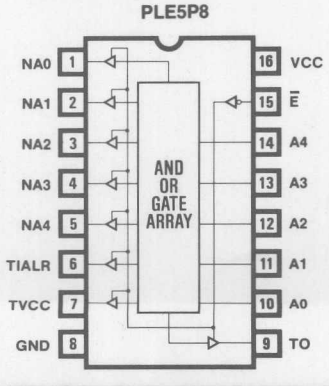


APPLYING 200KHz CLOCK SIGNAL TO THE CLK INPUT OF THE REGISTER GENERATES THE FOLLOWING TIMINGS:

1. I, A, AND L/R WIDTH : 50 usec
2. tD2 : 20 usec
3. tD : 5 usec
4. tVCCP : 30 usec
5. Tp : 20 usec

BECAUSE THE TIMING PATTERNS ARE STORED IN THE PROM, WE CAN EASILY CALIBRATE THE RELATIONS AND THE PERIOD AMONG THOSE SIGNALS TO MAKE AN OPTIMUM CONDITION.

A PORTION OF A TIMING GENERATOR FOR PAL LOGIC CIRCUIT ARRAY PROGRAMMING



```
PLE5P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5028                                          S. HORIKO 11/29/83
TIMING GENERATOR FOR PAL DEVICE SECURITY FUSE PROGRAMMING
MMI JAPAN
.ADD A0 A1 A2 A3 A4
.DAT NA0 NA1 NA2 NA3 NA4 TVCC TP01 TP11

; NEXT ADDRESS GENERATOR
;  ( THE COUNTER LOCKS UP AT COUNT-22 )

NA0  = /A4*          /A1*/A0        ; INCREMENTER (LSB)
     + /A4*           A1*/A0        ; INCREMENTER (LSB)
     +  A4*/A3*/A2*     /A0         ; INCREMENTER (LSB)
     +  A4*/A3* A2*/A1               ; INCREMENTER (LSB)

NA1  = /A4*          /A1* A0        ; INCREMENTER (BIT1)
     + /A4*           A1*/A0        ; INCREMENTER (BIT1)
     +  A4*/A3*/A2*/A1* A0          ; INCREMENTER (BIT1)
     +  A4*/A3*/A2* A1*/A0          ; INCREMENTER (BIT1)

NA2  = /A4*      A2*/A1             ; INCREMENTER (BIT2)
     + /A4*      A2*    /A0         ; INCREMENTER (BIT2)
     + /A4*     /A2* A1* A0         ; INCREMENTER (BIT2)
     +  A4*/A3* A2*/A1              ; INCREMENTER (BIT2)
     +  A4*/A3*/A2* A1* A0          ; INCREMENTER (BIT2)

NA3  = /A4* A3*/A2                  ; INCREMENTER (BIT3)
     + /A4* A3*    /A1             ; INCREMENTER (BIT3)
     + /A4* A3*         /A0        ; INCREMENTER (BIT3)
     + /A4*/A3* A2* A1* A0          ; INCREMENTER (BIT3)

NA4  = /A4* A3* A2* A1* A0          ; INCREMENTER (MSB)
     +  A4*/A3*/A2                  ; INCREMENTER (MSB)
     +  A4*/A3*    /A1              ; INCREMENTER (MSB)

; TIMING WAVEFORMS

TVCC = /A4                          ; TIMING FOR VCC
     +  A4*/A3*/A2*/A1
     +  A4*/A3*/A2*    /A0

TP01 = /A4*/A3* A2                  ; TIMING FOR PIN 01
     + /A4*/A3*     A1
     + /A4*/A3*         A0
     + /A4* A3*/A2*/A1*/A0

TP11 = /A4* A3* A2                  ; TIMING FOR PIN 11
     + /A4* A3*     A1
     +  A4*/A3*/A2*/A1
```

8

FUNCTION TABLE

A4 A3 A2 A1 A0 NA4 NA3 NA2 NA1 NA0 TVCC TP01 TP11

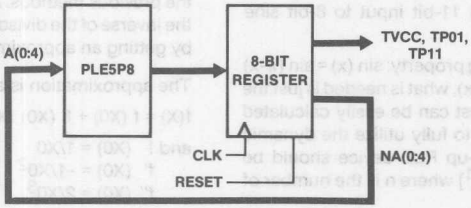
;	NNNNN						
;AAAAA	AAAAA	TIMING WAVEFORMS					
;43210	43210	TVCCP	TP01	TP11	; ## ;	COMMENTS	
LLLLL	LLLLH	H	L	L	; 01 ;	ASSERT TVCC, START HERE	
LLLLH	LLLHL	H	H	L	; 02 ;	ASSERT TP01	
LLLHL	LLLHH	H	H	L	; 03 ;		
LLLHH	LLHLL	H	H	L	; 04 ;		
LLHLL	LLHLH	H	H	L	; 05 ;		
LLHLH	LLHHL	H	H	L	; 06 ;		
LLHHL	LLHHH	H	H	L	; 07 ;		
LLHHH	LHLLL	H	H	L	; 08 ;		
LHLLL	LHLLH	H	H	L	; 09 ;	CLEAR TP01	
LHLLH	LHLHL	H	L	L	; 10 ;	ASSERT TP11	
LHLHL	LHLHH	H	L	H	; 11 ;		
LHLHH	LHHLL	H	L	H	; 12 ;		
LHHLL	LHHLH	H	L	H	; 13 ;		
LHHLH	LHHHL	H	L	H	; 14 ;		
LHHHL	LHHHH	H	L	H	; 15 ;		
LHHHH	HLLLL	H	L	H	; 16 ;		
HLLLL	HLLLH	H	L	H	; 17 ;		
HLLLH	HLLHL	H	L	H	; 18 ;		
HLLHL	HLLHH	H	L	L	; 19 ;	CLEAR TP11	
HLLHH	HLHLL	L	L	L	; 20 ;	CLEAR TVCC	
HLHLL	HLHLH	L	L	L	; 21 ;		
HLHLH	HLHLH	L	L	L	; 22 ;	LOOP HERE UNTIL RESET	

DESCRIPTION

THIS LOGIC SPECIFICATION IS A TIMING SIGNAL GENERATOR TO BE USED FOR SECURITY FUSE PROGRAMMING OF PAL DEVICES. A PLE5P8 FOLLOWED BY AN 8-BIT REGISTER ARE USED TO IMPLEMENT THIS FUNCTION.

THE PLE LOGIC CIRCUIT CONTAINS TWO FUNCTIONS IN THE SINGLE CHIP. THE FIRST FUNCTION IS A UNIQUE COUNTER USED FOR NEXT ADDRESS GENERATION. THE COUNTER INCREMENTS UP TO COUNT-21 AND THEN LOCKS UP THE INCREMENTAL OPERATION AT COUNT-22. THE SECOND FUNCTION IS A TIMING GENERATOR USED FOR DEFINING TIMING RELATIONSHIP AMOUNG VCC, P01, AND P11 SIGNALS.

THE SCHEMATIC IS AS FOLLOWS:



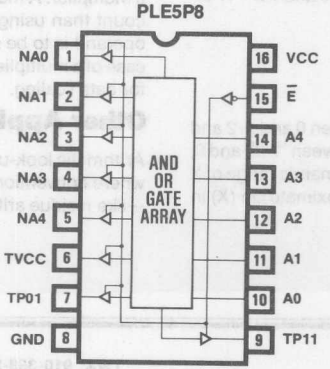
THIS LOGIC OUTPUTS A SEQUENCE OF TIMING PATTERNS DURING THE INCREMENTAL OPERATION AND THEN HOLDS ALL OUTPUTS LOW UNTIL A RESET SIGNAL FOR THE 8-BIT REGISTER IS APPLIED.

APPLYING 200 KHz CLOCK SIGNAL TO THE CLK INPUT OF THE REGISTER, THE FOLLOWING TIMINGS ARE GENERATED:

1. VCC WIDTH : 95 usec
2. TPP : 40 usec
3. tD : 5 usec

BY APPLYING THIS DESIGN METHOD, WE CAN EASILY GENERATE A SEQUENCE OF UNIQUELY DEFINED PATTERNS EACH TIME THE RESET PULSE IS APPLIED.

TIMING GENERATOR FOR PAL SECURITY FUSE PROGRAMMING



Fast Arithmetic Look-up

Fast Arithmetic Look-up

In performing arithmetic operations like trigonometric functions, multiplications and division, in order to reduce the delay, look-up tables are often used.

Sine Look-up

For trigonometric functions like sine function, it is very time-consuming to generate the function using the polynomial which represents the function. PLE devices can provide a very good alternative for sine look-up. An example is to use a 2Kx8 PLE device to do a sine look-up of an 11-bit input to 8-bit sine outputs.

Since sine function has the following property: $\sin(x) = \sin(\pi - x) = -\sin(\pi + x) = -\sin(2\pi - x) = \sin(2\pi + x)$, what is needed is just the sine function for $0 < x < \pi/2$; the rest can be easily calculated using the above relations. In order to fully utilize the dynamic range, the inputs of the sine look-up PLE device should be normalized to $(\pi/2) / (2^n) = \pi/[2^{n+1}]$ where n is the number of address lines to the device.

Since n is fixed for the PLE device chosen, and π is a constant, for the look-up table $\pi/[2^{n+1}]$ is a constant. Therefore, if the sine function of a given x is to be found, x will first be multiplied by the constant $[2^{n+1}]/\pi$ and sent to the address of the PLE device to get the final result.

Cos (x) is related to sine function as $\sin(\pi/2 - x)$. Thus the cosine function can also be found in the same manner by using $\pi/2 - x$ instead of just x. Other functions like tangent, secant etc., can also be found as a function of sine.

To increase the dynamic range of outputs, we can just use another PLE device to generate the less-significant bits of the sine function.

If a larger dynamic range is needed for the inputs, the result may be approximated using the Taylor series:

$$f(X) = f(X0) + f'(X0)(X - X0) + 1/2 f''(X0)(X - X0)^2 + \ldots$$

where f' and f'' are the first and second derivations of f. Since X0 by itself represents a resolution of 2^{-n}, and X is X0 concatenated with the rest of the bits, X – X0 must lie between 0 and $1/2^{-n}$. For f (X) = sin (X),

$$f(X0) = \sin(X0)$$
$$f'(X0) = \cos(X0)$$
and $f''(X0) = -\sin(X0)$

So f'' (X0) is between –1 and 0 for X0 lies between 0 and $\pi/2$ and $X - X0 < 2^{-n}$. Therefore, the last term will be between $1/2^n$ and 0, and as long as we do not want to expand the dynamic range of X beyond 2n-bits, it should be sufficient to approximate sin (X) in the first two terms:

$$\sin(X) + \sin(X0) + \cos(X0)(X - X0)$$

Since X–X0 is represented by only the bits after the more significant n-bits, and cos (X0) = sin $(\pi/2 - X0)$, the implementation will be very simple.

Division

Division will normally be much slower than multiplication. There are several ways to perform division. Bit-by-bit division restoring and nonrestoring algorithms are generally very slow. Another way is to use several bits at a time division which is faster than the previous methods. A third way is to multiply the dividend by the inverse of the divisor. The inverse of the divisor can be found by getting an approximation followed by iterations.

The approximation is again given by the Taylor series:

$$f(X) = f(X0) + f'(X0)(X - X0) + 1/2 f''(X0)(X - X0)^2 + \ldots$$

and $f(X0) = 1/X0$
$f'(X0) = -1/X0^2$
$f''(X0) = 2/X0^3$

Say X0 is 8-bits long and the first approximation of the inverse is found using a 256x8 PLE device. The first approximation can be obtained by subtracting $(X - X0)/(X0^2)$. Since the first approximation is limited by an error of approximately $(X - X0)^2/X0^2$, and if X0 at least 1, the error is limited by approximately $(X - X0)^2$. Since X0 has an 8-bit resolution, X–X0 is represented by the rest of the bits. The resolution of the second approximation will be about 16 bits. The third approximation is similarly deduced and has a resolution of about 32 bits, and the fourth has a resolution of about 64 bits.

The inverse thus obtained is then multiplied by the dividend to give the quotient.

Scaling

In arithmetic operations, scaling is sometimes needed. Scaling normally involves multiplication or division by a constant. If this constant can be expressed in 2^n where n is an integer, then scaling is simply shifting. Scaling with other constants may need a multiplier. A multiplier is more expensive and has a higher pin count than using a PLE device because the constant that the operand is to be scaled by is not required as an input as in the case of a multiplier. This will tremendously reduce the overhead for data scaling.

Other Applications

Arithmetic look-up are also very useful for arithmetic operations where conventional binary integral arithmetic is not applicable —like residue arithmetic, and distributed arithmetic.

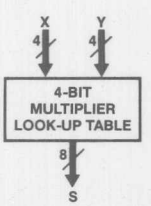
```
PLE8P8                    PLE CIRCUIT DESIGN SPECIFICATION
P5018                                VINCENT COLI 12/08/82
4-BIT MULTIPLIER LOOK-UP TABLE
MMI SANTA CLARA, CALIFORNIA
.ADD X0 X1 X2 X3 Y0 Y1 Y2 Y3
.DAT S0 S1 S2 S3 S4 S5 S6 S7


S7,S6,S5,S4,S3,S2,S1,S0 = X3,X2,X1,X0 .*. Y3,Y2,Y1,Y0    ; S = X * Y

FUNCTION TABLE

X3 X2 X1 X0 Y3 Y2 Y1 Y0 S7 S6 S5 S4 S3 S2 S1 S0

;-OPERANDS-     PRODUCTS
;XXXX  YYYY     SSSSSSSS      COMMENTS
;3210  3210     76543210
-----------------------------------------
 LLLL  LLLL     LLLLLLLL      0 *  0  =   0
 LLLH  HHHH     LLLLHHHH      1 * 15  =  15
 HHHH  LLLH     LLLLHHHH     15 *  1  =  15
 HHHH  HHHH     HHHLLLLH     15 * 15  = 225
-----------------------------------------
```

DESCRIPTION

THIS PLE8P8 PERFORMS 4-BIT LOOK-UP TABLE MULTIPLICATION. THE DEVICE ACCEPTS TWO 4-BIT OPERANDS (X3-X0 AND Y3-Y0) TO PRODUCE THE 8-BIT PRODUCT (S7-S0). THE PLE8P8 ALSO HAS THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

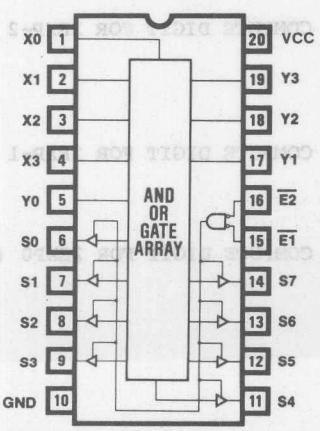
```
          X3 X2 X1 X0 }
        X              } TWO 4-BIT
          Y3 Y2 Y1 Y0 } OPERANDS
-------------------------
S7 S6 S5 S4 S3 S2 S1 S0
```

8-BIT PRODUCT



4-BIT MULTIPLIER LOOK-UP TABLE

PLE8P8



8

```
PLE5P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5023                                    PETER ZECHERLE 03/06/84
ARC TANGENT LOOK-UP TABLE
MMI GMBH MUNICH
.ADD A0 A1 A2 A3 A4
.DAT F0 F1 F2 F3 F4 F5 F6 F7


F0 =     A1*     /A3*/A4     ; COMPUTE DIGIT FOR 2EXP-7 (0.00078125) (LSB)
   +          A2*/A3
   +  A0*/A1*     A3*/A4
   + /A0* A1*        /A4
   + /A0*/A1*    /A3* A4
   +  A0*     A2
   +      A1* A2

F1 =     /A1*     A3*/A4      ; COMPUTE DIGIT FOR 2EXP-6 (0.015625)
   +  A0*         /A3* A4
   +      A1*    /A3* A4
   +          A2*/A3* A4
   +  A0* A1*    /A3
   +  A0*     A2*/A3
   + /A0* A1* A2*    /A4
   +  A0*    /A2* A3*/A4

F2 =  A0*         /A3*/A4      ; COMPUTE DIGIT FOR 2EXP-5 (0.03125)
   +      A1*/A2*    /A4
   +              A3* A4
   + /A0*     A2* A3*/A4
   +     /A1* A2* A3*/A4

F3 =      A1*/A2*    /A4      ; COMPUTE DIGIT FOR 2EXP-4 (0.0625)
   +     /A1* A2*    /A4
   + /A0*         A3*/A4
   +     /A1*     A3*/A4

F4 =     /A1*     A3*/A4      ; COMPUTE DIGIT FOR 2EXP-3 (0.125)
   +  A0* A1*/A2*    /A4
   +      A1* A2*/A3*/A4
   + /A0*         A3*/A4

F5 =  A0*/A1*        /A4      ; COMPUTE DIGIT FOR 2EXP-2 (0.25)
   +          A2*/A3*/A4
   +         /A2* A3*/A4
   + /A0*         A3*/A4

F6 =  A0*/A1*/A2*/A3          ; COMPUTE DIGIT FOR 2EXP-1 (0.5)
   +  A0* A1* A2* A3
   +                  A4

F7 =      A1                  ; COMPUTE DIGIT FOR 2EXP0 (1) (MSB)
   +          A2
   +              A3
   +                  A4
```

ARC TANGENT LOOK-UP TABLE (cont'd)

FUNCTION TABLE

;----ANGLE---- INTEGER					--------F = ARCTAN(A)-------- INTEGER	FRACTIONS							;ANGLE	---F = ARCTAN(A)--- LOOK-UP	CALCULATED
A4	A3	A2	A1	A0	F7	F6	F5	F4	F3	F2	F1	F0			
L	L	L	L	L	L	L	L	L	L	L	L	L	0	0.0000	0.0000
L	L	L	L	H	L	H	H	L	L	H	L	L	1	0.7813	0.7854
L	L	L	H	L	H	L	L	L	H	H	L	H	2	1.1016	1.1071
L	L	H	L	L	H	L	H	L	H	L	L	H	4	1.3203	1.3258
L	L	H	L	H	H	L	H	L	H	H	H	H	5	1.3672	1.3734
L	H	L	L	L	H	L	H	H	H	L	H	L	8	1.4531	1.4464
H	L	L	L	L	H	H	L	L	L	L	L	H	16	1.5078	1.5084
H	H	H	H	H	H	H	L	L	L	H	L	H	31	1.5391	1.5385

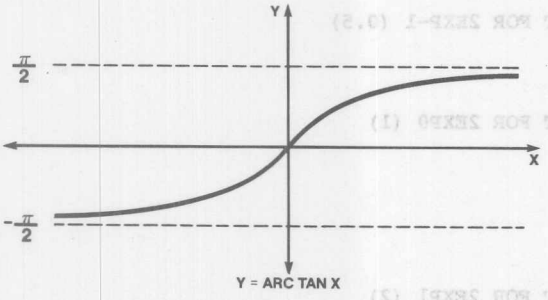
DESCRIPTION

THIS APPLICATION ILLUSTRATES THE CALCULATION OF THE ARC TANGENT FUNCTION USING A PLE5P8 AS A LOOK-UP TABLE. OTHER TRIGONOMETRIC FUNCTIONS (SUCH AS SINE, COSINE, COTANGENT, SECANT, COSECANT AND THEIR ARC INVERSE EQUIVALENT FUNCTIONS) OR HYPERBOLIC FUNCTIONS CAN ALSO BE CONSTRUCTED USING PLE DEVICES AS LOOK-UP TABLES.

F = ARCTAN(A) WHERE F = ARC TANGENT OF A
A = ANGLE IN RADIANS

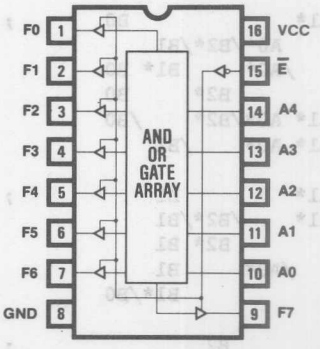
EXAMPLE: FOR A = 5, F = ARCTAN(5) = 1.3672

A PLE DEVICE WITH MORE INPUTS, SUCH AS THE PLE11P8, SHOULD BE USED TO CONSTRUCT A LOOK-UP TABLE WHEN ADDITIONAL ACCURACY IS REQUIRED.



ARC TANGENT LOOK-UP TABLE

PLE5P8



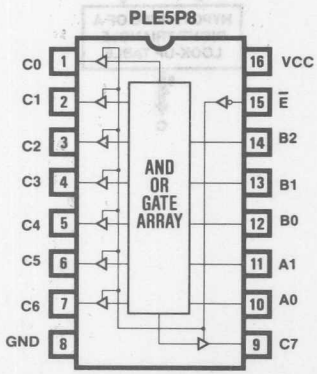
8

```
HYPOTENUSE OF A RIGHT TRIANGLE LOOK-UP TABLE
MMI GMBH MUNICH
.ADD A0 A1 B0 B1 B2
.DAT C0 C1 C2 C3 C4 C5 C6 C7


C0 =      A0*/B2* B1          ; COMPUTE DIGIT FOR 2EXP-5 (0.03125) (LSB)
   + /A1* A0* B2*/B1
   +  A1*    /B2*     B0
   +  A1*    /B2* B1
   +  A1*/A0* B2*/B1*/B0
   +  A1* A0*     B1* B0
   +      A0*/B2*     B0

C1 =      A0*     B1*/B0        ; COMPUTE DIGIT FOR 2EXP-4 (0.0625)
   + /A1* A0* B2
   +  A1*/A0*/B2*     B0
   +  A1*/A0* B2*    /B0
   +      A0* B2*     B0
   +  A1* A0*    B1

C2 =      A0*/B2*     B0        ; COMPUTE DIGIT FOR 2EXP-3 (0.125)
   + /A1* A0*/B2* B1
   +  A1*/A0* B2*/B1
   +  A1* A0* B2* B1
   +  A1*    /B2*/B1* B0

C3 = /A1* A0*/B2*/B1* B0        ; COMPUTE DIGIT FOR 2EXP-2 (0.25)
   +  A1*/A0*     B1*/B0
   +  A1*/A0* B2
   +  A1*     B2*     B0

C4 =  A1*/A0*/B2* B1            ; COMPUTE DIGIT FOR 2EXP-1 (0.5)
   +  A1* A0* B2*/B1* B0
   +  A1* A0*     B1*/B0

C5 = /A1*             B0        ; COMPUTE DIGIT FOR 2EXP0 (1)
   +      A0*/B2*/B1
   +     /A0*    B1* B0
   +         B2*     B0
   +  A1* A0*/B2*    /B0
   +  A1* A0*    /B1

C6 = /A1*             B1        ; COMPUTE DIGIT FOR 2EXP1 (2)
   +  A1*    /B2*/B1
   +         B2* B1
   +     /A0*    B1
   +             B1*/B0

C7 =          B2                ; COMPUTE DIGIT FOR 2EXP2 (4) (MSB)
   +  A1* A0*     B1* B0
```

;-LENGTH OF SIDES-					LENGTH OF THE HYPOTENUSE											
;SIDE A		SIDE B			INTEGER			FRACTION					SIDES		LENGTH OF HYPOTENUSE	
A1	A0	B2	B1	B0	C7	C6	C5	C4	C3	C2	C1	C0	;A	B	LOOK-UP	CALCULATED
L	L	L	L	L	L	L	L	L	L	L	L	L	0	0	0.00	0.00
L	L	L	L	H	L	L	H	L	L	L	L	L	0	1	1.00	1.00
L	L	L	H	L	L	H	L	L	L	L	L	L	0	2	2.00	2.00
L	L	H	L	L	H	L	L	L	L	L	L	L	0	4	4.00	4.00
L	H	L	L	L	L	L	H	L	L	L	L	L	1	0	1.00	1.00
H	L	L	L	L	L	H	L	L	L	L	L	L	2	0	1.00	1.00
H	L	L	H	L	L	H	L	H	H	L	L	H	2	2	2.78	2.83
H	L	H	L	L	H	L	L	L	H	H	H	H	2	4	4.47	4.47
H	H	L	H	L	L	H	H	H	L	L	H	H	3	2	3.59	3.61
H	H	H	H	H	H	H	H	L	H	H	H	H	3	7	7.47	7.62

8

HYPOTENUSE OF A RIGHT TRIANGLE LOOK-UP TABLE



HYPOTENUSE OF A RIGHT TRIANGLE LOOK-UP TABLE (cont'd)

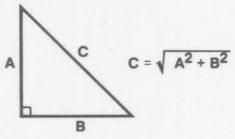
DESCRIPTION

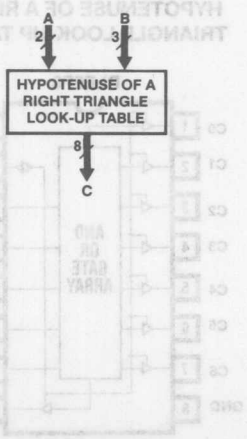
THE GENERATION OF COMPLEX ARITHMETIC FUNCTIONS SUCH AS THE PYTHAGOREAN THEOREM IS GENERALLY VERY DIFFICULT TO IMPLEMENT DIRECTLY IN HARDWARE. HOWEVER, IMPLEMENTING THE FUNCTION AS A LOOK-UP TABLE USING A PLE GREATLY SIMPLIFIES THE PROBLEM.

THIS EXAMPLE ILLUSTRATES HOW TO IMPLEMENT A LOOK-UP TABLE IN A PLE5P8 WHICH CALCULATES THE LENGTH OF THE HYPOTENUSE OF A RIGHT TRIANGLE AS A FUNCTION OF THE LENGTH OF THE TWO REMAINING SIDES OF THE TRIANGLE. THE THEOREM OF PATHAGOREAN STATES THAT THE LENGTH OF THE HYPOTENUSE OF A RIGHT TRIANGLE IS EQUAL TO THE SQUARE ROOT OF THE SUM OF THE SQUARE OF THE OTHER TWO SIDES OR C = SQRT(A**2 + B**2). THE INPUTS, "A" AND "B", CORRESPOND TO THE SIDES ADJACENT TO THE RIGHT ANGLE (I.E. 90 DEGREE ANGLE), WHILE THE OUTPUT, "C", CORRESPONDS TO THE SIDE OPPOSITE TO THE RIGHT ANGLE WHICH IS CALLED THE HYPOTENUSE.

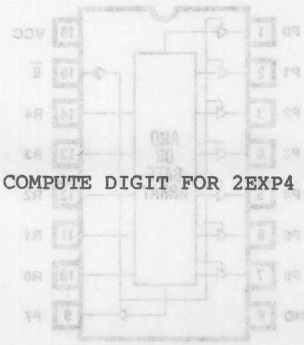
C = SQRT(A**2 + B**2) WHERE C = LENGTH OF SIDE C (THE HYPOTENUSE)
A = LENGTH OF SIDE A
B = LENGTH OF SIDE B

EXAMPLE: FOR A = 2 AND B = 4, C = SQRT(2**2 + 4**2) = 4.47





```
PLE5P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5025                                  PETER WITTFOTH 06/02/84
PERIMETER OF A CIRCLE LOOK-UP TABLE
MMI GMBH MUNICH
.ADD R0 R1 R2 R3 R4
.DAT P0 P1 P2 P3 P4 P5 P6 P7


P0 =     /R1* R2*/R3*/R4      ; COMPUTE DIGIT FOR 2EXP0 (1) (LSB)
   + /R0*/R1* R2*    /R4
   +      R1* R2*     R4
   +      R1*/R2*/R3
   +  R0*/R1*/R2* R3
   + /R0* R1*/R2
   +      R1*/R2*    /R4
   +     /R1*/R2*     R4

P1 =  R0*    /R2*/R3  ; COMPUTE DIGIT FOR 2EXP1 (2)
   + /R0* R1* R2*/R3
   + /R0*    /R2* R3
   +  R0*     R2* R3
   + /R0*     R2*/R3* R4
   +  R0*/R1* R2*    /R4
   +     /R1* R2* R3*/R4
   +  R0* R1*    R3* R4

P2 =  R0*/R1*    /R3*/R4      ; COMPUTE DIGIT FOR 2EXP2 (4)
   +  R0* R1*/R2*/R3* R4
   + /R0* R1* R2* R3* R4
   + /R0* R1*/R2*   /R4
   +      R1* R2*/R3*/R4
   +  R0* R1*    R3*/R4
   + /R0*/R1*/R2*    R4
   +     /R1* R2*/R3* R4
   +  R0*/R1*    R3* R4

P3 = /R0* R1*/R2*    /R4      ; COMPUTE DIGIT FOR 2EXP3 (8)
   +  R0*/R1*/R2* R3
   +  R0*/R1*/R2*     R4
   +  R0*    /R2* R3* R4
   +  R0*     R2*/R3*/R4
   + /R0*     R2* R3*/R4
   +      R1* R2* R3*/R4
   + /R0*     R2*/R3* R4
   +      R1* R2*/R3* R4
   + /R0* R1* R2*     R4
   + /R0*/R1* R2*/R3

P4 =  R0* R1*/R2*/R3          ; COMPUTE DIGIT FOR 2EXP4 (16)
   +      R1*/R2*/R3* R4
   + /R0*/R1* R2*/R3
   +  R0*/R1* R2*    /R4
   +     /R1*/R2* R3
   + /R0* R1*     R3*/R4
   +      R1* R2* R3*/R4
   +     /R1*     R3* R4
   + /R0*     R2* R3* R4
```



8

PERIMETER OF A CIRCLE LOOK-UP TABLE (cont'd)

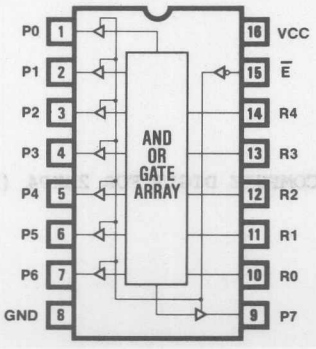
```
P5 =      R1* R2*/R3*/R4      ; COMPUTE DIGIT FOR 2EXP5 (32)
   +     /R1*/R2* R3*/R4
   +        /R2*/R3* R4
   +     R1*/R2*     R4
   +    /R1* R2* R3* R4
   + /R0* R1*/R2* R3
   + /R0*/R1* R2*     R4
   + /R0*     R2* R3* R4

P6 =  R0* R1*     R3*/R4      ; COMPUTE DIGIT FOR 2EXP6 (64)
   + /R0*/R1*    /R3* R4
   +          R2* R3*/R4
   +         /R2*/R3* R4
   +  R0* R1* R2* R3

P7 =  R0*     R2*     R4      ; COMPUTE DIGIT FOR 2EXP7 (128) (MSB)
   +     R1* R2*     R4
   +             R3* R4
```

FUNCTION TABLE

;---RADIUS----					-------PERIMETER-------									
; INTEGER					MSB		INTEGER				LSB			PERIMETER OF A CIRCLE
R4	R3	R2	R1	R0	P7	P6	P5	P4	P3	P2	P1	P0	;RADIUS	LOOK-UP	CALCULATED
L	L	L	L	L	L	L	L	L	L	L	L	L	0	0	0.0
L	L	L	L	H	L	L	L	L	L	H	H	L	1	6	6.3
L	L	L	H	L	L	L	L	L	H	H	L	H	2	13	12.6
L	L	L	H	H	L	L	L	H	L	L	H	H	3	19	18.8
L	L	H	L	L	L	L	L	H	H	L	L	H	4	25	25.1
L	H	L	L	L	L	L	H	H	L	L	H	L	8	50	50.3
H	L	L	L	L	L	H	H	L	L	H	L	H	16	101	100.5
H	H	H	H	H	H	H	L	L	L	L	H	H	31	195	194.8

PERIMETER OF A CIRCLE LOOK-UP TABLE

PLE5P8



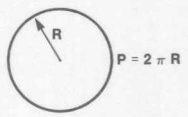
PERIMETER OF A CIRCLE LOOK-UP TABLE (cont'd)

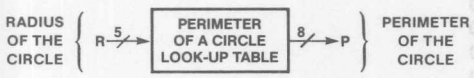
DESCRIPTION

THIS EXAMPLE ILLUSTRATES HOW TO IMPLEMENT A LOOK-UP TABLE IN A PLE5P8 FOR THE PERIMETER OF A CIRCLE AS A FUNCTION OF THE RADIUS. THE INPUT PINS (R4-R0), WHICH REPRESENT THE RADIUS OF A CIRCLE, ARE MULTIPLIED BY 2 TIMES PI IN ORDER TO CALCULATE THE PERIMETER OF A CIRCLE (P7-P0). THIS LOOK-UP TABLE IS VALID FOR RADII BETWEEN 0 AND 31. A PLE8P8 SHOULD BE USED INSTEAD IF A LARGER RADIUS RANGE (BETWEEN 0 AND 81) IS REQUIRED.

P = 2*PI*R WHERE P = PERIMETER OF THE CIRCLE
PI = 3.1415
R = RADIUS OF THE CIRCLE (BETWEEN 0 AND 31)

EXAMPLE: FOR R = 3, P = 2*PI*3 = 19





8

```
PERIOD OF OSCILLATION FOR A MATHEMATICAL PENDULUM LOOK-UP TABLE
MMI GMBH MUNICH
.ADD L0 L1 L2 L3 L4
.DAT T0 T1 T2 T3 T4 T5 T6 T7


T0 = /L4*    /L2*/L1* L0          ; COMPUTE DIGIT FOR 2EXP-5 (0.03125) (LSB)
   +     /L3*/L2* L1*/L0
   +      L3*/L2*     L0
   + /L4*     L2*/L1*/L0
   +  L4* L3*          L0
   +  L4*/L3*     L1
   +  L4*/L3* L2*    /L0

T1 =         /L2* L1*/L0          ; COMPUTE DIGIT FOR 2EXP-4 (0.0625)
   + /L4*/L3* L2*     L0
   + /L4* L3*/L2* L1
   + /L4*     L2*/L1*/L0
   +  L4*/L3*/L2* L1
   +     /L3* L2*/L1
   +  L4* L3*    /L1* L0
   +  L4* L3*     L1*/L0

T2 = /L4*/L3*     L1*/L0          ; COMPUTE DIGIT FOR 2EXP-3 (0.125)
   + /L4* L3*/L2*     L0
   +  L4*/L3*/L2*/L1*/L0
   +  L4*/L3*     L1* L0
   +  L4* L3*/L2* L1*/L0
   +  L4* L3* L2*/L1
   +  L4*     L2*     L0
   + /L4*/L3*    /L1* L0
   + /L4*/L3* L2*    /L0

T3 = /L4* L3*     L1* L0          ; COMPUTE DIGIT FOR 2EXP-2 (0.25)
   +  L4*/L3*     L1
   +  L4* L3* L2*/L1
   +     /L3*/L2*/L1* L0
   +     /L3* L2* L1* L0
   +      L3*/L2* L1* L0
   +      L3* L2*/L1* L0
   + /L4*     L2*/L1* L0

T4 = /L4*/L3*     L1*/L0          ; COMPUTE DIGIT FOR 2EXP-1 (0.5)
   +     /L3* L2* L1
   + /L4* L3* L2*/L1
   +  L4*/L3* L2
   +  L4*     L2* L1
   +          L2* L1*/L0
```

```
   + /L4*     /L2*/L1
   +       L3*      /L0
   +       L3*/L2
   +       L3*      /L1
   +  L4* L3

T6 = /L4*     /L2* L1* L0          ; COMPUTE DIGIT FOR 2EXP1 (2)
   + /L4* L3*         /L0
   + /L4*/L3* L2
   + /L4*/L3*     /L1

T7 =       L3* L2* L1* L0          ; COMPUTE DIGIT FOR 2EXP2 (4) (MSB)
   +  L4
```

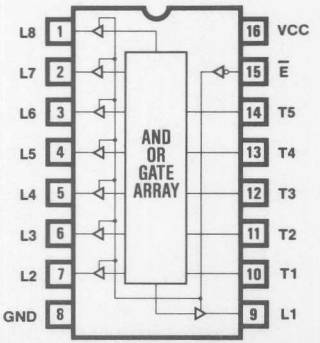
FUNCTION TABLE

;--AMPLITUDE-- INTEGER					--PERIOD OF OSCILLATION-- INTEGER				FRACTION					PERIOD OF OSCILLATION	
; L4	L3	L2	L1	L0	T7	T6	T5	T4	T3	T2	T1	T0	;AMPLITUDE	LOOK-UP	CALCULATED
L	L	L	L	L	L	L	H	L	L	L	L	L	1	2.0000	2.0050
L	L	L	L	H	L	L	H	L	H	H	L	H	2	2.8125	2.8356
L	L	L	H	L	L	L	H	H	L	H	H	H	3	3.4375	3.4728
L	L	H	L	L	L	H	L	L	L	H	H	H	5	4.4375	4.4834
L	H	L	L	L	L	H	H	L	L	L	L	L	9	6.0000	6.0151
H	L	L	L	L	H	L	L	L	L	H	L	L	17	8.2500	8.2670
H	H	H	H	H	H	L	H	H	L	H	L	H	32	11.3125	11.3423

8

PERIOD OF OSCILLATION FOR A MATHEMATICAL PENDULUM LOOK-UP TABLE

PLE5P8



PERIOD OF OSCILLATION FOR A PENDULUM LOOK-UP TABLE (cont'd)

DESCRIPTION

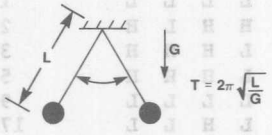
THIS PLE5P8 IS USED TO IMPLEMENT A LOOK-UP TABLE FOR THE PERIOD OF OSCILLATION OF A MATHEMATICAL PENDULUM. THE PERIOD OF OSCILLATION FOR MATHEMATICAL PENDULUM (T) IS DEPENDENT UPON ITS AMPLITUDE OF SWING (L) AND THE ACCELERATION DUE TO GRAVITY (G). THE PERIOD OF OSCILLATION IS CALCULATED USING THE FOLLOWING EQUATION:

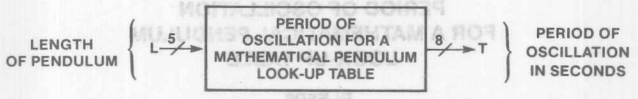
T = 2*PI*SQRT(L/G) WHERE T = PERIOD OF OSCILLATION IN SECONDS
PI = 3.14
L = AMPLITUDE OF SWING IN METERS
G = ACCELERATION DUE TO GRAVITY IN M/S/S
(9.81 M/S/S)

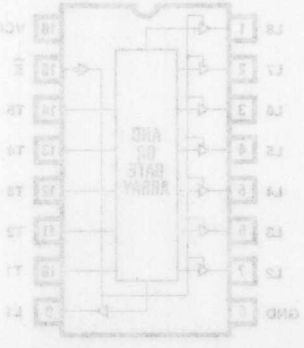
EXAMPLE: FOR L = 5, T = 2*PI*SQRT(5/G) = 4.4375

A PLE DEVICE WITH 5 INPUTS CAN BE USED TO CALCULATE THE PERIOD OF OSCILLATION FOR AMPLITUDES UP TO L = 32 METERS. PLE DEVICE WITH MORE INPUTS SHOULD BE USED TO CALCULATE LARGER PERIODS OF OSCILLATION.

THIS EXAMPLE DEMONSTRATES HOW EASY IT IS TO CONSTRUCT LOOK-UP TABLES FOR COMPLEX ARITHMETIC FUNCTIONS USING PLE DEVICES



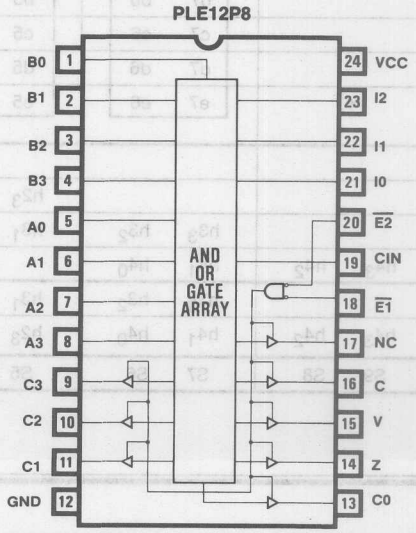




```
PLE12P8                                  PLE CIRCUIT DESIGN SPECIFICATION
P5017                                                    FRANK LEE 10/14/83
ARITHMETIC LOGIC UNIT
MMI SANTA CLARA, CALIFORNIA
.ADD A3 A2 A1 A0 B3 B2 B1 B0 CIN I2 I1 I0
.DAT C3 C2 C1 C0 Z V C

;***********************************************
;*  THIS DESIGN IS NOT YET SUPPORTED BY PLEASM  *
;***********************************************

C,C3,C2,C1,C0 = /S2*/S1* S0*/A3,/A2,/A1,/A0          ;B - A - 1 + CIN
                     .+. B3, B2, B1, B0.+. CIN
              + /S2* S1*/S0* A3, A2, A1, A0          ;A - B - 1 + CIN
                     .+./B3,/B2,/B1,/B0.+. CIN
              + /S2* S1* S0* A3, A2, A1, A0          ;A + B + CIN
                     .+. B3, B2, B1, B0.+. CIN
              +  S2*/S1*/S0*/A3,/A2,/A1,/A0          ;A XOR B
                     :*: B3, B2, B1, B0
              +  S2*/S1* S0* A3, A2, A1, A0          ;A + B
                     +  S2*/S1* S0* B3, B2, B1, B0
              +  S2* S1*/S0* A3, A2, A1, A0          ;A * B
                     * B3, B2, B1, B0
              +  S2* S1* S0                          ;PRESET

V             =  C:+: C3                             ;OVERFLOW

Z             = /C3*/C2*/C1*/C0                      ;ZERO

DESCRIPTION

THIS ALU CAN PERFORM 8 FUNCTIONS ON TWO 4-BIT OPERANDS A (A3-A0) AND
B (B3-B0) WITH CARRYIN (CIN) AND GIVES A 4-BIT RESULT C (C3-C0) WITH
CARRYOUT (C).  IT WILL ALSO GIVE STATUS AS OVERFLOW (V) AND ZERO (Z).
THE FUNCTION IS DETERMINED BY A 3-BIT FUNCTION SELECT CODE (S2-S0):
```

MODE	S2	S1	S0	FUNCTION
0	0	0	0	CLEAR
1	0	0	1	B - A - 1 + CIN
2	0	1	0	A - B - 1 + CIN
3	0	1	1	A + B + CIN
4	1	0	0	A XOR B
5	1	0	1	A + B
6	1	1	0	A * B
7	1	1	1	PRESET

ARITHMETIC LOGIC UNIT



Wallace Tree Compression

In performing arithmetic calculations, it may happen that more than two numbers are to be added together. Adding two numbers can be achieved by using a simple adder. If there are more than two numbers to be summed, several levels of adders may be needed. This often causes too much delay.

An alternative is to use Wallace Tree Compression. Suppose there are m numbers each of n-bits wide. Summation over these numbers will range from 0 to m x (2^n-1) which will take $\log_2 [m (2^n-1) + 1]$ bits (rounded UP to the nearest integer). For example, if there are five 2-bit numbers, i.e., m = 5, and n = 2, the sum will be bounded by 5 x (2^2-1) = 15 which will need a total of 4 bits.

One Wallace Tree Compression by itself will not be very useful. But consider if five 8-bit integers are added together. This technique enables vertical compression of these numbers in four groups. This type of vertical compression also eliminates the need of carry propagation. The five numbers are represented by:

A = (a7, a6, a5, a4, a3, a2, a1, a0)
B = (b7, b6, b5, b4, b3, b2, b1, b0)
C = (c7, c6, c5, c4, c3, c2, c1, c0)
D = (d7, d6, d5, d4, d3, d2, d1, d0)
E = (e7, e6, e5, e4, e3, e2, e1, e0)

where the 7^{th} bits are the most significant; the calculation is as follows:

The groups are assigned as follows:

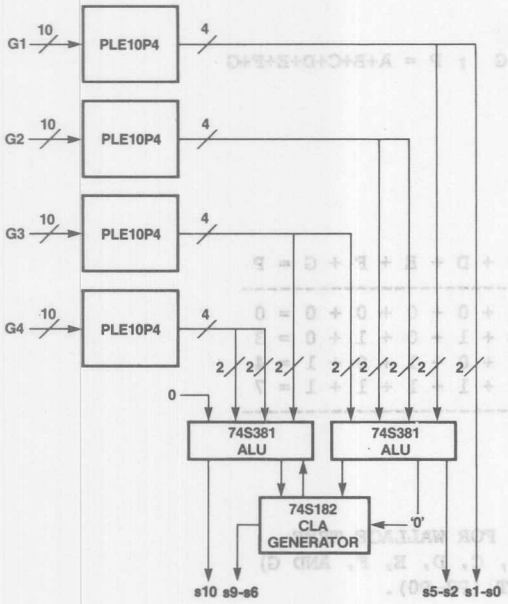
G1 : (a0, a1, b0, b1, c0, c1, d0, d1, e0, e1)
G2 : (a2, a3, b2, b3, c2, c3, d2, d3, e2, e3)
G3 : (a4, a5, b4, b5, c4, c5, d4, d5, e4, e5)
G4 : (a6, a7, b6, b7, c6, c7, d6, d7, e6, e7)

The above groups of bits can be compressed to:

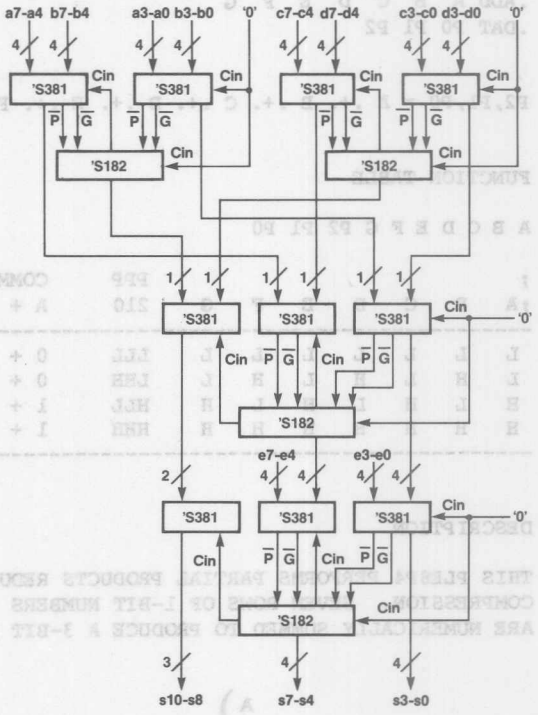
H1 : ($h1_3$, $h1_2$, $h1_1$, $h1_0$)
H2 : ($h2_3$, $h2_2$, $h2_1$, $h2_0$)
H3 : ($h3_3$, $h3_2$, $h3_1$, $h3_0$)
H4 : ($h4_3$, $h4_2$, $h4_1$, $h4_0$)

			G4		G3		G2		G1		
			a7	a6	a5	a4	a3	a2	a1	a0	= A
			b7	b6	b5	b4	b3	b2	b1	b0	= B
			c7	c6	c5	c4	c3	c2	c1	c0	= C
			d7	d6	d5	d4	d3	d2	d1	d0	= D
	+)		e7	e6	e5	e4	e3	e2	e1	e0	= E
							$h1_3$	$h1_2$	$h1_1$	$h1_0$	= H1
					$h2_3$	$h2_2$	$h2_1$	$h2_0$			= H2
			$h3_3$	$h3_2$	$h3_1$	$h3_0$					= H3
+)	$h4_3$	$h4_2$	$h4_1$	$h4_0$							= H4
			$h3_3$	$h3_2$	$h3_1$	$h3_0$	$h1_3$	$h1_2$	$h1_1$	$h1_0$	
+)	$h4_3$	$h4_2$	$h4_1$	$h4_0$	$h2_3$	$h2_2$	$h2_1$	$h2_0$			
S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0	= result

S1 and S0 are just $h1_1$ and $h1_0$. S10–S2 can be obtained through addition of other bits. The hardware implementation is a follows:



It needs four PLE10P4 devices, two 74S381 ALUs and one 74S182. An alternative is using ten 74S381 ALUs and four 74S182 Carry Lookahead Generators.

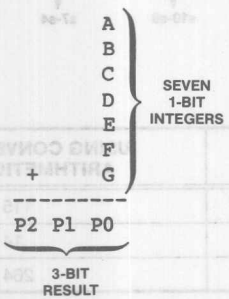


A comparison between the two architectures gives the following data:

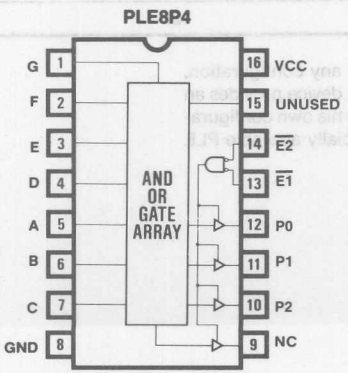
	USING WALLACE TREE COMPRESSION	USING CONVENTIONAL ARITHMETIC LOGIC
Delay (ns)	79	115
Number of components	7	14
Total number pins on the parts	128	264

Since Wallace tree compression can be of any configuration, there is no predefined part available. A PLE device provides an excellent solution. The designer may define his own configuration as long as it can be put in a commercially available PLE device.

```
P5019                                          VINCENT COLI 04/06/83
SEVEN 1-BIT INTEGER ROW PARTIAL PRODUCTS ADDER
MMI SANTA CLARA, CALIFORNIA
.ADD A  B  C  D  E  F  G
.DAT P0 P1 P2

P2,P1,P0 = A .+. B .+. C .+. D .+. E .+. F .+. G  ; P = A+B+C+D+E+F+G


FUNCTION TABLE

A B C D E F G P2 P1 P0

;                                    PPP     COMMENTS
;A   B   C   D   E   F   G           210     A + B + C + D + E + F + G = P
-------------------------------------------------------------------------
 L   L   L   L   L   L   L           LLL     0 + 0 + 0 + 0 + 0 + 0 + 0 = 0
 L   H   L   H   L   H   L           LHH     0 + 1 + 0 + 1 + 0 + 1 + 0 = 3
 H   L   H   L   H   L   H           HLL     1 + 0 + 1 + 0 + 1 + 0 + 1 = 4
 H   H   H   H   H   H   H           HHH     1 + 1 + 1 + 1 + 1 + 1 + 1 = 7
-------------------------------------------------------------------------


DESCRIPTION

THIS PLE8P4 PERFORMS PARTIAL PRODUCTS REDUCTION FOR WALLACE TREE
COMPRESSION.  SEVEN ROWS OF 1-BIT NUMBERS (A, B, C, D, E, F, AND G)
ARE NUMERICALLY SUMMED TO PRODUCE A 3-BIT RESULT (P2-P0).
```



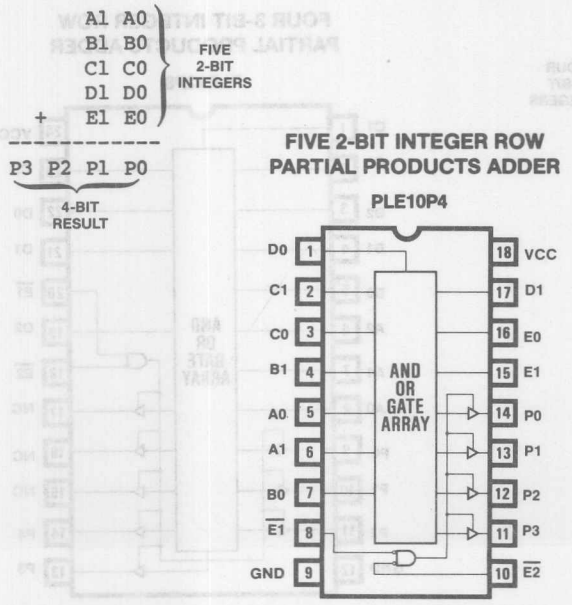
SEVEN 1-BIT INTEGER ROW PARTIAL PRODUCTS ADDER



```
FIVE 2-BIT INTEGER ROW PARTIAL PRODUCTS ADDER
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 B0 B1 C0 C1 D0 D1 E0 E1
.DAT P0 P1 P2 P3


P3,P2,P1,P0 = A1,A0 .+. B1,B0 .+. C1,C0 .+. D1,D0 .+. E1,E0  ; P = A+B+C+D+E


FUNCTION TABLE

A1 A0 B1 B0 C1 C0 D1 D0 E1 E0 P3 P2 P1 P0

;AA   BB   CC   DD   EE    PPPP    COMMENTS
;10   10   10   10   10    3210    A + B + C + D + E = P
-------------------------------------------------------
 LL   LL   LL   LL   LL    LLLL    0 + 0 + 0 + 0 + 0 =  0
 LH   LH   LH   LH   LH    LHLH    1 + 1 + 1 + 1 + 1 =  5
 HL   HL   HL   HL   HL    HLHL    2 + 2 + 2 + 2 + 2 = 10
 HH   HH   HH   HH   HH    HHHH    3 + 3 + 3 + 3 + 3 = 15
-------------------------------------------------------
```

DESCRIPTION

THIS PLE10P4 PERFORMS PARTIAL PRODUCTS REDUCTION FOR WALLACE TREE COMPRESSION. FIVE ROWS OF 2-BIT NUMBERS (A1-A0, B1-B0, C1-C0, D1-D0, AND E1-E0) ARE NUMERICALLY SUMMED TO PRODUCE A 4-BIT RESULT (P3-P0).

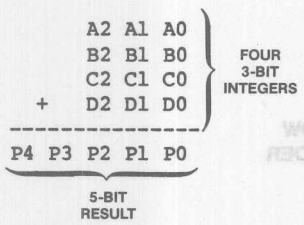


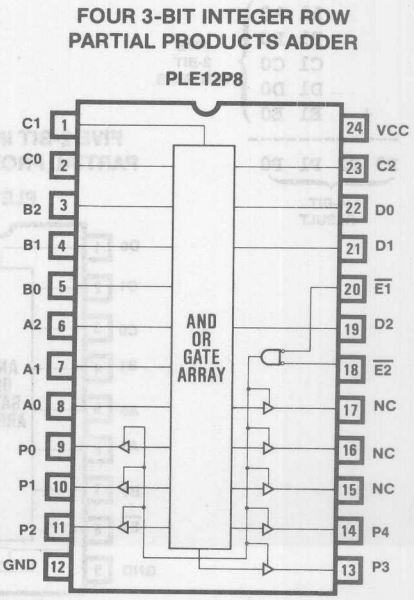
8

```
PLE12P8                          PLE CIRCUIT DESIGN SPECIFICATION
P5021                                          VINCENT COLI 02/10/83
FOUR 3-BIT INTEGER ROW PARTIAL PRODUCTS ADDER
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 A2 B0 B1 B2 C0 C1 C2 D0 D1 D2
.DAT P0 P1 P2 P3 P4


P4,P3,P2,P1,P0 = A2,A1,A0 .+. B2,B1,B0 .+. C2,C1,C0 .+. D2,D1,D0  ; P = A+B+C+D


FUNCTION TABLE

A2 A1 A0 B2 B1 B0 C2 C1 C0 D2 D1 D0 P4 P3 P2 P1 P0

;AAA   BBB   CCC   DDD   PPPPP     COMMENTS
;210   210   210   210   43210     A + B + C + D = P
-------------------------------------------------------
 LLL   LLL   LLL   LLL   LLLLL   = 0 + 0 + 0 + 0 = 0
 LLH   LLH   LLH   LLH   LLHLL   = 1 + 1 + 1 + 1 = 4
 LHL   LHL   LHL   LHL   LHLLL   = 2 + 2 + 2 + 2 = 8
 LHH   LHH   LHH   LHH   LHHLL   = 3 + 3 + 3 + 3 = 12
 HLL   HLL   HLL   HLL   HLLLL   = 4 + 4 + 4 + 4 = 16
 HHH   HHH   HHH   HHH   HHHLL   = 7 + 7 + 7 + 7 = 28
-------------------------------------------------------
```

DESCRIPTION

THIS PLE12P8 PERFORMS PARTIAL PRODUCTS REDUCTION FOR WALLACE TREE COMPRESSION. FOUR ROWS OF 3-BIT NUMBERS (A2-A0, B2-B0, C2-C0, AND D2-D0) ARE NUMERICALLY SUMMED TO PRODUCE A 5-BIT RESULT (P4-P0).

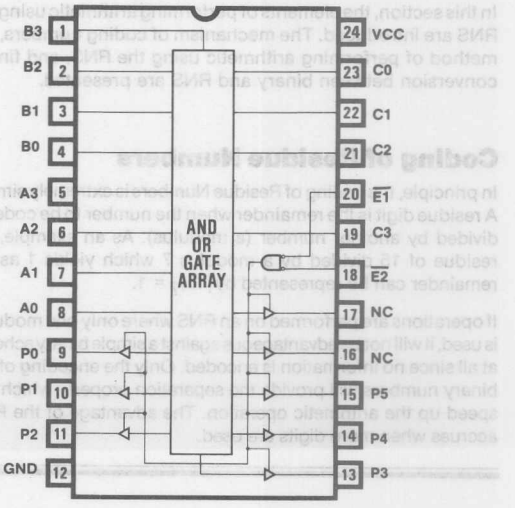




```
PLE12P8                                  PLE CIRCUIT DESIGN SPECIFICATION
P5022                                                 VINCENT COLI 08/10/83
THREE 4-BIT INTEGER ROW PARTIAL PRODUCTS ADDER
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 A2 A3 B0 B1 B2 B3 C0 C1 C2 C3
.DAT P0 P1 P2 P3 P4 P5

P5,P4,P3,P2,P1,P0 = A3,A2,A1,A0 .+. B3,B2,B1,B0 .+. C3,C2,C1,C0  ; P = A+B+C


FUNCTION TABLE

A3 A2 A1 A0 B3 B2 B1 B0 C3 C2 C1 C0 P5 P4 P3 P2 P1 P0

;AAAA   BBBB   CCCC   PPPPPP     COMMENTS
;3210   3210   3210   543210     A   +  B   +  C   =  P
-------------------------------------------------------
 LLLL   LLLL   LLLL   LLLLLL     0   +  0   +  0   =  0
 LLLH   LLLH   LLLH   LLLLHH     1   +  1   +  1   =  3
 LLHL   LLHL   LLHL   LLLHHL     2   +  2   +  2   =  6
 LHLL   LHLL   LHLL   LLHHLL     4   +  4   +  4   = 12
 HLLL   HLLL   HLLL   LHHLLL     8   +  8   +  8   = 24
 HHHH   HHHH   HHHH   HLHHLH    15   + 15   + 15   = 45
-------------------------------------------------------


DESCRIPTION

THIS PLE12P8 PERFORMS PARTIAL PRODUCTS REDUCTION FOR WALLACE TREE
COMPRESSION.  THREE ROWS OF 4-BIT NUMBERS (A3-A0, B3-B0, AND C3-C0)
ARE NUMERICALLY SUMMED TO PRODUCE A 6-BIT RESULT (P5-P0).
```

```
        A3 A2 A1 A0  }
        B3 B2 B1 B0  }  THREE 4-BIT INTEGERS
  +     C3 C2 C1 C0  }
-------------------
P5 P4 P3 P2 P1 P0
   6-BIT RESULT
```

THREE 4-BIT INTEGER ROW PARTIAL PRODUCTS ADDER



8

Residue Arithmetic using PLE Devices

Residue Arithmetic using PLE Devices

Conventional binary arithmetic can be replaced by another kind of computational methodology known as the Residue Number System. The use of this system allows integer arithmetic to be performed by arrays of PLE devices. The idea of PLE devices as arithmetic elements is simply to store pre-computed values of the arithmetic operation in the PLE memory cells and to use the input variables to the arithmetic as addresses to the PLE devices. Since we are computing the results of the arithmetic operations, the same PLE device organization may be used for many different functions. As an example, a 256x8-bit PLE device can be used as a 4x4-bit binary multiplier, or a 4+4-bit binary adder with the output multiplied by any 3-bit constant. It is this flexibility which holds so much appeal for the use of PLE devices as computational elements.

Introduction

Arithmetic operations often involve carry propagation. This propagation causes too much delay for high-speed arithmetic. The Residue Number System (RNS) provides the required separation property needed for high-speed arithmetic. Each digit of the RNS representation is coded into a certain number of bits. In performing the basic operations of addition, subtraction, and multiplication, no information is required to be passed between the digits. Therefore, the number of bits required for representing each digit can be partitioned so that commercially available PLE devices can be used to implement the arithmetic.

Basics of the Residue Number System

In this section, the elements of performing arithmetic using the RNS are introduced. The mechanism of coding numbers, the method of performing arithmetic using the RNS, and finally conversion between binary and RNS are presented.

Coding of Residue Numbers

In principle, the coding of Residue Numbers is extremely simple. A residue digit is the remainder when the number to be coded is divided by another number (a modulus). As an example, the residue of 15 divided by a modulus 7 which yields 1 as the remainder can be represented by $|15|_7 = 1$.

If operations are performed on an RNS where only one modulus is used, it will not be advantageous against a simple binary scheme at all since no information is encoded. Only the encoding of the binary numbers will provide the separation property which will speed up the arithmetic operation. The advantage of the RNS accrues when more digits are used.

Another example of encoding a number using 3 moduli to give a 3-digit RNS representation is as follows: let the moduli be m1 = 3, m2 = 4, m3 = 5. The residues of X = 25 will be shown as xi where i = 1, 2, 3. Thus,

$$X1 = |25|_{m1} = |25|_3 = 1$$
$$X2 = |25|_{m2} = |25|_4 = 1$$
$$X3 = |25|_{m3} = |25|_5 = 0$$

In the RNS using the moduli 3, 4, 5, the number 25 is represented as (1, 1, 0).

The number of unique representations for a set of moduli is the Least Common Mulitple (LCM) of the moduli. The most efficient set of moduli is one in which all moduli are pairwise relatively prime.

Tables 1 illustrates an example of a set of moduli (3, 4) which can represent 12 integers. Note that the representations of 0 and 12 are the same, since the representation repeats itself after 12 integers.

X	(3) x1	(4) x2
0	0	0
1	1	1
2	2	2
3	0	3
4	1	0
5	2	1
6	0	2
7	1	3
8	2	0
9	0	1
10	1	2
11	2	3
12	0	0

Table 1. Representation of 0 to 12 in RNS Using Moduli 3 and 4. The Representation Repeats Itself After 12 Intergers

In table 2, (4, 6) is the set of moduli uses. Since 4 and 6 are not relatively prime, the number of integers that can be represented is not the product of 4 and 6, but instead is the LCM of 4 and 6 which is 12. The representation again repeats itself once every 12 integers.

X	(3) x1	(4) x2
0	0	0
1	1	1
2	2	2
3	3	3
4	0	4
5	1	5
6	2	0
7	3	1
8	0	2
9	1	3
10	2	4
11	3	5
12	0	0
13	1	1
14	2	2
15	3	3
16	0	4
17	1	5
18	2	0
19	3	1
20	0	2
21	1	3
22	2	4
23	3	5
24	0	0

Table 2. Representation of 0x24 for Moduli 4 and 6. Since 4 and 6 are Not Relatively Prime, and Their LCM is Only 12, the Representation Again Repeats Itself Every 12 Integers

Negative numbers are formed in the same way negative numbers are formed in binary (two's complement) system. To form the two's complement of a number in binary, we subtract the number 2^B where B is the number of bits of the representation. In RNS, we subtract the RNS number from mi to form the negative. Table 1 can be rewritten as in table 3 for encoding of negative numbers.

X	(3) x1	(4) x2
0	0	0
1	1	1
2	2	2
3	0	3
4	1	0
5	2	1
-6	0	2
-5	1	3
-4	2	0
-3	0	1
-2	1	2
-1	2	3

Table 3. Representation of -6 to 5 in RNS using Moduli 3 and 4

Arithmetic Using the RNS

For two RNS numbers, X and Y, the result of the addition of the two numbers, Z, in RNS is given by:

$$|xi + yi|_{mi} = zi \text{ for all of the RNS digits.}$$

The same result is found for subtraction and multiplication. This means that arithmetic can be carried out between the same digits of the two numbers, X and Y, without interaction between adjacent digits. The arithmetic is therefore "carry-free". As an example, let us consider the following computation:

$$Z = (863 \times 3942) + (-862 \times 3942) = 3942$$

We only need sufficient dynamic range to represent the result; intermediate overflows can be ignored. Let us choose the following moduli for the RNS representation:

$$m1 = 7,\ m2 = 9,\ m3 = 11,\ m4 = 13$$
$$M = 9009$$

The above set can represent numbers in the range -4505 to 4504, and so this number range is sufficient for the calculation of this example. The computation is shown in table 4.

X	(7) x1	(9) x2	(11) x3	(13) x4	
3942	1	0	4	3	
863	2	8	5	5	
862	1	7	4	4	
-862	6	2	7	9	
863 x 3942	2	0	9	2	
-862 x 3942	6	0	6	1	
Z	1	0	4	3	= 3942

Table 4. Calculating Z = 863 x 3942 + (-862 x 3942) = 3942

Division and Scaling

Division of residue numbers is more complicated than addition, subtraction, or multiplication. If the dividend is exactly divisible by the divisor, the operation is easier. In this case, a division by a number is the same as a multiplication by the inverse of that number. The multiplication inverse of an integer X in modulo arithmetic can be found by finding the vector (d1, . . . ,dn) which satisfies the following:

$$|X.di|_{mi} = 1$$

For example, 95 divided by 5 in moduli 2, 7 and 9 can be done by first finding the vectors representing 95 and the inverse of 5.

$$|95|_2 = 1$$
$$|95|_7 = 4$$
$$|95|_9 = 5$$

So, for the multiplicative inverse of 5, we have:

$$|1/5|_2 = 1 \qquad |5 \times 1|_2 = 1$$
$$|1/5|_7 = 3 \text{ since } |5 \times 3|_7 = 1$$
$$|1/5|_9 = 2 \qquad |5 \times 2|_9 = 1$$

Therefore, $|95/5|_2 = ||95|_2 \times |1/5|_2|_2 = |1 \times 1|_2 = 1$

$$|95/5|_7 = ||95|_7 \times |1/5|_7|_7 = |4 \times 3|_7 = 5$$

$$|95/5|_9 = ||95|_9 \times |1/5|_9|_9 = |5 \times 2|_9 = 1$$

and the answer is 19.

The operation becomes more complicated when the dividend is not exactly divisible by the divisor or one of the moduli of the multiplicative inverse does not exist, say, if the residue of the divisor for that modulus is 0. In this case, we need to obtain the remainder and then subtract the remainder from the dividend and then perform the division. The problem in finding the remainder seems to be the same as performing the division itself. However, this type of division can be done in a process called scaling, which will not be discussed in detail in this paper.

In spite of the improvements made in implementing scaling algorithms, scaling still represents a major effort in any calculation. It is advisable to use RNS only on systems where many arithmetic operations can be performed for each scaling operation.

A System Using an RNS

An RNS is very useful in systems which have predefined operations and dynamic ranges. Moreover, it can only operate on integers, or at most, block floating-point numbers. Since the RNS involves conversions between integers and their RNS representations, and conversions by themselves are already time-consuming, the problem to be solved in the RNS system should be operation intensive.

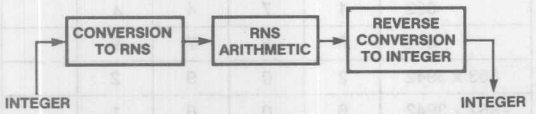


Figure 1. Architecture of an RNS

Conversion to RNS Representation

The conversion of an integer to RNS can be viewed as a mapping process. PLE devices provide a natural implementation for numbers ranging from 0 to 255, and the following moduli are arbitrarily chosen for conversion to RNS — 2, 11 and 15 (which can represent 330 integers), 8 bits of address are needed for the integer input and 9 outputs (1 for modulus 2, 4 for modulus 11 and 4 for modulus 15). In reality, only 8 outputs are needed because that bit of residue for modulus 2 is not required, since the least significant bit of the integer is also the residue of itself in modulus. In fact, a PLE8P8 will be sufficient.

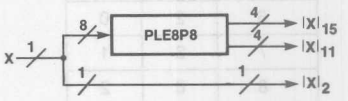


Figure 2. Mapping an 8-Bit Integer, X, to Its Residues on Moduli 2, 11 and 15

Another example is a 14-bit integer which is to be converted to RNS. A 14-bit address needs 16K address spaces for the mapping. 16K is too deep for a PLE device. An alternative is to use 4K-deep PLE devices. PLE12P4 and PLE12P8 devices and a selector (e.g., a PLE5P8 to control the PLE devices (See Figure 3)). The PLE5P8 device will decode two of the address bits and will selectively enable one of the four sets of PLE devices as the mapping set, thus deepening the effective address to 16K.

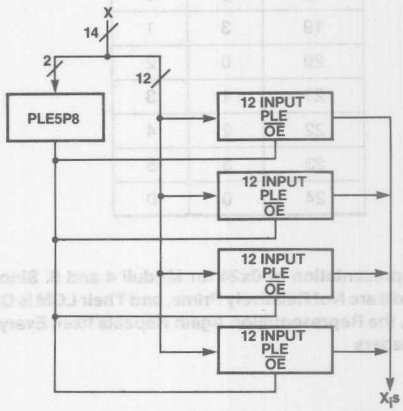


Figure 3. Mapping a 14-Bit Integer, X, to Its Residues by Selectively Enabling the Outputs of One of the Four Sets of 12-Input PLE Devices

This method of expansion is not effective with bigger integers. If the integer is N-bit and the PLE address space available is M-bit, then 2^{N-M} sets of PLE devices will be needed. Besides, as the dynamic range increases, the width of the outputs will also increase about proportionally. An alternative method is to use two or more levels of PLE devices to generate the residues. The first level generates the remainders from the more significant bits of the integer and the products of some of the moduli. These remainders are in turns concatenated with the rest of the bits to become the inputs to the second level PLE devices.

For example, for a 16-bit integer 43689, and let us use (2, 11, 13, 15, 23) as the set of moduli. We may choose 23, 30 and 143 as the moduli for the first level. The first level consists of PLE12P4s and PLE12P8s which generate the remainders of the most significant 12 bits of 43689 which is 2730. We know that $|2730|_{23}$ will be at most 22 and can therefore be represented by a 5-bit number; $|2730|_{15}$ will be at most 14 and can be represented by another 4-bit number; and $|2730|_{143}$ will be at most 142 and can be represented by a 6-bit number. The 5-bit number represented by $|2730|_{23}$ will be concatenated with the least significant 4 bits of the integer and gives a 9-bit number which can perform another division by 23 to give the final $|43689|_{23}$; the 4-bit number represented by $|2730|_{15}$ will be concatenated with the least significant 4 bits of the integer and gives an 8-bit number which can perform another division by 15 to give the final $|43689|_{15}$; the 6-bit number represented by $|2730|_{143}$ will be concatenated with the least siginificant 4 bits of the integer and gives a 10-bit number which can perform another division by 11 and 13 to give the final $|43689|_{11}$ and $|43689|_{13}$. As in the first example, $|43689|_2$ is just the least significant bit of the integer.

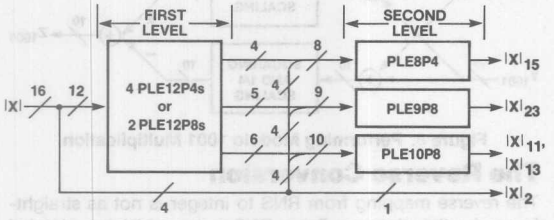


Figure 4. Mapping a 16-Bit Integer X to Residues in Modulo 2, 11, 13, 15, and 23 Using Two-Level Mapping. The First Level Gives Remainders from the More Significant Twelve Bits, While the Second Level Finds the Final Residues

In some circumstances, although an N-bit integer only has a dynamic range of 2^N, the intermediate calculations may overflow. It is sometimes necessary to add some other moduli to boost up the dynamic range for the intermediate calculations.

Arithmetic Operations In RNS

The arithmetic operations of the RNS is different from regular arithmetic in that even simple addition must be performed in modulo arithmetic. Simple ALU may not be able to handle this arithmetic. Again, PLE devices are proven to be most useful. A PLE8P4 device can perform addition, subtraction, or multiplication on two 4-bit residue numbers and give a 4-bit modulo result.

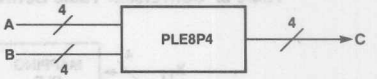


Figure 5. Calculating C = A + B, A - B, B - A, or A x B Using PLE8P4

If the modulus is large, say greater than 64, the combined number of bits for two residues will be greater than the number of address bits for the largest of the commercially available PLE device. Of course, more than one device can be used to deepen the effective address space. In this case, for every additional bit of a modulus, two more bits of address will be needed — one for each operand. In other words, for each additional bit of a modulus the address space of operation will be quadrupled. It is not very effective when the modulus grows too large. Fortunately, for both addition and multiplication, there are more efficient procedures.

Large Modulus Addition

Table 5 shows the contents required for the addition operations in modulus 11. There is a lot of redundancy in the table which can be compressed by reducing what should be eight bits of inputs to five bits. What we need is just another level of mapping. There are a total of 121 combinations for a number of modulus 11 operating on another operand of the same modulus. In reality, only numbers ranging from 0 to 10 can be represented in modulus 11. The sum ranges from 0 to 20 (not in modulus 11). This range can be represented by a new set of submoduli (3, 7) which is five bits wide. In fact, any new set of submoduli which has a dynamic range of at least twenty-one can be used. The operands in modulus 11 will be converted to their representations in submoduli 3 and 7. The addition is done in the submoduli and the result is reconverted back to modulus 11 RNS (see Table 6).

x_{11} / 4

y_{11} / 4

	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

Table 5. Addition Table in Modulo 11 Arithmetic

8

x + y	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
|x + y| 11	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5	6	7	8	9
|x + y| 7	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6
|x + y| 3	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2

Table 6. Conversion Table Between Modulo 11 Arithmetic and Modulo 3 and 7 Arithmetic

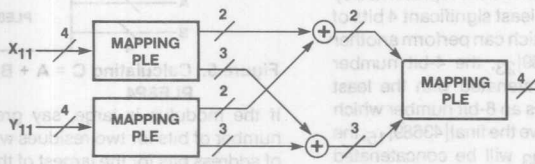


Figure 6. Calulating Addition of Two Numbers in Modulo 11 Using Submoduli Operations

Large Modulus Multiplication

The solution to this problem in multiplication is similar. For example, if two RNS digits in modulus 91 is to be multiplied, (7, 13) may be chosen as a set of submoduli. The representation of an RNS digit in modulus 91 needs 7-bits. These 7-bits are first mapped to two RNS digits — in modulo 7 which needs 3-bits; and in modulo 13 which needs 4-bits. The representations of the two operands in the two moduli can then be multiplied and give the result in modulo 7 and modulo 13. The result is then converted back to modulo 91. Unfortunately, this scheme can be used only when the modulus can be expressed as a product to two integers which are relatively prime. But, in this case, the RNS digit may simply be represented as the residue of the two smaller integers instead of using them as submoduli.

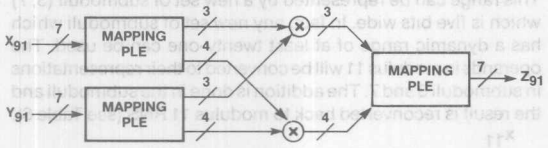


Figure 7. Calculating Multiplication of Two Numbers in Modulo 91 Using Submoduli Operations

Suppose another modulus 101 is used. 101 is a prime number and RNS in modulus 101 ranges from 0 to 100. The real dynamic range of the product of two numbers in modulus 101 is 0 to 10000, which is already too large for a PLE address space. For this modulus, we may use three 4 K-deep PLE devices to deepen the address space. For a modulus like 1001, it may not be too efficient to use this scheme. Instead, since:

$$xy = [(x + y)^2 - (x - y)^2] / 4$$
$$\text{or} \quad = [x + y)^2] / 4 - [(x - y)^2] / 4$$

we may do x + y and x − y first and then do the squaring of the sum and the difference scaled by a factor of 4. Since the final product of two integers must be an integer, the squaring and scaling may be performed in one operation with the fractional part discarded. The way to obtain x + y and x − y is the same as what was discussed earlier in the "Large Modulo Addition" session.

In any event, operations on residues of large moduli are slower and involve more hardware and are not recommended.

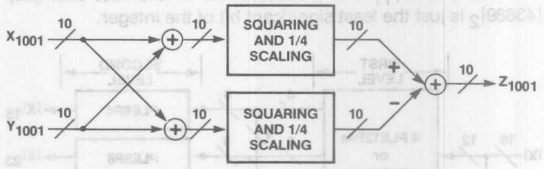


Figure 8. Performing Modulo 1001 Multiplication

The Reverse Conversion

The reverse mapping from RNS to integer is not as straightforward as the other way. For an RNS system which has a total of twelve bits for all the residues, we can still use 12-input PLE devices to convert. We may also use several sets of 12-input PLE devices to reverse map the RNS if the integer is not much longer. But for very long integers, we may need to use the general algorithm for the reverse map:

1. Find $M = m1 \times m2 \times \ldots \times m_{n-1}$ (where n is the number of moduli)
2. Find ti = M/mi
3. Find $X = |x1t1 + x2t2 + \ldots + xn{-}1tn{-}1|M$

In hardware implementation, ti's are all known beforehand, We can map xi's to get the xiti's. Then we may perform Wallace Tree Compression (see the session on this subject in this handbook for more information) on the xiti's to give two-level operands which add to the final sum and divide it by M to get X. Again, PLE devices provide the best solution for Wallace Tree Compression.

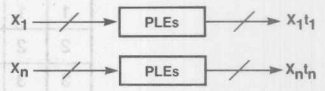


Figure 9a. Reverse Mapping to Get $X_i t_i$

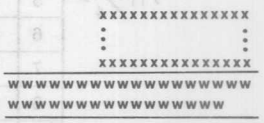


Figure 9b. Modulo M Wallace Tree Compression to Reduce the Number of Levels for Summation to 2 Followed by an Addition and Division to get $X = |x_1t_1 + \ldots + x_nt_n| M$

Conclusion

Memory elements provide excellent solutions to mapping functions — for control purposes, for arithmetic operations and general logic replacements. This paper investigates the possibility of using PLE devices as arithmetic units. In fact, for logic like residue number arithmetic, there is no better solution than to use these devices.

Acknowledgement

Portions of this article were extracted from *"Integer Arithmetic Using PROMs"* by Dr. G. A. Jullien of the University of Windsor, Canada.

Distributed Arithmetic Using PLE™ Devices

Distributed Arithmetic Using PLE Devices

In digital signal processing, sum-of-product type of operations are often necessary. These operations take the form of:

$$y = \sum_{i=1}^{M} a_i x_i \qquad \text{where } a_i\text{'s are some constants}$$

If real multiplications are to be performed on every product term, it will need a total of M multiplications and M-1 additions. Multiplication operations normally take much longer than simple addition. An alternative to calculate equations of the above form is by using distributed arithmetic.

Suppose there is an N-bit integer X given by :

$$x = [x(N-1), x(N-2), \ldots, x(1), x(0)]$$

or equivalently:

$$x = \sum_{j=0}^{N-1} x(j)\, 2^j$$

where x(N-1) is the most significant bit. The equation:

$$y = \sum_{i=0}^{M} a_i x_i$$

can be expressed as:

$$y = \sum_{i=1}^{M} a_i \left(\sum_{j=0}^{N-1} x_i(j)\, 2^j \right) = \sum_{j=0}^{N-1} 2^j \left(\sum_{i=1}^{M} a_i x_i(j) \right)$$

Now, let:

$$H(j) = \sum_{i=1}^{M} a_i x_i(j)$$

Since H (j) is independent of i and since a_1's are all constants, we precompute for every x (j) = $[x_1(j), x_2(j), \ldots, x_M(j)]$ the values of H (j).Then x (j) can be used as the address of PLE devices whose outputs are the precomputed result H (j).

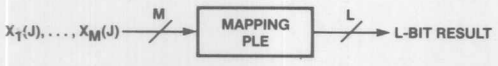


Figure 1. Mapping the j^{th} Bit from Each of the x_i's to An L-bit Result

If there are M bits of data and the result is L-bit wide, and if M is very large, say 20, and L is 8, then we need 20 bits of address lines if we want to use only PLE mapping. Since 20 bits of address translate to 1M words, and there is no available 1M-deep PLE device on the market, it is not realistic to use PLE mapping. Instead, H (j) can be partitioned as follows:

$$H(j) = \sum_{i=1}^{20} a_i x_i(j) \qquad \text{for } M = 20$$

$$= \sum_{i=1}^{10} a_i x_i(j) + \sum_{i=11}^{20} a_i x_i(j)$$

the 20-bit address can be separated to two 10-bit addresses and each of them is individually mapped. The two outputs will then be added together to give H (J). An implementation of this mapping is shown in figure 2.

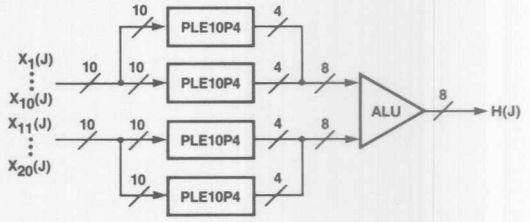


Figure 2. Mapping the j^{th} Bit of Each of x_i's to an L-bit Result When There Are Too Many x's (20 in This Case)

There is another alternative for implementing a sum-of-product operation: by using a multiplier accumulator (MAC).

The main constraint on distributed arithmetic is that one set of the multiplicands must be fixed, i.e. ai's in this case, for the sum-of-product mapping while a MAC will allow flexibility.

There are normally some constraints on the width for the data bus from which the operands are loaded. If all the operands are new, it will need M cycles to load in the operands anyway, distributed arithmetic offers no advantages over MAC since distributed arithmetic needs to wait for all the operands to be loaded in before any operation can start while MAC can perform a multiplication and an addition every cycle. M cycles will be needed anyway for the complete operation using a MAC while distributed arithmetic may take even longer.

On the other hand, for operations like convolutions where one set of operands are fixed and only one new variable operand is needed for every result, distributed arithmetic will be a better solution since it can give a result in every clock-cycle while a MAC will need M-cycles (because recalculations of all the product terms are necessary). An implementation for convolution is shown in Figure 3.

PLE™ is a trademark of Monolithic Memories.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2374
TWX: 910-338-2376

Monolithic Memories

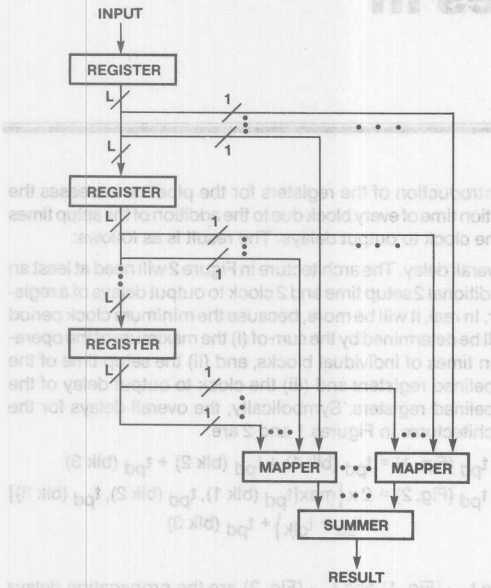


Figure 3. An Implementation of a Distributed Arithmetic System for Convolution

There is another way to implement distributed arithmetic through bit-serialization:

From H (j), the sum-of-product of y can be obtained as:

$$y = \sum_{j=0}^{N-1} 2^j H(j)$$

To implement this equation, consider that the least significant bit of the result is to be used only for rounding purposes only. Only the more significant bits will be retained. The computation can be performed in the following way:

1) For j = 0,
   $y_0 = 2^0 H(0) = H(0)$
2) For j = 1 to N-1
   $y_j = H(j) + 1/2\, H(j-1)$

Note that the second term of the last equation means that the previous result (y_{j-1}) is shifted right one-bit; the last bit of y_{j-1} is truncated.

The implementation of such a system is shown in Figure 3. The system consists of a shift register, a mapper (PLE circuits, or PLE circuits with adders), an accumulator, and an ALU.

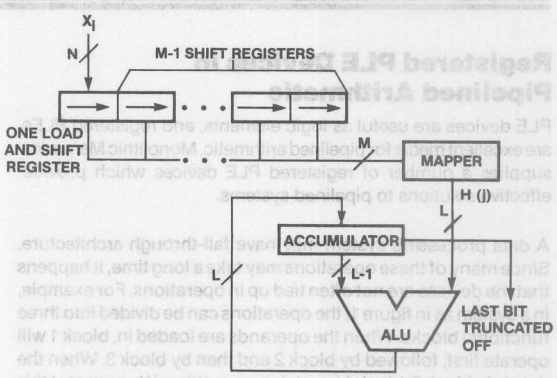


Figure 4. A Bit-Serialization Implementation for a Distributed Arithmetic System

The operations are as follows:

1) Load x_i onto the load and shift register at clock 0.
2) Load H (0) onto accumulator and shift all registers at clock 1.
3) From clock k (between clock 2 to clock N-1), the content of the accumulator will be replaced by the sum of H (k-1) and the more significant N-1 bits of the current accumulator value.
4) For clock N, the following are performed:
   a) Repeat step 3. At the end of the operation, the accumulator contains the value of the result (scaled by the number of shifts).
   b) x_{i+1} is loaded onto the load and shift register. The shifting frequency is equal to N times the basic rate.

Due to the fact that there are a number of shift operations necessary for each data load, this method is recommended for the following conditions:

1) This design is under cost, power dissipation, and board space constraints.
2) This design is for high M-to-N-ratio array multiplications.

8

Registered PLE Devices in Pipelined Arithmetic

PLE devices are useful as logic elements, and registered PLEs are excellent media for pipelined arithmetic. Monolithic Memories supplies a number of registered PLE devices which provide effective solutions to pipelined systems.

A data processing system may have fall-through architecture. Since many of these operations may take a long time, it happens that the devices are not often tied up in operations. For example, in a system as in figure 1, the operations can be divided into three functional blocks. When the operands are loaded in, block 1 will operate first, followed by block 2 and then by block 3. When the data is in block 2, block 1 is not doing anything. We cannot at this time put in the next set of operands because changes in operands may disturb the operation in block 2.

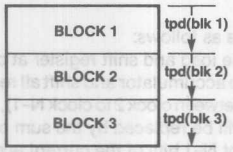


Figure 1. An Example of the Fall-Through Approach to Arithmetic Operation

A solution to this is by registering the operands and signal paths when the operations is switched to block 2; and by registering the operands and signal paths again when the operations is carried out in block 3. The result is stated in figure 2. This architecture is called the pipelined structure. It makes the loading of the second set of operands possible even before the first result is out, thus increasing the throughput.

Figure 2. Pipelined Arithmetic Operation

The introduction of the registers for the pipeline increases the operation time of every block due to the addition of the setup times and the clock to output delays. The result is as follows:

1) Overall delay. The architecture in Figure 2 will need at least an additional 2 setup time and 2 clock to output delays of a register. In real, it will be more, because the minimum clock period will be determined by the sum of (i) the maximum of the operation times of individual blocks, and (ii) the setup time of the pipelined registers and (iii) the clock to output delay of the pipelined registers. Symbolically, the overall delays for the architectures in Figures 1 and 2 are:

$$t_{pd}\text{ (Fig. 1)} = t_{pd}\text{ (blk 1)} + t_{pd}\text{ (blk 2)} + t_{pd}\text{ (blk 3)}$$

$$t_{pd}\text{ (Fig. 2)} = 2 \times \{\max[t_{pd}\text{ (blk 1)}, t_{pd}\text{ (blk 2)}, t_{pd}\text{ (blk 3)}] + t_{su} + t_{clk}\} + t_{pd}\text{ (blk 3)}$$

Where t_{pd} (Fig. 1) and t_{pd} (Fig. 2) are the propagation delays of the architectures in figure 1 and figure 2 respectively; t_{pd} (blk 1) t_{pd} (blk 2), t_{pd} (blk 3) are the propagation delays of block 1, block 2, and block 3 respectively; and t_{su} and t_{clk} are the setup time and clock-to-output delay of the registers respectively.

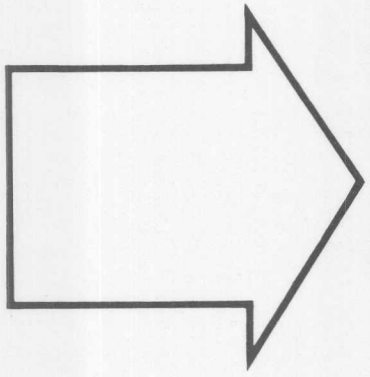
2) Throughputs of clock rate. The architecture in figure 1 has a throughput period of (t_{pd} (blk 1) + t_{pd} (blk 2) + t_{pd} (blk 3) + t_{su} + t_{clk}), assuming that the operands are coming from and the result is going to some registers; the architecture in Figure 2 has a throughput period of (max[t_{pd} (blk 1), t_{pd} (blk 2), t_{pd} (blk 3)] + t_{su} + t_{clk}) which is faster.

PLE devices are useful as logic elements, and registered PLE devices are excellent media for pipelined arithmetic. Monolithic Memories supplies a number of registered PLE devices which provide effective solutions to pipelined systems.

Applications for pipeline arithmetic include array and digital signal processing.

PLE™ is a trademark of Monolithic Memories.

2175 Mission College Boulevard, Santa Clara, CA 95054 Tel: (408) 970-9700 TWX: 910-338-2376 TWX: 910-338-2374



Contents Section 9

Testing Your PAL Devices

Manouchehr Vafai

Introduction

The advantage of Programmable Array Logic (PAL®) circuits as a basic building block of digital system is now well established.

PAL circuits are a unified group of devices which combine programmable flexibility with high speed and extensive selection of interface options.

The architecture of PAL circuits consists of programmable-AND-OR gate arrays, output-registers and I/O feedback as shown in Figure 1.

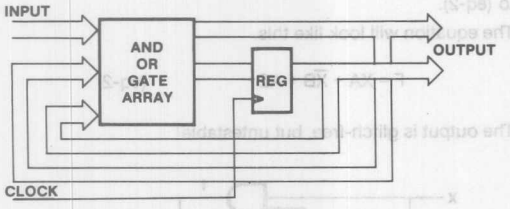


Figure 1. PAL Circuit Architecture

The increased system speed, reduced chip count and availability of a CAD tool called PALASM™ software should leave no doubt for design engineers that they have made a right choice in choosing PAL circuits.

The HAL circuit family is the masked program version of a PAL circuit. HAL® circuits will provide the users a cost-effective solution for large quantities and is unique in that it is a gate array with a programmable prototype.

The following steps are required when designing with PAL circuits.

- Familiarity with Demorgan's law.
- Familiarity with the Karnaugh maps.
- Ability to express logic equation in Sum-of-Product form.
- Ability to write simple seed vector for function table.
- Familiarity with different PAL circuits.

Programming PAL Circuits

PAL circuits will be programmed using PALASM software.

PALASM software is the CAD tool developed by Monolithic Memories to facilitate the process of programming. PALASM software is a Fortran IV program which assembles and simulates PAL circuits design specifications. It generates PAL circuit fuse patterns in formats compatible with PAL circuits or PROM programmers.

Besides generating PAL circuit fuse pattern in different programming formats, PALASM software does the following:

- Assembles PAL circuit design specification and reports error messages.
- Simulates the Function Table.
- Tests each product term for Stuck at zero (SA0) and stuck at one (SA1) faults.

The purpose of writing vectors is to prove that a device is capable of performing it's function before it is put in a system. PALASM software will exercise the vectors and will report any discrepancy. Writing vectors will raise confidence that a device will function properly at least in the design level. The simulator also transfers the function table vectors to a set of universal test vectors which may be used for functional testing after the device is programmed.

When a new system is transferred to production, the system designer hands over the responsibility for the system to the test engineering department, who now determines how and what test should be performed to ensure proper operation of the system. At this point the system designer transmits the necessary information for understanding the system operation. Unfortunately, much information is lost at this point. Test engineers usually have a hard time understanding how the system works with insufficient information. It is the design engineer who best knows the operation of his PAL circuit design, and it is the design engineer who can quickly specify a few seed vectors to give the test a starting point in solving the future problem.

Design for Testability

In short the only way to control a digital circuit is to apply a known value to it's input. Fault simulation has been the best technique of yielding a quantitive measure of test effectiveness. Fault simulation will test stuck-at-0 (SA0) and stuck-at-1 (SA1) of input and output lines. By generating test vectors that will test for each product term for (SA0) and (SA1) faults, then by observing the corresponding output and comparing it with the fault-free output, one can conclude whether a fault can be detected or not.

Consider the following example:

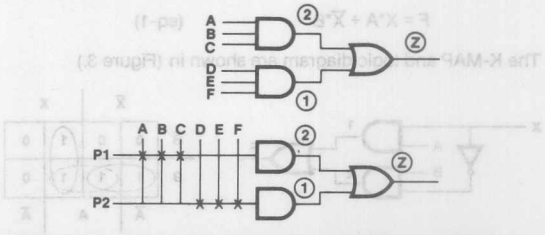


Figure 2. Logic Diagram and It's PAL Circuit Implementation

9

A B C D E F Z
1 1 1 0 X X 1 (vector-1)

The (vector-1) selects a product term P1. Under a fault-free condition, the output (2) will be high (we can observe this); however, under a fault condition the output will be low. In other words, one can conclude that either product term (P1) is (SA0) or outputs Z (Figure 2.) are (SA0).

Now consider vector 2.

A B C D E F G
0 0 0 0 0 0 0 (vector-2)

As it can be seen that both of the product terms are low, if the observed output is high, one can conclude that either product terms or outputs are Stuck-at-one.

Fault simulation grading is used by Monolithic Memories to evaluate candidates design for transfering from a PAL circuit to a HAL circuit.

In designing with PAL circuits, four different cases should be considered.

1. A purely combinational circuit where output is function of input.
2. A purely combinational circuit where output is function of input and feedback from output.
3. A purely sequential logic where output is function of input and feedback from output.
4. A combinational-sequential logic where output is function of input, feedback from combinational output and feedback from sequential output.

In cases 1 and 2 we can define a structured way of writing function table. Cases 3 and 4, on the other hand, because of dependency of the device on the previous state of the device, impose a relatively more sophisticated scheme of testing strategy.

In the following examples the various techniques which might be helpful in testability of PALs, will be discussed.

Example 1: Glitch-free and Testable

Suppose we want to implement (EQ-1) using any of the combinational PAL.

$$F = X^*A + \overline{X}^*B \qquad \text{(eq-1)}$$

The K-MAP and logic diagram are shown in (Figure 3.)

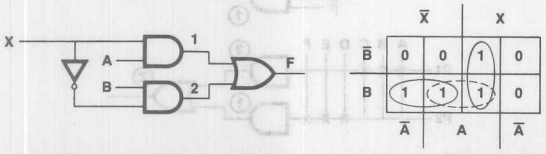


Figure 3. Logic Diagram and It's K-Map

The above logic is testable because we have full control over each node for (SA0) and (SA1) test.

The implementation using PAL circuits is as follows:

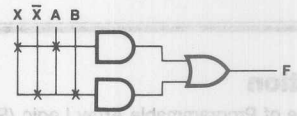


Figure 4. PAL Circuit Implementation of the Logic

Ideally the output should always be high if both inputs are high. The circuit is not glitch-free, the output might momentarily drop to low if we change the state of X, due to propagation delay between X and $\overline{X}$.

The problem will be solved by including a redundant (AB) term to (eq-2).

The equation will look like this.

$$F = XA + \overline{X}B + AB \qquad \text{(eq-2)}$$

The output is glitch-free, but untestable!

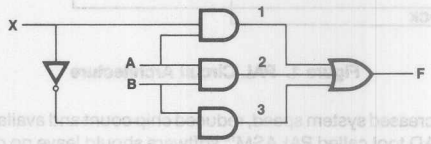


Figure 5. A Glitch-free Circuit

Node (2) is not Observable for (SA0). One can not force node (2) to one and keep node (1) and (3) in the low state. So the redundant product term is untestable.

This circuit can be made testable by the addition of control signal (Y) as follows:

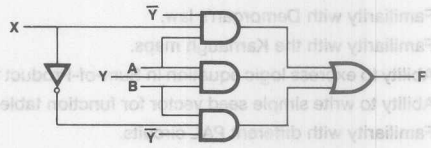


Figure 6. Glitch-free and Testable Logic

Now the logic is glitch-free and testable.

Example 2: Untestable Logic — A Simple Example

The logic F := $\overline{F}$ is untestable

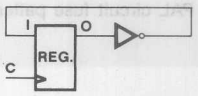


Figure 7. Implementation of F := $\overline{F}$

The initial state of the oscillator is unknown; this system can be made testable as follows:

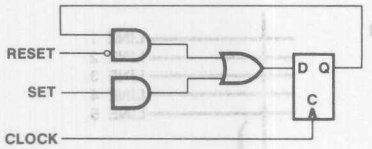


Figure 8. Implementation of F := $\overline{F}$ with SET and RESET

It has been done by addition of two control signals (RESET and SET) and one extra product term.

Illegal States

Upon power-up the initial state of output registers are unknown; this might force the device into one of the "illegal states".

The design engineer should be worried about the illegal state at design time. For example let's look at modulo-6 state machine

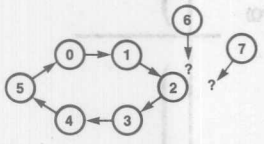


Figure 9. State Transition Diagram for a Modulo-6 Counter

The design engineer might ignore the other possible state (6, 7) but his ignorance might be costly at test time. If upon power-up the machine starts at either of (6) or (7) state, there is no way to control the state-machine. The best solution is to force both of the illegal states into one of the known states.

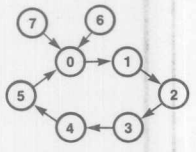


Figure 10. A Modulo 5 Counter with No Illegal State

Example 3: Design "pitfall" Case One

Consider the implementation of the following example

Q1 := I1 * Q1

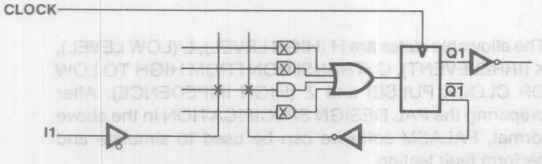


Figure 11. Implementation of Q1 := I1 * Q1 Using a PAL

If Q1 falls to zero, it will stay there forever. The logic needs a control signal for output reset.

Example 4: Design "Pitfall" Case Two

Consider the implementation of the following equation:

$\overline{Q0} = \overline{A}*\overline{Q1}*Q0 + A*Q1*\overline{Q0} + \overline{Q0}$

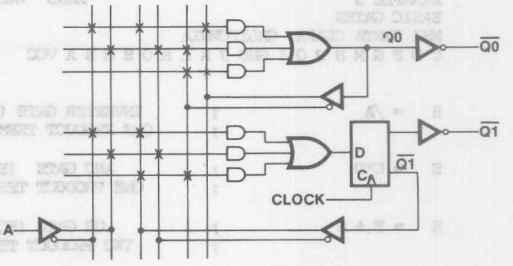


Figure 12. Implementation of $\overline{Q0} = \overline{A}*\overline{Q1}*Q0 + A*Q1*\overline{Q0} + \overline{Q0}$

If $\overline{Q0}$ goes to one it will stay there forever, the logic needs a control signal to clear it's output.

Hard Array Logic (HAL) Devices

The HAL device is the Hard Array version of a PAL device.

HAL logic circuits are the best choice for designs that are firm and volumes are large enough to justify the initial cost. Besides having Boolean equation in PAL DESIGN SPECIFICATION format the user should provide the following.

1. A FUNCTION TABLE which gives enough information about the operation of the device. Normally this FUNCTION TABLE shall test a minimum of 50% "Stuck at fault" grading using PALASM or TEGAS fault grading test.
2. The FUNCTION TABLE shall be constructed such that the device may be initilized to a known state within a specified number of steps (or clocks).

The HAL CIRCUIT SPECIFICATION is the input file used with PALASM software for the HAL's. The input format as shown in example 5 is as follow:

- Line 1 HAL circuit part number
- Line 2 user's part number followed by originator's name and the date
- Line 3 device application name
- Line 4 user's company name, city, state
- Line 5 pin list which is a sequence of symbolic names separated by one or more spaces. All pins including VCC and GND must be named
- Line M the logic equation which are used to generate metal masks from the provided equations

PAL®, and HAL® are Registered Trademarks of Monolithic Memories

```
HAL12H6                                  HAL DESIGN SPECIFICATION
EXAMPLE 5                                 MANO  VAFAI  04/20/83
BASIC GATES
MMI SANTA CLARA, CALIFORNIA
C D F G M N P Q I GND J K L R O H E B A VCC


B   = /A                ;        INVERTER GATE (EQ - 1)
                        ;      ONE PRODUCT TERM: #1 (/A)

E   = C*D               ;          AND GATE  (EQ - 2)
                        ;        ONE PRODUCT TERM: #1 (C*D)

H   = F + G             ;           OR GATE (EQ - 3)
                        ;         TWO PRODUCT TERMS: #1 (F), #2 (G)

L   = /I + /J + /K      ;          NAND GATE (EQ -4)
                        ; THREE PRODUCT TERMS: #1(/I), #2(/J), #3(/K)

O   = /M*/N             ;          NOR GATE  (EQ-5)
                        ;      ONE PRODUCT TERM: #1 (/M*/N)

R   = P*/Q + /P*Q       ;          XOR GATE  (EQ-6)
                        ;     TWO PRODUCT TERMS: #1(P*/Q), #2(/P*Q)

FUNCTION TABLE

A B C D E F G H I J K L M N O P Q R
;AB CDE FGH IJKL MNO PQR     OPTIONAL COMMENTS FIELD
-------------------------------------------------------------------
 LH XXX XXX XXXX XXX XXX     (EQ-1,PT-1)          SA0 TEST
 HL XXX XXX XXXX XXX XXX     (EQ-1,PT-1)          SA1 TEST
 XX HHH XXX XXXX XXX XXX     (EQ-2,PT-1)          SA0 TEST
 XX LLL XXX XXXX XXX XXX     (EQ-2,PT-1)          SA1 TEST
 XX XXX HLH XXXX XXX XXX     (EQ-3,PT-1)          SA0 TEST
 XX XXX LHH XXXX XXX XXX     (EQ-3,PT-2)          SA0 TEST
 XX XXX LLL XXXX XXX XXX     (EQ-3,PT-1,2)        SA1 TEST
 XX XXX XXX LHHH XXX XXX     (EQ-4,PT-1)          SA0 TEST
 XX XXX XXX HLHH XXX XXX     (EQ-4,PT-2)          SA0 TEST
 XX XXX XXX HHLH XXX XXX     (EQ-4,PT-3)          SA0 TEST
 XX XXX XXX HHHL XXX XXX     (EQ-4,PT-1,2,3)      SA1 TEST
 XX XXX XXX XXXX LLH XXX     (EQ-5,PT-1)          SA0 TEST
 XX XXX XXX XXXX HHL XXX     (EQ-5,PT-1,2)        SA1 TEST
 XX XXX XXX XXXX XXX HLH     (EQ-6,PT-1)          SA0 TEST
-------------------------------------------------------------------

DESCRIPTION
THE MAIN PURPOSE OF THIS EXAMPLE IS TO FAMILIARIZE
THE USER WITH WHAT WE MEAN BY "FUNCTION TABLE",
PRODUCT TERM(PT) COVERAGE, STUCK-AT-0(SA0) AND
 STUCK-AT-ONE (SA1) TESTS.
```

LINE 1
LINE 2
LINE 3
LINE 4
LINE 5
LINE M
LINE N

EXAMPLE 5

- Line N the function table which begins with the key word "FUNCTION TABLE." It's followed by a pin list which may be in a different order and polarity from the pin list in line 5. VCC and GND cannot be listed. The pin list is followed by dashed line; e.g.;_ _ _ _ which in turn is followed by a list of vectors, one vector per line. One state must be specified for each pin name and optionally separated by spaces. A vector is a sequence of states listed in the same order as the pin list and followed by an optional comment.

The allowable states are H (HIGH LEVEL), L (LOW LEVEL), X (IRRELEVENT), C (TRANSITION FROM HIGH TO LOW OR CLOCK PULSE) and Z (HIGH IMPEDENCE). After preparing the PAL DESIGN SPECIFICATION in the above format, PALASM software can be used to simulate and perform fault testing.

```
 1 XXXXXXXXXXXXXXXXH01
 2 XXXXXXXXXXXXXXXXL11
 3 11XXXXXXXXXXXXXXHXX1
 4 00XXXXXXXXXXXXXXLXX1
 5 XX10XXXXXXXXXXXXHXX1
 6 XX01XXXXXXXXXXXXHXX1
 7 XX00XXXXXXXXXXXXLXX1
 8 XXXXXXXX0X11HXXXXXX1
 9 XXXXXXXX1X01HXXXXXX1
10 XXXXXXXX1X10HXXXXXX1
11 XXXXXXXX1X11LXXXXXX1
12 XXXX00XXXXXXXXHXXXX1
13 XXXX11XXXXXXXXLXXXX1
14 XXXXXX10XXXXXHXXXXX1

PASS SIMULATION
 PRODUCT:   1 OF EQUATION.  6 UNTESTED(SA1)FAULT
 PRODUCT:   2 OF EQUATION.  6 UNTESTED(SA1)FAULT
 PRODUCT:   2 OF EQUATION.  6 UNTESTED(SA0)FAULT

NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE =  8

NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE =  9

PRODUCT  TERM  COVERAGE                    = 85%
```

$P * \overline{Q}$

$R = P.\overline{Q} + \overline{P}.Q \quad EQ - 6$

$\overline{P} * Q$

FAULT-TESTING

The following information is reported to the user

— Total number of SA1 Faults. (8 in example 5)

— Total number of detected SA0 faults. (9 in example 5)

— $\dfrac{\text{SA1 faults + SA0 faults}}{\text{2 * total number of product terms}}$

$* 100\ \% \quad \left(\dfrac{8+9}{2*10} \quad *100\% = 85\%\ \text{ex-5}\right)$

— One vector may detect more than one SA0 OR SA1 FAULTS (vector # 11 in example 5)

— The user is reported with a message which tells him the product term for which it was not tested. (PRODUCT TERM 1 & 2-EQ 6, in example 5)

The following vectors can be added to the function table in example 5 in order to achieve 100% fault coverage.

```
AB CDE FGH IJKL MNO PQR        COMMENTS (EXAMPLE 5)
XX XXX XXX XXXX XXX LHH        (EQ-6,PT2)      SA0 TEST
XX XXX XXX XXXX XXX HHL        (EQ-6,PT-1,2)   SA1 TEST
```

PALASM™ software has tested the above function table for example 5, the result is as follows:

```
BASIC GATES
 1 XXXXXXXXXXXXXXXXH01
 2 XXXXXXXXXXXXXXXXL11
 3 11XXXXXXXXXXXXXXHXX1
 4 00XXXXXXXXXXXXXXLXX1
 5 XX10XXXXXXXXXXXXHXX1
 6 XX01XXXXXXXXXXXXHXX1
 7 XX00XXXXXXXXXXXXLXX1
 8 XXXXXXXX0X11HXXXXXX1
 9 XXXXXXXX1X01HXXXXXX1
10 XXXXXXXX1X10HXXXXXX1
11 XXXXXXXX1X11LXXXXXX1
12 XXXX00XXXXXXXXHXXXX1
13 XXXX11XXXXXXXXLXXXX1
14 XXXXXX10XXXXXHXXXXX1
15 XXXXXX01XXXXXHXXXXX1
16 XXXXXX11XXXXXLXXXXX1
PASS SIMULATION
NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE = 10
NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE = 10
PRODUCT  TERM  COVERAGE                    =100%
```

PALASM™ is a trademark of Monolithic Memories.

PAL®20RA10 Design for Testability

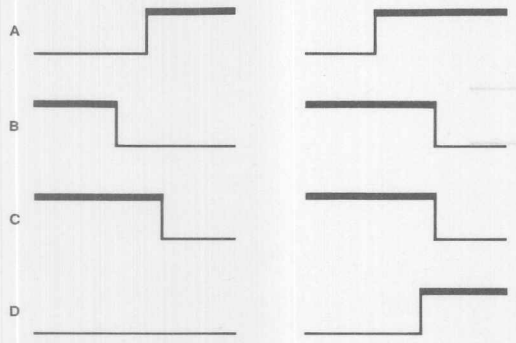
Edwin Young

This article is written to help customers of the PAL20RA1 recognize some fundamental design-for-testability issues which may arise due to the part's unique architecture. Customers should understand that these issues represent design criteria which Monolithic Memories will use to accept PAL20RA10 patterns for test generation/fault grading and for estimating the resource cost to test engineering if accepted. This article does not address the BUSINESS REQUIREMENTS such as the need for acceptable test vectors and the acceptability of a particular pattern for processing as a HAL® device.

The designer who wishes to use a 20RA10 in his/her design must bear in mind that although the part has preloadability, certain designs could diminish the effectiveness of this feature. The following rules are presented to help establish Test Engineering acceptance standards for the 20RA10. Additional general guidelines applicable are available in the PAL Handbook article reprint "Testing Your PAL Devices" by M. Vafai.

Avoid False Latching Situations

The equation D = (A*B) + (C+D) and its variants are susceptible to latching hazards since ATE may have considerable input skew. Of course, from a testing viewpoint, such implementations should be avoided. But if they must be implemented, care must be exercised in developing the function table so as to account for the possibility of latching. The designer must adopt and stick to some guideline such as "no more than one input undergoes a change in logic value per vector" when specifying the function table.

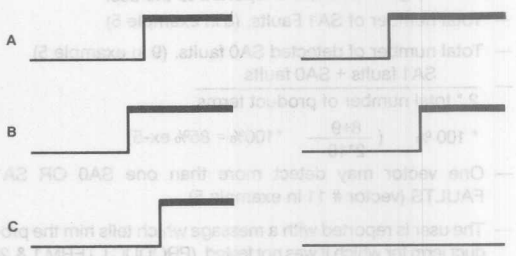


Assume A, B and C are primary inputs while D is a fed-back output. The waveforms to the left show two possible outcomes for output D depending on the skew of inputs A and B, which is a function of tester calibration.

The latch problem described is not unique to the 20RA10 but is clearly applicable to any PAL with asynchronous outputs with feedback (e.g., 16R4). The designer should realize, however, that false latching may occur on a 20RA10 even if all outputs are registered. Consider the equation set D:= C and D.CLKF = A*B for a simple registered 20RA10 output. The resulting waveforms would look similar to those of the previous asynchronous example. The important distinction here is that a 20RA10 has programmable asynchronous clocks rather than a single 'master clock' pin which can cause difficulties in testing.

Allow Data to Setup Prior to Clocking

The previous two pitfalls were examples of flaky latching due to glitches during testing. Consider the equation set C := B and C.CLKF = A for a registered output. The following example shows a definite positive latching...but of flaky (skewed) data.



Assume B is a primary input. Then the timing for situations at far left and left may be with A as feedback and as primary input respectively.

This example illustrates another aspect of the programmable asynchronous clock feature of the 20RA20: Clock pulses can have critical minimal or no delay relative to data setup time. Note that all other registered PAL devices have dedicated common clock pins to which delayed pulses are applied by the ATE to allow sufficient data setup time and ATE input skew.

Avoid Unreachable States

The 20RA10 may be preloaded to any state desired for testing purposes. Unfortunately, the desired state may not exist long enough for the simulator or ATE to use it. With all other preloadable PALs, any arbitrary state may be preloaded into the registers on a given test vector and the state will persist into the next vector providing the required conditions to detect some fault/s. This means all stuck-at-type faults *possibly* detectable can be detected. With the 20RA10, the preloaded state may feed back to assert state dependent resets or presets on one or more

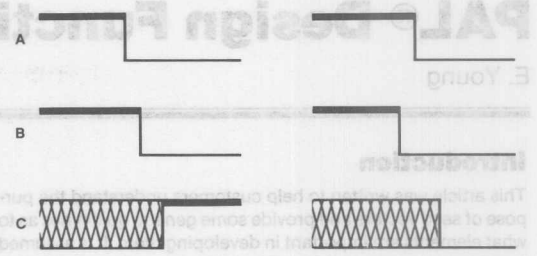
PAL® and HAL® are registered trademarks of Monolithic Memories.

registers. Consequently, the desired state may last only a few nanoseconds after the preload vector is complete before changing to some new state. Since the desired state is not stable going into the vector immediately following the preload vector, the faults expected to be detected become non-detectable.

Another problem arises whenever output control logic is a function of state. In this case, assume the desired state for detecting faults is preload and is stable in the next vector. If this state provides the conditions necessary to detect faults and also disables the outputs, then the faults will be effectivley masked from detection.

Caution on Individual Register Bypass Mode

The 20RA10 allows the designer to permanently and independently bypass any register. Those registers not permanently bypassed may be bypassed under program control by setting both SET and RESET nodes to logic high. In this 'bypass mode', the register's D node is multiplexed to the output rather than its Q node. There is generally no test problem in going into bypass mode. The pitfall is in returning to 'register mode' operation, which only the 20RA10 can do. Consider the equation set C.RSTF = A and C.SETF = B of a simple registered output and



An indeterminate state on the output can occur if both primary inputs A and B go to logic low on the same vector.

the following possible waveforms. A race condition will occur to see whether set or reset operation prevails in going from bypass to register mode. There are two methods by which to get known states for testing purposes:

1) Clock a known value into the register on the next vector or;
2) Set RESET to logic low on one vector and then SET to low on the next or vice versa.

9

PAL® Design Function and Test Vectors

E. Young

Introduction

This article was written to help customers understand the purpose of seed vectors and provide some general guidelines as to what elements are important in developing them. It is assumed that the reader has read the "PALASM™ Manual" and the "PAL® Handbook" article reprint *Testing Your PAL Devices.*

In general, PAL®/HAL® devices are required to provide a function table or "Seed Vectors" to Monolithic Memories in order to ensure that parts shipped have a high degree of reliability for the application intended. Ideally, these vectors should accomplish three objectives:

1) Initialize the PAL device preferably in the same way as in the actual system;
2) Exercise the customer's functions thoroughly, emulating actual system operation as closely as possible;
3) Provide a high degree of fault coverage.

Initialization

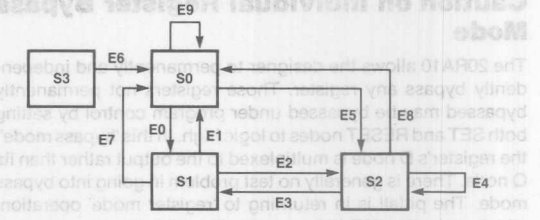
Seed vectors which initialize the PAL logic circuit consist of one or more vectors placed at the very beginning which will bring both combinatorial and registered outputs to a stable and known logic state (1 or 0). This is necessary in the system also so that its operation upon power-up is predictable. Furthermore, care should be taken to ensure that the initialization state is a legal state of the state machine for which the PAL device is intended.

Exercise Functions

The essential functions for which the PAL device was originally designed must be exercised fully. This will assure that the tested parts work the way they were intended to. In addition to essential "designed-for" functions, it is prudent to include general test exercises such as verifying that outputs don't change in the absence of clock pulses and checking to see that inputs in the "don't care" (X) state don't produce adverse responses. General test exercises help to reinforce the validity of a design and can uncover overlooked design errors. After a set of exercises has been decided upon, the next step is to write them in a format suitable for simulation purposes.

The designer may have originally defined the functions in terms of equations, state diagrams, truth tables, etc. Truth tables are readily reformatted to PALASM1 syntax "Function Tables" and exercises with the simulation option (code=S). State diagrams can be converted by expressing each state and input edge in binary vector format and sequencing them according to the diagram's flow. The customer should become thoroughly familiar with the syntax of the function table description (see the PALASM Manual for a detailed treatment of syntax) before attempting to translate truth tables, etc.

The following simple example demonstrates how exercising seed vectors might be derived from a designer's state diagram:



Assume the following state and edge definitions accompany the diagram:

STATE 0 = LL	EDGE 0 = LL
STATE 1 = LH	EDGE 1 = LH
STATE 2 = HL	EDGE 2 = LL
(ILLEGAL) STATE 3 = HH	EDGE 3 = LH
	EDGE 4 = LL
	EDGE 5 = HH
	(INITIALIZING) EDGE 6 = HL
	(INITIALIZING) EDGE 7 = HL
	(INITIALIZING) EDGE 8 = HL
	(INITIALIZING) EDGE 9 = HL

From the above information, it is possible to create the truth table for the diagram and then the function table representation:

EDGE	PRESENT STATE	NEXT STATE
AB	CD	CD
HL	XX	LL
HH	LL	HL
LL	HL	HL
LH	HL	LH
LH	LH	LL
LL	LL	LH
LL	LH	HL

PAL® and HAL® are registered trademarks of Monolithic Memories.
PALASM™ is a trademark of Monolithic Memories.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2374
TWX: 910-338-2376
Monolithic Memories MMI

FUNCTION TABLE REPRESENTATION		
FUNCTION TABLE		
AB	CD	
HL	LL	INITIALIZE DEVICE
HH	HL	TEST EDGE 5
LL	HL	TEST EDGE 4
LH	LH	TEST EDGE 3
LH	LL	TEST EDGE 1
LL	LH	TEST EDGE 0
LL	HL	TEST EDGE 2

Fault Coverage

Another criterion for seed vector completeness is "fault coverage". Fault coverage is an empirical method and is more quantitative than functional exercising — indeed, no knowledge of the circuit's intended function is necessary or assumed (although it could help) while developing fault coverage vectors.

Fault coverage, being an empirical approach to determining a logic circuit's reliability, uses the concept of "failure models" to grade the effectiveness of a given set of test vectors. This is called "fault grading". In fault grading a set of vectors, a fault coverage value is calculated that is simply the ratio of detected faults to total faults expressed as a percentage.

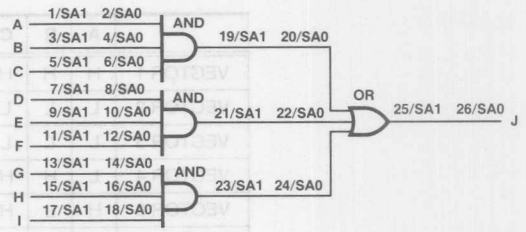
Test vectors may be graded against one or more failure models. Some well-known models include single stuck-at-1/stuck-at-0, pattern sensitivities, shorts and opens and multiple stuck-at models. Selection of a failure model (or models) for fault grading fundamentally depends on the model's empirical effectiveness for screening bad parts and will be affected by a number of factors including circuit technology and fault simulator capabilities.

The most common and primary fault coverage failure model considered by "TGEN" at Monolithic Memories is the classic single stuck-at-1/stuck-at-0 failure model. "TGEN" automatically appends test vectors which test for the following additional failure models where applicable: 1) Adjacencies, 2) Clock, 3) Tri-state.

"TGEN" has a specified minimum value of fault coverage for PAL and HAL devices based on the single stuck-at failure model. The minimum values are determined by current "TGEN" policy (see your FAE) and reflect the economic trade-off between acceptable levels of reliability and the cost of test generation for maximum coverage. PAL and HAL devices for which the specified minimum values cannot be attained will require the customer's written waiver for low coverage prior to production release of the pattern.

The fault coverage percentage determined by "TGEN" is different from the percentage determined by selecting the fault testing option (code=F) of PALASM1 software. In PALASM1 software fault coverage is based on product term coverage (PTC). PTC is still the ratio of detected to total faults except that "detected" and "total" fault sums refer to stuck-at faults on product term outputs only. PTC ignores stuck-at faults which occur anywhere else. A more accurate procedure is to calculate the coverage based on all the circuit nodes where a stuck-at condition may occur. When every node (fault site or wire) is considered, the coverage calculated correlates to the design's testability better and will generally be a much lower value than PTC. "TGEN" goes one step further in conservatism by calculating fault coverage on a "collapsed" fault basis. Fault collapsing simply divides all the stuck-at faults into groups such that, within a group, if one fault is detected, then all the others in the group are detected too. The advantage of collapsing it that only one representative fault in a group needs to be selected for test generation and if it is detected, then the other "equivalent" faults are detected by definition. This saves time and effort on test generation for equivalent faults. Calculations on a collapsed fault basis treat each group as one fault.

The following simplified example demonstrates the difference in fault coverage calculations using a collapsed fault list:



Note: This example is a simplified one for illustrative purposes only and does not show the effects of faults normally associated with input or output buffers. Also, some partial collapsing has already been done (i.e., input faults of "OR" gate are collapsed into output faults of "AND" gates).

Assume the above circuit is to be realized as a PAL or HAL device. Suppose some seed vectors are provided also, as shown here:

	A	B	C	D	E	F	G	H	I
VECTOR 1	H	H	H	L	L	L	L	L	L
VECTOR 2	L	L	L	H	H	H	L	L	L
VECTOR 3	L	L	L	L	L	L	H	H	H
VECTOR 4	L	L	L	L	L	L	L	L	L

The seed vectors on the previous page yield various values for fault coverages corresponding to the method of calculation as shown in the table below.

	PTC	EVERY NODE	COLLAPSED	
SUBSET OF TOTAL FAULTS CONSIDERED FOR CALCULATIONS THAT ARE DETECTED BY EACH VECTOR (SHOWN AT FAR RIGHT)	20	2,4,6,20,26	2	Vector 1
	22	8,10,12,22	8	Vector 2
	24	14,16,18,24	14	Vector 3
	19,21,23	19,21,23,25	19	Vector 4
TOTAL FAULTS CONSIDERED FOR CALCULATIONS	19,20, 21,22, 23,24	1,2,3,4,5,6,7,8,9,10,11, 12,13,14,15,16,17,18, 19,20,21,22,23,24,25,26	1,2,3,5,7, 8,9,11,13, 14,15,17,19	
PERCENT COVERAGE	6/6 = 100%	17/26 = 65%	4/13 = 31%	

As can be seen from the above example, given the same seed vectors, PALASM1 software would show 100% coverage whereas "TGEN" would show 31% coverage. Notice that the poor coverage by "TGEN" is due to none of the input nodes being tested for stuck-at-1. In most instances, a better set of test vectors can improve the coverage significantly. For the above example, the reader can verify that the following slight modification of the seed vectors would yield 100% coverage for all calculation methods:

	A	B	C	D	E	F	G	H	I
VECTOR 1	H	H	H	L	L	L	L	L	L
VECTOR 2	L	L	L	H	H	H	L	L	L
VECTOR 3	L	L	L	L	L	L	H	H	H
VECTOR 4	L	H	H	L	H	H	L	H	H
VECTOR 5	H	L	H	H	L	H	H	L	H
VECTOR 6	H	H	L	H	H	L	H	H	L

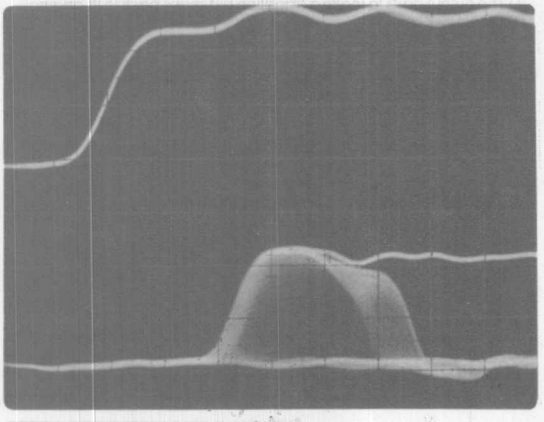
Testability

The previous sections described some essentials for comprehensive seed vector set. Variations on how fault coverage is calculated was covered also. However, no matter how it is calculated, fault coverage is only as good as the testability of the circuit permits. Using the stuck-at failure model, the customer must consider both absolute and practical fault coverages achievable for his PAL/HAL logic circuit design. Certain testability factors, such as redundancy, number of test points (outputs) or reconvergence, affect absolute (i.e., theoretical maximum) coverage. Other factors, including preloadable state machines, the amount of feedback and overall controllability, will affect practical coverage since many faults may be potentially detectable but uneconomical to detect due to excessive vectors or difficult to reach states. As testability is improved, absolute and practical fault coverage will usually increase.

METASTABILITY

A Study of the anomalous behavior of synchronizer circuits

Danesh M. Tavana



INTRODUCTION

This article will summarize the results of the studies performed on synchronizer circuits. The information presented may be used by system designers to gain insight into the anomalous behavior of edge-triggered flip-flops. Understanding flip-flop behavior and applying some simple design practices can result in an increased reliability of any system.

METASTABILITY

In the digital world a bit represents the fundamental unit of measure. The output state of any digital device is either "HIGH" (a voltage level above VIH) or "LOW" (a voltage level below VIL) as shown in figure 2. Under the proper operating conditions the register in figure 1 outputs a HIGH or a LOW on the rising edge of the clock within a nominal delay called the "clock to out" delay. If the setup and hold times are violated the register has a small probability of entering a third region of operation called the "metastable" state. Metastable is a Greek word meaning "in between" and it is a state between HIGH and LOW. Even though most synchronizers snap out of metastability in a short period of time, theoretically this state can persist indefinitely. Some of the registers built from older technologies had metastable states which lasted as long as a few microseconds. When the output of a device goes into metastability the clock to out delay will be grossly affected. This may alter the system's worst case propagation delay and potentially lead to a system crash!

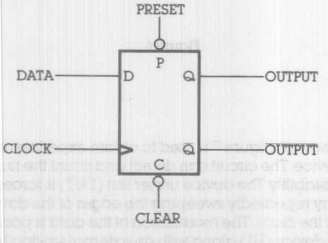


Figure 1

SYNCHRONIZERS

The design of a synchronous digital system is based on the assumption that the maximum propagation delay of a flip-flop and any other gates are known. A digital system is free of hazardous race conditions and timing anomalies if the maximum propagation delay in the system does not exceed the clock's period. In systems where an asynchronous input is interfaced with a clocked device such as a flip-flop, the maximum specified propagation delay of this device may no longer be valid if certain electrical parameters are violated. Computer peripherals, an operator's keyboard, or two independently clocked subsystems are instances where there is a possibility of interfacing an asynchronous input which will violate the synchronizer's electrical parameters.

A popular device typically used in synchronized systems is the edge-triggered register shown in figure 1. The edge-triggered register will properly synchronize the incoming data to the system's clock as long as its operating conditions are satisfied. Table 1 summarizes these specifications for Monolithic Memories Inc's (MMI) 74LS374 register. It is difficult to guarantee setup and hold time requirements when the data is asynchronously interfaced to a register. The violation of setup or hold time in a register has a probability of initiating a misbehavior termed "Metastability."

SYMBOL	PARAMETER	COMMERCIAL MIN.	TYP.	MAX.	UNIT
V_{CC}	Supply Voltage	4.5	5	5.5	V
T_A	Operating free air temp.	0		75	°C
t_w	Width of clock	15			ns
t_{su}	Setup time	20			ns
t_h	Hold time	0			ns

Table 1

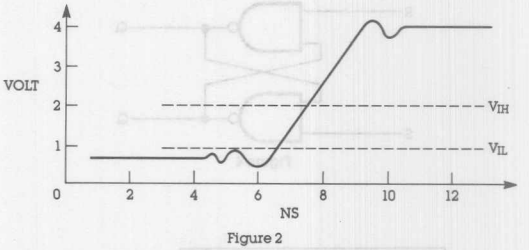


Figure 2

The diagrams in figure 3 illustrate some examples of waveforms in the metastable condition. From the waveforms it is evident that the outputs are distorted under metastable conditions. Figure 3d shows the output of a typical 74LS374 register manufactured by Monolithic Memories. Monolithic Memories family of bipolar devices exhibit superior metastable hardened performance due to their high speed bipolar technology and advance Schottky TTL circuit design techniques. Most of these devices typically snap out of metastability in a flashing 15 nanoseconds.

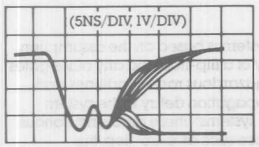
WHY THE SYNCHRONIZER FAILS

Before attempting to explain how the synchronizer's internal circuitry fails let's take a look at an interesting problem.

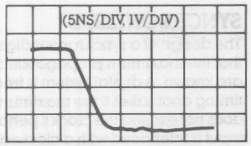
PROBLEM: In the SR type latch shown in figure 4 what happens if the set (S) and the reset (R) inputs are simultaneously raised from a LOW voltage level to a HIGH level?

9

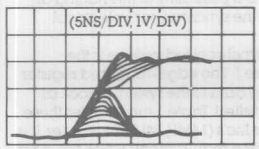
transition and will quickly oscillate to a final steady state of either HIGH or LOW (see figure 3a). To demonstrate this result the reader is encouraged to do this excercise either mentally or to actually build the circuit and view the output on the oscilloscope.



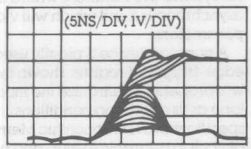
(a) CROSS-TIED NAND GATES



(b) REGISTER WITH NORMAL BEHAVIOR



(c) MONOLITHIC MEMORIES INC'S PAL16R4A



(d) MONOLITHIC MEMORIES INC'S 74LS374

Figure 3

Clock driven master-slave flip-flops contain the same type of cross tied RS latch within their internal circuitry. The NAND gate equivalent of the master-slave D type flip-flop is shown in figure 5. The gates circled in this figure can potentially behave similar to the above problem. If the clock and data are triggered within a specific window of one another the output may have an oscillatory behavior before settling down.

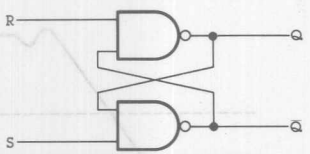
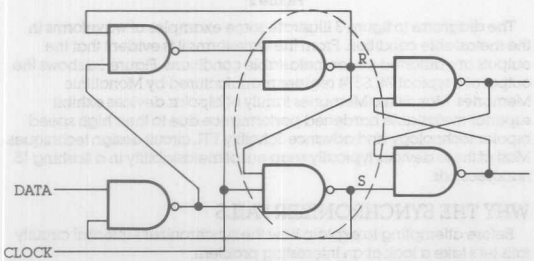


Figure 4



Cross tied RS latch structure is seen in the master-slave edge triggered flip-flop.

Figure 5

This section will show how to characterize the [illegible] triggered flip-flop with an asynchronous data interface. If the setup and hold times of the flip-flop are satisfied the output behaves properly (figure 6a). One of the four possible events below can take place if the flip-flop goes metastable:

1) The output starts to make a transition but snaps back to its original state (figure 6b).

2) The output makes a complete transition but the maximum propagation delay of the device is exceeded (figure 6c).

3) The output starts oscillating and retains its present state (figure 6d).

4) The output oscillates to a new state (figure 6e).

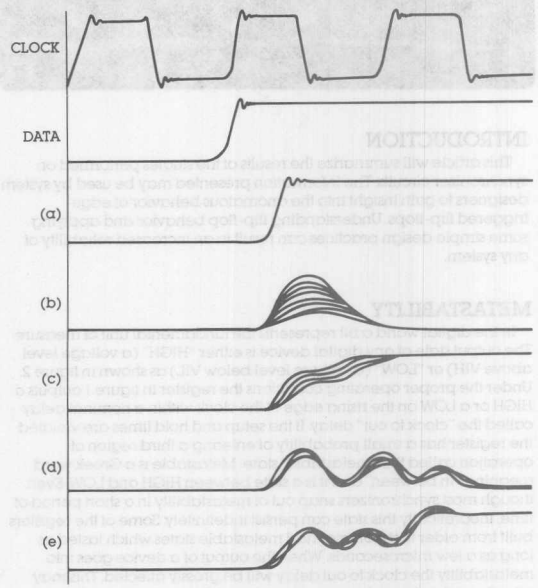


Figure 6

The circuit shown in figure 7 is used to obtain experimental results of a metastable device. The circuit can detect and count the number of events of metastability. The device under test (DUT) is forced into metastability by repeatedly sweeping the edges of the data past the rising edges of the clock. The modulation of the data is possible by using a comparator device (U1) along with an external sawtooth waveform. Thousands of transitions are created within the setup and hold time window of the DUT. Sweeping the data edges past the low to high clock transitions simulates an asynchonous input and increases the probability of getting a metastable failure on the output ($\overline{Q}$) of the DUT.

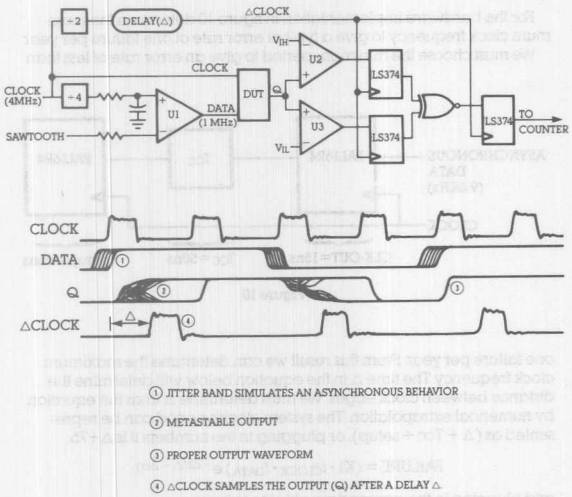


Figure 7

If the output of the device goes into metastability it will be detected by the comparator pair (U2) and (U3). The comparators will have complementary outputs if the output (Q) of DUT is anywhere between VIH and VIL. The outputs of the comparators are latched by a delayed version of the clock (ΔClock). The EXCLUSIVE-NOR gate followed by the register signal the event of metastability to an external counter.

The variable delay (Δ) between the two clocks will sample the output at various locations on the time axis. As this delay is varied the event of metastability is sampled and counted at these locations by our circuit. Therefore the output of our circuit measures the rate of metastability versus time delay. The real behavior of a metastable output can thus be effectively characterized with this scheme, that is, we can determine the length of time a metastable condition will persist and the density distribution of the metastable event.

Three 74374 devices and four PAL devices are used in this experiment. The plots of metastable failure versus time are shown in figures 8a,b. The next section will discuss in detail the characteristics of these plots.

EXPERIMENTAL RESULTS

Various graphs of metastability failure rate versus delay time are illustrated in figure 8. We can conclude from these graphs that the rate of metastability failure decreases as the sample clock (ΔCLOCK) moves farther and farther away from the DUT clock. The pictures shown in figure 9 have captured repeated events of metastability on the oscilloscope.

Let's take a closer look at one of the graphs to examine the behavior of the device. The PAL16R4A-4 device exhibits one count per second if the delay (Δ) is 60 nanoseconds. As the delay (Δ) is decreased, the rate increases exponentially until the delay equals 32 ns at which point the rate flattens out and remains fixed. The 32 ns forms the knee of our graph and will be referred to as Δo. The rate will remain constant if the delay (Δ) is decreased past the knee of our graph. Further reduction in the delay will place the sampling clock's rising edge prior to data transitions and thus the error rate vanishes to zero. The time at which the rate goes to zero is marked with an (X) on the graphs. By using this time (X), and another location on the graph such as the time where only one error per second occurs, we can associate an approximate range of metastability for different devices. This range of metastability is referred to as the "mean time to snap out of metastability". From the graph it is evident that the mean time to snap out of metastability for the PAL16R4A-4 logic circuit is the difference between 60 ns and 25 ns which is 35 ns.

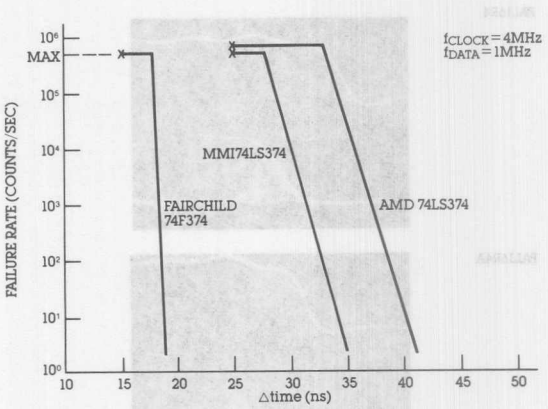


Figure 8a

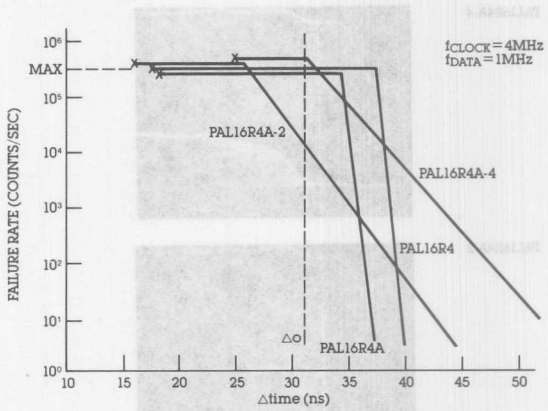


Figure 8b

All of the graphs illustrated can be quantified by an equation of the form:

$$\log \text{FAILURE} = \log \text{MAX} - b(\Delta - \Delta o)$$

Since a natural logarithm is a constant multiple of base 10 logarithm we can rewrite the above equation as:

$$a \cdot \ln \text{FAILURE} = a \cdot \ln \text{MAX} - b\,(\Delta - \Delta o)$$

In the above equation the MAX value is representative of the maximum metastability failure rate in our device. This MAX value is closely related to the frequency at which a metastable condition may occur in our device. The frequency at which metastability occurs is simply a constant multiple of the product of CLOCK and DATA frequency.

$$\text{MAX} = K1 \cdot f_{CLOCK} \cdot f_{DATA}$$

Substituting this in our original equation we get:

$$a \cdot \ln \text{FAILURE} = a \cdot \ln (K1 \cdot f_{CLOCK} \cdot f_{DATA}) - b(\Delta - \Delta o)$$

$$\ln \text{FAILURE} = \ln (K1 \cdot f_{CLOCK} \cdot f_{DATA}) - b/a(\Delta - \Delta o)$$

$$\text{FAILURE} = (K1 \cdot f_{CLOCK} \cdot f_{DATA})\, e^{-k2\,(\Delta - \Delta o)}$$

9

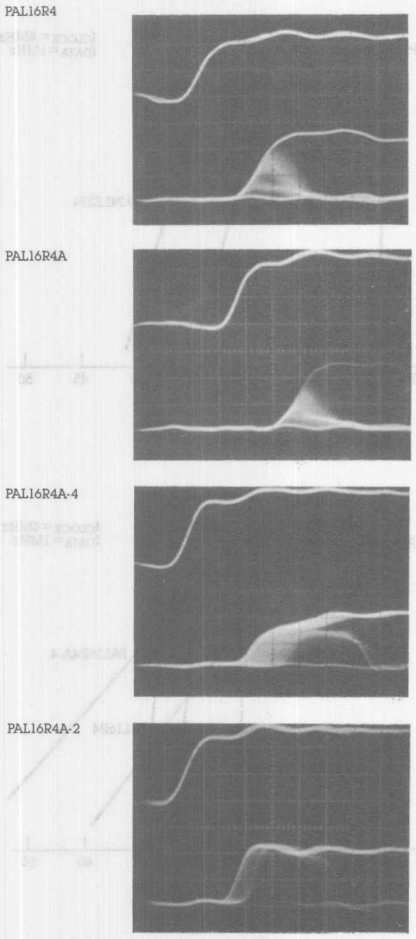


Figure 9 (2v/DIV, 5ns/DIV)

Table 2 gives the three important parameters which can be used by system designers to fully characterize the metastable behavior of the mentioned devices. These parameters can be obtained for different devices by duplicating this experiment. An example is given below to show how the information on table 2 may help the designer in the design of asynchronous systems.

MANUFACTURER	DEVICE	K_1 (Sec)	K_2 (ns^{-2})	$\triangle o$ (ns)
MMI	PAL16R4	1×10^{-7}	4.3	37
	PAL16R4A	1×10^{-7}	4.3	34.5
	PAL16R4A-2	1×10^{-7}	.64	25
	PAL16R4A-4	1×10^{-7}	.5	31
	74LS374	2×10^{-7}	1.8	27.5
AMD	74LS374	2×10^{-7}	2.0	34.5
FAIRCHILD	74F374	2×10^{-7}	11.5	17.5

Table 2

EXAMPLE

For the hardware implementation in figure 10 determine the maximum clock frequency to give a typical error rate of one failure per year. We must choose the minimum period to give an error rate of less than

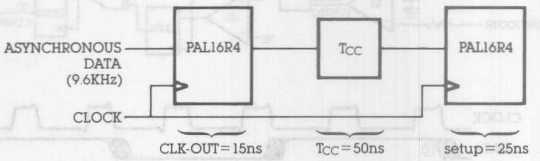


Figure 10

one failure per year. From this result we can determine the maximum clock frequency. The time Δ in the equation below will determine the distance between clock edges. We must determine Δ from the equation by numerical extrapolation. The system clock's period can be represented as (Δ + Tcc + setup), or plugging in the numbers it is Δ+75.

$$\text{FAILURE} = (K1 \cdot f_{CLOCK} \cdot f_{DATA})\, e^{-K2(\Delta - \Delta o)}$$

and plugging in the appropriate values we have:

$$3.2EE-8 = [(1EE-7)(1/(\Delta+75ns))(9600)]\, e^{-[(4.3)(\Delta-37)]}$$

Solving for Δ, we see that it is approximately 43 nanoseconds. The system period is thus seen to be the sum of 43ns and 75ns or 118ns. The maximum clock frequency is the inverse of the period or approximately 8 MHz.

CONCLUSION

Synchronization of two independent pulse trains is possible through the use of edge triggered registers. The electrical characteristics of the flip-flop are affected when the setup and hold times of the device are violated. This misbehavior is termed "metastability" and its probability of occurrence can be derived for a given system. The factors which affect this probability and the length of time which a metastable condition persists are influenced by the technology of the device as well as by the circuit design techniques.

An important fact which needs to be stressed is that even if a register's output goes metastable, the system may not necessarily fail if the register snaps out in time to satisfy the system's worst case timing requirement. The following design practices are suggested when using synchronizers:

Try to minimize the number of locations where asynchronous signals enter your system.

Clocking the asynchronous inputs through two pipelined registers can greatly reduce the error rate.

Use a single clock within your local system environment. For multiple system clocks, derive all the clock signals from a single source to assure synchronization between different devices within the system.

When analyzing the worst case timing of your system, add the time to snap out of metastability to any register in an asynchronous data path.

A single PAL® with registers can be your best choice for state machine analysis of asynchronous events. As the registers have virtually identical setup times, the simultaneous observation of a metastable event by different register states are likely to be the same. Contrasted to a distributed system of observing register states with different setup times, the PAL system of register states with identical setup times is a superior synchronizer.

Avoid edge sensitive devices on the output paths of the registers which have asynchronous inputs. The glitch created when the synchronizer goes metastable is enough to trigger the edge sensitive device. The use of level sensitive devices is generally a better design practice.

PAL devices can be effective synchronizers where various registering schemes are easily implemented.

Fast 64x64 Multiplication Using 16x16 Flow-Through Multiplier and Wallace Trees*

Marvin Fox, Chuck Hastings and Suneel Rajpal

The Monolithic Memories SN54/74S556 is a high-speed fully-parallel 16x16 multiplier and it provides the entire 32-bit product on a flowthrough basis from a single part. It is available in an 84-pin Leadless Chip Carrier (LCC) and 88-pin, pin-grid array packages. 8x8 40-pin array-multipliers such as the SN54/74S557/8 have been available for several years, however there is a large parts count for implementing longer wordlengths.

This paper describes the design philosophy and internal architecture of the 'S556 and appiications for larger wordlength multiplications such as 32, 48, and 64 bits using these multipliers and high-speed PROMs and ALUs also available from Monolithic Memories.

The system advantages for using the 'S556 over the MPY-16H-class multipliers is also discussed; the main advantages being the availability of the entire product each cycle and the space savings on the board.

9

* This paper is a slightly updated version of the paper by the same name which appeared in the Northcon/83 Professional Program Session Record, Session 24 reprint, paper 24/2, 10-12 May 1983. A modified version subsequently also appeared in the Mini Micro West/83 Professional Program Session Record, Session 14, paper 14/2, 8-11 November 1983.

2175 Mission College Boulevard, Santa Clara, CA 95054 Tel: (408) 970-9700 TWX: 910-338-2374
TWX: 910-338-2376

Monolithic Memories

Summary

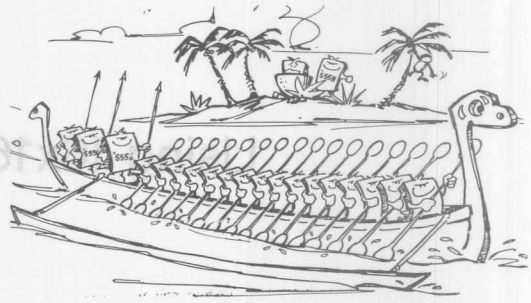
Multiplication is one basic digital-computer operation which can readily be speeded up by employing massive parallelism. "Cray multiplication" techniques, first used in large special-purpose computers a quarter of a century ago, are now commonplace in high-performance systems.

Essentially, in Cray Multiplication a full adder is placed in every position which would be occupied by a partial-product bit in a pencil-and-paper binary multiplication example (r1, r2). This technique may be applied within an LSI integrated circuit, in a system, or in both at once; it may or may not be modified by using "Booth-multiplication" approaches (r3, r4, r5).

8x8 40-pin Cray-multiplier integrated circuits have been available for several years, with a useful "flow-through" architecture. However the parts count for implementing full-blown Cray multiplication with practical scientific-computation word-lengths has been quite large. There have, of course, been several 16x16 Cray-multiplier 64-pin integrated circuits available; however, these have been unable, because of pin limitations, to furnish an entire 32-bit product in parallel. As a result, long-word-length multiplication cannot be performed economically on a flowthrough basis using these parts; some sort of clocking and multiplexing scheme is necessary to use them whenever the wordlength exceeds 16 bits, or else they must be duplicated outright.

Now there is a 16x16 Cray-multiplier part, the Monolithic Memories SN54/74S556, which provides the entire 32-bit product on a flow-through basis from a single part. The 'S556 has been designed to use the new 84-pin leadless-chip-carrier (LCC) and 88-pin pin-grid array packages, rather than compromising the architecture of the part because of the pin limitations (64 at most) of dual-in-line (DIP) packages.

This paper describes the design philosophy and internal architecture of the 'S556. It also shows how long-word-length multipliers may be built up from arrays of individual Cray-multiplier integrated circuits and programmable read-only memories (PROMs); the latter are used as "Wallace-tree" adders. Part-count and performance comparisons are made, for the representative word length of 64 bits, between implementations based on 64-pin 16x16 devices and implementations using 'S556s, in two different architectures; one which aims at lower cost and is a compromise between Cray multiplication and traditional shift-and-add multiplication.



"... MULTIPLICATION ... CAN READILY BE SPEEDED UP BY EMPLOYING MASSIVE PARALLELISM ..."

'S556 Architecture

The 'S556, shown in Figure 1, is a 16x16 Cray multiplier designed with an ultra-high-speed array of 256 adders, internally organized to the shift-and-add technique for multiplication (r1, r2). In place of the usual ripple-carry adders used in multiplier designs to sum up the final product bits, the 'S556 uses a carry-lookahead adder.

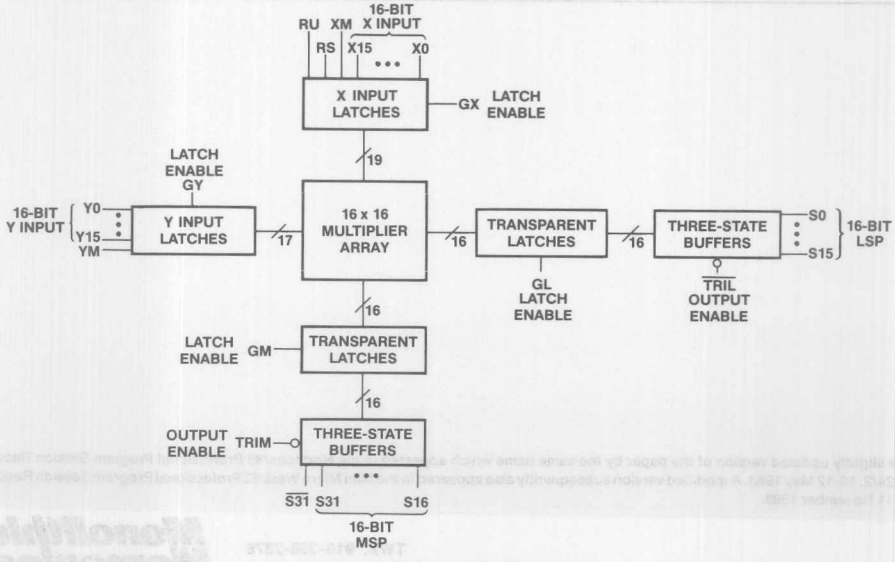


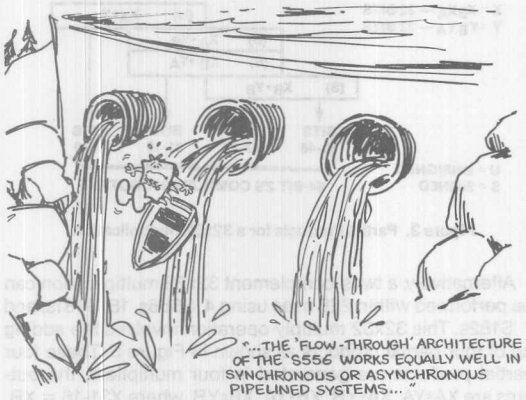
Figure 1. The 'S556 Architecture

The "flow-through" architecture of the 'S556 works equally well in synchronous or asynchronous pipelined systems. Latches are available to hold the input operands and the resulting double-length product, to increase the throughput rate in pipelined systems. If the designer does not wish to use these latches, they may be disabled, and the 'S556 then operates as a pure memoryless arithmetic network.

The 'S556 accepts operands in either unsigned or signed twos-complement form. When used in pipelined architectures, the 'S556 is capable of supplying 32-bit products at a 12.5 MHz repetitive throughput rate. The 'S556 has three-state outputs, controlled by the TRIL and TRIM control inputs.

Rounding-control input pins are provided on the 'S556 for rounding either unsigned or signed operands. Rounding is allowed in either of two binary positions, to support either "fractional-arithmetic" or "integer-arithmetic" positioning of a single-length rounded result.

The more traditional shift-and-add technique was chosen for the internal design of the 'S556 adder network because of the compactness, simplicity, and lower power requirement of this implementation. The Booth-algorithm approach, which groups the multiplier bits to effectively reduce the number of rows in the array, was considered (r3, r4, r5). However this approach also has penalties, in that it increases the width of each row from 16 to 18 bits, and the width of the final adder from 18 to 24 bits. Intrinsically, both the shift-and-add technique and the Booth-algorithm technique require 31 logic delays in the multiplier array using a ripple-carry final adder. At this point, the use of a carry-lookahead adder structure results in major speed improvements.

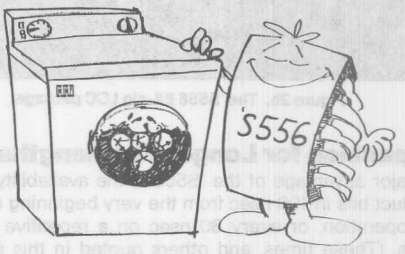


Here again there are tradeoffs. In MSI bipolar circuits, carry-lookahead parts are reasonable to construct with scanning widths of up to 4 bits, with a carry-out available (r5). Beyond that, the circuit gets bulky and power-hungry. Parallel "banking" of 4-bit-adder groups may be used to extend this limit, but here again 4 to 5 banks is as far as this approach can be reasonably pushed. With parallel banking the 24-bit adder required by the Booth-algorithm technique can be implemented using 6 banks of 4-bit adders; this exceeds the limit of 4 to 5 banks. This shows that the Booth-algorithm implementation requires fewer horizontal rows of adders, which translates to shorter propagation delays as compared to shift-and-add technique; however the final adder in the Booth-algorithm implementation is slower than the final adder in the shift-and-add technique implementation.

The 'S556 internal design uses an Emitter-Coupled-Logic (ECL) circuit implementation, based on Monolithic Memories' new washed-emitter process. ECL was chosen here over TTL and Emitter-Follower Logic, both of which have been used in previous Monolithic Memories Cray-multiplier designs (r3, r4). Here, ECL also turns out to have the most compact circuit-layout form, requiring 82 square mils of chip surface area per full adder. Emitter Function Logic (EFL) was chosen for one portion of the design, the carry-lookahead tree, because it interfaces easily with single-ended ECL outputs. All latches are implemented in ECL, to interface easily with the TTL/ECL buffers at the inputs and the ECL/TTL buffers at the outputs. The input latches introduce one ECL delay, but there is zero additional delay at the outputs as the output latches are incorporated right into the ECL/TTL translators.

The 'S556 is a universal multiplier aimed at a flow-through-type-processor architecture. Latches are used since registers cannot implement a flow-through architecture directly.

To be sure, the currently-available 16x16 multipliers from TRW and AMD, which use 64-pin dual-in-line packages do have a feed-through capability on the output registers. This capability allows latch-like transparency on the output registers, but nowhere else, since the parts are pin-limited and input and output data must in some cases share the same pins. Such an implementation consumes considerably more chip area and power than a purely latch design.



"...THE 'S556 INTERNAL DESIGN USES MONOLITHIC MEMORIES' NEW WASHED-EMITTER PROCESS..."

Many users who wish to use registers to achieve pipelined operation can find ways to do so using the 'S556s' internal latches. Usually pipelining can be achieved by choosing the proper phasing and pulse width of the latch gate-control signals without resorting to using external registers. Of course, external registers may be used when absolutely necessary.

The 'S556 will be supplied in an 84-pin Leadless Chip Carrier (LCC), and also in an 88-pin pin-grid-array package, with an integral heat sink. Both Commercial and Military grade parts will be available. The pinout is shown in Figure 2a. A photograph of the 84-pin LCC package is shown in Figure 2b.

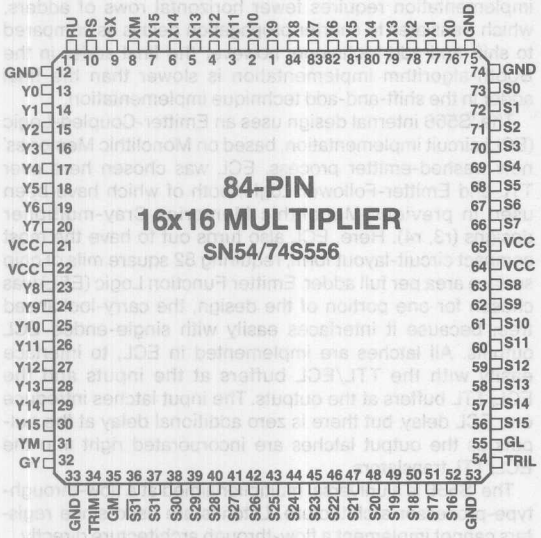


Figure 2a. The 'S556 Pinout Diagram

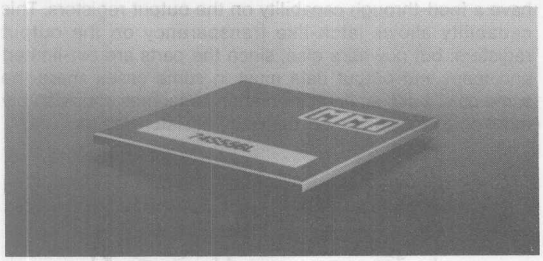


Figure 2b. The 'S556 84-pin LCC package

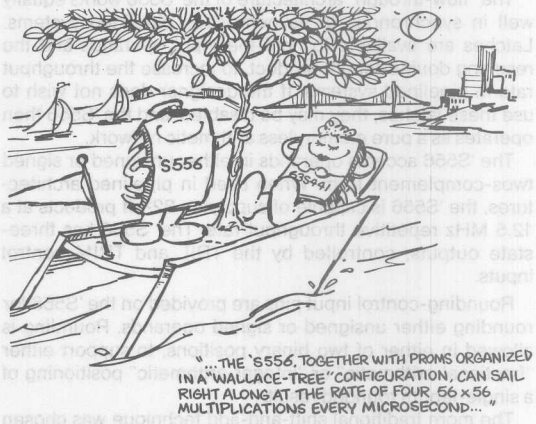
Expansion for Longer Wordlengths

A major advantage of the 'S556 is the availability of all 32 product bits in 100 nsec from the very beginning of a multiply operation, or every 80 nsec on a repetitive pipelined basis. (These times, and others quoted in this paper, are worst-case rather than typical.) Thus, the 'S556 is especially suited for longer-wordlength arithmetic units.

Other commercially-available multipliers, of the TRW MPY-16H class, are packaged in 64-pin 900-mil DIPs, which require a circuit board area of approximately 1" x 3.25". Moreover, these parts operate more slowly in expanded configurations, as the most-significant half and the least-significant half of the 32-bit double-length product must be obtained on two successive clock cycles.

Totally-Parallel 32-bit Multiplier

The 'S556, together with PROMs organized in a "Wallace-Tree" configuration, can sail along at the rate of four 56x56 multiplications every microsecond. An unsigned 32-bit multiplication can be performed using 4 'S556 multipliers, 11 63S481A PROMs used as "Wallace-Tree adders" (r1), and 16 'S381 and 5 'S182 used to form a 64-bit adder. The multipliers supply the partial products which are positioned as shown



in Figure 3. The difference is that only unsigned operands are used, and only positive partial products are added. The three rows of partial products which overlap are added by using PROMs which "compress" these three rows to 2 rows, which are then added in the 64-bit adder. The compression technique is discussed in greater detail in the description, later on, of the 64-bit multiply operation. Using the above configuration, an unsigned 32x32 multiply operation can be performed in less than 175 nsec worst-case allowing for a 75-nsec 'S556 multiplier delay, a 30-nsec 63S481A PROM delay and a 64-nsec 64-bit adder delay.

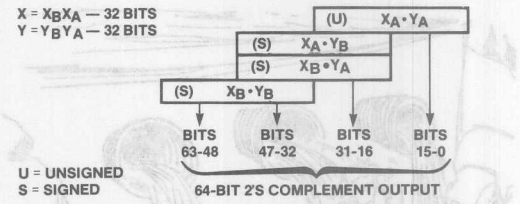
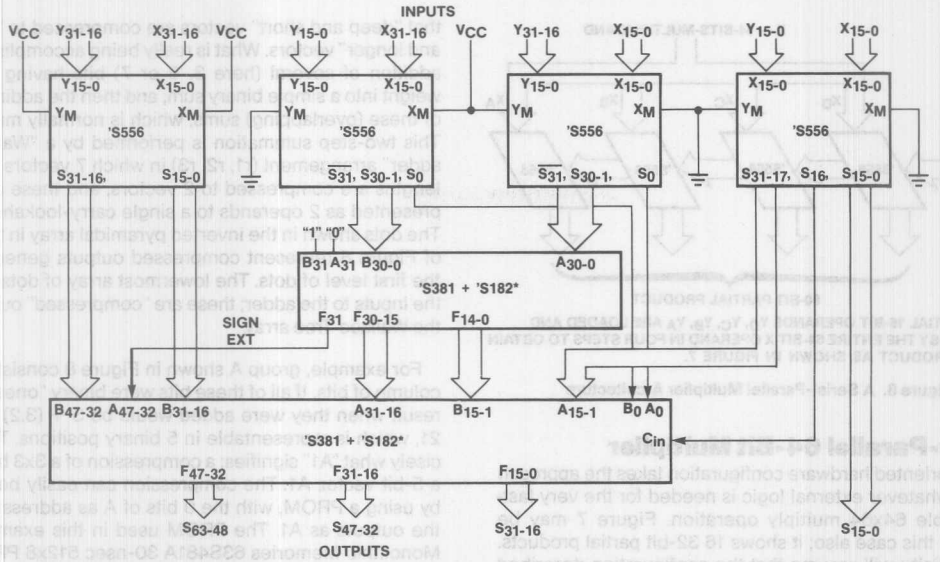


Figure 3. Partial Products for a 32x32 Multiplication

Alternatively, a twos-complement 32x32 multiplication can be performed within 228 nsec using 4 'S556s, 18 'S381s, and 7 'S182s. This 32x32 multiply operation involves the adding up of four partial products as shown in Figure 3. These four partial products are generated in four multipliers; the outputs are XA*YA, XA*YB, XB*YA, XB*YB, where X31-16 = XB, X15-0 = XA, Y31-16 = XB, Y15-0 = XA.

The implementation of this twos-complement 32x32 multiplier is shown in Figure 4. The outputs of the 16x16 multipliers are connected to two levels of adders to give a 64-bit product. The first level of adders is needed to add the two central partial products of Figure 2, XA*YB and XB*YA. Notice the technique which is used to generate the "sign extension" or the most-significant sum bit of the first level of adders. The 'S556 provides as a direct output the complement of the most-significant product bit; having this signal immediately speeds up the sign-extension computation, and reduces the external parts count.



* THESE ARE ADDER BLOCKS USING THE 'S381, A 4-BIT ALU FUNCTION GENERATOR, TO PERFORM A HIGH SPEED ADD OPERATION. THE 'S182 IA A LOOK-AHEAD CARRY GENERATOR AND IT REDUCES THE PROPAGATION DELAY. ALL THE ABOVE PARTS ARE AVAILABLE FROM MONOLITHIC MEMORIES INCORPORATED.

TOTAL MULTIPLY TIME = MULTIPLIER DELAY + ADDER LEVEL 1 DELAY + ADDER LEVEL 2 DELAY = 75 + 64 + 64 = 203 nsec

Figure 4. Implementation of the 32x32 Multiplier

For example, the inputs to the adder in the most significant position are the $\overline{S31}$ outputs from the two central multipliers. The sign extension of the addition of XA*YB and XB*YA is defined as

SIGN EXT = $\overline{A}.\overline{B}. + \overline{A}.C + \overline{B}.C$, where

A is the most-significant bit of the term XA*YB;

B is the most-significant bit of the term XB*YA; and

C is the carry-in to the most-significant bits of XA*YB and XB*YA, in the adder.

The sign extension can be computed as the negation of the carry-out term of three terms, A, B, and C. This term corresponds to the negative of the carry-out of the bit position just one place to the right of the most-significant bit position of the first level of adders. The negative of the carry-out can be generated by presenting a carry-out and a binary "one" to the most significant bit of the adder. The generated sum bit then corresponds to the negation of the carry-out of the previous stage, which is the sign extension required to be added to the 16 most-significant bits of the XB*YB partial product term.

The second level of adders, which performs a 40-bit add function, is fairly straightforward. These adders can be implemented using 'S381 four-bit ALUs and 'S182 carry-bypasses ("carry-lookahead generators") which are available from Monolithic Memories, Inc. and from other vendors.

Other configurations such as 48x48 multipliers can be designed using the same methodology. Figure 5 shows the alignment of the partial products from 9 'S556s for the 48x48 case.

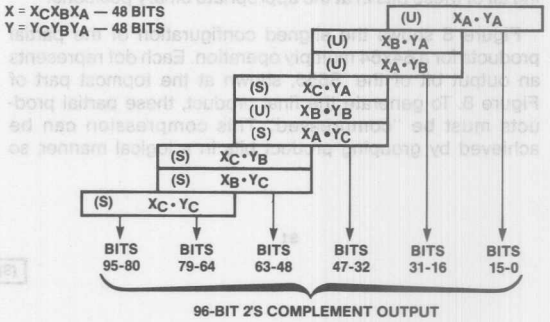
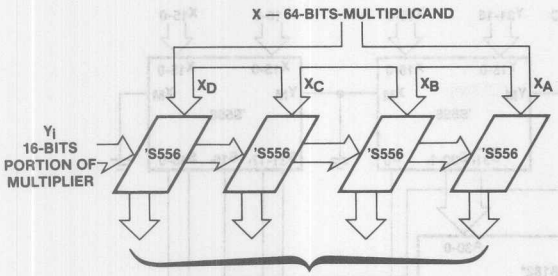


Figure 5. Partial Products for a 48x48 Multiplication

Serial-Parallel Multiplier

In applications where speed can be sacrificed, it is possible to implement an alternative solution using fewer multipliers, at some penalty in speed, but still with a very significant speed gain over other methods of multiplication. Figure 6 shows a plausible method of performing a 64x64 multiply operation, in four cycles. Each cycle generates four partial products, each of which is 32 bits wide; these must be added in at the appropriately-aligned bit positions to generate an 80-bit partial product, in logic external to the multipliers. On the next cycle another 80-bit partial product is generated, and is added to the previous 80-bit partial product at the appropriate alignment offset. Figure 7 shows the 16 32-bit partial products aligned appropriately to their binary weighting, for the entire time-sequenced multiply process. The final 128-bit product can be obtained from the addition of the four 80-bit partial products on successive clock cycles.



THE Y_I - PARTIAL 16-BIT OPERANDS Y_D, Y_C, Y_B, Y_A ARE LOADED AND MULTIPLIED BY THE ENTIRE 64-BIT X OPERAND IN FOUR STEPS TO OBTAIN A 128-BIT PRODUCT AS SHOWN IN FIGURE 7.

Figure 6. A Serial-Parallel Multiplier Architecture

Totally-Parallel 64-Bit Multiplier

A speed-oriented hardware configuration takes the approach of using whatever external logic is needed for the very fastest possible 64x64 multiply operation. Figure 7 may be applied in this case also; it shows 16 32-bit partial products. (For simplicity, will assume that the configuration described here deals strictly with unsigned integers, so that the 16 partial products are unsigned.) Since 16 'S556s are being used, then the 32-bit partial products corresponding to *all* of the combinations of the partitioned multiplier and the partitioned multiplicand are all available at the same time. Now comes the crucial aspect of the design, which involves adding all of these bits in at the appropriate binary positions!

Figure 8 shows the aligned configuration of the partial products for a 64x64 multiply operation. Each dot represents an output bit of the 'S556, shown at the topmost part of Figure 8. To generate the final product, these partial products must be "compressed." This compression can be achieved by grouping product bits in a logical manner, so that "deep and short" vectors are compressed to "shallower and longer" vectors. What is really being accomplished is the addition of several (here 3, 5 or 7) bits having the same weight into a simple binary sum; and then the adding up of all of these (overlapping) sums, which is normally much easier. This two-step summation is performed by a "Wallace-tree-adder" arrangement (r1, r2, r3) in which 7 vectors of varying lengths are compressed to 2 vectors, and these are in turn presented as 2 operands to a single carry-lookahead adder. The dots shown in the inverted pyramidal array in the middle of Figure 8 represent compressed outputs generated from the first level of dots. The lowermost array of dots represent the inputs to the adder; these are "compressed" outputs from the Wallace-Tree array.

For example, group A shown in Figure 8 consists of a 3x3 column of bits. If all of these bits were binary "ones" then the result when they were added would be 3 + (3.2) + (3.4) = 21, which is representable in 5 binary positions. This is precisely what "A1" signifies; a compression of a 3x3 block, A, to a 5-bit vector, A1. The compression can easily be achieved by using a PROM, with the 9 bits of A as address lines, and the outputs as A1. The PROM used in this example is the Monolithic Memories 63S481A 30-nsec 512x8 PROM; only five of the eight output bits are used. Designers may prefer to group the bits in a different configuration from the one suggested in Figure 8; many other arrangements are possible. For example, one may group another column of three bits and thereby reduce a 4x3 block to a 6-bit vector, using 63S3281s, which are (40-nsec 4Kx8 PROMs); this approach would give a different pattern than the one in the middle of Figure 8.

Similar compressions for Group B to B1 can be performed using 63S441/1A 1Kx4 PROMs. This configuration compresses five 2-bit vectors to a 4-bit vector, which fits the 10-bit input address and 4-bit output word of the 1Kx4 PROM.

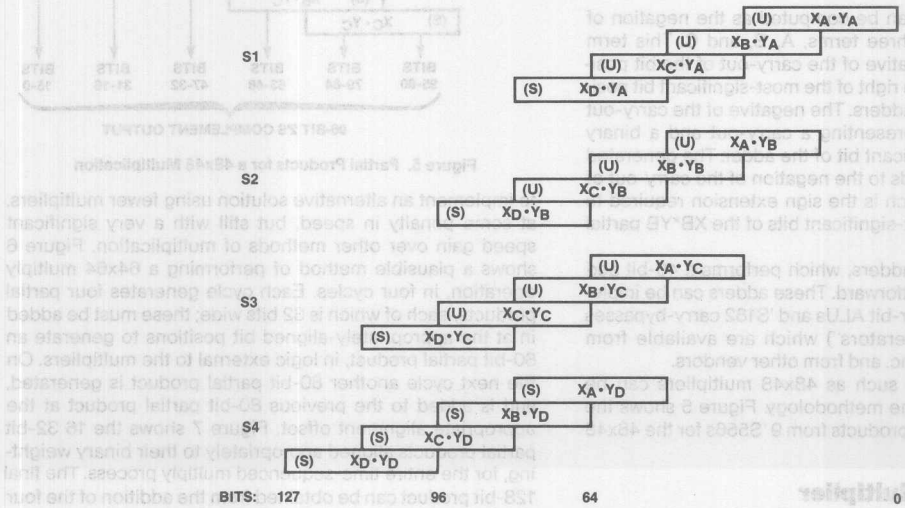


Figure 7. Partial Product Alignment for a Serial-Parallel Multiplier

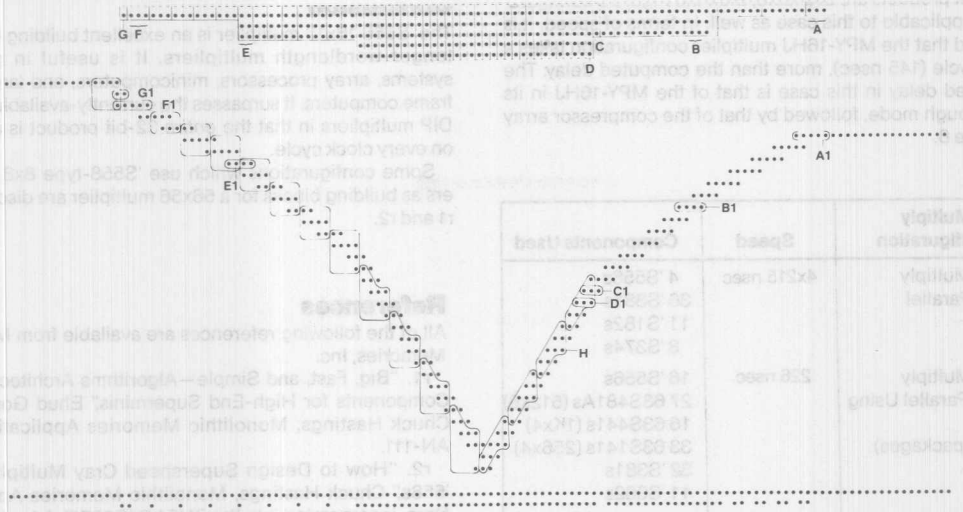


Figure 8. Partial Product/Bit Grouping for a Totally Parallel 64x64 Multiplier

Other compressions shown are C and D groups to C1 and D1 respectively. The C group is handled by compressing five 1-bit vectors to a 3-bit vector. The D group is handled by compressing seven 1-bit vectors to a 3-bit vector. The C and D groups can be compressed using 'S141/1A 256x4 PROMs. Similarly, groups E, F, G, and H are compressed to E1, F1, G1 and H1 respectively. All the above mentioned PROMs are available from Monolithic Memories.

The second level of dots has some groups of four columns. These four-column groups contain 3 bits in the least-significant bit position, and 2 bits in the remaining columns. These 9 inputs can be compressed using a 63S481A 30-nsec 512x8 PROM, to a vector 5 bits wide. For parts-counting purposes, the same 63S481A PROM type is used for all the compressions in the middle of Figure 8.

To aid users in the programming of PROMs for these and other Wallace-tree applications, or in fact any other applications exploiting PROMs as logic elements, Monolithic Memories provides *P*rogrammable *L*ogic *E*lement *AS*se*M*bler (PLEASM), a portable computer program written in FORTRAN. PLEASM provides a simple method for generating a PROM truth table. The user has only to supply equations which define the arithmetic/Boolean function needed within the PROM; PLEASM does all the drudgery of figuring out the code values which are needed in each PROM location.

Sample PLEASM source codes are shown at the end of this paper. For example, the entire 1Kx4 PROM which reduces the five 2-bit vectors to a 4-bit vector can be specified, using PLEASM, in 15 or fewer lines of code. Without PLEASM or its equivalent, the user would have had to specify the contents of 1024 PROM locations, after computing the corresponding code values for those locations.

Performance Comparisons

The bottom line for any hardware-architecture analysis is how fast the system runs, and what it costs in circuit-board real estate and dollars. With this understanding, a performance table is derived, based on three configurations.

The first is the configuration of Figure 7, using 4 'S556 multipliers; the entire multiplication takes four clock cycles. In addition to the multiplier ICs, a 64-bit adder is needed for the four partial products, which effectively furnishes a 80-bit partial product on every cycle. A 64-bit adder can be used to do the addition, since the least-significant partial-product bits are available directly. The 80-bit partial product has to be shifted 16 bits and then added to the second 80-bit partial product, which implies a need for a 64-bit register and an 80-bit ALU, which together serve as an accumulator.

The second configuration is the totally-parallel design using 16 'S556 multipliers plus PROMs and ALUs, shown in Figure 8.

The third configuration uses TRW-MPY16H-class 64-pin 16x16 multipliers. The entire 32-bit product of an MPY-16H is available on two successive clock cycles, as the product lines are shared with the incoming data. An additional 145 nsec is added to the MPY-16H time to allow for the necessary clocking and multiplexing steps to occur: effectively, the operands cannot be pipelined at one clock cycle as may be done in the 'S556 architecture. Even if the pin-compatible Am29516 multiplier is used, a cycle is still wasted, as two cycles are needed to clock the entire multiplier.

There is one way around this problem, when using the Am29516 multiplier; twice as many multipliers are used, and a pair of adjacent multipliers receive the same input operands. One multiplier of the pair then outputs the least-significant half of the product, and the other multiplier of the pair outputs the most-significant half of the product; thus, the two paired Am29516 64-pin DIPs are functionally a quasi-equivalent of the 84-pin 'S556, albeit they require many times the circuit board area.

The analysis in the Table 1 assumes the use of 16 MPY-16HJ multipliers. 16 16-bit registers are needed to hold the 16 halves of the various different partial products. After the

16 32-bit products are available, then the Figure 8 configuration is applicable to this case as well. In terms of speed, it is assumed that the MPY-16HJ multiplier configuration takes a clock cycle (145 nsec), more than the computed delay. The computed delay in this case is that of the MPY-16HJ in its feedthrough mode, followed by that of the compressor array of Figure 8.

Multiply Configuration	Speed	Components Used
64x64 Multiply Serial-Parallel	4x215 nsec	4 'S556s 36 'S381s 11 'S182s 8 'S374s
64x64 Multiply Totally-Parallel Using 'S556s (84-pin packages)	226 nsec	16 'S556s 27 63S481As (512x8) 16 63S441s (1Kx4) 33 63S141s (256x4) 32 'S381s 11 'S382s
64x64 Multiply Clocked Parallel Using MPY-16HJ (64-pin packages)	481 nsec	16 MPY-16HJ 32 'S374s 27 63S481s (512x8) 16 63S441s (1Kx4) 33 63S141s (256x4) 32 'S381s 11 'S382s

Table 1. Performance Comparisons

Conclusion

The 'S556 16x16 multiplier is an excellent building block for longer-wordlength multipliers. It is useful in graphics systems, array processors, minicomputers, and large mainframe computers. It surpasses the currently-available 64-pin-DIP multipliers in that the *entire* 32-bit product is available on every clock cycle.

Some configurations which use 'S558-type 8x8 multipliers as building blocks for a 56x56 multiplier are discussed in r1 and r2.

References

All of the following references are available from Monolithic Memories, Inc.

r1. "Big, Fast, and Simple—Algorithms Architecture, and Components for High-End Superminis," Ehud Gordon and Chuck Hastings, Monolithic Memories Application Note AN-111.

r2. "How to Design Superspeed Cray Multipliers with '558s," Chuck Hastings, Monolithic Memories Application Note, incorporated into the SN54/74S557/8 data sheet.

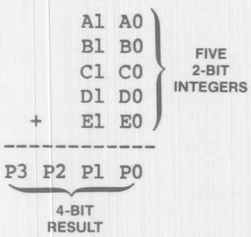
r3. "Real-Time Processing Gains Ground with Fast Digital Multiplier," Shlomo Waser, *Electronics,* 9/29/77.

r4. "State-of-the-Art in High Speed Arithmetic Integrated Circuits," Shlomo Waser, *Computer Design,* 6/1978.

r5. "Doing Your Own Thing in High-Speed Arithmetic," Chuck Hastings, *Conference Proceedings of the 6th West Coast Computer Faire,* pages 492-510, 4/5/81. Also Monolithic Memories Conference Proceedings reprint CP-102.

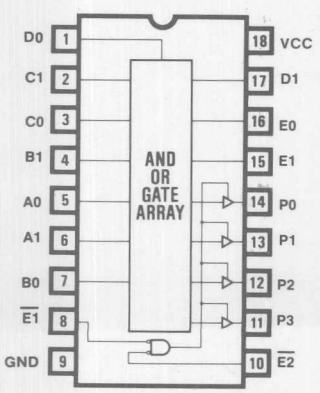
Appendix – Sample PLEASM Source Listing – To Reduce Group B In Figure 8

```
PLE10P4                                                PLE DESIGN SPECIFICATION
P5020                                                    VINCENT COLI 08/22/83
FIVE 2-BIT INTEGER ROW PARTIAL PRODUCTS ADDER
MMI SANTA CLARA, CALIFORNIA
.ADD A0 A1 B0 B1 C0 C1 D0 D1 E0 E1
.DAT P0 P1 P2 P3


P3,P2,P1,P0 = A1,A0 .+. B1,B0 .+. C1,C0 .+. D1,D0 .+. E1,E0   ; P = A+B+C+D+E


FUNCTION TABLE

A1 A0 B1 B0 C1 C0 D1 D0 E1 E0 P3 P2 P1 P0

;AA   BB   CC   DD   EE    PPPP     COMMENTS
;10   10   10   10   10    3210     A + B + C + D + E = P
-------------------------------------------------------------
 LL   LL   LL   LL   LL    LLLL     0 + 0 + 0 + 0 + 0 =  0
 LH   LH   LH   LH   LH    LHLH     1 + 1 + 1 + 1 + 1 =  5
 HL   HL   HL   HL   HL    HLHL     2 + 2 + 2 + 2 + 2 = 10
 HH   HH   HH   HH   HH    HHHH     3 + 3 + 3 + 3 + 3 = 15
-------------------------------------------------------------


DESCRIPTION

THIS PLE10P4 PERFORMS PARTIAL PRODUCTS REDUCTION FOR WALLACE TREE
COMPRESSION.  FIVE ROWS OF 2-BIT NUMBERS (A1-A0, B1-B0, C1-C0,
D1-D0, AND E1-E0) ARE NUMERICALLY SUMMED TO PRODUCE A 4-BIT RESULT
(P3-P0).
```



FIVE 2-BIT INTEGER ROW PARTIAL PRODUCTS ADDER

PLE10P4



Northcon/83
Electronics Show & Convention
May 10-12, 1983
Portland, Oregon

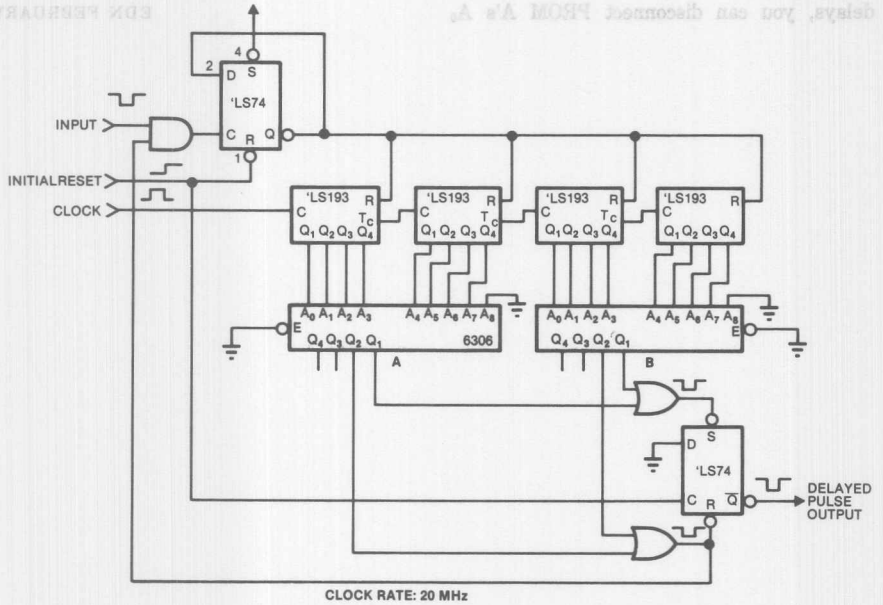
Cascade Chapter, ERA
Portland and Seattle Sections IEEE
Portland and Seattle Chapters, NWPCA

PROMs yield delayed pulses

Rick Wegner
Storage Tech Corp, Louisville, CO

If you need a highly accurate, delayed pulse with adjustable width, the circuit shown in the **figure** will do the job. The circuit can operate at repetition rates as high as 20 MHz, a limitation set by the bipolar PROMs' access time (in this case, 50 to 60 nsec) and the counters' maximum clock rate (32 MHz for the example's 74LS193). To generate a delayed pulse from the original input pulse, program the PROMs to yield a logic Low upon reaching the desired delay count and a logic High at the end of the delayed pulse's period.

The delay count equals the desired delay divided by the clock period. In the schematic shown, you need an OR gate because PROM A goes through several intermediate counts before both counters attain the final delay count. To set up your desired pulse width,



(a)

CLOCK RATE: 20 MHz
DESIRED DELAY: 1.599650 mSEC
DESIRED PULSE WIDTH: 400 nSEC

PROM A

ADDRESS		OUTPUT				
HEX	BINARY	Q_4	Q_3	Q_2	Q_1	HEX
01	1	1	1	0	1	B
F9	249	1	1	1	0	E

PROM B

ADDRESS		OUTPUT				
HEX	BINARY	Q_4	Q_3	Q_2	Q_1	HEX
7C	124	1	1	1	0	E
7D	125	1	1	0	1	B

ALL OTHER ADDRESSES ARE PROGRAMMED WITH OUTPUTS HIGH (F).

(b)

Obtain accurate pulse delays and widths *with this PROM-based circuit. The circuit operates with clock rates as high as 20 MHz and allows you to program very precise pulse parameters. You can extend the scheme's capabilities by adding more counters and additional PROMs.*

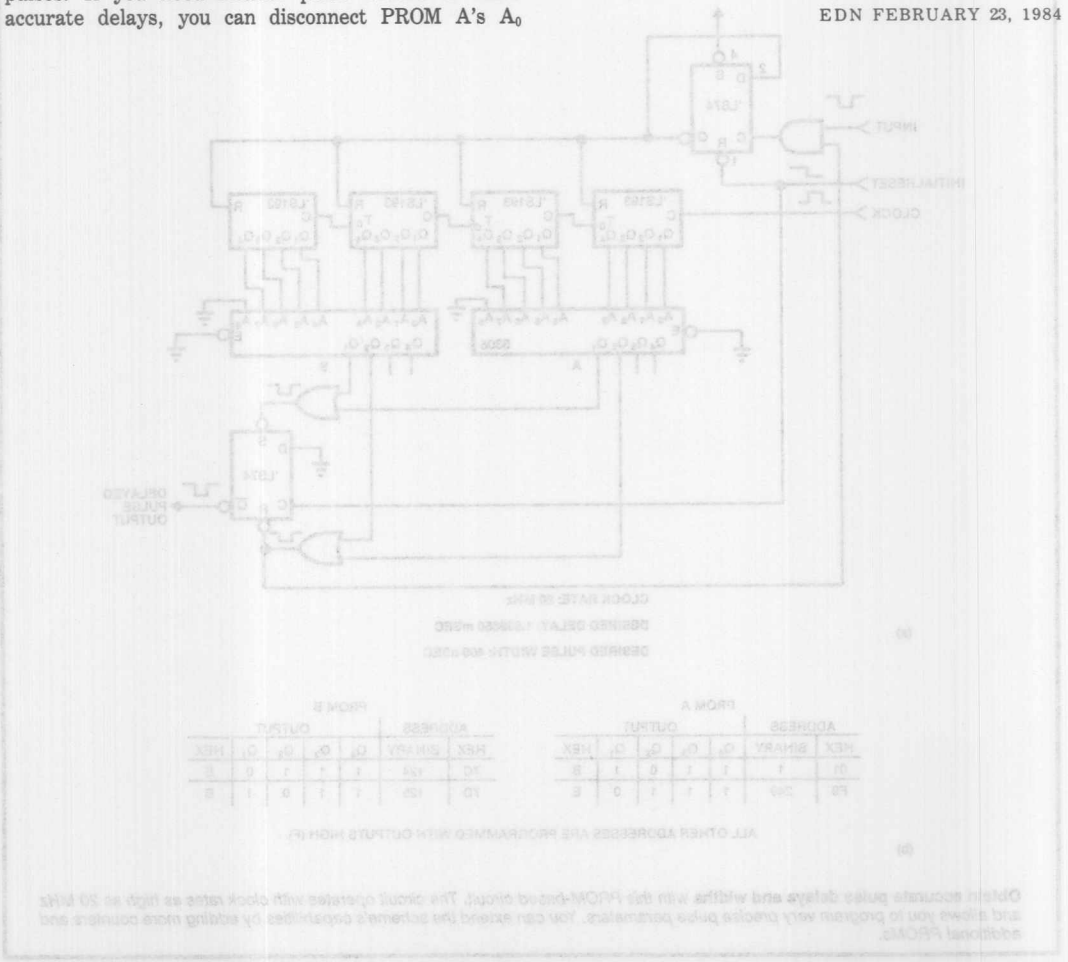
generate another pulse that represents the added delay (equal to the desired delay plus the pulse width).

These two pulses then serve to set and reset a flip flop that generates a delayed pulse with the programmed width. You can obtain additional pulses by using the PROMs' other two outputs. The feedback resets the circuit so that the next input pulse can start the delay counting again. A specific program example is shown in (**b**).

The design has several modification possibilities. If you need a longer delay (without sacrificing accuracy), you can add more counters and PROMs. Moreover, an 8-output PROM allows the generation of more delayed pulses. If you need smaller pulse widths or more accurate delays, you can disconnect PROM A's A_0 left in the schematic. This action allows the determination of a new set of delay counts, effectively doubling the input clock-rate capability.

What are the limitations of this circuit? First, because all address inputs must change simultaneously, the circuit demands synchronous counters. Second, you shouldn't use large-capacity EPROMs, because their increased access time reduces the maximum clock rate, thus reducing the accuracy of both the delay and the pulse width. **EDN**

EDN FEBRUARY 23, 1984



High-Speed PROMs with On-Chip Registers and Diagnostics

Vincent J. Coli, Stephen M. Donovan and Frank Lee

A family of High-Speed Registered and Diagnostic PROMs offers new savings for system designers. The Registered PROM family features on-chip D-type output registers which are useful in pipelined systems and state machines. In addition to output registers, the Diagnostic PROMs feature a Shadow Register which makes it easier for system designers to include diagnostics in microprogrammed systems. Architectures and applications for these devices are discussed in this paper.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2376 TWX: 910-338-2374

High-Speed PROMs with On-Chip Registers and Diagnostics

Vincent J. Coli, Stephen M. Donovan and Frank Lee

A family of High-Speed Registered and Diagnostic PROMs offers new savings for system designers. The Registered PROM family features on-chip D-type ouput registers which are useful in pipelined systems and state machines. In addition to output registers, the Diagnostic PROMs feature a Shadow Register which makes it easier for system designers to include diagnostics in microprogrammed systems. Architectures and applications for these devices are discussed in this paper.

Architectures

In digital systems, it is natural to have a PROM followed by a register. This structure is particularly useful in microprogramming and state machine design. The Registered PROM family includes an on-chip Output Register as illustrated in Figure 1. By integrating these two building blocks into one chip, the following benefits are realized:

1. 2-to-1 chip count reduction
2. PC-Board space saving
3. Reduced power consumption
4. Eliminate the PROM output buffer and register input buffer and their associated delays.

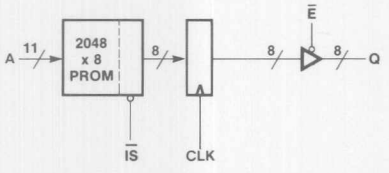


Figure 1. Registered PROM Block Diagram with Synchronous Initialization

In addition to the on-chip Output register, the Diagnostic PROMs include extra circuitry to perform system level diagnostics, DOC (On-Chip). Specifically, a buried Shadow Register with shifting capability and a 2:1 multiplexer are provided. A block diagram illustrating the Diagnostic PROM architecture is given in Figure 2.

Shadow register diagnostics allows observation and control of all points in a digital system by scanning through the Shadow Register. As a result, test vector generation is greatly simplified and a high degree of fault coverage can be easily obtained. A standalone 8-Bit Diagnostic Register (SN54/74S818 shown in Figure 3b) is also available. Several references are listed at the end of this paper which provide a detailed description of diagnostic architecture and how to use it, including Session 16 of this conference (see r4).

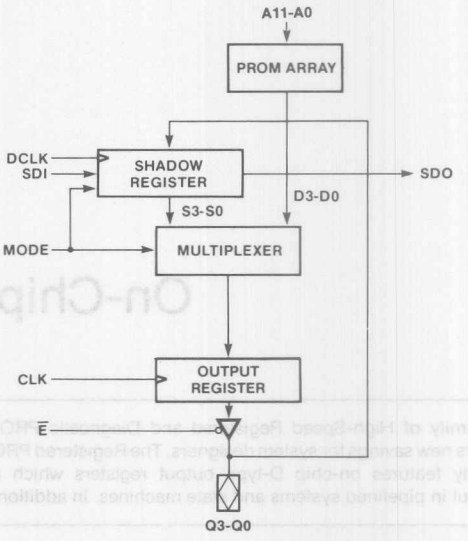


Figure 2. Diagnostic PROM Block Diagram

Product Families
Registered PROMs

The Registered PROMs are configured in 8-bit wide organizations with densities of 4K, 8K, and 16K. The following Registered PROMs are available:

53/63RA481	— 512 words x 8-bit memory with both synchronous and asynchronous three-state enables and preset and clear functions
53/63RS881	— 1024 words x 8-bit memory with both synchronous and asynchronous three-state enables and 16 synchronous initialization words
53/63RA1681	— 2048 words x 8-bit memory with asynchronous three-state enable and 16 synchronous initialization words
53/63RS1681	— 2048 words x 8-bit memory with synchronous three-state enable and 16 synchronous initialization words

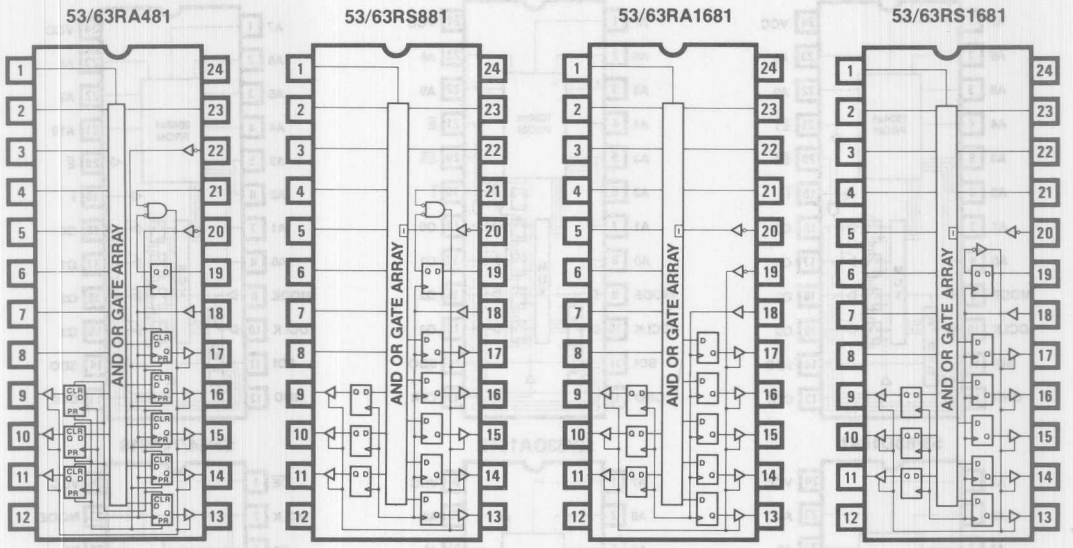


Figure 3a. Logic Symbols for the Registered PROM Family

Diagnostic PROMs

The Diagnostic PROMs are configured in 4-bit wide organizations with densities of 4K, 8K and 16K. The following Diagnostic PROMs are available:

53/63DA441 — 1024 words x 4-bit memory with asynchronous initialization and two asynchronous three-state enables

53/63DA442 — 1024 words x 4-bit memory with both asynchronous and synchronous three-state enables

53/63DA841 — 2048 words x 4-bit memory with asynchronous initialization and asynchronous three-state enable

53/63D1641 — 4096 words x 4-bit memory with asynchronous three-state enable

53/63DA1643 — 4096 words x 4-bit memory with asynchronous initialization and totem-pole outputs

Both the Registered PROMs and Diagnostic PROMs are available in space-saving 24-pin SKINNYDIP® (0.3- inch wide) packages and are specified over both commercial and military temperature ranges.

Features

Edge Triggered Registers

Data from the PROM is loaded into the Output Register on the rising edge of the clock. The use of the term "register" is to be distinguished from the term "latch," in that a register contains master-slave flip-flops while a latch contains gated flip-flops. In other words a register is edge-triggered while a latch is level-sensitive. The output of a register will change only on the rising edge of the clock. A latch holds whatever input data is present on the falling edge of the clock. The distinguishing advantage of a register is that its output will only change on the rising edge of the clock, while a latch becomes transparent (output follows input) when the clock is HIGH. As a result, system timing is simplified and faster microcycle times can be obtained.

Asynchronous Programmable Initialization

The Output Register can be loaded with a user-programmable initialization word. Each flip-flop in the Output Register may be individually programmed to either a HIGH state or a LOW state so that when the Initialize pin (I) is active (LOW), the Output Register will now contain this initialization word. Note that the initialization operation will occur independent of a clock pulse. Also, this feature is a superset of a preset and clear function. Therefore programmable initialization can be used to generate any arbitrary microinstruction for system reset or interrupt. This feature if offered in several of the Diagnostic PROMs.

SKINNYDIP® is a registered trademark of Monolithic Memories.

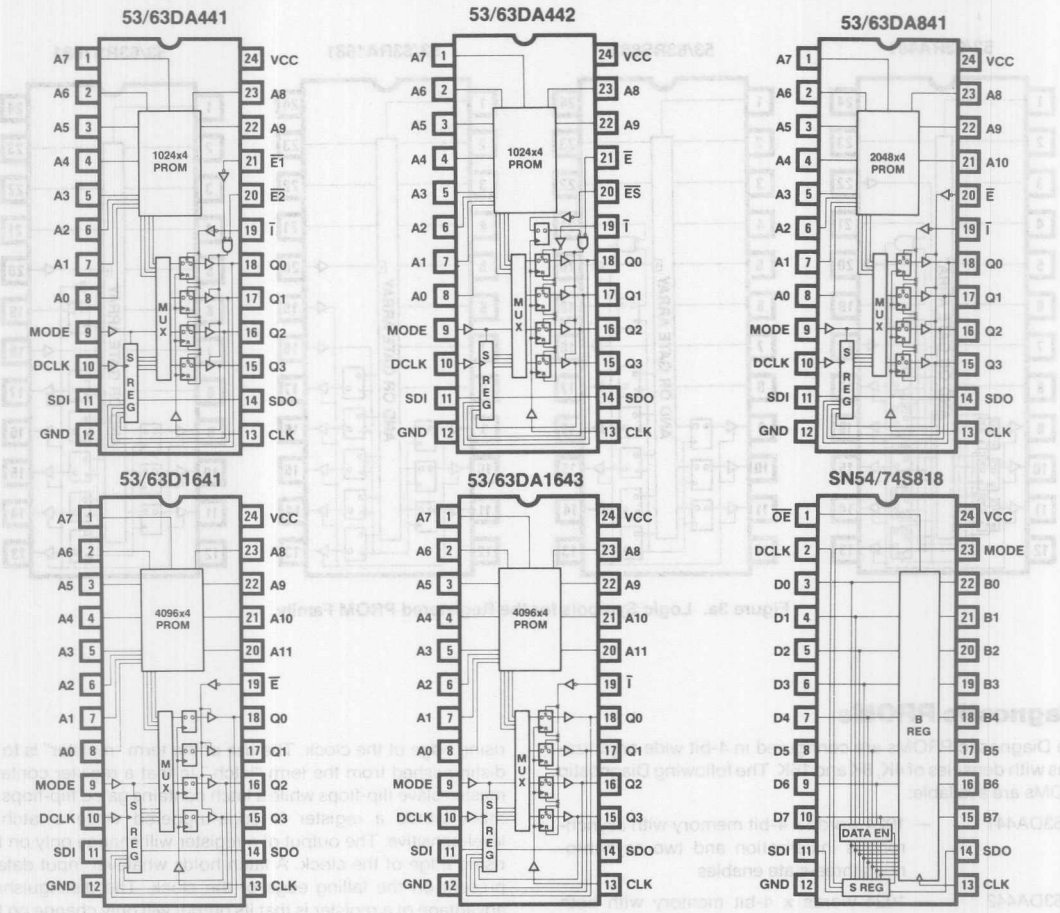


Figure 3b. Logic Symbols for the Diagnostic PROM Family

Synchronous Programmable Initialization

This feature provides sixteen user-programmable synchronous initialization words. As illustrated in the Block Diagram (Figure 4), with the synchronous initialize pin ($\overline{IS}$) LOW, one of sixteen column words (A3-A0) will be loaded into the Output Register following the clock pulse and independent of the row addresses (A9-A4). This is useful for implementing a small (≤16 word) reset or interrupt routine. With all $\overline{IS}$ column words (A3-A0) programmed to the same pattern, the $\overline{IS}$ function will be independent of both row and column addressing and may be used as a single pin control. This feature is offered in several of the Registered PROMs.

Three-State Drivers

The output of the register is buffered by three-state drivers which are compatible with low-power Schottky three-state bus standards. Thus VOL is 0.5 volts at IOL of 24 mA, VOH is 2.4 volts at IOH of -3.2 mA, and IOS minimum is guaranteed to be -20 mA. These hefty standards provide ample drive to meet the requirements of many bus standards.

Synchronous and Asynchronous Enables

Both synchronous and asynchronous output enable options are available. The synchronous output enable ($\overline{ES}$, see figure 5a), which is sampled on the rising edge of the clock, is used when more than one PROM is bused together to increase word length. In this case the enables effectively become the most significant address bits and, as such, must be registered just as data. Stated another way, when the clock goes high, the address is free to change , requiring enable information to be remembered somewhere. It is most appropriate to store the enable information. When the enable is not used, or when the outputs are to be gated onto some type of bus, the registered enable tends to get in the way. For this reason, the asynchronous output enable option ($\overline{E}$, see Figure 5b) is offered to allow direct control of the enable independent of the clock. For parts which have both synchronous and asynchronous output enables (see Figure 5c), outputs are enabled if, and only if, $\overline{ES}$ is LOW during the last rising edge of the clock and $\overline{E}$ is LOW.

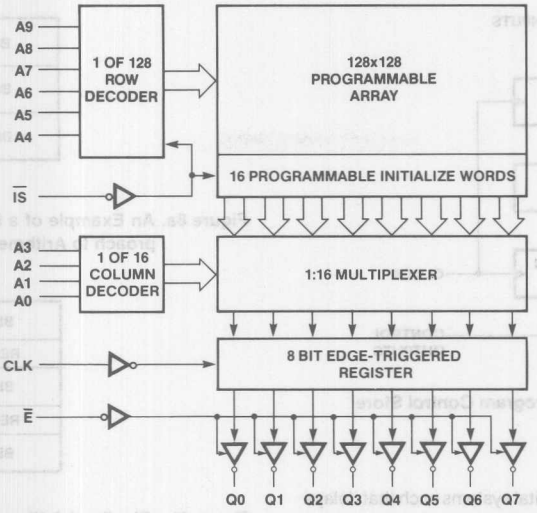


Figure 4. Block Diagram of 24-pin 53/63RA1681 Registered PROM. The 2Kx8 Registered PROM Contains Sixteen Programmable Initialization Words

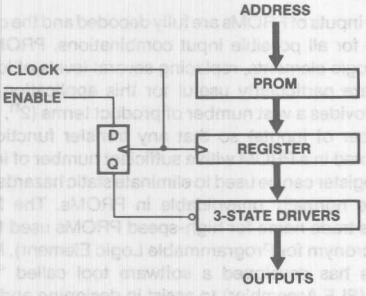


Figure 5a. Synchronous Output Enable ($\overline{ES}$)

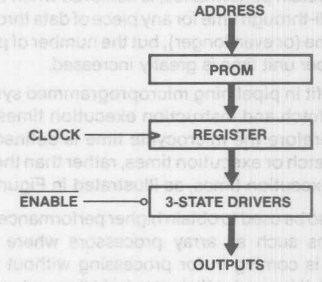


Figure 5b. Asynchronous Output Enable ($\overline{E}$)

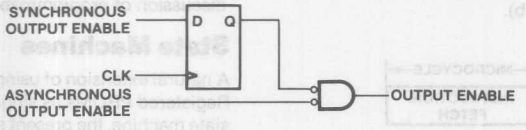


Figure 5c. Enabling of Outputs for Both Synchronous ($\overline{ES}$) and Asynchronous ($\overline{E}$) Output Enables

Application Areas

Microprogram Control Store

Microprogramming is the technique of using control programs stored in high-speed memory, such as bipolar PROMs, to instruct a digital system to perform various functions. A typical microprogram control store architecture is given in Figure 6.

The Microprogram Sequencer generates the addresses for the Microprogram Memory which stores the control program. The Microprogram register assures that all bits change simultaneously after the clock pulse and allows for pipelining instruction fetch and instruction execution. Some bits from the register are fed back to the sequencer while others are used for system control. This field of bits is called a Microword.

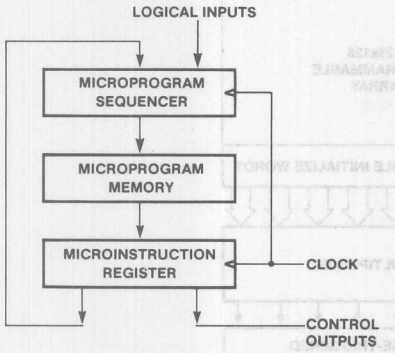


Figure 6. Typical Microprogram Control Store

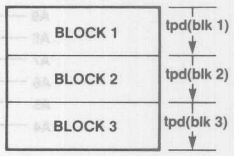


Figure 8a. An Example of a Nonpipelined (Fall-through) Approach to Arithmetic Operation

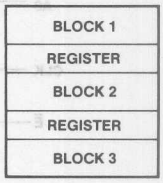


Figure 8b. Pipelined Arithmetic Operation. Note That t_{PD} = MAX [t_{PD} (blk 1), t_{PD} (blk 2), t_{PD} (blk 3)] + t_{PD} (reg)

Pipelined Systems

Pipelining is the art of designing digital systems such that delays associated with causal operations occur in parallel. A complex operation is divided into several smaller stages which are performed during clock cycles. Just as a widget traveling down an assembly line, each stage is operating on a piece of information which the previous stage operated on during the previous clock cycle. Maximum utilization of the hardware, which translates into maximum system performance, is achieved when the pipeline is full. The fall-through time for any piece of data through the system is the same (or even longer), but the number of pieces of data processed per unit time is greatly increased.

Clearly the benefit in pipelining microprogrammed systems is that instruction fetch and instruction execution times can be overlapped. Therefore the microcycle time is defined as the longer of either fetch or execution times, rather than the sum of both fetch and execution times, as illustrated in Figure 7.

Pipelining can also be used to obtain higher performance in data-intensive systems such as array processors where a large amount of data is coming in for processing without passing through the CPU. It is very inefficient to hold the next set of data until the previous data has propagated through all of the logic blocks in the system (Figure 8a). It is more efficient to pipeline the system and load new data after the previous data has been passed to the next block (Figure 8b).

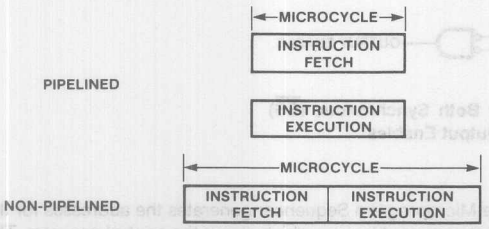


Figure 7. Length of Microcycle for Pipelined and Non-Pipelined Systems. Note that Delays are Overlapped in the Pipelined System, While Delays are Summed in the Non-Pipelined System

Programmable Logic Elements (PLE) Devices

Since the inputs of PROMs are fully decoded and the outputs are definable for all possible input combinations, PROMs can be used as logic elements, replacing several levels of logic gates. PROMs are particularly useful for this application since the PROM provides a vast number of product terms (2^n, where n is the number of inputs) so that any transfer function can be implemented in a PROM with a sufficient number of inputs. The Ouput Register can be used to eliminate static hazards (glitches) which are normally unavoidable in PROMs. The Monolithic Memories trade name for high-speed PROMs used for logic is "PLE" (acronym for Programmable Logic Element). Monolithic Memories has developed a software tool called "PLEASM" software (PLE Assembler) to assist in designing and programming PROMs as PLEs. PLEASM is available for many computers and may be requested through the Monolithic Memories IdeaLogic Exchange. References r6, r7 and r8 offer an in-depth discussion of programmable logic applications for PROMs.

State Machines

A natural extension of using PROMs as logic elements is to use Registered PROMs as single-chip State Machines. In a classic state machine, the present state (or output) is a function of both the present inputs and the previous state. The combinatorial logic is implemented in the PROM array and the Output Register is used to store the state. One or more of the Registered PROM outputs are connected to address inputs in order to provide the state of the machine. The abundance of product terms in a PROM used to implement combinatorial logic translates into an unlimited combination of states. For example, a 2Kx8 Registered PROM can implement a 4-input, 8-output machine with any combination of 128 states. States and inputs can be traded off to provide a wide range of possible state machines. The programmable initialization feature is convenient to initialize the state machine.

Some Application Examples
64-Bit Microcontroller

A 4096-word by 64-bit wide microcontroller can be constructed using sixteen 4096x4 Diagnostic PROMs (53/63D1641) and one Programmable Array Logic (PAL®) chip. This controller supplies thirty-six control signals, four status select bits, and two addresses of twelve bits each (Figure 9) — one for the Next address and one for the Jump address.

In this design, three PROMs are used to store the Next address while an additional three PROMs are used to store the Jump address. Note that three 4-bit wide PROMs provide sufficient inputs to address the full 4096 words of Microprogram memory. One PROM is used to store four status select inputs to the PAL device which is used as a multiplexer for test conditions. A PAL16C1 logic circuit or PAL20C1 device is ideal for this Test Mux since these parts provide many inputs (16 and 20 respectively) and complementary output (both true and inverted) polarities. The remaining nine PROMs are used to store the 36-bit Microcontrol word. Note that a Microprogram sequencer is not used in this architecture.

The Microcontrol signals control various parts of the CPU and other external blocks such as memory and I/O. For certain microinstructions, some operations may involve a Jump. The 4-bit status select PROM will select a status bit from the test conditions to a pair of complementary outputs which will enable either the Next address or the Jump address. The address enabled will point to the Next microinstruction in the bank of PROMs. If no conditional Jump is needed, both the Next address and the Jump address will be the same.

For example, a Jump will be performed if bit 11 of the test conditions is set; the status select bits will be 1011 (which represent 11) and the status to be tested and its complement will appear on the outputs of the PAL device. Noting that the output enables of the Diagnostic PROMs are active LOW, the true PAL device output controls the NEXT address PROMs, while the inverted PAL device output controls the Jump address PROMs. If the test status is TRUE, the PAL device output disables the Next address PROMs while the inverted PAL device output enables the Jump address PROMs. The reverse will occur when the test status is FALSE. This Next/Jump decision is illustrated in Figure 10.

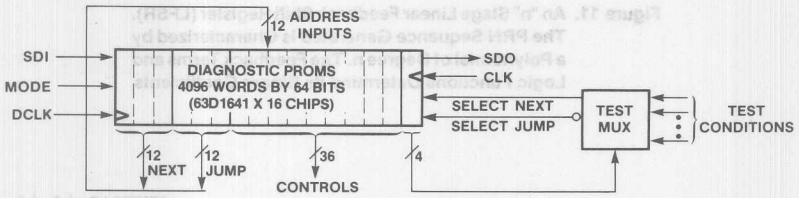


Figure 9. 64-bit Microcontroller Using Sixteen 4Kx4 Diagnostic PROMs and One PAL Device

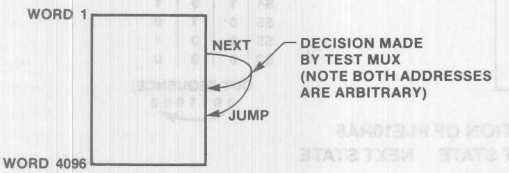


Figure 10. Microprogram Memory Map Illustrating the Next Address/Jump Address Decision Made by the Test Multiplexer

The decision cycle time is computed by the following equation:

$$f_{MAX} = \frac{1}{t_{su} + t_{CLK} + t_{PD} + t_{PXZ}}$$

where

t_{su} = address setup time for the diagnostic PROM
t_{CLK} = clock to output delay of the PROM
t_{PD} = propagation delay in the outside logic
t_{PXZ} = output enable/disable delay for the diagnostic PROM.

Note that the decision time can be decreased if the Next/Jump decision is made one clock cycle ahead and stored using a synchronous enable. This scheme will reduce the decision time by an amount equal to the propagation delay through the PAL Test Mux, but microcoding this system will become much more complex.

Fewer PROMs would be required if an even/odd Jump address scheme were used (such as only allowing Jumps to certain paragraphs), however this decreases the flexibility of PROM addressing.

Pseudo Random Number Generator

In the data path, a Registered PROM can be used to implement complex functions such as a Pseudo Random Number (PRN) Generator. PRN sequences are useful in encoding and decoding of information in signal processing and communication systems. They are used for data encryption in secure communication links, and error detection and correction codes in data communication systems. PRN sequences are also utilized as test vectors for testing digital systems and as reference white noise in many signal processing applications.

There are many techniques for generating PRN sequences. The most common technique is to use "n" stages of linear shift registers with feedback paths to determine a polynomial which characterizes a PRN sequence. Figure 11 illustrates a typical mechanism for generating PRN sequences.

The advantage of using a PROM (or PLE) device for implementing PRN sequences is that any polynomial can be quickly customized in it. In data encryption systems where the code is frequently changed for protection from mischievous eavesdroppers, a PROM can be used to generate a new code each time or several codes can be implemented in the same PROM.

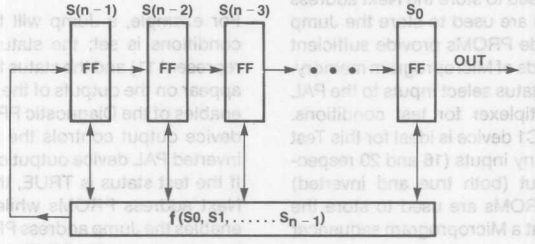


Figure 11. An "n" Stage Linear Feedback Shift Register (LFSR). The PRN Sequence Generated is Characterized by a Polynomial of Degree n. The Feedback Terms and Logic Functions Determine its Binary Coefficients

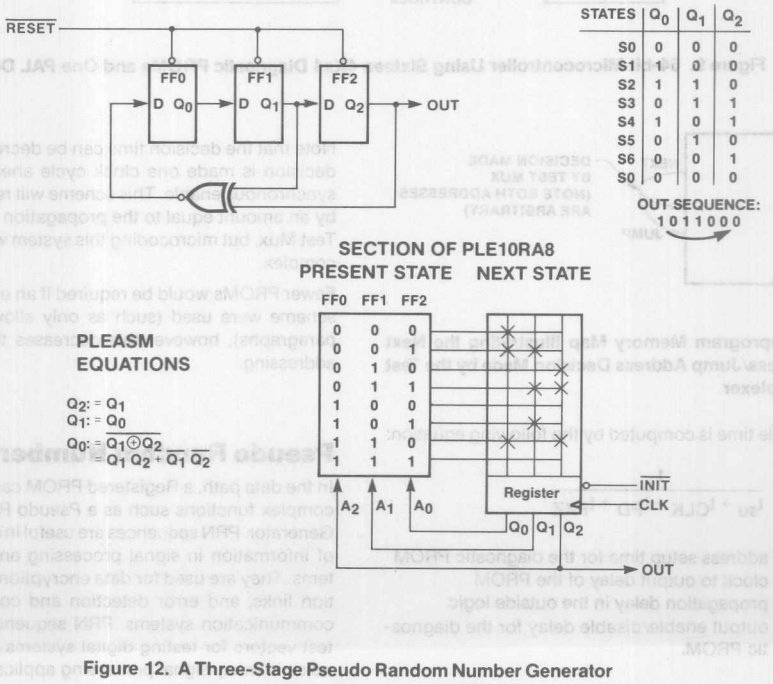


Figure 12. A Three-Stage Pseudo Random Number Generator Implemented in a Registered PROM (PLE)

An example of a PRN generator implemented in a Registered PROM is shown in Figure 12. A linear 2-input XOR function is used to generate a PRN sequence characterized by a polynomial of degree 3. The PRN sequence is of maximum length with period 7.

Cyclical Redundancy Check (CRC) is widely used for Error Detection in data communication. Both serial and parallel CRC can be performed depending on the nature of application. In serial data transfer on Local Area Networks, or between peripheral and main memory, serial CRC is the preferred and perhaps the most efficient technique. However, systems employing wide data buses for high-speed short-distance data transfer require a high-speed mechanism of ensuring data integrity. In these applications, parallel CRC might be the better alternative.

The implementation of an M-bit parallel CRC is more complex than its serial counterpart. Although both use Linear Feedback Shift Register (LFSR) configurations, the parallel implementation requires M-bit carry look-ahead circuitry to process the M data bits simultaneously (see reference r9). The equations for this carry look-ahead represent the output of each stage in the LFSR after every shift of an M-bit string of data. These equations contain a large number of XOR operations which make it very efficient to implement in a Registered PROM.

To illustrate with a practical example, Figure 14 shows the serial implementation of the CRC generator polynomial

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

also called the CRC-CCITT standard. Figure 16 shows the 8-bit carry look-ahead equations for an 8-bit parallel CRC implementation of the same polynomial. These equations are derived in reference r9, where an implementation in four PAL devices is also shown with a maximum delay of 90 ns. Figure 15 shows an implementation in only three Registered PROMs and one SSI chip. The maximum delay is 50 ns.

The speed of operation of parallel CRC implemented in Registered PROMs will remain the same for any generator polynomial and M. Increasing the complexity of the carry look-ahead equations only increases the number of devices required to implement them. It does not increase the delay.

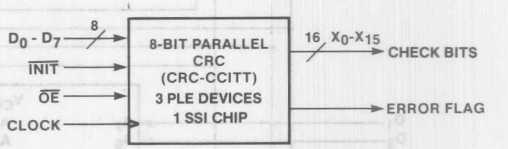


Figure 13. Block Diagram of an 8-Bit Parallel CRC

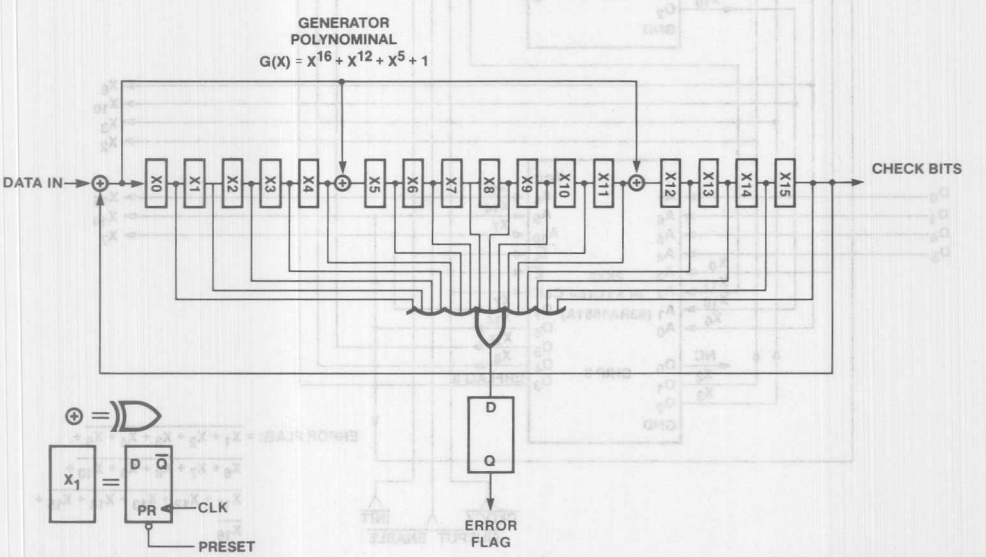


Figure 14. A 16-Bit Linear Feedback Shift Register (LFSR) Implementing a Serial CRC Generator

9

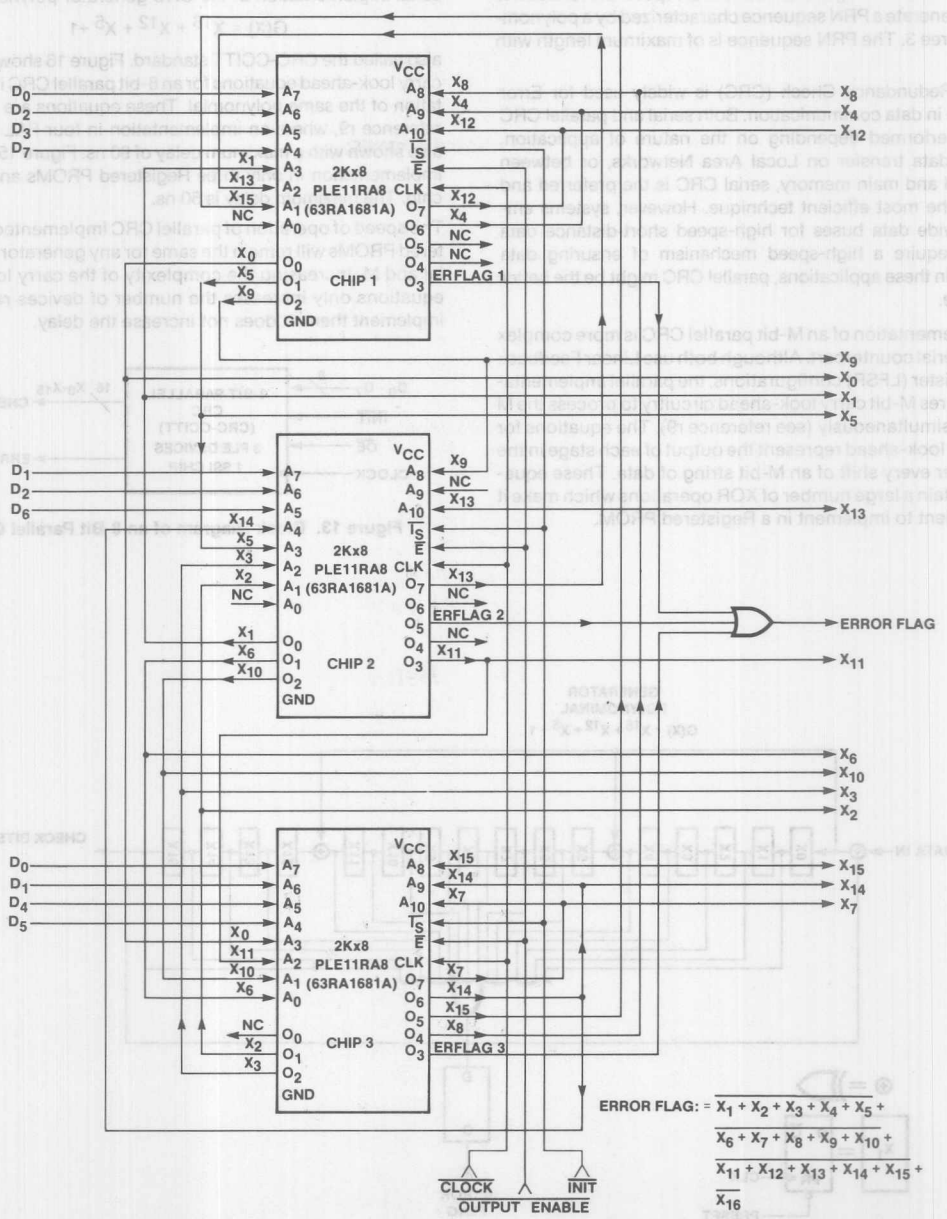


Figure 15. Diagram Showing How to Connect Three Registered PROM (or PLE) Devices Together to Implement 8-Bit Parallel CRC. The Error Flag is Valid on the Next Clock Pulse After All the Data Has Been Clocked In

X0 (n + 1) := X8 (n) ⊕ X12(n) ⊕ D(3) ⊕ D(7) chip1
X1 (n + 1) := X9 (n) ⊕ X13(n) ⊕ D(2) ⊕ D(6) chip2
X2 (n + 1) := X10(n) ⊕ X14(n) ⊕ D(1) ⊕ D(5) chip3
X3 (n + 1) := X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3
X4 (n + 1) := X12(n) ⊕ D3 ..... chip1
X5 (n + 1) := X8 (n) ⊕ X12(n) ⊕ X13(n) ⊕ D(2) ⊕ D(3) ⊕ D(7) chip1
X6 (n + 1) := X9 (n) ⊕ X13(n) ⊕ X14(n) ⊕ D(1) ⊕ D(2) ⊕ D(6) chip2
X7 (n + 1) := X10(n) ⊕ X14(n) ⊕ X15(n) ⊕ D(0) ⊕ D(1) ⊕ D(5) chip3
X8 (n + 1) := X0 (n) ⊕ X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3
X9 (n + 1) := X1 (n) ⊕ X12(n) ⊕ D(3) chip1
X10(n + 1) := X2 (n) ⊕ X13(n) ⊕ D(2) chip2
X11(n + 1) := X3 (n) ⊕ X14(n) ⊕ D(1) chip2
X12(n + 1) := X4 (n) ⊕ X8 (n) ⊕ X12(n) ⊕ X15(n) ⊕ D(0) ⊕ D(3) ⊕ D(7) chip1
X13(n + 1) := X5 (n) ⊕ X9 (n) ⊕ X13(n) ⊕ D(2) ⊕ D(6) chip2
X14(n + 1) := X6 (n) ⊕ X10(n) ⊕ X14(n) ⊕ D(1) ⊕ D(5) chip3
X15(n + 1) := X7 (n) ⊕ X11(n) ⊕ X15(n) ⊕ D(0) ⊕ D(4) chip3

where Xi (n + 1) is the next state value of the corresponding register i, i = 0, ..., 15
Xi (n) is the present value of the corresponding register i, i = 0, ..., 15
D (n) is the parallel input data bits, where n = 0, ..., 7

Figure 16. Carry Look-Ahead Equations for 8-Bit Parallel CRC with G(X). The Equations are Partitioned into Parts for Efficient Implementation in Three Chips

Summary

There are many interesting applications for high-speed Registered and Diagnostic PROMs. The integration of a Shadow Register in the Diagnostic PROM greatly simplifies system level diagnostics.

Acknowledgements

The Pseudo Random Number Generator and the Parallel CRC application design examples originated from Zahir Ebrahim and Vivian Kong, colleagues of ours at Monolithic Memories. These two applications are reprinted from Monolithic Memories Application Note AN-126 (see reference r6).

"...THE DIAGNOSTIC PROMS AND DIAGNOSTIC REGISTERS HELP YOU TO ANALYZE YOUR SYSTEMS CONVENIENTLY !..."

References

r1. *"Shadow Register Architecture Simplifies Digital Diagnosis"* John Birkner, Vincent Coli and Frank Lee. Monolithic Memories Application Note AN-123.

r2. *"New PROM Architecture Simplifies Microprogramming"* John Birkner, Vincent Coli and Frank Lee, Electro 1983, Session 24.

r3. *"Testing Algorithms for LSI PALs"*, Imtiyaz Bengali and Vincent Coli, Wescon 1983, Session 13.

r4. *"Diagnostic Devices and Algorithms for Testing Digital Systems"* Imtiyaz Bengali, Vincent Coli and Frank Lee, Electro 1984, Session 16.

r5. *"Registered PROMs Impact Computer Architecture"* John Birkner, Monolithic Memories Application Note AN-107.

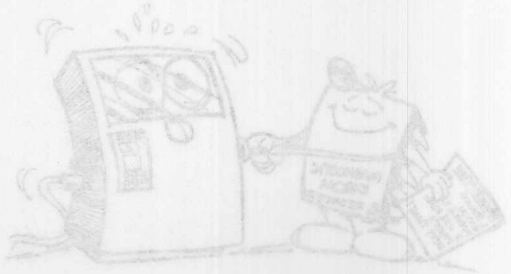
r6. *"PROMs and PLEs: An Application Perspective"* Zahir Ebrahim, Monolithic Memories Application Note AN-126.

r7. *"High Speed Bipolar PROMs Find New Applications as Programmable Logic Elements"* Vincent Coli and Frank Lee, 9th West Coast Computer Faire 1984.

r8. *"PLE Programmable Logic Element Handbook"* Monolithic Memories, Inc.

r9. *"Implementation of Serial/Parallel CRC Using PAL Devices"* Vivian Kong, Monolithic Memories Application Note AN-125.

PAL® (Programmable Array Logic) and SKINNYDIP® are registered trademarks of Monolithic Memories.
DOC™, PLE™ and PLEASM™ are trademarks of Monolithic Memories.



Diagnostic Devices and Algorithms for Testing Digital Systems*

Imtiyaz M. Bengali, Vincent J. Coli and Frank Lee

A new concept called Diagnostics-On-Chip (DOC) was introduced in the industry recently. A series of new products with shadow register diagnostic capability is coming. These new products use this new concept and will provide a cost-effective solution to the issue of testability for digital systems.

* This paper is a slightly modified version of the paper by the same name which appeared in the *Electro'84 Professional Program Session Record*, Session 16 reprint, paper 16.1: 15-17 May 1984.

Testing Digital Systems

Imtiyaz M. Bengali, Vincent J. Coli and Frank Lee/Electro 84

A new concept called Diagnostics On-Chip (DOC) was introduced in the industry recently. A series of new products with shadow register diagnostic capability is coming. These new products use this new concept and will provide a cost effective solution to the issue of testability for digital systems.

Introduction

In developing a digital system, cost is a very sensitive issue. For the OEMs, cost itself can be categorized as R & D, manufacturing, marketing, testing and maintenance costs, etc. The strategy is to reduce the overall cost for a system. Since marketing cost is about the same for all systems of a certain type and R & D cost is a one-time expense, it will be beneficial to put in diagnostic features to reduce the future expense in testing and maintenance.

If a large system goes down, it will not be practical to test all the chips individually. An alternative is to have built-in test circuits. If an error occurs, it can be located by running a test sequence through the system. It will definitely save a lot of time and expense compared to using tens or hundreds of man hours to debug the system manually.

Basics of Diagnostics

The test problem has two major facets:

1. Test generation.
2. Test verification.

Test generation is the process of determining the test sequence for a circuit which will demonstrate its correct operation. Test verification is to prove that the circuit works with the test vectors. Fault simulation has been the best technique of yielding a quantitative measure of test effectiveness. Test sequences are automatically generated and verified in the circuit after simulating a single "stuck-at-type" of fault in it. By observing the circuit outputs, faults can be detected and a quantitative measure of test effectiveness can be evaluated.

This technique is efficient for testing combinatorial circuits, especially smaller circuits where the test sequence is trivial. For a more complex circuit, many techniques are available, such as D-Algorithm, Compiled Code Boolean Simulation and Adaptive Random Test Generation.

The techniques for combinatorial circuits are inefficient and ineffective for sequential circuits. As a first approximation, one can treat a sequential circuit as being purely combinatorial within each clock cycle and test it with the above techniques for a particular state. Every time the machine makes a transition to a new state, the test sequence is different. This is a very costly way of testing the circuit, especially if it has many states. Moreover, one should have the knowledge of initial state, illegal states, and sequences to bring the machine out from an illegal state into a known state. All these problems pertaining to testing of sequential circuits have given rise to the concept of "design for testability".

The key concepts are CONTROLLABILITY and OBSERVABILITY. Control and observation of a network are essential to implement its test procedure. Various designs for testability methods have evolved in the last five to six years. All of these methods have the same objective—to be able to control and observe critical points in a network. These techniques allow test generation problems to be completely reduced to the generation of test vectors for combinatorial logic.

For example, consider the case of the AND gate:

Figure 1. A Simple 2-Input AND Gate

In order to test for a stuck-at-1 (sa1) fault, it is necessary to put 'A' to '0', 'B' to '1', and observe output 'C' for '0' or '1'. If '0' is observed at C, then the AND gate is good for sa1 fault; otherwise there is a fault. In order to fully test the AND gate, the following test vectors are to be exercised:

A	B	C	
0	1	0	Detect sa1
1	0	0	
1	1	1	Detect sa0

Table 1. A Set of Test Vectors Fully Covering all Stuck-At-Faults of the AND Gate in Figure 1

As the circuit becomes more complex, it is more difficult to control and observe every signal path. Thus, it becomes essential to give serious thought to the testability of the circuit through the design phase. One approach is to adopt structured design methodology. Ideally, this means that the design is totally synchronous with the system clock.

Most structured design practices are built upon the concept that if the values in all of the registers can be controlled to any specific value, and if they can be observed with a straightforward operation, then the test generation, and possibly the fault simulation task, can be reduced to doing test generation and fault simulation for a combinatorial network. A control signal can switch the memory elements from their normal mode of operation to a mode that makes them controllable and observable.

A simple but effective way to convert a sequential network to a combinatorial one is by breaking the feedback loop and inserting the test data in place of the sequential data in the feedback registers.

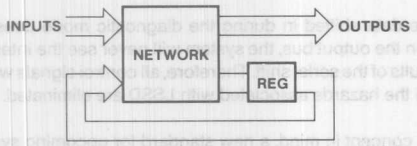


Figure 2a. A Simplified Representation of a Sequential Network

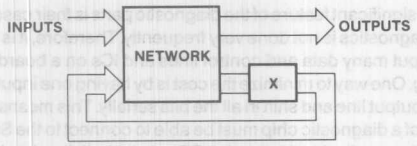


Figure 2b. The Feedback Path on the Sequential Network is Broken in Order to Reduce the Network to a Pseudo-Combinatorial One

There are many methods of design for testability practiced in the industry, like LSSD, ScanPath, Scan/Set, Random Access, and BILBO. All of these techniques require additional hardware, mostly shift registers, in order to input test sequences and to observe critical points in the circuit. It appears that additional cost in terms of special hardware has to be incurred for designing testability and structured design. But since the cost of hardware is declining, the trade-off is advantageous in the reduction of testing cost of bigger circuits.

Moreover, the circuit is well monitored and documented. Thus when the boards are in the field, and if there is a fault in a particular board, each block of the circuit can be monitored efficiently and the fault can be easily diagnosed, thus reducing maintenance cost in the future.

Previous Diagnostic Schemes

There are two basic methods to load in test vectors: parallel loading, and serial scanning.

Parallel loading of data in and out requires very wide input and output buses and is not worthwhile. Besides, it would not be effective to store and analyze the results. Built-in digital circuit observer (BIDCO) is a modified example of parallel loading of diagnostic data using a pseudorandom number generator to generate test vectors.

Serial scanning needs several clock cycles to load in or shift out the test results. It may take several minutes to run all the diagnostic vectors through the system. Considering the time needed to analyze the results and repair, the time taken to run the diagnostic vectors is insignificant. Examples of serial scan diagnostics techniques are level-sensitive scan design (LSSD) and Diagnostic-On-Chip (DOC). LSSD involves shifting of diagnostic data into latches, testing the system with that data and then shifting out the test result. DOC uses a buried register, called a shadow register, through which diagnostic data is shifted in and out.

Shadow Register—Why?

For the LSSD, outputs from the microcontrol store will contain some intermediate data when diagnostic microinstructions are shifted in and test results are shifted out. If several control signals are used to drive several ports on the same bus, it is possible that more than one port may be enabled at the same time by the intermediate data (as shown in Figure 4), thus creating a bus fight. The result may be hazardous to your system. Other hazards such as disk crashes are also possible. Designing with LSSD forced compromises in system design.

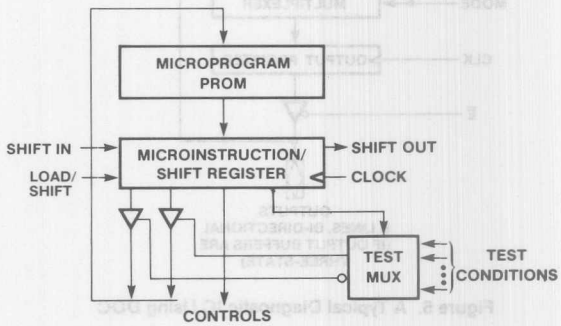


Figure 3. Serial Scanning Techniques Simplify Testing by Serially Shifting in Test Data and Shifting Out Test Result

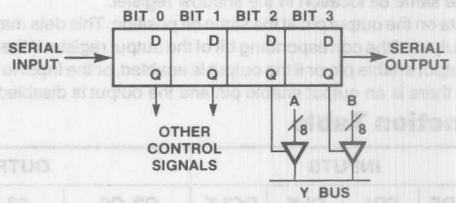


Figure 4. Potential Bus Fight May Appear as Port A and Port B may Both be Enabled when Test Data or Result is Shifted Through the Register Bits (Also Called Three-State Overlap)

If the diagnostic data is shifted into some buried registers which are not directly tied to the control lines, the above problem can be avoided. This is the concept of Diagnostic-On-Chip (DOC) which uses shadow register diagnostics.

An additional feature for a shadow register is to permit test vectors to be shifted in during normal execution, which means it is not necessary to hold up the system too long in order to perform diagnosis.

A shadow register is basically a buried register with shift capability (Figure 5). There is also an output register whose

9

outputs appear to the rest of the system. Each output flip-flop has an associated flip-flop in the shadow register. An output flip-flop drives a three-state buffer before going to the output pin. If the output is disabled, the output pin may be converted to an input pin. This feature is very important if the output is driving a bus and sampling of data on the bus desired.

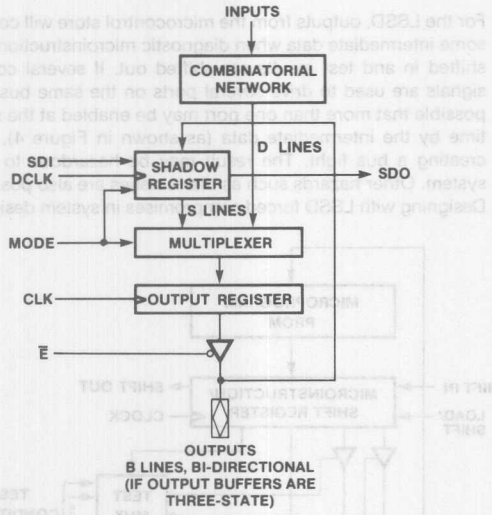


Figure 5. A Typical Diagnostic IC Using DOC

The input to any bit of the output register is multiplexed from one of two sources:
1) The less significant bit location in the shadow register (or SDI for the least significant bit). This operation is just a simple shift register.
2) The same bit location in the shadow register.
3) Data on the output pin at the same bit position. This data may be the output of the corresponding bit of the output register if there is no output enable pin or if the output is enabled, or the input to that pin if there is an output enable pin and the output is disabled.

The input to a bit of the shadow register is multiplexed from one of three sources:
1) The corresponding input bit from the memory array.
2) The corresponding bit location in the shadow register.

Since the data shifted in during the diagnostic mode does not appear on the output bus, the system will never see the intermediate results of the serial shift. Therefore, all control signals will be valid and the hazards associated with LSSD are eliminated.

With this concept in mind, a new standard for upcoming system diagnostics can now be presented.

Cascadability of the Diagnostic ICs

One very significant feature of the diagnostic parts is their cascadability. Diagnostics is not done very frequently. Therefore, it is very costly to put many data and control lines and ICs on a board just for testing. One way to minimize the cost is by having one input line and one output line and shift in all the bits serially. This means that the SDO of a diagnostic chip must be able to connect to the SDI of another diagnostic chip. Noting that SDI can be both the data input or the control input, SDO must contain the most significant bit of the shadow register if SDI is the data input, and must pass the content of SDI if SDI is used as a control signal.

There is only one data input and one data output to the diagnostic parts. When serial data is shifted in or shifted out, data has to be passed from one diagnostic chip to another. Since SDI must be passed from chip to chip (if it is used for control), it is necessary for logic designers to make sure the fall-through time of SCI to the last chip and the setup time from SDI to DCLK are satisfied.

The Diagnostic IC Family

A family comprised of 4K, 8K and 16K Diagnostic PROMs (DPROM) with 4-bit output organizations and 8-bit Diagnostic Register is available from Monolithic Memories. These devices are packaged in industry standard 24-pin SKINNYDIP® (0.30-inch wide) packages and are specified over both commercial and military temperature ranges.

Function Table

INPUTS				OUTPUTS			OPERATION
MODE	SDI	CLK	DCLK	Q3-Q0	S3-S0	SDO	
L	X	↑	*	Qn ← PROM	HOLD	S3	Load output register from PROM array
L	X	*	↑	HOLD	Sn ← Sn-1 S0 ← SDI	S3	Shift shadow register data
L	X	↑	↑	Qn ← PROM	Sn ← Sn-1 S0 ← SDI	S3	Load output register from PROM array while shifting shadow register data
H	X	↑	*	Qn ← Sn	HOLD	SDI	Load output register from shadow register
H	L	*	↑	HOLD	Sn ← Qn	SDI	Load shadow register from output bus
H	H	*	↑	HOLD	HOLD	SDI	No operation†

* Clock must be steady or falling. † Write back from shadow register to input bus (SN54/74S818 Diagnostic Register only).

Table 2. Operation of the Diagnostic ICs in a Digital System

The Diagnostic component series consists of the following products (see Figure 6 below for the Logic Symbols):

53/63DA441 — 1024 words x 4-bit memory with asynchronous initialization and two asynchronous three-state enables

53/63DA442 — 1024 words x 4-bit memory with asynchronous initialization and both asynchronous and synchronous three-state enables

53/63DA841 — 2048 words x 4-bit memory with asynchronous initialization and asynchronous three-state enable

53/63DA1641 — 4096 words x 4-bit memory with asynchronous three-state enable

53/63DA1643 — 4096 words x 4-bit memory with asynchronous initialization and totem-pole outputs

SN54/74S818 — 8-bit register with asynchonous three-state enable and write-back capability to the inputs, basically for loading of writeable control store (WCS). Even in the case of non-writeable control store, diagnostic registers should also be used in breaking the loops of the sequential system

The introduction of the DPROMs and diagnostic register results in a new standard for diagnostics. Noting that the diagnostic devices need controls over two independent registers and a multiplexer, a number of overhead pins are necessary. These overhead pins must be defined in a way that the diagnostic parts can be cascadable.

The diagnostic ICs need the following pins in addition to those used in a similar part without the diagnostic features:
1) Diagnostic Clock (DCLK)—The diagnostic clock is used to clock the shadow register.
2) MODE—This pin is used in selecting the data to the registers. For the output register, MODE = LOW indicates that the output register is being used as a normal register; MODE = HIGH indicates that the next state of the output register will be obtained from the shadow register. For the shadow register, MODE = LOW indicates serial data from SDI (see below) is shifted in every diagnostic clock; MODE = HIGH switches SDI from a data input to a control input. See below for details.

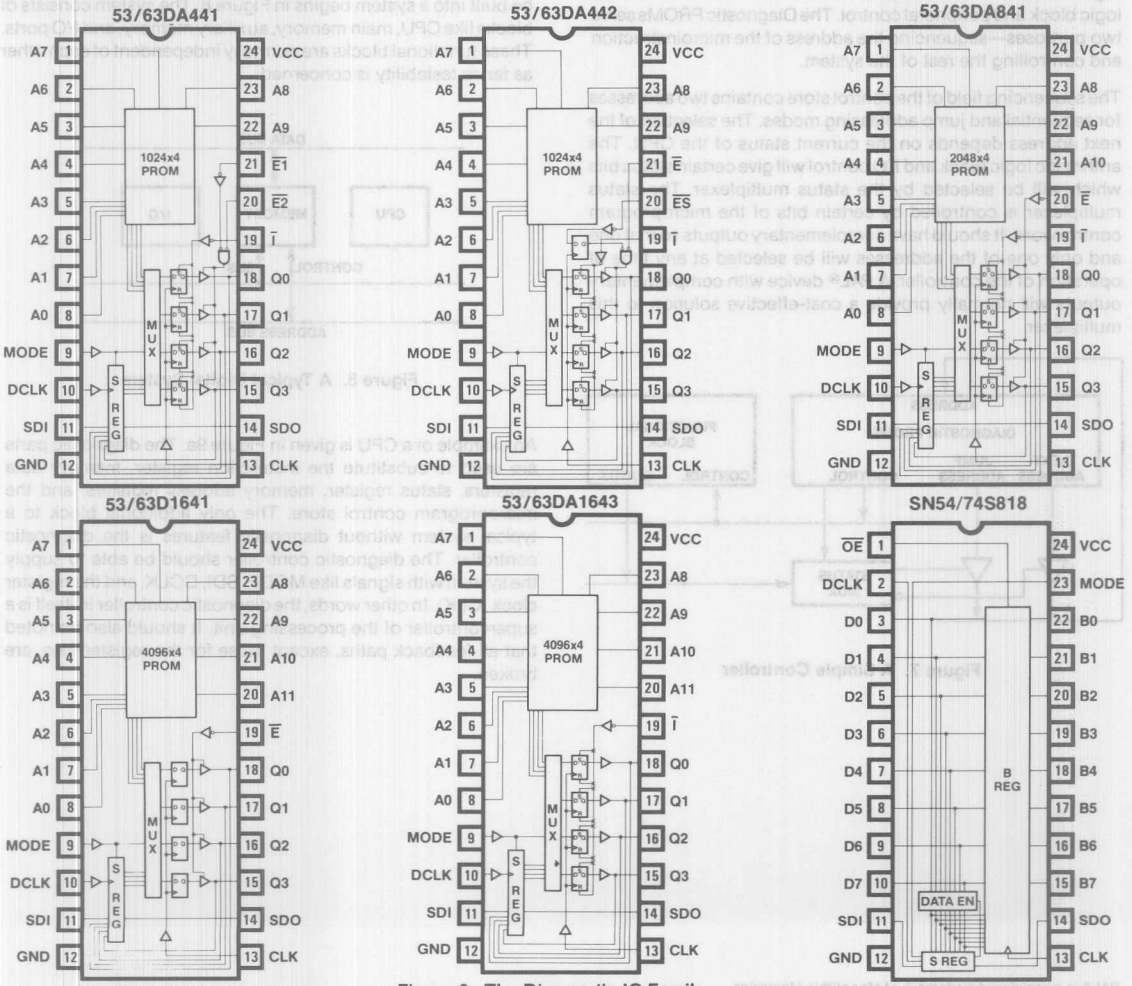


Figure 6. The Diagnostic IC Family

3) Serial Data In (SDI)—When MODE = LOW, this pin is used for shifting serial data in. When MODE = HIGH, SDI serves as a control pin. If MODE = HIGH and SDI = LOW, data from the output pins will be loaded to the shadow register on the next DCLK. MODE = HIGH and SDI = HIGH indicate a reserved operation for diagnostic PROMs, and is used for write-back for the diagnostic register.
4) Serial Data Out (SDO)—When MODE = LOW, this pin carries the shift-out bit of the shadow register. When MODE = HIGH, the SDI becomes a control pin and the control signal should be passed along if several diagnostic parts are connected together serially. So SDO should carry SDI along in this case

This standard is being used in designing all current and future diagnostic devices.

Some Applications Examples

A simple controller can be constructed using Diagnostic PROMs together with other functional blocks such as the arithmetic logic block and peripheral control. The Diagnostic PROMs serve two purposes—sequencing the address of the microinstruction and controlling the rest of the system.

The sequencing field of the control store contains two addresses for sequential and jump addressing modes. The selection of the next address depends on the current status of the CPU. The arithmetic logic block and I/O control will give certain status bits which will be selected by the status multiplexer. The status multiplexer is controlled by certain bits of the microprogram control word. It should have complementary outputs so that one and only one of the addresses will be selected at any time of operation of the controller. A PAL® device with complementary outputs will normally provide a cost-effective solution to this multiplexer.

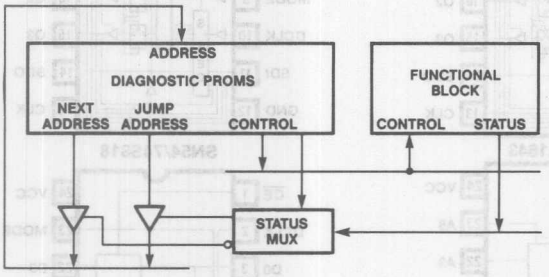


Figure 7. A Simple Controller

A continue statement can be implemented by having both addresses programmed to the next sequential address while an unconditional jump can be done by programming both addresses to the Jump address.

The control PROMs provide signals to control various functional blocks of the controller and other external blocks such as memory and I/O.

Since the other functional blocks such as the arithmetic logic and I/O control of the system also have sequential logic, it may be necessary to break the loops in those blocks so that diagnosis can be done on the whole system. The diagnostic registers can be incorporated in those blocks in places such as the registers (say, memory address registers, memory data registers, and instruction registers, etc.) The diagnostic data and result shifted into and out of the CPU can also be shifted through the diagnostic registers.

Another more detailed example of how the diagnostic parts can be built into a system begins in Figure 8. The system consists of blocks like CPU, main memory, auxiliary memory, and I/O ports. These functional blocks are normally independent of each other as far as testability is concerned.

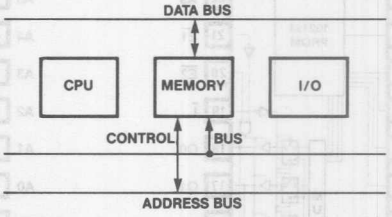


Figure 8. A Typical Digital System

An example of a CPU is given in Figure 9a. The diagnostic parts are used to substitute the instruction register, memory data registers, status register, memory address registers, and the microprogram control store. The only additional block to a typical system without diagnostic features is the diagnostic controller. The diagnostic controller should be able to supply the system with signals like MODE, SDI, DCLK, and the register clock (CLK). In other words, the diagnostic controller in itself is a supercontroller of the processing unit. It should also be noted that all feedback paths, except those for the register files, are broken.

PAL® is a registered trademark of Monolithic Memories.

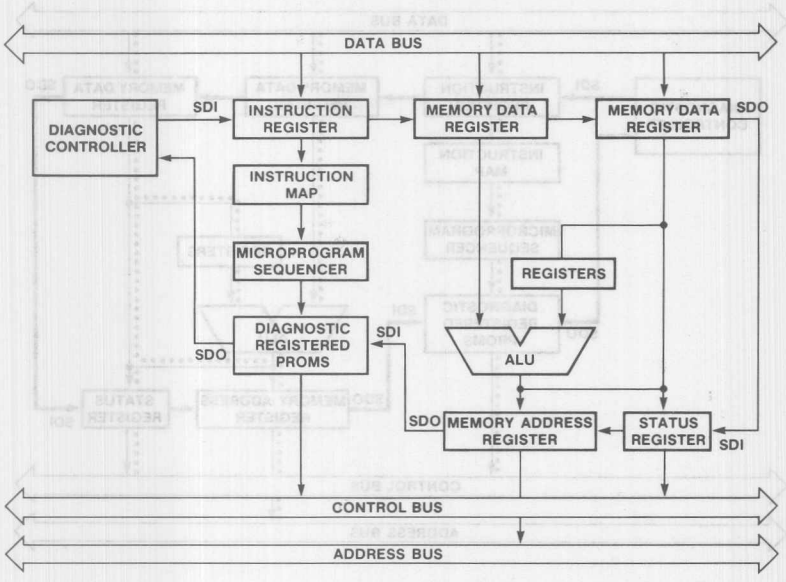


Figure 9a. A CPU Using DPROMs and Diagnostic Registers

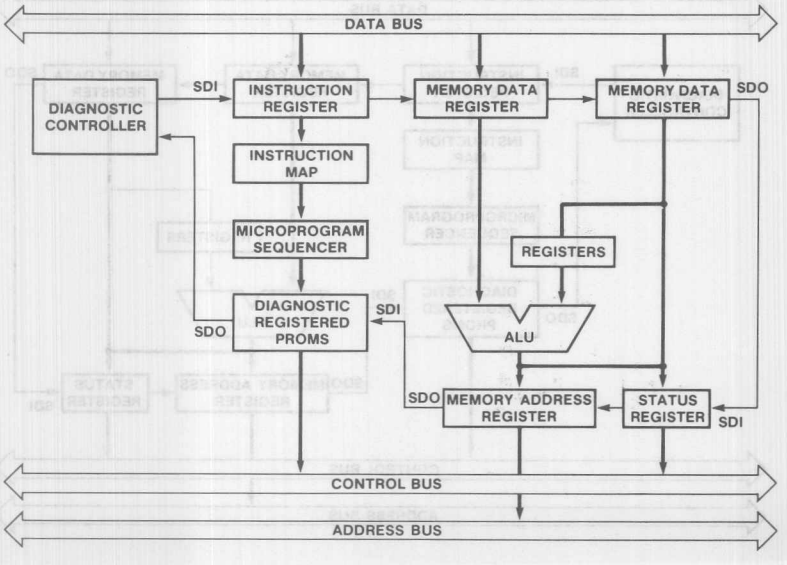


Figure 9b. Data Flow During Normal CPU Operation

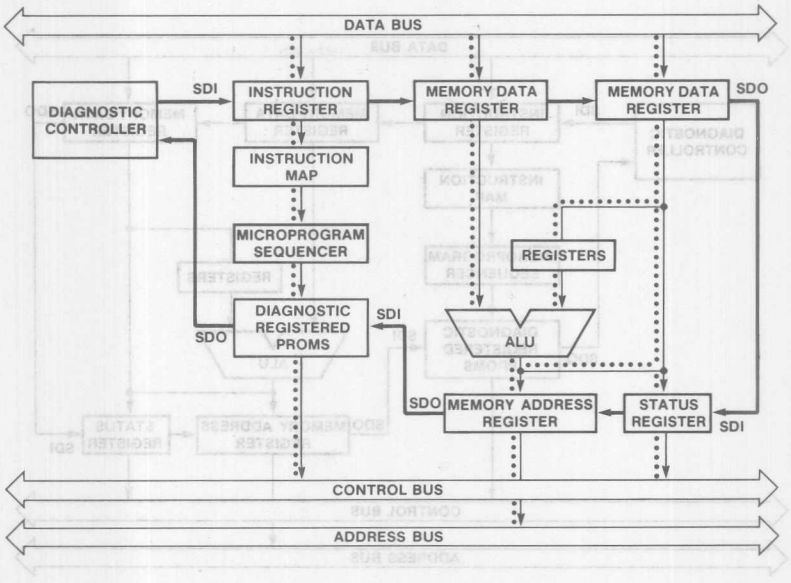


Figure 9c. Shifting on of Test Vector. Dotted Lines Represent Data Flow in the CPU if the Loading of Test Vector is Hidden in Normal CPU Operations

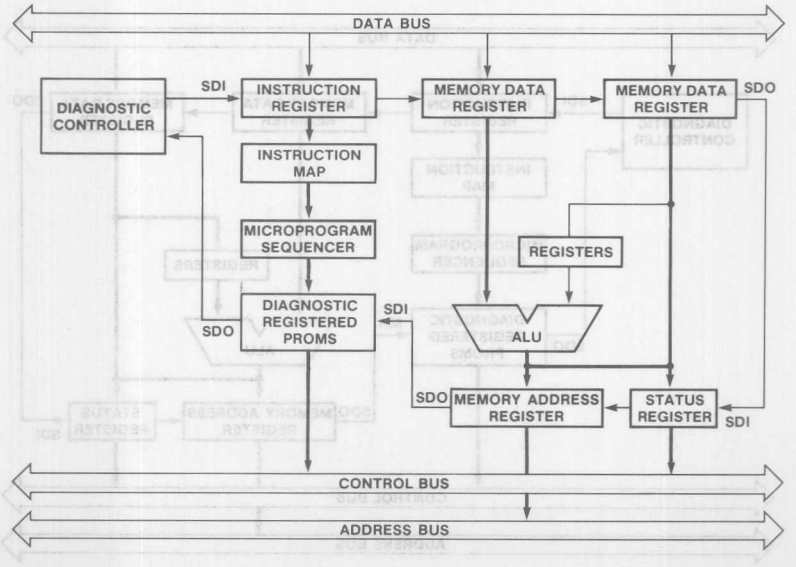


Figure 9d. After the Last Bit of the Test Data is Shifted In, the Output Registers Will be Loaded With the Test Vector Which Will be Used to Test the System (Control the System)

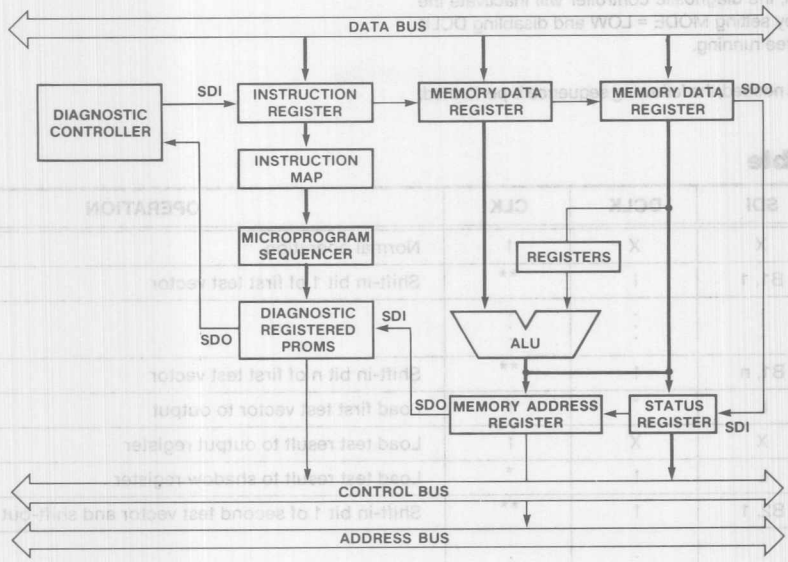


Figure 9e. The Test Result is Then Loaded Back Into the Output Register (Observe the System)

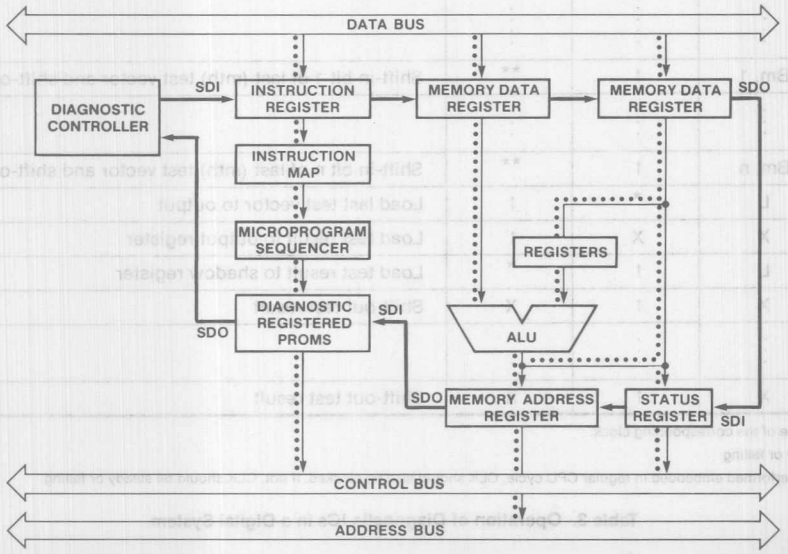


Figure 9f. The Test Result is Shifted Out at the Same Time the Next Test Vector is Shifted In. Again Dotted Lines Represent Data Flow in the CPU if the Loading of Test Vector is Hidden in Normal CPU Operations

In normal operation, the diagnostic controller will inactivate the diagnostic feature by setting MODE = LOW and disabling DCLK and have the CLK free running.

When diagnostics is needed, the following sequence is performed:

Function Table

MODE	SDI	DCLK	CLK	OPERATION
L	X	X	↑	Normal operation
L	B1, 1	↑	**	Shift-in bit 1 of first test vector
⋮	⋮	⋮	⋮	
L	B1, n	↑	**	Shift-in bit n of first test vector
H	L	*	↑	Load first test vector to output
L	X	X	↑	Load test result to output register
H	L	↑	*	Load test result to shadow register
L	B2, 1	↑	**	Shift-in bit 1 of second test vector and shift-out test result
⋮	⋮	⋮	⋮	
L	B2, n	↑	**	Shift-in bit n of second test vector and shift-out test result
H	L	*	↑	Load second test vector to output
L	X	X	↑	Load test result to output register
H	L	↑	*	Load test result to shadow register
⋮	⋮	⋮	⋮	
L	Bm, 1	↑	**	Shift-in bit 1 of last (mth) test vector and shift-out test result
⋮	⋮	⋮	⋮	
L	Bm, n	↑	**	Shift-in bit n of last (mth) test vector and shift-out test result
H	L	*	↑	Load last test vector to output
L	X	X	↑	Load test result to output register
H	L	↑	*	Load test result to shadow register
L	X	↑	X	Shift-out test result
⋮	⋮	⋮	⋮	
L	X	↑	X	Shift-out test result

↑ Indicates a rising edge of the corresponding clock.

* Clock must be steady or falling

** If diagnosis is to be performed embedded in regular CPU cycle, CLK should also be clocked. If not, CLK should be steady or falling.

Table 3. Operation of Diagnostic ICs in a Digital System

A block diagram of a simple diagnostic controller is shown in Figure 10. The central control unit of this controller may be a disk-based unit or even another PROM. Since in normal operation MODE remains LOW and only CLK is active, it is possible to include a switch in Figure 10 so that the diagnostic controller will be inactive (see Figure 11).

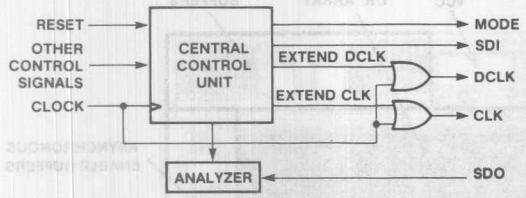


Figure 10. A Block Diagram of a Diagnostic Controller

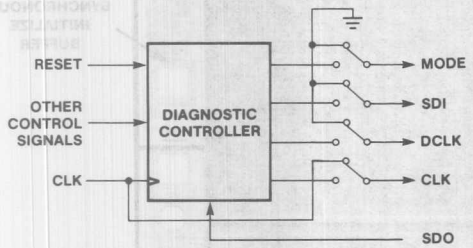


Figure 11. Including a Switch to Disconnect the Diagnostic Controller from the CPU

Some Final Thoughts

More complicated systems may have co-processors, DMA, I/O ports, etc., in addition to the CPU. A top-down approach will be very efficient in testing such systems by first locating the defective board, followed by the locating of the defective part in that board.

The diagnostic PROMs and registers can also be used in minicomputers, data storage devices, and peripherals.

Besides being used for diagnostics, the serial shifting feature present in a diagnostic component can also be applied to serial character generators, other serial to parallel and parallel to serial converters, serial code generators, etc.

References

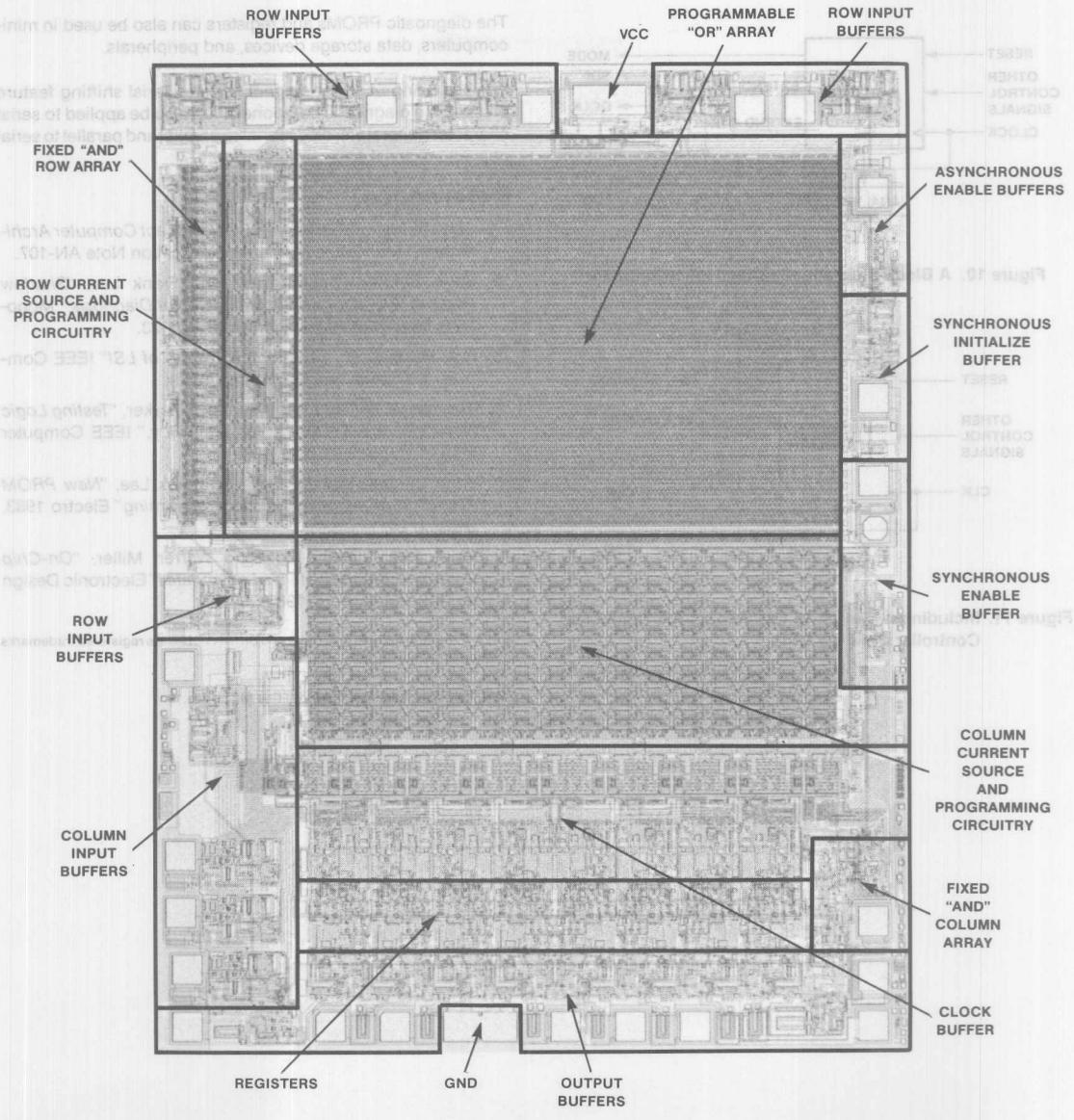
1. John Birkner, *"Registered PROMs Impact Computer Architecture,"* Monolithic Memories Application Note AN-107.
2. John Birkner, Vincent Coli and Frank Lee, *"Shadow Register Architecture Simplifies Digital Diagnosis,"* Monolithic Memories Application Note AN-123.
3. R.A. Rasmussen, *"Automated Testing of LSI"* IEEE Computer Magazine, pp. 69-78 (Mar. 1982).
4. Thomas W. Williams and Kenneth P. Parker, *"Testing Logic Networks and Designing for Testability,"* IEEE Computer Magazine, pp. 9-21 (Oct. 1979).
5. John Birkner, Vincent Coli and Frank Lee, *"New PROM Architecture Simplifies Microprogramming"* Electro 1983, Session 24.
6. Frank Lee, Vincent Coli and Warren Miller, *"On-Chip Circuitry Reveals System's Logic States"* Electronic Design pp. 119-124 (Apr. 14, 1983).

PAL® (Programmable Array Logic) and SKINNYDIP® are registered trademarks of Monolithic Memories

DPROM™ is a trademark of Monolithic Memories.

9

1Kx8 Registered PROM Metalization



A Compiler for Programmable Logic in FORTH*

Michael Stolowitz

This paper describes the implementation of a very compact compiler, written in the FORTH language, which is used as a design aid in the generation of digital systems using *P*rogrammable *A*rray *L*ogic elements (PAL®) devices. The compiler produces a fuse map for generation of a target part from an algebraic description of the input/output relationships. Many of the techniques used are applicable to other areas.

* This paper is a slightly modified version of the paper by the same name which appeared in the 1982 FORML Conference Proceedings, pages 257-265, 6-8 October 1982; available from the FORTH Interest Group, P.O. Box 1105, San Carlos, CA 94070. (FORML stands for *FORTH Modification Laboratory.*)

PAL® is a registered trademark of Monolithic Memories.

A Compiler for Programmable Logic in FORTH

Michael Stolowitz

Background

The schematic diagram of a typical PAL device is shown in Figure 1. Note that this part contains multiple logic device elements and outputs from gates or flip-flops. Each intersection in the matrix on the left represents a connection made by a fuse. These fuses may be blown (one time only) to customize the part for a particular application. The digital-system designer writes a set of Boolean equations for the outputs as functions of the inputs. This is most conveniently done using signal names from the application as shown in Figure 2. These expressions must be translated into a hexadecimal data file for transmission to a "programmer" test instrument, capable of blowing the undesired fuses to produce the target part.

It is not impossible to encode this data manually, but the process is tedious and provides considerable opportunity for error. A FORTRAN program (PALASM™) is available, but FORTRAN is not usually available on microprocessor-based development systems, and the used of remote systems is not convenient. For the above application and for the reasons stated, the program PAL device was implemented in FORTH.

Implementation

The PALs device program uses the same concept as many of the FORTH-based assemblers: namely, an environment is created which allows the source data to be *interpreted* directly. As in the case of the assemblers, this approach led to a surprisingly short program.

In a FORTH-style assembler, a single pass is made of the source data. The interpreter must be able to process each element of the input stream as it is encountered. In an assembler, each of the operation-code mnemonics is an executable word, whose function is to generate a particular machine-language instruction. In PALs, when the logical expressions are interpreted, each of the signal names is an *executable FORTH word.* The function of an output signal name is to make the row of fuses associated with its first term current, and to set all of the fuses in this row to be blown. The function of an input signal name is to cause the fuse to be spared which is at the intersection of the current row and the column associated with the named input. The output of the PALs program is the *fuse map,* which is created in its final format by the interpretation of the logical expressions. The fuse map for the application of Figure 2 is shown in Figure 3.

All of the signal names must be defined *before* the logical expressions are encountered by the interpreter. Since the rows and columns of the matrix are associated with specific pins of the PAL device, the signal names are created by the declarations which associate signal names with pins. When the declarations are interpreted, the pin numbers (i.e., the *words* 1.-20. and not the integers 1-20) have already been defined. Pin numbers are *defining words* whose function it is to create signal names.

The various types of PAL logic circuits differ in number of inputs and outputs. For this reason, the various PAL type are *executable words.* The function of a PAL type is to cause the appropriate set of definitions for pin numbers to be placed in the dictionary. For this reason, the PAL type is executed before any of the pin/signal declarations.

The only word which precedes the PAL type in the input is the word PALs. While this happens to be the name of the program, it is also an executable word whose function it is to make PAL the current and context vocabulary. A PAL logic circuit contains definitions for all of the PAL type, and for Boolean operators in the FORTH vocabulary.

Details

The 16R4 was selected as an example (Figure 1) because it contains most of the features found in the PAL device family, including: input signals which are available in both true and inverted forms, gate outputs which may have individual three-state controls, and clocked outputs. As shown in the example, the output signals are connected to columns in the fuse matrix, and may therefore be used as inputs. Many of these features require some consideration by the program.

Negation — Inputs may be used in expressions in either their true or complemented forms. The operator "/" sets a flag called INVERT, which causes the column number of the next input executed to be incremented. The column number of the complemented signal is always one greater than the column number of the corresponding true signal.

Logical "OR" — Execution of the Boolean "+" increments the current row number, selecting the next term of the current output.

Feedback — The use of internal feedback means that an output signal name may appear as a factor in a term, i.e., outputs may sometimes be inputs depending on the context. The question of which function an output signal name is to perform is resolved with a flag called IN/OUT, which indicates which type of signal is expected. IN/OUT is set by outputs, and cleared by inputs. The Boolean "*" is used to set the flag again between consecutive inputs.

Ouptut Inversion — PALS requires that all expressions be written as the sum of a group of terms. For logical consistency, the designer must indicate an inverted output by defining the expression for the *complement* of the output, which he indicates with a "/" (the inversion operator) in front of the output signal name.

PAL® is a registered trademark of Monolithic Memories.
PALASM™ is a trademark of Monolithic Memories.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2376 TWX: 910-338-2374
Monolithic Memories

Logical vs. Electrical Expression — The use of a negation operator as described above results in electrical expressions rather than logical expressions. The operator indicates that the input must be LOW, and ignores the fact that the signal might be assertive-HIGH or assertive-LOW. A naming convention was developed which allows the expressions to be interpreted logically. The convention is to begin the names of all assertive-LOW signals with the symbol: "/". When reading expressions, the "/" is pronounced "NOT" whether it is part of the name or an independent operator. Double negatives are not pronounced.*

The Code — The complete code for PALS may be found in Figures 4 and 5. The compiler itself is on four FORTH screens. The two additional screens show examples of PAL device-type definitions for six of the most common types. The six screens are usually printed on a single page.

The FORTRAN program PALASM™ (1981 version) appears on pages 3-58 through 3-63 of the second edition of the *PAL® Programmable Array Logic Handbook*. Subsequent updates of the source code have not been published, but are available (subject to a charge and an agreement) from the Monolithic Memories Inc. IdeaLogic Exchange.

A partial *glossary* of the PALS code follows:

MAP is the structure in which the output is generated.

PWR2 is used to make a bit mask.

ADDR converts column and row to byte and bit within the map.

FUSE is used to toggle a bit in the map.

(INPUT) is what input signals do.

INPUT creates pins which create input signals.

NEXT-TERM is used by output signals and "+" to select a row, and to clear its fuses.

OUTPUT creates pins which create output signals.

PAL. prints the symbolic map as in Figure 3.

PAL-TYPE is the defining word for PAL types.

* Editor's note: in this respect, Mike Stolowitz's program allows a syntax convention (double negative) which isn't supported by PALASM.

Summary

One of the objectives of the PALS compiler was that the source data and documentation for the resultant part should be one and the same. This insures that the documentation is always up to date, since it must have been created first. It also eliminates the possibility of documentation errors occurring when documentation is produced after-the-fact by an independent process.

This application program is in use by people who have little or no knowledge of FORTH. A FORTH system with the compiler has been modified to load and run as a CP/M .COM file. The program accepts the name of a text file, which it interprets a line at a time.

The techniques described above have also been adapted to the compilation of other forms of programmable logic, including Field-Programmable Logic Arrays (FPLAs) and Field-Programmable Logic Sequencers (FPLSs).

Why is the PALs program so simple in FORTH? In FORTH, it is rarely necessary to write code for the entire application. It is only necessary to extend what already exists to include the application.

The author, Michael Stolowitz, is a digital hardware/software systems consultant specializing is personal computer systems and peripheral devices. Since this paper was published, he has considerably extended the features of the PALS program from what is described here; in particular, PALS can now produce JEDEC-format files. Questions may be addressed to Mr. Stolowitz at (415) 837-3887, 335 Merrilee Place, Danville, CA 94526.

FORTH Self-Study References

1. Leo Brodie, *Starting FORTH,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981; ISBN 0-13-842930-8 (hardcover), 0-13-842922-7 (paperback).
2. Anita Anderson and Martin Tracy, *Mastering FORTH,* Brady Communications Company Inc., Bowie, MD, 1984; ISBN 0-89303-660-9.
3. Leo Brodie, *Thinking FORTH,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984; ISBN 0-13-917576-8 (hardcover), 0-13-917568-7 (paperback).

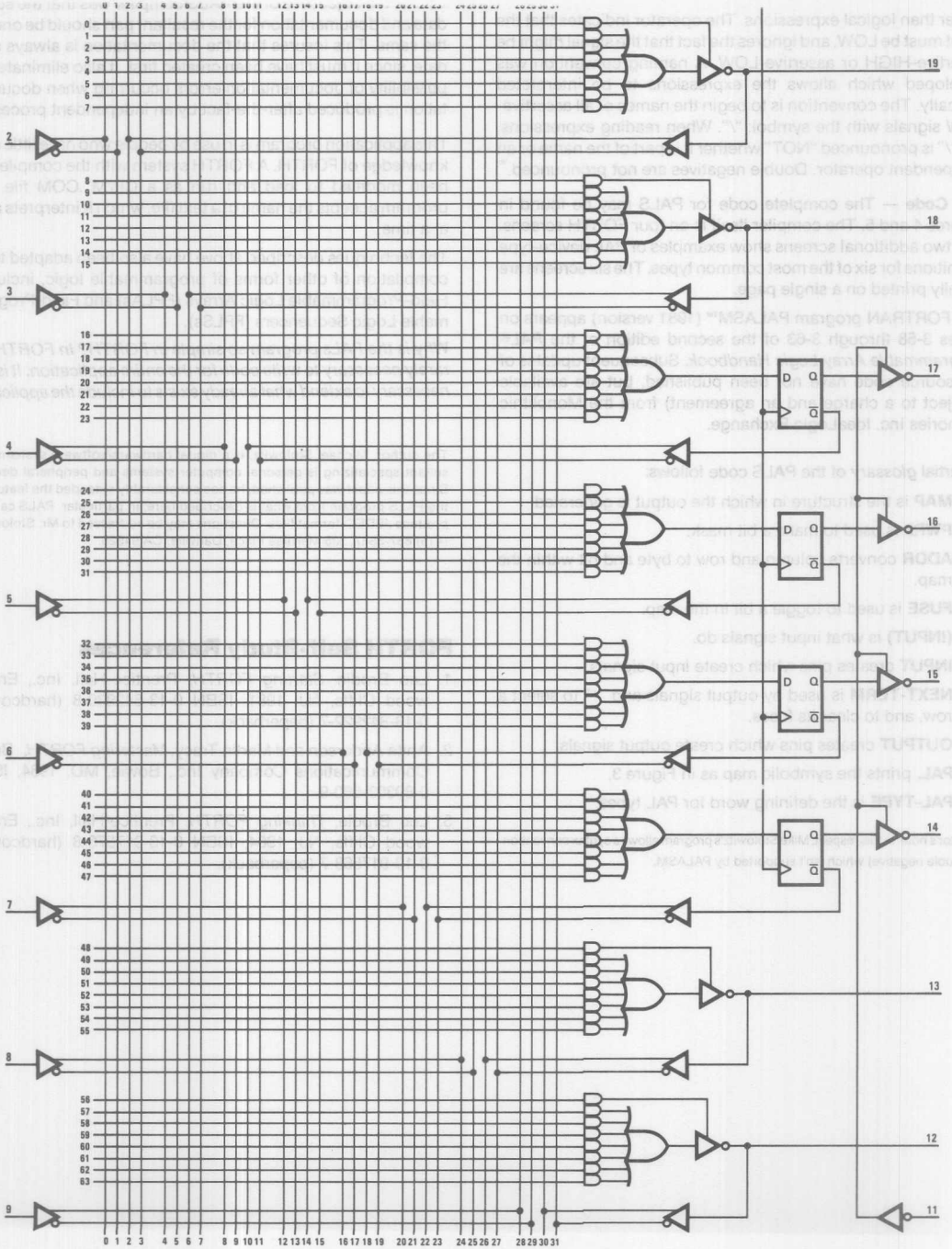


Figure 1. Logic Diagram, PAL16R4

```
\ PROJECT        FORML 1982
\ FUNCTION       EXAMPLE PAL
\ BY             MICHAEL STOLOWITZ
\ FILE           EXAMPLE.TXT
\ DISK           FORML
\ REVISION       10/04/82

PALS   16R4

1.    PHI                        11. /OE
2.    PHI                        12.  AØ
3.   /SET-TWO                    13.  SYNC-INH
4.    CDSB                       14. /LATCH-DATA
5.    WAIT                       15.  STB-ENA
6.   /AS                         16. /TWO-CYC
7.   /MAS                        17. /WAIT
8.   /UDS                        18.  SYNC
9.   /LDS                        19. /DTACK-INH
10.   GND                        20.  VCC

/ SYNC           =   / CDSB  ( TRI-STATE ENABLE )
                 +    /MAS
                 +    /UDS       *    /LDS
                 +   / SYNC      *   / SYNC-INH  *   / PHI

/ SYNC-INH       =     HI  ( TRI-STATE ENABLE )
                 +    /AS
                 +   / SYNC-INH  *   / SYNC
                 +   / /TWO-CYC  *  / STB-ENA  *  / PHI

/ /TWO-CYC       =     SYNC       *  / /SET-TWO  *  / /AS
                 +  / /TWO-CYC    *     STB-ENA  *  / /AS

/ STB-ENA        =   / SYNC  *  / STB-ENA
                 +   / SYNC  *  /WAIT

/ /WAIT          =     WAIT

/ /LATCH-DATA    =  / /TWO-CYC      *   /WAIT  *   / /AS
                 +  / /LATCH-DATA  *  / /AS

/ /DTACK-INH     =     HI  ( TRI-STATE ENABLE )
                 +    /UDS  *  /LDS
                 +  / /SET-TWO
                 +  / /TWO-CYC

/ AØ             =     HI  ( TRI-STATE ENABLE )
                 +    / SYNC-INH  *  / /UDS  *  /LATCH-DATA
                 +      SYNC      *  / /UDS  *  /LATCH-DATA


PAL.
```

Figure 2. Example of Source Data for PAL16R4 Application

9

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

 8   ---- ---- -X-- ---- ---- ---- ---- ----
 9   ---- ---- ---- ---- ---- X--- ---- ----
10   ---- ---- ---- ---- ---- ---- X--- X---
11   -X-- ---X ---- ---- ---- ---- ---X ----
12   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
13   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
14   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
15   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

16   ---- ---- ---- X--- ---- ---- ---- ----
17   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
18   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
19   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
20   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
21   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
22   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
23   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

24   ---- -XX- ---- ---- -X-- ---- ---- ----
25   ---- ---- ---- ---X -XX- ---- ---- ----
26   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
27   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
28   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
29   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
30   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
31   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

32   ---- ---X ---- ---- ---X ---- ---- ----
33   ---- ---X --X- ---- ---- ---- ---- ----
34   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
35   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
36   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
37   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

40   ---- ---- --X- ---X -X-- ---- ---- ----
41   ---- ---- ---- ---- -X-- ---X ---- ----
42   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
43   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
44   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
45   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
46   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
47   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

Figure 3. Fuse Map for PAL16R4 Application

```
SCR #99      (63H)
  Ø \ fuse addr pwr2 map in/out invert last-out row h/l   15Sep82mcs
  1
  2 CREATE MAP   512 ALLOT
  3 VARIABLE IN/OUT                  VARIABLE INVERT
  4 VARIABLE LAST-OUT                VARIABLE H/L
  5 VARIABLE ROW
  6
  7 : PWR2 (S n -- 2**n ) 1 SWAP ?DUP IF Ø DO DUP + LOOP THEN ;
  8
  9 : ADDR ( col -- addr bit ) ( addr = r5 r2 r1 rØ c4 c3 c2 c1 cØ )
 10    ROW @  8 /MOD  SWAP  32 *  ROT +  SWAP   ( bit = r4 r3 )
 11    4 /MOD  256 *  ROT +  MAP +  SWAP  PWR2 ;
 12
 13 : FUSE (S col -- )
 14    INVERT @ +   Ø INVERT !   ADDR TOGGLE ;
 15


SCR #1ØØ     (64H)
  Ø \ (input) input next-term                          15Sep82mcs
  1
  2 : (INPUT) (S 'col -- )
  3    @  FUSE  Ø IN/OUT ! ;
  4
  5 : INPUT (S col -- )  ( pin's dot name    )
  6       CREATE  ,  DOES>
  7         (S     -- )  ( pin's signal name )
  8     @  CREATE  ,  DOES>
  9    IN/OUT @  Ø=  ABORT" Output Required!"  (INPUT) ;
 10
 11 : NEXT-TERM (S -- )
 12    LAST-OUT @  DUP @  Ø=  ABORT" No More Terms!"
 13    -1 OVER +!   2+  DUP  @  ROW !  1 SWAP +!
 14    32 Ø DO  I FUSE  LOOP  1 IN/OUT ! ;
 15


SCR #1Ø1     (65H)
  Ø \ output twk pal.                                  15Sep82mcs
  1 : OUTPUT (S row #rows col -- )  ( pin's dot name    )
  2    CREATE , , ,  DOES>
  3       (S          -- )  ( pin's signal name )
  4    CREATE ,  DOES>
  5    @  IN/OUT @  IF   ( input expected  )
  6    DUP @  31 >  ABORT" No Internal Feedback!"  (INPUT)
  7                 ELSE ( output expected )
  8    INVERT @  H/L @  +  1 -  ABORT" Invert Output Equation!"
  9      2+ LAST-OUT !  Ø INVERT !  NEXT-TERM  THEN ;
 10 : TWK ( col -- )    ROW @  SWAP ADDR  OVER C@  OR  SWAP C! ;
 11 : PAL. CR  64 Ø DO  I ROW !  I 3 .R  2 SPACES
 12    32 Ø DO  I  3 AND  Ø=  IF SPACE THEN
 13      I ADDR  SWAP C@ AND  IF ." -" ELSE ." X" THEN  LOOP
 14    CR  I  8 MOD  7 =  IF CR THEN  ?KEY IF LEAVE THEN  LOOP ;
 15
```

Figure 4. The PALS FORTH Program (Screens 1-3)

```
SCR #102      (66H)
 0 \ pal-type pal pals + / * = HI NC                          30Sep82mcs
 1 : PAL-TYPE (S +scr h/l )
 2    CREATE , , DOES> 2@ H/L !  [ BLK @ ] LITERAL + LOAD
 3    MAP 512 ERASE  0 INVERT ! ;
 4 : SKIP ( -- ) BL WORD DROP ;
 5 VOCABULARY PAL   IMMEDIATE   PAL DEFINITIONS
 6 : NC       (S -- ) BL WORD DROP ;
 7 : +        (S -- ) NEXT-TERM ;
 8 : /        (S -- ) 1 INVERT ! ;
 9 : *        (S -- ) IN/OUT @ ABORT" Input Expected!" 1 IN/OUT ! ;
10 : HI       (S -- ) 0 IN/OUT ! ;
11 : =        (S -- ) ;
12 1 0 PAL-TYPE 10L8   1 1 PAL-TYPE 10H8  2 0 PAL-TYPE 16R4
13 2 0 PAL-TYPE 16R6   2 0 PAL-TYPE 16R8  2 0 PAL-TYPE 16L8
14 FORTH DEFINITIONS
15 : PALS     (S -- ) [COMPILE] PAL  DEFINITIONS ;


SCR #103      (67H)
 0 \ 10L8 10H8                                                30Sep82mcs
 1  2 INPUT  1.       0 INPUT  2.       4 INPUT  3.       8 INPUT  4.
 2 12 INPUT  5.      16 INPUT  6.      20 INPUT  7.      24 INPUT  8.
 3 28 INPUT  9.     : 10. SKIP ;      30 INPUT 11.    : 20. SKIP ;
 4 56 2 BL OUTPUT 12.              48 2 BL OUTPUT 13.
 5 40 2 BL OUTPUT 14.              32 2 BL OUTPUT 15.
 6 24 2 BL OUTPUT 16.              16 2 BL OUTPUT 17.
 7  8 2 BL OUTPUT 18.               0 2 BL OUTPUT 19.
 8 : TWEEK (S -- )
 9    FORTH  8 0 DO 2 0 DO J 8 * I + ROW !
10    6 TWK 7 TWK 11 TWK 14 TWK 15 TWK 18 TWK
11    19 TWK 22 TWK 23 TWK 26 TWK 27 TWK LOOP LOOP PAL ;
12
13
14
15


SCR #104      (68H)
 0 \ 16R4 16R6 16R8 16L8                                      30Sep82mcs
 1
 2 : 1. SKIP ;       0 INPUT  2.       4 INPUT  3.       8 INPUT  4.
 3 12 INPUT  5.     16 INPUT  6.      20 INPUT  7.      24 INPUT  8.
 4 28 INPUT  9.     : 10. SKIP ;      : 11. SKIP ;    : 20. SKIP ;
 5
 6 56 8 30 OUTPUT 12.              48 8 26 OUTPUT 13.
 7 40 8 22 OUTPUT 14.              32 2 18 OUTPUT 15.
 8 24 8 14 OUTPUT 16.              16 8 10 OUTPUT 17.
 9  8 8  6 OUTPUT 18.               0 8  2 OUTPUT 19.
10
11 : TWEEK (S -- ) ;
12
13
14
15
```

Figure 5. The PALS FORTH Program (Screen 4) and Definitions (Screens 5-6)

High-Speed Bipolar PROMs Find New Applications As Programmable Logic Elements*

Vincent J. Coli and Frank Lee

Classic applications for bipolar PROMs include instruction storage for microprogram control store and software for microprocessor programs. However, due to a new design methodology and state-of-the-art performance, PROMs are finding increasing numbers of applications as Programmable Logic Element (or PLE) devices. This paper will cover the architecture, applications, and software support for PLE devices.

9

* This paper is a slightly modified version of the paper by the same name which appeared in the Conference Proceedings of the 9th West Coast Computer Faire, pages 40-47, April 1984.

Vincent J. Coli and Frank Lee

Classic applications for bipolar PROMs include instruction storage for microprogram control store and software for microprocessor programs. However, due to a new design methodology and state-of-the-art performance, PROMs are finding increasing numbers of applications as Programmable Logic Element (or PLE) devices. This paper will cover the architecture, applications, and software support for PLE devices.

Fuse-Programmable Logic Families

A typical combinatorial Boolean equation can be written in sum-of-product form, which consists of several AND gates summed at an OR gate. In general, a set of combinatorial Boolean equations with n inputs (I0, I1, . . . , In–1) and m outputs (O0, O1, . . . , Om–1) can be generated through one level of AND gates followed by one level of OR gates. Custom logic functions can be defined using programmable logic.

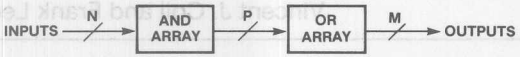


Figure 1. Structure of Programmable Logic Devices

Fuse-programmable devices normally consist of two levels of logic — AND-array and OR-array — as suggested above. There are three basic types of fuse-programmable devices — PROM (Programmable Read Only Memory), PLA (Programmable Logic Array), and PAL® (Programmable Array Logic) devices. Which arrays are fuse-programmable distinguish these three types of devices.

PLAs offer the greatest flexibility since both the AND and OR arrays are programmable. This flexibility comes with the cost of lower performance, higher power dissipation, and generally higher price.

A PAL device has only the AND-array programmable; the OR-array is fixed. Each output has an OR gate associated with it which sums a fixed number of product terms (AND combinations). Statistically there is only a limited number of product terms in any equation. So the flexibility of a PLA is normally not needed. This is a compromise between flexibility and cost and performance.

The OR-array is programmable in a PROM, but the fixed AND-array consists of all combinations of literals for each of the input variables. For example, there are 32 product terms available in a PROM with 5 inputs a,b,c,d,e (corresponding to words 0 through 31 in the PROM memory):

```
/a*/b*/c*/d*/e     (Word  0)
/a*/b*/c*/d* e     (Word  1)
/a*/b*/c* d*/e     (Word  2)
      .                .
      .                .
      .                .
a* b* c*/d* e      (Word 29)
a* b* c* d*/e      (Word 30)
a* b* c* d* e      (Word 31)
```

where '*' represents the Boolean AND operator and '/' represents the Boolean NOT or inverter operator. The fuses in the OR-array are programmed to select the desired AND combinations.

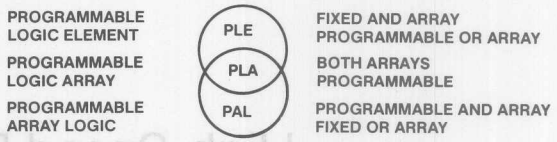


Figure 2. Structural Difference Between PLE (PROM), PLA and PAL Devices. Note that the PAL and PLE Logic Circuits Complement Each Other. The PAL Device has Many Input Terms While the PLE Device is Rich in Product Terms

The existence of all combinations of literals for all inputs makes it possible to define functions which cannot be implemented in a PLA or a PAL device. For example, a 5-input Exclusive-OR (XOR) function can be implemented using sixteen product terms. This may exceed the number of product terms available in a PAL device and will consume too many product terms in a PLA, but can be constructed quite efficiently in a PROM. It is important to realize that any combination of inputs can be decoded in a PROM as long as sufficient input pins are provided since a PROM provides 2^n product terms (where n is the number of inputs). Another way of looking at this is that PROMs store the logic transfer function in a memory. The fixed AND-array (or AND-plane) consists fo the row and column decoders while the fuses in the OR-array (or OR-plane) are the bits in the memory. In a memory, a fuse blown versus a fuse intact distinguishes a HIGH from a LOW.

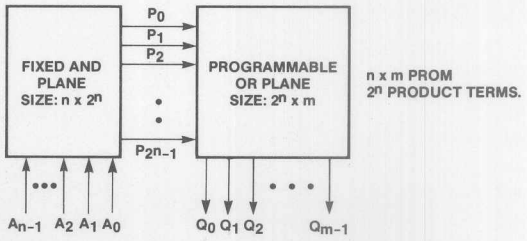


Figure 3. Block Diagram of a PROM Viewed as PLE Device. Notice that the PLE Provides Many (2^n, Where n is the Number of Inputs) Product Terms. A By-Product of this is Programmable Output Polarity: Either Active-High or Active-Low Output Polarities are Available

Due to this special characteristic of abundant product terms, PROMs are also often used as logic devices. In this paper, PROMs are referred to as PLE (Programmable Logic Element) devices.

Advantages of PLE Devices

PLE devices provide a cost-effective solution for many applications. Here are just some of the advantages of PLE devices:

1) Customizable Logic — The designeer is limited to standard functions if SSI/MSI devices are used. The designer can create his own logic chips using PLE devices.

2) Design Flexibility — Modification of design is possible even without redesigning the PC board. For example, the address space of a microprocessor-based system can be reconfigured by merely programming a new device if the decoding is implemented in a PLE device. This feature comes in handy if you want to upgrade a system which originally used 64-K Dynamic RAMs to now use state-of-the-art 256-K Dynamic RAMs.

3) Reduce Errors — Errors are sometimes unavoidable and oftentimes quite expensive. Programmable devices make it easier and less expensive to correct errors.

4) Reduction of Printed Circuit Board Space—PLE devices save PC board space since several SSI/MSI functions can be integrated into a single package.

5) Fast Turnaround Time — With existing commercial programmers and development software support, a prototype of the custom tailored PLE device will be ready in just a few minutes.

A Great Performer!

PLE Applications

PLE applications include random logic replacement, decoder/encoders, code converters, custom ALUs, error detection and correction, look-up tables (both trigonometric and arithmetic), data scaling, compression arithmetic like Wallace Tree adders, distributed arithmetic, and residue arithmetic.

Several levels of random logic chips can be replaced by one PLE logic circuit. As discussed earlier, PLE devices can implement logic in sum of products form.

Despite the existence of dedicated encoders and decoders, many of these functions are application dependent. A standard 3-to-8 decoder/demultiplexer (74S138) can be used in decoding applications. But the decoding scheme may require several 3-to-8 decoder/ demultiplexers and additional SSI OR-gates. On the other hand, a PLE device can be customized to perform the required decoding function with no additional gates. Simple decoders, such as those used for decoding memory chip selects from address lines, can be implemented in a PLE device with five to ten inputs. More complex decoding may require eight to twelve inputs.

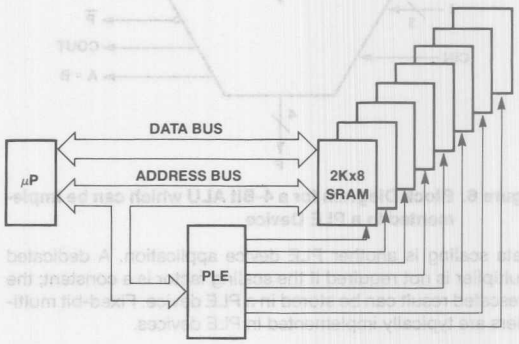


Figure 4. PLE Address Decoding Application. The PLE Device Selects One of Eight 2Kx8 Static RAMs by Decoding Several Microprocessor Address Lines

PLE devices also offer a very flexible solution for code conversion applications. Translations of codes such as from ASCII to EBCIDIC, Binary to BCD (Binary Coded Decimal), or BCD to Gray code can be implemented in PLEs. The 74S184 Binary-to-BCD Converter is actually a 32x8 PROM.

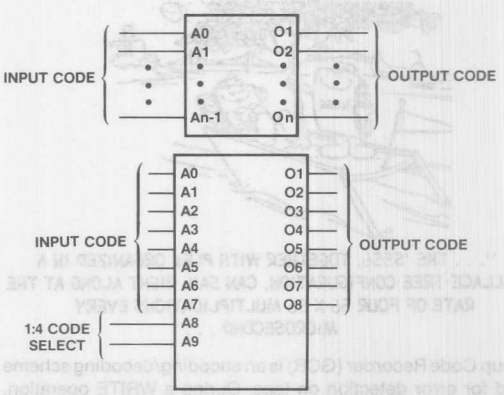


Figure 5. Two Examples of PLE Code Converters. The Second Example Illustrates How to Use Two Inputs as Code Select Lines so that Four Converters can be Provided in One PLE Device

9

Standard ALUs (such as the 74S181) may not provide a very specailized function which a particular system requires, such as BCD arithmetic. In this case a PLE device is again a good alternative. Although the PLE device may be slower than a dedicated ALU, the presence of this specialized function is critical. For example, a 4-bit ALU can be constructed in a PLE device with twelve inputs (A3-A0, B3-B0, I2-I0, Cin) and eight outputs (F3-F0,/G,/P, Cout, A = B). Any eight functions can be implemented.

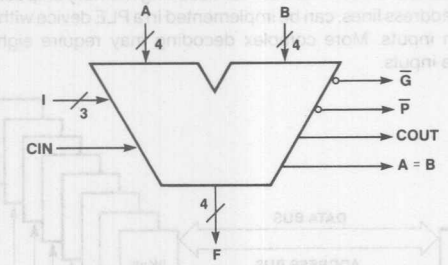


Figure 6. Block Diagram for a 4-Bit ALU which can be Implemented in a PLE Device

Data scaling is another PLE device application. A dedicated multiplier is not required if the scaling factor is a constant; the prescaled result can be stored in a PLE device. Fixed-bit multipliers are typically implemented in PLE devices.

Column compression technique (also called Wallace Tree Compression) is used when expanding an array of several smaller parallel multipliers to perform large wordlength multiplication. These smaller multipliers will generate partial products (intermediate results) which must be numerically summed according to bit significance in order to calculate the final wordlength multiplication. Many levels of 2-input bus adders can be used to add these partial products, but the carry propagation delays may be too long. However, partial product adders implemented in PLE devices can do compression of many levels without passing carries. Thus, the summmation will be much faster.

"... THE 'S556, TOGETHER WITH PLEs ORGANIZED IN A WALLACE-TREE CONFIGURATION, CAN SAIL RIGHT ALONG AT THE RATE OF FOUR 56 X 56 MULTIPLICATIONS EVERY MICROSECOND ..."

Group Code Recorder (GCR) is an encoding/decoding scheme used for error detection on tape. During a WRITE operation, each 8-bit word is divided into two 4-bit nibbles. Both nibbles are then encoded into 5-bit codes before being recorded onto tape. Both 5-bit codes are decoded back to the original 4-bit nibbles and then combined during a READ operation. PLE circuits are exceptionally useful in mapping the 4-bit data to the 5-bit code and back.

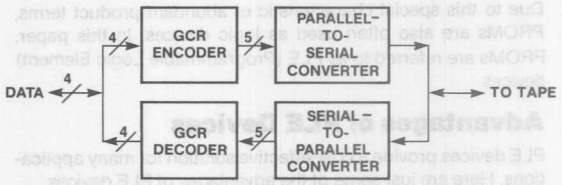


Figure 7. GCR Encoder/Decoder Block Diagram

Exclusive-OR gates, being half adders, are very prevalent in Error Detection and Correction (EDC) schemes. Many SSI chips are required to implement this function while PLA and PAL devices may not provide sufficient product terms. PLE devices are again an ideal solution.

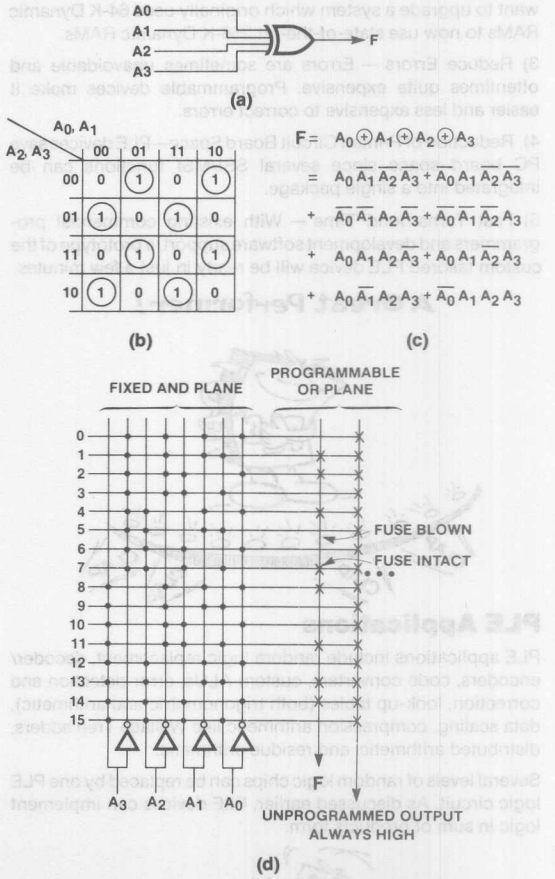


Figure 8. Exclusive-OR Gates can be Implemented in PLE Very Efficiently. A 4-Input XOR Gate (a) Maps into a Checkerboard Pattern in a Karnaugh Map (b) and Requires Eight Product Terms (c). The PLE Implementation is Shown in (d). An 8-input XOR Gate Requires Sixteen Product Terms

In many applications, the speed of the converging series used to generate the trigonometric functions is too slow and the accuracy obtained by direct table look-up requires too much hardware. A good compromise between speed and hardware is to store an approximation to the function in a PLE device. Then use this approximation as a starting point for an iterative algorithm (such as Newton-Raphson) to obtain additional accuracy. High-Speed division, multiplication, and square-root calculations can be performed in a similar manner.

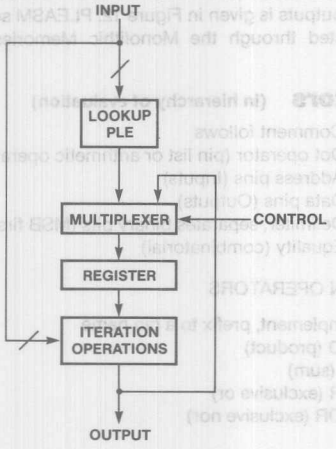


Figure 9. PLE Look-Up Tables and Iteration Loops can be Used to Generate Very Accurate Trigonometric and Arithmetic Functions. An Approximation to the Function is Stored and Additional Accuracy is Gained Using Iteration Operations

Distributed arithmetic is used for performing convolution operations without using multiplier/accumulators. If the coefficients are constant, a look-up table for convolution can be stored in a PLE device, thus replacing the multiplier.

Residue arithmetic (also called Carry-Independant arithmetic) is a technique used to perform very fast integer arithmetic. High speed is achieved by using numbers in residue representation so that the sequential delay of carries on digits of higher significance is eliminated. A Residue Numbering System (RNS) is determined using an optimum moduli when designing the system. Conversion to and from residue representation are basic mapping functions which can be conveniently done in a PLE device. Also, since operations in residue arithmetic are performed using modulo addition and multiplication without carries, these operations can also be done using PLE devices. In general, residue arithmetic should only be used for integer arithmetic which requires intensive operations.

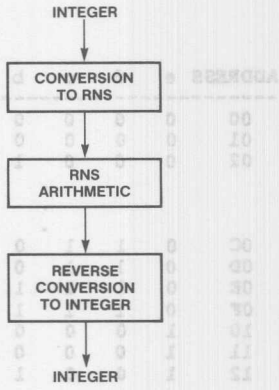


Figure 10. Architecture of a System Based on RNS. An Integer Number is Converted to RNS Representation Using PLE Devices, Then the RNS Arithmetic is Performed Using Some Other PLE Devices, and Finally the RNS Result is Converted Back to Integer Representation Again Using PLE Devices

Restrictions

The basic restrictions for using PLE devices to replace SSI/MSI parts are:

1) Since a memory element has a product term for every combination of literals of all the input terms, static hazard is normally unavoidable. For example, there are 5 inputs available in a 32 x 8 PROM. In order to generate a function like:

$$f = a^* b^* c^* d$$

The actual implementation inside the PROM will be:

$$f = a^* b^* c^* d^*/e + a^* b^* c^* d^* e$$

If a = b = c = d = HIGH, according to the first equation, we shall expect f to remain HIGH independent of e changing. In the actual PROM implementaton, there will be no hazard if e stays either HIGH or LOW. But if e changes, depending on whether e or /e will occur first, there exists the possibility that both product terms in the second equation will be LOW momentarily, which may cause a static logic hazard (HIGH to LOW to HIGH) for f. This hazard is commonly called a "glitch". Static hazards are not a problem for many applications, like those offered in this paper, but extreme care must be taken if the output of a PLE device is used to strobe another device.

ADDRESS	e	d	c	b	a	f
00	0	0	0	0	0	0
01	0	0	0	0	1	0
02	0	0	0	1	0	0
.						
.						
.						
0C	0	1	1	0	0	0
0D	0	1	1	0	1	0
0E	0	1	1	1	0	0
0F	0	1	1	1	1	1
10	1	0	0	0	0	0
11	1	0	0	0	1	0
12	1	0	0	1	0	0
.						
.						
.						
1D	1	1	1	0	1	0
1E	1	1	1	1	0	0
1F	1	1	1	1	1	1

Figure 11. This Truth Table Graphically Illustrates the Possible Glitch (HIGH to LOW to HIGH Hazard) for the Function f = a* b* c* d Implemented in a 32x8 PROM. Address 0F and 1F Contain a 1 while All Other Locations Contain a 0 for Output f. If Address Input e Should Change, the PROM Decoders Could Momentarily Selct a Location Containing a 0

2) Although PROMs are available with registered outputs, internal feedback from the outputs and buried registers are not yet available in PROMs. External connections from some outputs to inputs must be made for applications which require feedback (such as in state machines). However, Registered PROMs without feedback are useful for pipelining (overlap instruction fetch and execution) in order to increase system throughput.

PLEASM Software Support

Monolithic Memories has developed a software tool to assist in designing and programming PROMs as PLE devices. This package, called "PLEASM" (PLE Assembler), is available for several computers including the VAX/VMS and IBM PC/DOS. PLEASM converts design equations (Boolean and arithmetic) into truth tables and formats compatible with PROM programmers. A simulator is also provided to test a design using a Function Table before actually programming the PLE device. The PLEASM operators are listed below and the PLEASM catalog of operations is given on the next page. A sample PLE Design Specification (source code for PLEASM software) with PLEASM outputs is given in Figure 12. PLEASM software may be requested through the Monolithic Memories IdeaLogic Exchange.

Operators (in hierarchy of evaluation)

;	Comment follows
.	Dot operator (pin list or arithmetic operator follows)
ADD	Address pins (Inputs)
DAT	Data pins (Outputs)
,	Delimiter, separates binary bits (MSB first)
=	Equality (combinatorial)

BOOLEAN OPERATORS

/	Complement, prefix to a pin name
*	AND (product)
+	OR (sum)
:+:	XOR (exclusive or)
:*:	XNOR (exclusive nor)

ARITHMETIC OPERATORS

.*.	Multiply (numeric multiplication)
.+.	Plus (numeric addition)

Monolithic Memories PLEASM version 1.2D © copyright 1984 Monolithic Memories

PLEASM — PLE Assembler — provides the following options:

C Catalog	— Prints the PLEASM catalog of operations
E Echo Input	— Prints the PLE design specifications
T Truth Table	— Prints the entire truth table
B Brief Table	— Prints only used addresses in the truth table
H Hex Table	— Prints the truth table in HEX form
S Simulate	— Exercises the function table in the logic equations
I Intel Hex	— Generates INTEL HEX programming format
A ASCII Hex	— Generates ASCII HEX programming format
Q Quit	— Exits PLEASM

```
PLE5P8                                          PLE DESIGN SPECIFICATION
P5000                                                 VINCENT COLI 10/03/82
BASIC GATES
MMI SANTA CLARA, CALIFORNIA
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8

O1 =  I0                                              ; BUFFER

O2 = /I0                                              ; INVERTER

O3 =  I0  *  I1  *  I2  *  I3  *  I4                  ; AND GATE

O4 =  I0  +  I1  +  I2  +  I3  +  I4                  ; OR GATE

O5 = /I0  + /I1  + /I2  + /I3  + /I4                  ; NAND GATE

O6 = /I0  * /I1  * /I2  * /I3  * /I4                  ; NOR GATE

O7 =  I0 :+: I1 :+: I2 :+: I3 :+: I4                  ; EXCLUSIVE OR GATE

O8 =  I0 :*: I1 :*: I2 :*: I3 :*: I4                  ; EXCLUSIVE NOR GATE


FUNCTION TABLE

I0 I1 I2 I3 I4 O1 O2 O3 O4 O5 O6 O7 O8

;INPUT    -  -   OUTPUTS  FROM  BASIC  GATES  -  -
;01234   BUF  INV  AND   OR  NAND  NOR  XOR  XNOR   COMMENTS
-------------------------------------------------------------------
 LLLLL    L    H    L    L    H     H    L    L     ALL ZEROS
 HHHHH    H    L    H    H    L     L    H    H     ALL ONES
 HLHLH    H    L    L    H    H     L    H    H     ODD CHECKERBOARD
 LHLHL    L    H    L    H    H     L    L    L     EVEN CHECKERBOARD
-------------------------------------------------------------------


DESCRIPTION

THIS EXAMPLE ILLUSTRATES THE USE OF PLEs TO IMPLEMENT THE BASIC GATES:
BUFFER, INVERTER, AND GATE, OR GATE, NAND GATE, NOR GATE, EXCLUSIVE OR
GATE, AND EXCLUSIVE NOR GATE.

NOTE ALSO THAT THREE-STATE OUTPUTS ARE PROVIDED WITH ONE ACTIVE LOW
OUTPUT ENABLE CONTROL (/E).

PLEASM GENERATES THE PROM TRUTH TABLE FROM THE LOGIC EQUATIONS AND
SIMULATES THE FUNCTION TABLE IN THE LOGIC EQUATIONS.
```

Figure 12a. PLE Design Specification. This is the Source Code for PLEASM Software. PLEASM Software Generates the Truth Table and Programming Formats from the Equations. PLEASM Software Also Exercises the Function Table in the Equation and Reports Errors

```
BASIC GATES
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8

 ADD    A0 A1 A2 A3 A4   O1 O2 O3 O4 O5 O6 O7 O8
-------------------------------------------------
   0    L  L  L  L  L    L  H  L  L  H  H  L  L
   1    H  L  L  L  L    H  L  L  H  H  L  H  H
   2    L  H  L  L  L    L  H  L  H  H  L  H  H
   3    H  H  L  L  L    H  L  L  H  H  L  L  L
   4    L  L  H  L  L    L  H  L  H  H  L  H  H
   5    H  L  H  L  L    H  L  L  H  H  L  L  L
   6    L  H  H  L  L    L  H  L  H  H  L  L  L
   7    H  H  H  L  L    H  L  L  H  H  L  H  H
   8    L  L  L  H  L    L  H  L  H  H  L  H  H
   9    H  L  L  H  L    H  L  L  H  H  L  L  L
  10    L  H  L  H  L    L  H  L  H  H  L  L  L
  11    H  H  L  H  L    H  L  L  H  H  L  H  H
  12    L  L  H  H  L    L  H  L  H  H  L  L  L
  13    H  L  H  H  L    H  L  L  H  H  L  H  H
  14    L  H  H  H  L    L  H  L  H  H  L  H  H
  15    H  H  H  H  L    H  L  L  H  H  L  L  L
  16    L  L  L  L  H    L  H  L  H  H  L  H  H
  17    H  L  L  L  H    H  L  L  H  H  L  L  L
  18    L  H  L  L  H    L  H  L  H  H  L  L  L
  19    H  H  L  L  H    H  L  L  H  H  L  H  H
  20    L  L  H  L  H    L  H  L  H  H  L  L  L
  21    H  L  H  L  H    H  L  L  H  H  L  H  H
  22    L  H  H  L  H    L  H  L  H  H  L  H  H
  23    H  H  H  L  H    H  L  L  H  H  L  L  L
  24    L  L  L  H  H    L  H  L  H  H  L  L  L
  25    H  L  L  H  H    H  L  L  H  H  L  H  H
  26    L  H  L  H  H    L  H  L  H  H  L  H  H
  27    H  H  L  H  H    H  L  L  H  H  L  L  L
  28    L  L  H  H  H    L  H  L  H  H  L  H  H
  29    H  L  H  H  H    H  L  L  H  H  L  L  L
  30    L  H  H  H  H    L  H  L  H  H  L  L  L
  31    H  H  H  H  H    H  L  H  H  L  L  H  H
-------------------------------------------------

HEX CHECK SUM = 00F3C
```

Figure 12b. Truth Table. PLEASM Software Generates This Truth Table which can be Used for Verifying Your Design

```
BASIC GATES
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8

  ADD      HEX ADDRESS      HEX DATA
-----------------------------------
    0          000             32
    1          001             D9
    2          002             DA
    3          003             19
    4          004             DA
    5          005             19
    6          006             1A
    7          007             D9
    8          008             DA
    9          009             19
   10          00A             1A
   11          00B             D9
   12          00C             1A
   13          00D             D9
   14          00E             DA
   15          00F             19
   16          010             DA
   17          011             19
   18          012             1A
   19          013             D9
   20          014             1A
   21          015             D9
   22          016             DA
   23          017             19
   24          018             1A
   25          019             D9
   26          01A             DA
   27          01B             19
   28          01C             DA
   29          01D             19
   30          01E             1A
   31          01F             CD
-----------------------------------

HEX CHECK SUM = 00F3C
```

Figure 12c. Hex Table. PLEASM Software Generates This Truth Table in Hexadecimal Form for Verification of Locations in the PLE

```
32 D9 DA 19 DA 19 1A D9 DA 19 1A D9 1A D9 DA 19 .
DA 19 1A D9 1A D9 DA 19 1A D9 DA 19 DA 19 1A CD .

00F3C
```

Figure 12d. ASCII Hex Programming Format. PLEASM Software Generates this ASCII Hex Programming Format with Hex Check Sum. Control Characters are Included so that the Information can be Down-Loaded Directly to a PROM Programmer

```
:1000000032D9DA19DA191AD9DA191AD91AD9DA1940
:10001000DA191AD91AD9DA191AD9DA19DA191ACD54
:00000001FF
```

Figure 12e. Intel Hex Programming Format. PLEASM Software Generates this Intel Hex Programming Format with a Hex Check Sum Following Every 16 Bytes of Data

9

PLE Family

Monolithic Memories carries a family of fast PROMs which can be used as Memory or PLE devices. Since the critical parameter for logic applications is speed, our series of fast PROMs have worst-case memory access times (or propagation delays) ranging from 15 ns for small PROMs to 40 ns for large PROMs. The Logic Symbols for four of the PLE devices are given in Figure 13 and a summary of the PLE family is given below:

PLE Selection Guide

PART NUMBER	INPUTS	OUTPUTS	PRODUCT TERMS	OUTPUT REGISTERS	t_{PD} (ns) MAX*
PLE5P8	5	8	32		25
PLE5P8A	5	8	32		15
PLE8P4	8	4	256		30
PLE8P8	8	8	256		28
PLE9P4	9	4	512		35
PLE9P8	9	8	512		30
PLE10P4	10	4	1024		35
PLE10P8	10	8	1024		35†
PLE11P4	11	4	2048		35
PLE11P8	11	8	2048		35
PLE12P4	12	4	4096		35
PLE12P8	12	8	4096		40
PLE9R8	9	8	512	8	15
PLE10R8	10	8	1024	8	15
PLE11RA8	11	8	2048	8	15
PLE11RS8	11	8	2048	8	15

*Clock to output time for registered outputs

† Preliminary data.

NOTE: Commercial limits specified.

Acknowledgements

Several of the designs discussed in this paper were proposed by our good friend and colleague Ulrik Mueller, who is now studying Computer Science in his native country, Denmark, and our Monolithic Memories Pal Zahir Ebrahim. Special thanks also go to Ranjit Padmanabhan for writing the PLEASM simulator.

Summary

There are many interesting applications for high-speed PROMs used as PLE devices. A software package called "PLEASM" software is available as a development tool.

References

1. "PAL Programmable Array Logic Handbook", 3rd edition, J. Birkner, V. Coli, Monolithic Memories, Inc.
2. "Systems Design Handbook", Monolithic Memories, Inc.
3. "Bipolar LSI 1984 Databook", 5th edition, Monolithic Memories, Inc.
4. "PROMs and PLEs: An Application Perspective", Z. Ebrahim, Monolithic Memories Application Note AN-126.
5. "An Introduction to Arithmetic for Digital Designers", S. Waser, M.J. Flynn, Holt, Rinehart & Winston, N.Y., 1982.

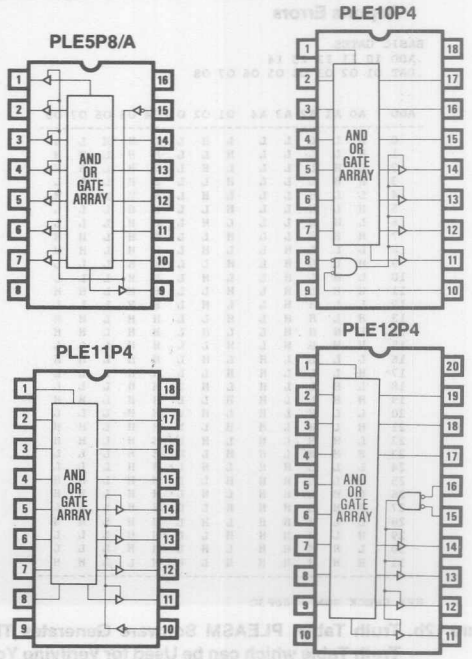


Figure 13. Four Sample PLE Logic Symbols

ABEL™, A Complete Design Tool For Programmable Logic

Michael J. Holley
DATA I/O
10525 Willows Rd. N.E.
Redmond, WA 98073-9746

As the use of PAL® and PLE devices (PROMs) increases, the need for high-level design tools becomes necessary. Designers need easier, faster, and more efficient ways to design with such programmable devices. With the more complex devices currently being introduced to the market, this need is even greater. Additionally, a designer should be able to specify logic designs in a way that makes sense in engineering terms; he or she should not have to learn a new way of thinking about designs.

ABEL™, a complete logic design tool for PAL devices, PLE devices and FPLA devices meets these requirements. ABEL™ incorporates a high-level design language and a set of software programs that process logic designs to give correct and efficient designs.

The ABEL™ design language offers structures familiar to designers: state diagrams, truth tables, and Boolean equations. The designer can choose any of these structures or combine them to describe a design. Macros and directives are also available to simplify complex designs.

The ABEL™ software programs process designs described with the high-level language. Processing includes syntax checking, automatic logic reduction, automatic design simulation, verification that a given design can be implemented in a chosen device, and automatic generation of design documentation.

To use ABEL™, the designer uses an editor to create a source file containing an ABEL™ design description. He then processes the source file with the ABEL™ software programs to produce a programmer load file. The programmer load file is used by logic and PLE programmers to program devices. Several programmer load file formats are supported by ABEL™ so that different programmers may be used.

The source file created by the designer must contain test vectors if simulation is to be performed. Test vectors describe the desired (expected) input-to-output function of the design in a truth table format. The ABEL™ simulator applies the inputs contained in the test vectors to the design and checks the obtained outputs against the expected outputs in the vectors. If the outputs obtained during simulation do not match those specified in the test vectors, an error is reported.

Following are two designs described in the ABEL™ design language. These designs would be processed to verify their correctness and to reduce the number of terms required to implement them. The first design is for a PAL device, the second for a PLE logic circuit.

ABEL™ is a trademark of DATA I/O.

6809 MEMORY ADDRESS DECODER

Address decoding is a typical application of programmable logic devices, and the following describes the ABEL™ implementation of such a design.

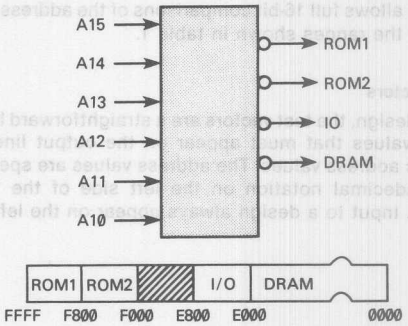


Figure 1. Block Diagram: 6809 Memory Address Decoder

Design Specification

Figure 1 shows a block diagram for the design and a continuous block of memory divided into sections containing dynamic RAM (*DRAM*), I/O (*IO*), and two sections of ROM (*ROM1* and *ROM2*). The purpose of this decoder is to monitor the six high-order bits (*A15-A10*) of a sixteen-bit address bus and select the correct section of memory based on the value of these address bits. To perform this function, a simple decoder with six inputs and four outputs is designed with a 14L4 PAL device.

Table 1 shows the address ranges associated with each section of memory. These address ranges can also be seen in figure 1.

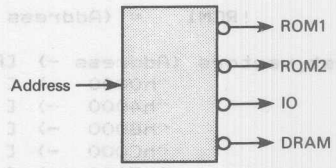


Figure 2. Simplified Block Diagram: 6809 Memory Address Decoder

Design Method

Figure 2 shows a simplified block diagram for the address decoder. The address decoder is implemented with simple

PAL® is a registered trademark of Monolithic Memories.

simplification is achieved by grouping the address bits into a set named *Address*. The lower-order ten address bits that are not used for the address decode are given "don't care" values in the address set. In this way, the designer indicates that the address in the overall design (that beyond the decoder) contains sixteen bits, but that bits 0-9 do not affect the decode of that address. This is opposed to simply defining the set as, *Address = [A15,A14,A13,A12,A11,A10]*, which ignores the existence of the lower-order bits. Specifying all 16 address lines as members of the address set also allows full 16-bit comparisons of the address value against the ranges shown in table 1.

DRAM	0000–DFFF
I/O	E000–E7FF
ROM2	F000–F7FF
ROM1	F800–FFFF

Table 1. Address Ranges for 6809 Controller

Test Vectors

In this design, the test vectors are a straightforward listing of the values that must appear on the output lines for specific address values. The address values are specified in hexadecimal notation on the left side of the "->"" symbol. Input to a design always appear on the left side of the test vectors. The expected outputs are specified to the right of the "->" symbol. The designer chose in this case to use the symbols *H* and *L* instead of the binary values 1 and 0 to describe the outputs. The correspondence between the symbols and the binary values was defined in the constant declaration section of the source file, just above the section labeled *equations*.

```
module m6809a
title '6809 memory decode
Jean Designer     Data I/O Corp Redmond WA    24 Feb 1984'

                          U09      device   'P14L4';
A15, A14, A13, A12, A11, A10  pin 1,2,3,4,5,6;
ROM1, IO, ROM2, DRAM          pin 14,15,16,17;

H, L, X       = 1, 0, .X.;
Address  = [A15, A14, A13, A12,  A11, A10, X, X,  X, X, X, X,  X, X, X, X];

equations
        !DRAM    = (Address <= ^hDFFF);

        !IO      = (Address >= ^hE000) & (Address <= ^hE7FF);

        !ROM2    = (Address >= ^hF000) & (Address <= ^hF7FF);

        !ROM1    = (Address >= ^hF800);

test_vectors (Address -> [ROM1, ROM2, IO, DRAM])
              ^h0000  -> [  H,   H,   H,    L ];
              ^h4000  -> [  H,   H,   H,    L ];
              ^h8000  -> [  H,   H,   H,    L ];
              ^hC000  -> [  H,   H,   H,    L ];
              ^hE000  -> [  H,   H,   L,    H ];
              ^hE800  -> [  H,   H,   H,    H ];
              ^hF000  -> [  H,   L,   H,    H ];
              ^hF800  -> [  L,   H,   H,    H ];
end m6809a
```

Figure 3. Source File: 6809 Memory Address Decoder

Seven-Segment Display Decoder

This display decoder decodes a four-bit binary number to display the decimal equivalent on a seven-segment LED display. The design incorporates the ABEL™ truth table format and is implemented on a RA5P8 PLE.

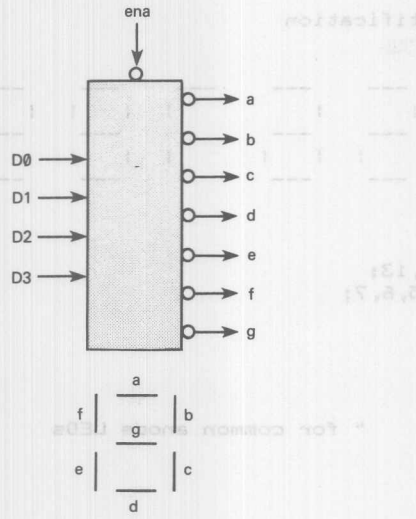


Figure 4. Block Diagram: Seven-Segment Display Decoder

Design Specification

Figure 4 shows a block diagram for the decoder design and a drawing of the display with each of the seven segments labeled to correspond to the decoder outputs. To light up any one of the segments, the corresponding line must be driven low. Four input lines *D0-D3* are decoded to drive the correct output lines. The outputs are named *a, b, c, d, e, f,* and *g* corresponding to the display segments. All outputs are active low. An enable, *ena*, is provided. When *ena* is low, the decoder is enabled; when *ena* is high, all outputs are driven to high impedance.

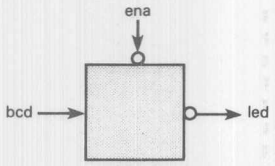


Figure 5. Simplified Block Diagram: Seven-Segment Display Decoder

Design Method

Figures 5 and 6 show the simplified block diagram and the source file for the ABEL™ implementation of the display decoder. The FLAG statement is used to make sure that the programmer load file is in the Motorola Exorciser format. The binary inputs and the decoded outputs are grouped into the sets *bcd* and *led* to simplify notation. The constants *ON* and *OFF* are declared so that the design can be described in terms of turning a segment on or off. To turn a segment on, the appropriate line must be driven low, thus *ON* is declared as 0 and *OFF* as 1.

The design is described in two sections: an equations section and a truth table section. The decoding function is described with a truth table that specifies the outputs required for each combination of inputs. The first line of the truth table (the truth table header) names the inputs and outputs. In this example, the inputs are contained in the set named *bcd* and the outputs are in *led*. The body of the truth table defines the input-to-output function. Because the design decodes a number to a seven-segment display, values for *bcd* are expressed as decimal numbers, and values for *led* are expressed with the constants *ON* and *OFF* that were defined in the declarations section of the source file. This makes the truth table easy to read and understand; the incoming value is a number and the outputs are on and off signals to the LED.

The input and output values could have just as easily been described in another form. Take for example the line in the truth table:

```
5 -> [ON,OFF,ON,ON,OFF,ON,ON]
```

This could have been written in the equivalent form:

```
[0,1,0,1] - > 36
```

In this second form, 5 was expressed as a set containing binary values, and the LED set was converted to decimal. (Remember that *ON* was defined as 0 and *OFF* was defined as 1.) Either of the two forms is valid, but the first is more appropriate for this design. The first form can be read as, "the number five turns on the first segment, turns off the second,..." whereas the second form cannot be so easily translated into terms meaningful for this design.

Test Vectors

The test vectors for this design test the decoder outputs for the ten valid combinations of input bits. The enable is also tested by setting *ena* high for the different combinations. All outputs should be at high impedance whenever *ena* is high. If they are not, an error has occurred.

Summary

Two designs described with the ABEL™ design language have been shown. The first design shows how Boolean equations with logical and relational operators are used to describe an address decoder. The second design shows how a truth table describes a seven-segment display decoder design for a PLE logic circuit. In both designs, test vectors were written to test the function of the design using ABEL™'s simulator. In addition to the Boolean equations and truth table shown in these examples, ABEL™ features a state diagram structure. The state diagram allows the designer to fully describe state machines in terms of their states and state transitions.

Regardless of the method used to describe logic, ABEL™'s automatic logic reduction and simulation ensure that the design uses as few terms as possible and that it operates as the designer intended. The end results are savings in time, devices, board space, and money.

```
module  bcd7rom flag '-d82'
title 'seven segment display decoder
Data I/O Corp   Redmond WA   16 Mar 1984'
"
"          a
"        ---        BCD-to-seven-segment decoder similar to the 7449
"     f! g !b
"        ---           segment identification
"     e! d !c
"        ---
"  ---   ---   ---   ---   ---   ---   ---
" !   ! !   ! !   ! !   ! !   ! !   ! !   !
"  ---   ---   ---   ---   ---   ---   ---
" !   ! !   ! !   ! !   ! !   ! !   ! !   !
"  ---   ---   ---   ---   ---   ---   ---

U6      device  'RA5P8';

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 -> led)
" input                 outputs
"                a     b     c     d     e     f     g
        0 -> [ ON,   ON,   ON,   ON,   ON,   ON,  OFF];
        1 -> [OFF,   ON,   ON,  OFF,  OFF,  OFF,  OFF];
        2 -> [ ON,   ON,  OFF,   ON,   ON,  OFF,   ON];
        3 -> [ ON,   ON,   ON,   ON,  OFF,  OFF,   ON];
        4 -> [OFF,   ON,   ON,  OFF,  OFF,   ON,   ON];
        5 -> [ ON,  OFF,   ON,   ON,  OFF,   ON,   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];

test_vectors ([ena,bcd]  -> led)
" inputs                  outputs
"                a     b     c     d     e     f     g
  [L,1]  -> [OFF,   ON,   ON,  OFF,  OFF,  OFF,  OFF];
  [L,2]  -> [ ON,   ON,  OFF,   ON,   ON,  OFF,   ON];
  [L,3]  -> [ ON,   ON,   ON,   ON,  OFF,  OFF,   ON];
  [L,4]  -> [OFF,   ON,   ON,  OFF,  OFF,   ON,   ON];
  [L,5]  -> [ ON,  OFF,   ON,   ON,  OFF,   ON,   ON];
  [L,6]  -> [ ON,  OFF,   ON,   ON,   ON,   ON,   ON];
  [L,7]  -> [ ON,   ON,   ON,  OFF,  OFF,  OFF,  OFF];
  [L,8]  -> [ ON,   ON,   ON,   ON,   ON,   ON,   ON];
  [L,9]  -> [ ON,   ON,   ON,   ON,  OFF,   ON,   ON];
  [L,0]  -> [ ON,   ON,   ON,   ON,   ON,   ON,  OFF];
  [H,5]  -> [  Z,    Z,    Z,    Z,    Z,    Z,    Z];
  [H,9]  -> [  Z,    Z,    Z,    Z,    Z,    Z,    Z];
  [H,0]  -> [  Z,    Z,    Z,    Z,    Z,    Z,    Z];
end     bcd7rom
```

Figure 6. Source File: Seven-Segment Display Decoder

CUPL™ The Universal Compiler For Programmable Logic

CUPL is the first software CAD tool designed especially for the support of all programmable logic devices (PLDs), including PALs and PROMs. It was developed specifically for YOU, the Hardware Design Engineer. Each feature of the CUPL language has been chosen to make using programmable logic easier and faster than conventional TTL logic design.

MAJOR FEATURES OF CUPL

1. UNIVERSAL

   a. PRODUCT SUPPORT: CUPL supports products from every manufacturer of programmable logic. With CUPL you are free to use not only PALS, but also other programmable logic devices.

   b. PALASM CONVERSIONS: CUPL has a PALASM to CUPL language translator which allows for an easy conversion from your previous PALASM designs to CUPL.

   c. LOGIC PROGRAMMER COMPATIBILITY: CUPL produces a standard JEDEC download file and is compatible with any logic programmer that uses JEDEC files.

2. HIGH LEVEL LANGUAGE

High Level Language means that the software has features that allow you to work in terms that are more like the way you think than like the final PLD programming pattern. Examples of these are:

   a. FLEXIBLE INPUT: CUPL gives the engineer complete freedom in entering logic descriptions for their design:

      - EQUATIONS
      - TRUTH TABLES
      - STATE MACHINE SYNTAX

   b. EXPRESSION SUBSTITUTION: This allows you to pick a name for an equation and then, rather than write the equation each time it is used, you need only use the name. CUPL will properly substitute the equation during the compile process.

c. SHORTHAND FEATURES: Instead of writing out fully expanded equations CUPL provides various shorthand capabilities such as:

- LIST NOTATION: Rather than [A7,A6,A5,A4,A3,A2,A1,A0]
  CUPL only requires [A7..0]

- BIT FIELDS: A group of bits may be assigned to a name, asin FIELD ADDR = [A7..0]
  Then ADDR may be used in other expressions

- RANGE FUNCTION: Rather than
```
A15 & !A14 #
A15 &  A14 & !A13 #
A15 &  A14 &  A13 & !A12
```
  CUPL only requires ADDR:[8000..EFFF]

- THE DISTRIBUTIVEPROPERTY:
  From Boolean Algebra, where A & (B # C)
  is replaced by A & B # A & C

- DeMORGAN'S THEOREM:
  From Boolean Algebra, where !(A & B)
  is replaced by !A # !B

3. SELF DOCUMENTING

CUPL provides a template file which provides a standard "fill-in-the-blanks" documentation system that is uniform among all CUPL users. Also, CUPL allows for free form comments through out your work so there can be detailed explanations included in each part of the project.

4. ERROR CHECKING

CUPL includes a comprehensive error checking capability with detailed error messages designed to lead you to the source of the problem.

5. LOGIC REDUCTION

CUPL contains the fastest and most powerful minimizer offered for Programmable Logic equation reduction. The minimizer allows the choice of various levels of minimization ranging from just fitting into the target device to the absolute minimum.

6. SIMULATION

With CSIM, the CUPL Simulator, you can simulate your logic prior to programming an actual device. Not only can this save devices but it can help in debugging a system level problem.

7. TEST VECTOR GENERATION

Once the stimulus/response function table information has been entered into the simulator, CSIM will verify the associated test vectors and append them to the JEDEC file for downloading to the logic programmer. The programmer will verify not only the fuse map, but also the functionality of the PLD, giving you added confidence in the operation of your custom part.

8. EXPANDABILITY

CUPL is designed for growth so as new PALs and other devices are introduced you will be kept current with updated device libraries and product enhancements.

DESIGN EXAMPLE USING CUPL

In the following design example, a single PAL (or PROM) is used to replace four TTL packages on the interface card for an IBM-PC computer. The Prototype I/O Channel Interface Card, as supplied by IBM, uses four SSI packages to decode the ten bit I/O address and control the direction and enable for the bus buffer on the PCB. The PAL approach conserves real estate and also adds flexibility to decode not only the pre-assigned address, but the ability to change the board address to any location in the I/O map by merely replacing the programmable device.

1. CIRCUIT OPERATION

The inputs to the decoding logic are the expansion bus addresses A0 thru A9. The logic compares the address on the expansion bus and asserts the "IO_DECODE" signal when the correct address range of 3F0-3FF is seen. In addition, the "ENABLE" signal is also asserted if either the I/O READ or I/O WRITE signals are active during this time. The READ signal, which controls the direction of the data bus buffer, is asserted whenever I/O READ is active and AEN, the DMA Address enable signal is inactive. The AEN signal is negated when the microprocessor has control of the address bus and is generating an I/O cycle.

First, all device pins are assigned in the logic description file (see figure 1) using CUPL's pin declaration statements. Note the use of indexed variables for the address bus allows a simple assignment for pins 1 thru 8. The active polarity for input and output pins are made in these declarations, so the designer need only be concerned with the logic instead of voltage levels.

The address bus is assigned a name using the FIELD statement. This lets the designer then describe the desired address range with the single equation:

```
range = ioadr:[300..31F] ;
```

instead of the difficult to understand

```
range = a9 & a8 & !a7 & !a6 & !a5 ;
```

This range expression is then used in the output equation for IO_DECODE and ENABLE. Since ENABLE may be asserted whenever IOR or IOW are true, the intermediate variable IOREQ is created to define this condition. The resultant CUPL equation for ENABLE is simply

```
enable = range & ioreq ;
```

Finally, the READ signal is created using the active IOR and the inactive AEN signals as follows:

```
read = ior & !aen ;
```

Note that for a device such as the PAL16L8 which has a fixed inverting buffer on all of its output pins, CUPL will automatically convert the logic equations when an output is desired to be active-level HI, as with the READ output above.

3. CUPL OUTPUT FILES

CUPL will create a standard JEDEC output file which is compatible with most logic programmers. A simple serial download link is all that is usually required to transfer the fuse information to the programmer. In addition, CUPL generates an extensive documentation file which assists the designer in analyzing his/her design. Figure 2 shows a small section of this file, illustrating such features as pin and variable names, product term utilization, and other information.

```
PARTNO     P90001234  ;
NAME       PCIO  ;
DATE       02/14/85 ;
REV        01 ;
DESIGNER   Kahl/Osann ;
COMPANY    Assisted Technology ;
ASSEMBLY   PC Proto Board ;
LOCATION   U2 ;

/*************************************************************************/
/* This device provides a one-chip I/O interface for an equivalent  */
/* of the IBM-PC proto board.  This logic description may be placed */
/* in either a PROM or PAL without alteration.                      */
/*************************************************************************/
/* Allowable Target Device Types : PAL--) PAL16L8, PAL16P8           */
/*                                 PROM-) PLE12P4                    */
/*************************************************************************/

/**  Inputs  **/

PIN [1..8]     = [a2..9]    ; /* CPU Address bits 0 thru 9 */
PIN 9          = aen        ; /* DMA Address Enable */
PIN 17         = !ior       ; /* I/O Read Strobe  (active LO) */
PIN 18         = !iow       ; /* I/O Write Strobe (active LO) */

/**  Outputs  **/

PIN 12         = read        ; /* Direction Control For Bus Buffer */
PIN 13         = !enable     ; /* Enable For Bus Buffer */
PIN 14         = !io_decode  ; /* Decoded I/O Strobe for On Board Use */


/** Declarations and Intermediate Variable Definitions **/

field ioadr = [a9..2] ; /* Name the I/O Address Bus "ioadr" */

ioreq = ior # iow ;     /* Define I/O Request */

range = ioadr:[300..31F] & !aen ; /* Decoded I/O Address Range and */
                                  /* not DMA cycle */

/**  Logic Equations  **/

enable = range & ioreq ;

io_decode = range ;

read = ior & !aen ;

/* Change the intermediate variable "range" for other I/O Locations */
```

Figure 1.

```
CUPL            2.02a
Device          p1618  DLIB-c-18-5
Partno          P90001234
Name            PCIO
Revision        01
Date            02/14/85
Designer        Kahl/Osann
Company         Assisted Technology
Assembly        PC Proto Board
Location        U2
================================================================================
                                Symbol Table
================================================================================

Pol   Name            Ext     Pin     Type    Used    Max
---   ----            ---     ---     ----    ----    ---

      a2                        1       V       -       -
      a3                        2       V       -       -
      a4                        3       V       -       -
      a5                        4       V       -       -
      a6                        5       V       -       -
      a7                        6       V       -       -
      a8                        7       V       -       -
      a9                        8       V       -       -
      aen                       9       V       -       -
!     enable                    13      V       2       7
!     io_decode                 14      V       1       7
      ioadr                     0       F       -       -
!     ior                       17      V       -       -
      ioreq                     0       I       2       -
!     iow                       18      V       -       -
      range                     0       I       1       -
      read                      12      V       2       7
      enable          oe        13      D       1       1
      io_decode       oe        14      D       1       1
      ior             oe        17      D       1       1
      iow             oe        18      D       1       1
      read            oe        12      D       1       1


LEGEND    D : default var    F : field      I : intermediate var
          U : undefined      V : var        X : extended var
          N : node           M : extended node
```

Figure 2.

CUPL-GTS
DRAW LOGIC SCHEMATICS FOR PAL DESIGNS!

In recent years, programs like CUPL and ABEL have become available to provide high level language support for PAL designs. These languages allow the designer to represent a PAL function in terms of high-level equations, truth tables or state machines. All of these logic description formats are non-graphical in nature and require a good working knowledge of the computer they run on.

Many hardware designers, however, are most comfortable with the traditional logic schematic and have historically had little reason to use a computer in the design process. Use of a high-level PAL design language presents most of us with a variety of simultaneous unknowns:

1. The computer and its operating system.
2. The full screen editor necessary to create the logic description file.
3. The logic compiler or assembler language syntax.
4. Boolean algebra theory.
5. PAL architectures.

Where this combination places an unnecessary burden on the designer, an alternative is now available.

CUPL-GTS is a powerful combination of hardware and software which turns an IBM-PC type computer into a programmable logic workstation which allows the user to draw logic schematics for the function of a PAL. A basic premise in creating CUPL-GTS was to provide a friendly environment where the user is isolated from the traditional keyboard as much as possible. To this end, virtually all functions can be actuated with one button by way of the mouse and a series of pop-up menus which ease the user's task. An area is provided at the top of the CUPL-GTS screen for prompting the user regarding the next operation in a command sequence. Highlighting of various elements on the screen is coordinated with these prompts to enhance their effectiveness. For the most part, the user need only utilize the conventional keyboard for defining symbolic names for wires, pins, objects, and files.

An on-screen HELP facility is provided to aid the user with CUPL-GTS commands. In addition to the basic set of object types which can be easily picked from a pop-up menu, the ability to call up macro-objects is also provided. These macro-objects have been previously drawn using CUPL-GTS and stored away on the disk under their own symbolic name.

After a logic schematic has been entered, the user may quickly check to see if the design fits in a specific PAL. This is done by selecting the "Translate to PLD" command from the main menu which automatically invokes the GTS translation programs. These programs run in an on-screen window which overlays the graphical information, providing feedback in the form of error messages displayed in this window. Following the automatic execution of these programs, the cursor is returned to the user who can then continue to work in the graphics environment without ever having fully left. In this way many errors can be quickly determined and remedied without ever having to let go of the mouse.

When the user wishes a hard copy version of a design, the print command from the main menu may be selected. This causes the GTS print program to execute in an on-screen window according to the printer configuration file (PRINTCAP) which is stored on the disk. The PRINTCAP file allows the user to configure the GTS print function for any dot matrix printer they might have.

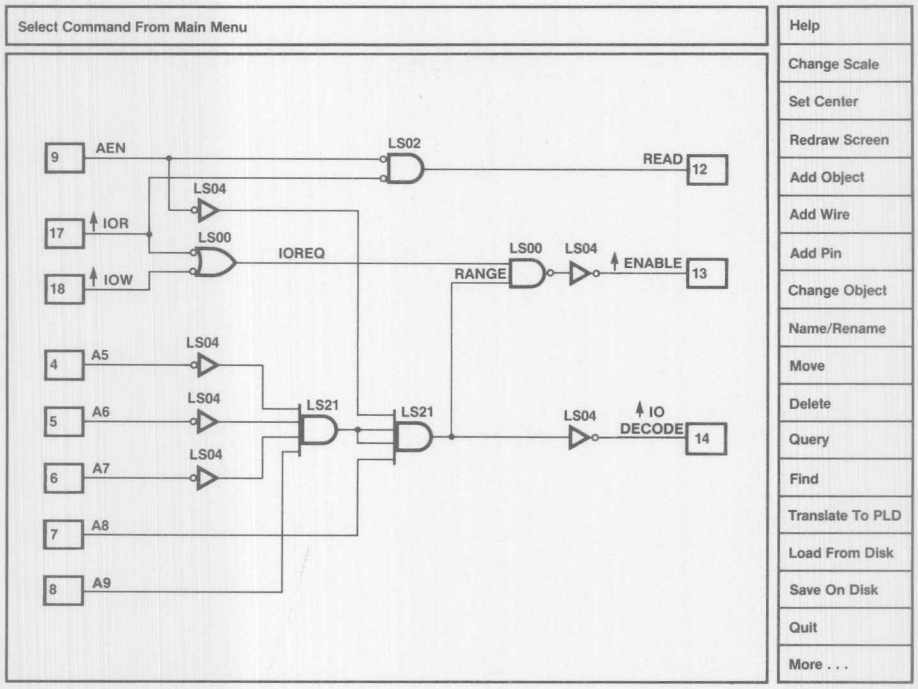
Often a logic description not fit in a particular PAL due to a logic capacity (product-term) limitation. When this occurs, the universal capability of CUPL-GTS will easily allow the user to try placing this same logic in a different PAL of similar architecture.

Since CUPL-GTS incorporates CUPL the high level language in its internal operation, it also benefits from CUPL's powerful "Quine Procedure" logic minimizer. This is especially advantageous for CUPL-GTS as logic descriptions showing many levels of gates can be very deceptive in their ability to consume the logic capacity of a PAL. The presence of the logic minimizer can eliminate unnecessary and redundant logical functions, and maximizes the probability that a design will fit in a target PAL.

Also included with CUPL-GTS is the CUPL simulator, CSIM, which allows the user to simulate a logic design prior to physically creating a programmed PAL. Not only can this save devices, but it can help significantly in debugging a system level problem.

CUPL-GTS is desinged for growth and expandability. As new programmable logic devices are introduced users will be kept current with updated device libraries and product enhancements.

Most of us first use PAL devices to replace TTL in order to shrink a design and/or add functionality. The following example shows how the simple I/O decoder design previously discussed would appear on the CUPL-GTS screen prior to translation to a PAL16L8, PAL16P8 or PLE12P4.



A CUPL-GTS Design Screen

9