Programmable Logic Design Guide

National Semiconductor Corporation

Literature Order
Number 320100-001

A Corporate Dedication to Quality and Reliability

National Semiconductor is an industry leader in the manufacture of high quality, high reliability integrated circuits. We have been the leading proponent of driving down IC defects and extending product lifetimes. From raw material through product design, manufacturing and shipping, our quality and reliability is second to none.

We are proud of our success . . . it sets a standard for others to achieve. Yet, our quest for perfection is ongoing so that you, our customer, can continue to rely on National Semiconductor Corporation to produce high quality products for your design systems.

Charlie Sporck

Charles E. Sporck
President, Chief Executive Officer
National Semiconductor Corporation

Wir fühlen uns zu Qualität und Zuverlässigkeit verpflichtet

National Semiconductor Corporation ist führend bei der Herstellung von integrierten Schaltungen hoher Qualität und hoher Zuverlässigkeit. National Semiconductor war schon immer Vorreiter, wenn es galt, die Zahl von IC Ausfällen zu verringern und die Lebensdauern von Produkten zu verbessern. Vom Rohmaterial Uber Entwurf und Herstellung bis zur Auslieferung die Qualität und die Zuverlässigkeit der Produkte von National Semiconductor sind unübertroffen.

Wir sind stolz auf unseren Erfolg, der Standards setzt, die für andere erstrebenswert sind. Auch ihre Ansprüche steigen ständig, Sie als unser Kunde können sich auch weiterhin auf National Semiconductor verlassen.

La Qualité et La Fiabilité:

Une Vocation Commune Chez National Semiconductor Corporation

National Semiconductor Corporation c'est l'un des leaders industriels qui fabrique des circuits intégrés d'une très grande qualité et d'une fiabilité exceptionelle. National a été le premier à vouloir faire chuter le nombre de circuits intégrés defectueux et a augmenter la durée de vie des produits. Depuis les matières premières, en passant par la conception du produit sa fabrication et son expédition, partout la qualité et la fiabilité chez National sont sans équivalents.

Nous sommes fiers de notre succès et le standard ainsi défini devrait devenir l'objectif à atteindre par les autres sociétés. Et nous continuons à vouloir faire progresser notre recherche de la perfection; il en résulte que vous, qui êtes notre client, pouvez toujours faire confiance à National Semiconductor Corporation, en produisànt des systèmes d'une très grande qualité standard.

Un Impegno Societario di Qualità e Affidabilità

National Semiconductor Corporation è un'industria al vertice nella costruzione di circuiti integrati di altà qualità ed affidabilità. National è stata il principale promotore per l'abbattimento della difettosità dei circuiti integrati e per l'allungamento della vita dei prodotti. Dal materiale grezzo attraverso tutte le fasi di progettazione, costruzione e spedizione, la qualità e affidabilità National non è seconda a nessuno.

Noi siamo orgogliosi del nostro successo che fissa per gli altri un traguardo da raggiungere. Il nostro desiderio di perfezione è d'altra parte illimitato e pertanto tu, nostro cliente, puoi continuare ad affidarti a National Semiconductor Corporation per la produzione dei tuoi sistemi con elevati livelli di qualità.

Charlie Sporck

Charles E. Sporck
President, Chief Executive Officer
National Semiconductor Corporation

Programmable Logic Design Guide

Bipolar Memory
National Semiconductor Corporation
Santa Clara, California

TRADEMARKS

Following is the most current list of National Semiconductor Corporation's trademarks and registered trademarks.

Abuseable™
Anadig™
ANS-R-TRAN™
Auto-Chem Deflasher™
BI-FET™
BI-FET II™
BI-LINE™
BIPLAN™
BLC™
BLX™
Brite-Lite™
BTL™
CIM™
CIMBUS™
Clock✓Chek™
COMBO™
COPS™ microcontrollers
DATACHECKER®
DENSPAK™
DIB™
Digitalker®
DISCERN™
DISTILL™
DNR™
DPVM™
ELSTAR™
E-Z-LINK™
GENIX™
HEX 3000™
INFOCHEX™
Integral ISE™
Intelisplay™
ISE™
ISE/06™
ISE/08™
ISE/16™
ISE32™
Macrobus™
Macrocomponent™
Meat✓Chek™
Microbus™ data bus (adjective)
MICRO-DAC™
µtalker™
Microtalker™
MICROWIRE™
MICROWIRE/PLUS™
MOLE™
MST™
National®
NAX 800™
Nitride Plus™
Nitride Plus Oxide™
NML™
NOBUS™
NSC800™
NSX-16™
NS-XC-16™
NURAM™
OXISS™
Perfect Watch™
Pharma✓Chek™
PLAN™
Polycraft™
POSitalker™
QUAD3000™
RAT™
RTX16™
Script✓Chek™
Shelf-Chek™
SERIES/800™
Series 32000™
SPIRE™
STAR™
Starlink™
STARPLEX™
STARPLEX II™
SuperChip™
SYS32™
TAPE-PAK™
TDS™
TeleGate™
The National Anthem®
Time✓Chek™
TLC™
Trapezoidal™
TRI-CODE™
TRI-POLY™
TRI-SAFE™
TRI-STATE®
XMOS™
XPU™
Z STAR™
883B/RETS™
883S/RETS™

PAL® is a registered trademark of Monolithic Memories, Inc.

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation 2900 Semiconductor Drive, P.O. Box 58090, Santa Clara, California 95052-8090 (408) 721-5000
TWX (910) 339-9240

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied, and National reserves the right, at any time without notice, to change said circuitry or specifications.

Preface

The CLASS™ Revolution

The nature of logic design is changing and National Semiconductor is leading this change into software-based logic and systems design through the use of structured programmable logic arrays. We welcome you to join us in the CLASS revolution. CLASS stands for Complete Logic And Software Solutions, and it exemplifies National's commitment to the design, development and support of programmable logic devices, and to the software-based design tools that can make the logic and system designer's task easier.

Table of Contents

4.0 The Programmable Logic Family

5.0 How to Design With Programmable Logic

6.0 Software Support

7.0 Testing and Reliability

8.0 Applications

List of Illustrations

Figure No. **Page No.**

Figure No. **Page No.**

Figure No. **Page No.**

Figure No. **Page No.**

Figure No. **Page No.**

List of Tables

Table No. **Page No.**

Table No. **Page No.**

Programmable Logic Design Guide

1

Introduction

1.1 PURPOSE OF THIS DESIGN GUIDE

This book was conceived to fill the need for a comprehensive Design Guide about Field-Programmable Logic Devices. The Guide is organized to serve both the experienced programmable logic user and the uninitiated. The primary objective of this guide is to introduce the uninitiated logic designer to programmable logic and to take the designer through a step-by-step approach to logic design by using programmable logic devices. The Guide is comprehensive in that it covers all aspects of design, including: Boolean logic basics, sequential and combinational circuit basics, testing, and applications. Every effort has been made to clearly illustrate points with examples. National Semiconductor invites comments and suggestions from our users on improving this Design Guide.

1.2 OVERVIEW OF PROGRAMMABLE LOGIC

Programmable Logic has been used for many years as the means of customizing logic design. The early devices were primarily mask-programmed and were developed by computer manufacturers for in-house use while the vast majority of other logic users were relegated to the world of standard SSI/MSI devices. Then, in the mid to late seventies, along came fuse-programmable logic. The logic devices could actually be customized by the designer who used external pulses generated by simple programmer equipment. Now logic designers had devices that could be customized instantly and that offered higher integration than standard logic. Field-programmable logic devices became the first, true semicustom logic that was widely available for both the small and the larger user.

Today, the user can choose from a variety of speeds, power, packages, logic features and vendors.

The logic designer's task is being simplified even further by the rapid development of software tools that actually perform some of the design tasks such as logic minimization, higher level Boolean representation, device selection, and test vector generation. The final goal is to simply specify input-output or state descriptions in a high-level language to obtain a completely programmed and functionally tested device.

Technology developments are also taking place to achieve field-programmable logic devices in low-power CMOS technology and high-speed ECL technology.

1.3 NATIONAL SEMICONDUCTOR, THE LEADER

National Semiconductor entered the field programmable logic marketplace in 1980 with the introduction of the PAL® device family. By 1984 National had taken the leadership of this market through technological advances and customer support. In particular, National is the first company to come out with the 15 ns high-performance family of PAL devices. National also has the broadest product line of programmable-logic products that will include CMOS and ECL products. National Semiconductor is committed to maintaining its leadership in this area through technological innovation, customer support and product quality.

PAL is a registered trademark of and used under license to Monolithic Memories, Inc.

2

Programmable Logic Basics

2.1 WHAT IS PROGRAMMABLE LOGIC?

Programmable logic devices are essentially uncommitted logic gates where the user determines the final logic configuration of the device. Hence, programmable logic devices are true semicustom products. A major feature of these devices is field-programmability, which offers almost instant customization. A mask-programmable option is also available for volume applications. The internal structure of these devices is a fuse-programmable interconnection of AND gates, OR gates, and Registers. These devices allow the user to design combinational as well as sequential circuits. The basic programmable array is AND-OR logic in the familiar Sum-of-Products (SOP) representation. The conventional schematic representation is shown in Figure 2.1.1.

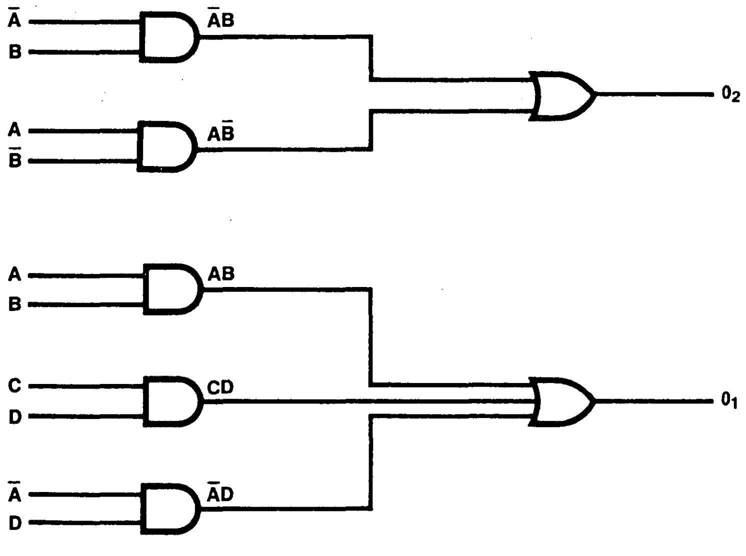


Figure 2.1.1 Conventional Representation

Its programmable logic equivalent is shown in figure 2.1.2.

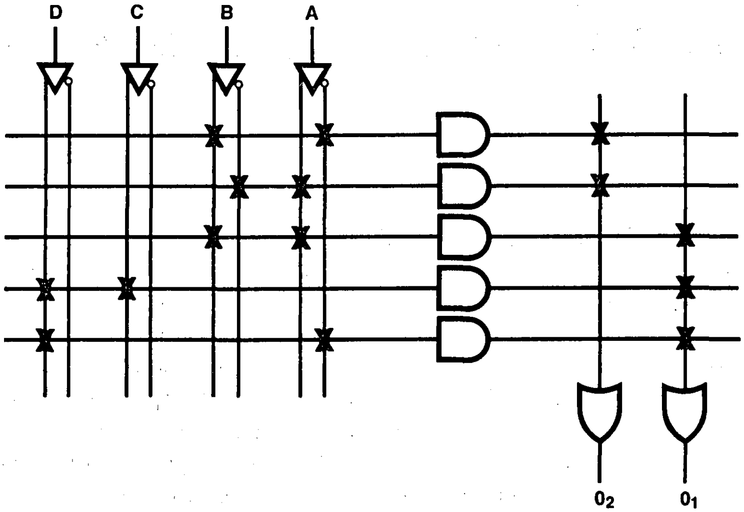


Figure 2.1.2 Programmable Logic Representation

Various programmable logic products are built around this structure by adding features and other logic elements such as programmable Active-Low or Active-High outputs, output registers, internal feedback, and state registers.

A definition of programmable logic is not complete without including software. An important part of these products is the software and design automation tools that aid systems design with programmable logic devices.

2.2 USER-BENEFITS OF PROGRAMMABLE LOGIC

The use of programmable logic devices in systems design presents the user with many benefits, some of which are obvious and some of which are not. The versatility and power of programmable logic devices can be demonstrated through the most common benefits described below.

Reduced Board Space

Today, programmable logic typically implements from 4 to 20 SSI and MSI logic devices on a single chip. PC board real estate is one of the most valuable and limited items in a system and programmable-logic devices are ideal for reducing board space. This can allow the system manufacturer to reduce the size of a system or to increase the logic power for a system of a given size.

Fast Systems Design

Fast turnaround in systems design can be achieved. Systems can be prototyped quickly by using available design automation development tools. Standard design tools reduce the need for manual design and documentation. After the first prototype has been built, modifications and correction to the logic can also be made quickly, without having to rewire or rework the PC board. The net result is that the programmable-logic user can enjoy a competitive advantage in the marketplace by bringing new products to market early.

Design Flexibility

Systems design is generally an iterative process. It starts with ideas and concepts and then progresses through an iterative series of evaluation, modification, and refinement of the original design. Numerous logic configurations have to be evaluated in this process and the painless way to perform these evaluations is through the use of programmable logic. All of the changes can be made at the CAD terminal, which will also ensure that the documentation is updated to include the changes.

With the use of programmable logic, the designer is not limited to standard off the shelf parts and, therefore, can use non-standard logic structures. The engineer now simply chooses what is needed instead of taking only what is available.

Design flexibility derived from using programmable logic means logic changes are easy in all phases of the system life cycle. For example, logic changes can be made during prototyping, during system testing, during system production, and in the field.

Many manufacturers need to be able to perform some final customization to the system. The use of programmable logic allows this customization to be implemented quickly.

Multilevel Logic Reduction

The designer can compress multiple levels of logic into a two-level AND-OR structure through the use of programmable logic, thus simplifying the design and in many cases obtaining a speed and/or power advantage. An example is shown on the following page in Figure 2.2.1.

Cost Reduction

The systems manufacturer can realize cost reduction by the use of programmable logic through a variety of factors, including:

- Lower component cost through
  - PC board area reduction.
  - Reduction in connectors used.
  - Simpler back panel.
  - Smaller power supplies.
  - Reduced cooling.

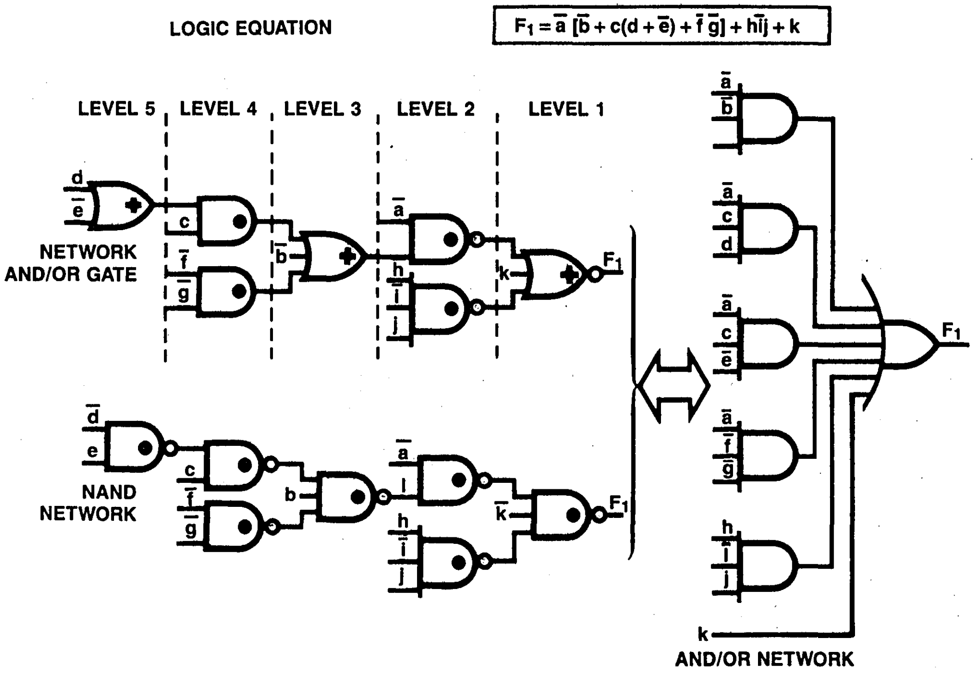


Figure 2.2.1 Multilevel Logic Reduction

- Lower design and development cost through
  - Quick-turnaround software-supported design.
  - Easy-to-make changes.
  - Computerized documentation.
  - Simplified layout.
- Lower manufacturing cost through
  - Fewer component insertions.
  - Fewer boards to manufacture.
  - Less component, board and system testing.
- Lower service costs through
  - Improved reliability.
  - Fewer spare parts.
  - Faster logic fixes.

Example to Illustrate Lower Component Costs

Table 2.2.1 is an example of the elements of component cost. The costs used are typical of those found in the industry and will have to be modified to reflect a specific situation.

Cost Variable	Cost Range $	Ave Cost $	Cost/IC $
Purchasing, Receiving, Inventory	0.01–0.03	0.02	0.02
Incoming Inspection	0–0.15	0.08	0.08
PC Board	10–100	30.00	0.30
Assembly Labor	0.10–0.40	0.20	0.20
Connectors, Wire, etc.	30–100	60.00	0.10
Power Supplies, Cooling	45–120	60.00	0.10
System Assembly	40–80	60.00	0.10
Rack, Cabinet, Panels	20–50	30.00	0.05
Total Overhead			0.95
IC Cost	0.12–2.00		0.50
Total IC Cost in System			1.45

Table 2.2.1 Typical Component Cost Structure

Assume a system with 600 SSI/MSI ICs. The total cost of the system is therefore as follows:

SSI/MSI System Cost = 600 × $1.45 = $870

PAL devices are used to replace the SSI/MSI discrete logic devices. The replacement can be accomplished at various efficiencies, where efficiency is defined as:

Efficiency = Average number of SSI/MSI devices replaced by one PAL.

If we assume that the cost of programming a PAL device is $0.40 then the total cost of a PAL based system is as follows:

PAL based system cost =

$$\frac{600}{\text{Efficiency}} \times (\text{PAL Device Price} + \text{Overhead} + \text{Programming Cost})$$

$$\frac{600}{\text{Efficiency}} \times (\text{PAL Device Price} + \$0.95 + \$0.40)$$

Various efficiencies and PAL device prices are substituted in the above equation to obtain the PAL based system costs in Table 2.2.2 below.

Efficiency Factor (EF)	SSI/MSI System Cost (1)	PAL Device System Cost (2) at a PAL Device Purchase Price of				Your SSI/MSI System Cost	Your PAL Device System Cost @ /PAL Device
		$8.00	$6.00	$4.00	$3.00		
3:1	870	1870	1470	1070	870		
4:1	870	1403	1103	803	653		
6:1	870	935	735	535	435		
8:1	870	701	551	401	325		

(1) Cost = 600 ICs x 1.45/IC = $870
(2) Cost = [600 ÷ EF] x [PAL Device price + Overhead + Programming Cost]
= [600 ÷ EF] x [PAL Device price + 0.95 + 0.40]
= [600 ÷ EF] x PAL Device price + 1.35

Table 2.2.2 System Cost Comparison Between SSI/MSI Based System and PAL Device Based System.

Most users realize at least a 4:1 ratio and at today's PAL device prices, it is clearly more economical to use PAL devices. Furthermore, as prices decline, even low efficiencies become economical.

Example of Cost Reduction Through Reliability Improvements

A simple example is used here to illustrate the power of PAL devices to improve system reliability. Assume that systems fail for only two reasons:

- External connection failures (70%)
  - Solder.
  - Connectors.
  - Back plane wiring.
- IC failures (30%)

A hypothetical system is defined as having 5 boards each with 100 SSI/MSI devices. With the following assumptions:

- System is in use for 3 years.
- Single device failure probability is 0.01% within the 3 years.
- Single device failure will cause board failure, which will result in system failure.
- 100 systems are sold.
- $1000 cost for each system failure.

The system failure probabilities and expected costs are computed below.

SSI/MSI device-related board failure probability $= 1 - (0.9999)^{100} = 0.009989$

SSI/MSI device-related system failure probability $= 1 - (0.990011)^{5} = 0.0489583$

External connection failure probability $= \dfrac{0.0489583 \times 70}{30} = 0.114236$

Total system failure probability within the three years = 0.1631943
Total Expected Cost from system failures during the three years = $1000 × 100 × 0.1631943 ≈ $16,000

The logic designer now uses PAL devices and other LSI devices to realize a 5:1 SSI/MSI chip replacement. The system will now have one board. The system failure probability and expected cost of the PAL device-based system is computed below:

Device-related board failure probability $= 1 - (0.9999)^{100} = 0.009989$

External connection failure probability $= \dfrac{0.009989 \times 70}{30}$

$= 0.023307666$

Total PAL device-based system failure probability = 0.033296666

Total Expected Cost of PAL device based system = $1000 × 100 × 0.033296666

≈ $3300

On comparing the expected costs from system failures of the SSI/MSI based system to that of the programmable-logic based system, there is approximately a 5:1 ratio of cost in favor of the programmable-logic based system.

This example is somewhat simplistic and some gross assumptions were made to illustrate the advantages of using programmable logic. In reality, the actual reliability improvement will depend on numerous factors that have not been addressed here.

Small Inventory

The programmable logic family can be used to replace up to 90% of TTL components. This allows the user to lower inventory costs considerably, in addition to simplifying the inventory system.

2.3 ELEMENTS OF PROGRAMMABLE LOGIC

The first programmable integrated circuit logic device was the diode matrix. It was introduced in the early 1960s. This approach featured rows and columns of metallization, connected at the crosspoints with diodes and aluminum fuses (Figure 2.3.1). These fuses could be selectively melted, leaving some of the crosspoints open and others connected. The result was a diode-logic OR matrix.

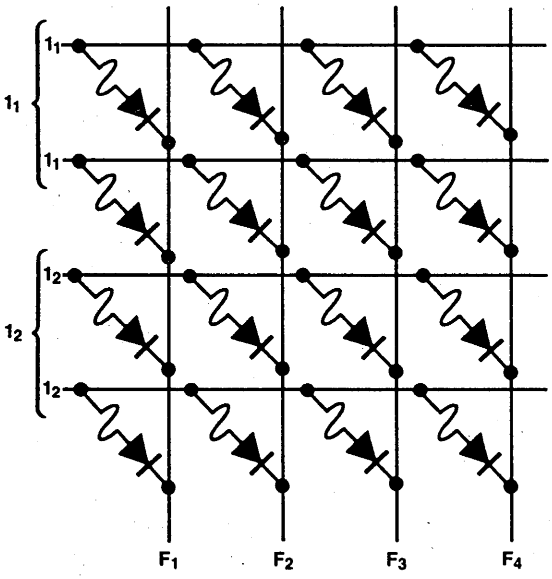


Figure 2.3.1 Diode OR Matrix

The PROM

Integrated circuit designers added input decoders and output buffers to the basic diode matrix, creating the field-programmable read-only memory (PROM) (Figure 2.3.2). This extended the programmable-logic concept considerably, since the input variables could now be encoded. It also reduced the number of pins required per input variable. At the same time, the input circuitry, along with the output buffers, provided TTL compatibility, the lack of which was one of the drawbacks of the diode matrix. For the sake of simplicity and clarity, the programmable diode matrix is shown at a simple crosspoint in Figure 2.3.3

A decoder is nothing more than a collection of AND gates that combine all the inputs to produce product terms. The basic logic implemented by the PROM is AND-OR with the AND gates all preconnected on the chip, making this portion fixed.

The OR matrix is implemented with diode-fuse interconnections, making it programmable. Thus, the PROM is an AND-OR logic element with fixed AND matrix and programmable OR.

There are many advantages to using PROMs as logic devices. Because they are used in many applications, they are produced in high volume. Also, the PROM is a universal logic solution. In other words, all of the product terms of the input variables are generated. This makes it possible to implement any AND-OR function of these variables.

On the less positive side, it is difficult to accomodate a large number of variables with PROMs. For each variable added to the PROM, not only does the package increase by one pin, but the size of the fuse matrix doubles. For example, an eight-function, five-variable PROM (32 × 8) requires a 256-fuse element matrix. An eight-function, six-variable device (64 × 8) requires a 512-element matrix. As a practical matter, PROMs are limited in the maximum number of input variables they can be designed to handle. Manufacturers are currently producing no larger than 13-input PROMs.

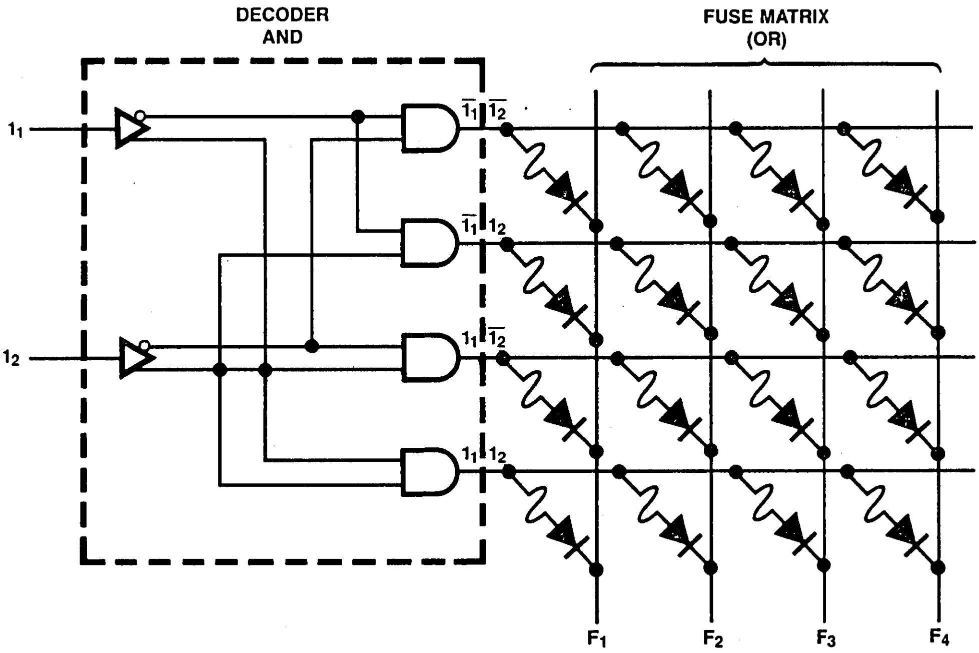


Figure 2.3.2 4 × 4 Bit Prom

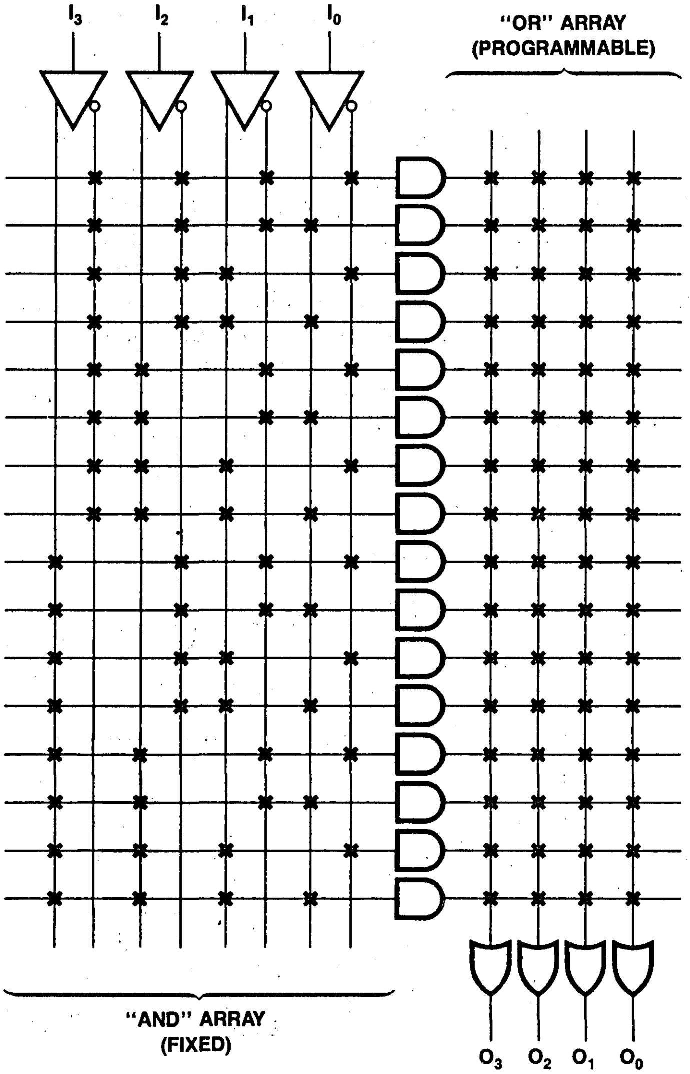


Figure 2.3.3 PROM with 16 Words × 4 Bits

The FPLA

The Field-Programmable Logic Array (FPLA) overcomes some of the size restrictions of PROMs because its designers recognized that not all product terms are required to implement most logic functions. By having a second fuse matrix (an AND matrix), the FPLA allows the designer to select and program only those product terms used in each specific function (Figure 2.3.4). These product terms are then combined in the OR fuse array to form an AND-OR logic equation.

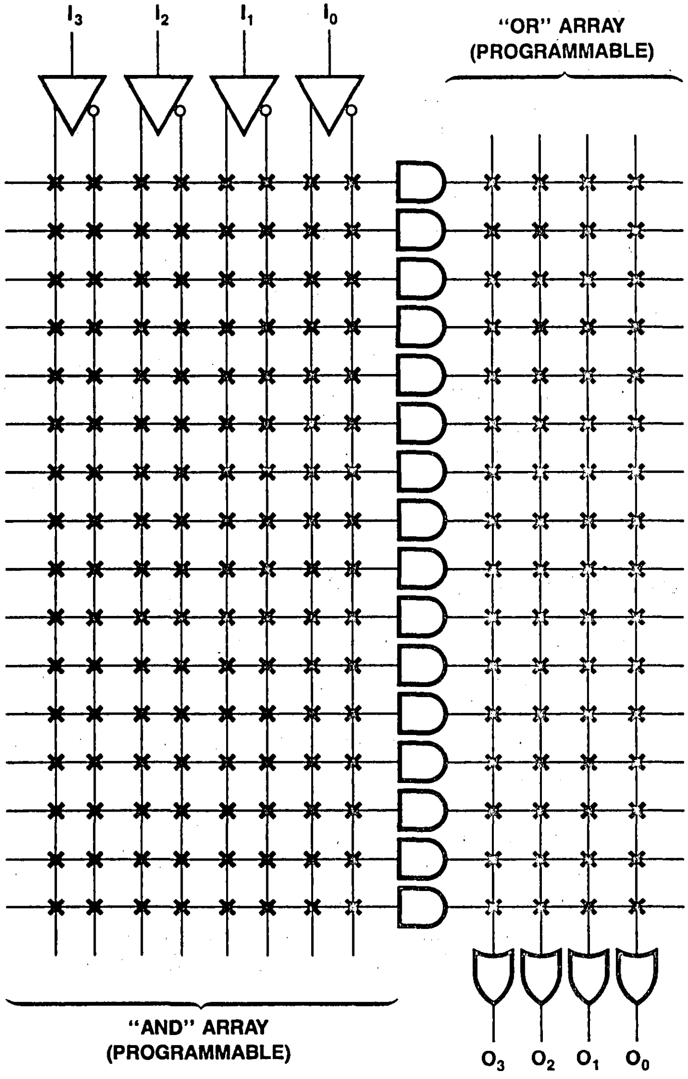


Figure 2.3.4 FPLA with 4 Inputs, 4 Outputs, and 16 Products

In addition to specifying the number of inputs and functions, the FPLA manufacturer must also specify the number of product terms available, since there are less than 2^n terms (with n as the number of input variables). The fact that the number of product terms is less than 2^n is what allows the FPLA to accommodate larger values of n, i.e., more inputs. This is in contrast to the PROM where the number of product terms is always equal to 2^n.

Although the FPLA usually requires less fuses to implement a given logic function, the additional fuse matrix does pose some difficulties of its own. The biggest problem is the circuitry required to select and program these fuses — circuitry that is not used in the final logic solution, but which is paid for in die area. This "chip overhead" cost is not significant if the FPLA's capabilities are fully utilized, but it does become significant for less complex problems that leave unused logic.

As has been shown, PROMs provide all of the product terms for a limited number of input variables in generating AND-OR logic functions. FPLAs, on the other hand, provide a limited number of product terms for a larger number of input variables. However, the FPLA is unrestricted in combining the product terms in the OR matrix, which adds considerable flexibility to this device.

Because of the dual fuse matrix and the overhead cost of the circuitry required for programming, the FPLA cannot be used economically in some low complexity logic problems. The cost saving associated with the removal of the AND matrix (by hardwiring it) is evident when one compares price. PROMs cost less than FPLAs. As mentioned, however, the PROM approach significantly restricts the number of input variables.

The PAL (Programmable Array Logic) Device

Savings similar to those of PROMs could be made without the penalty of restricting the input variables, by removing the OR matrix from the FPLA, or hardwiring it. In the PAL device concept (Figure 2.3.5), the AND fuse array allows the designer to specify the product terms required. The terms are then hardwired to a predefined OR matrix to form AND-OR logic functions.

An immediate observation is that because the OR gates in PAL devices are prewired, the degree to which the product terms can be combined at these OR gates is restricted. PAL devices partially compensate for this by offering different part types that vary the OR-gate configuration. Specifying the OR-gate connection therefore becomes a task of device selection rather than of programming, as with the FPLA. With this approach, PAL devices eliminate the need for a second fuse matrix with little loss in overall flexibility.

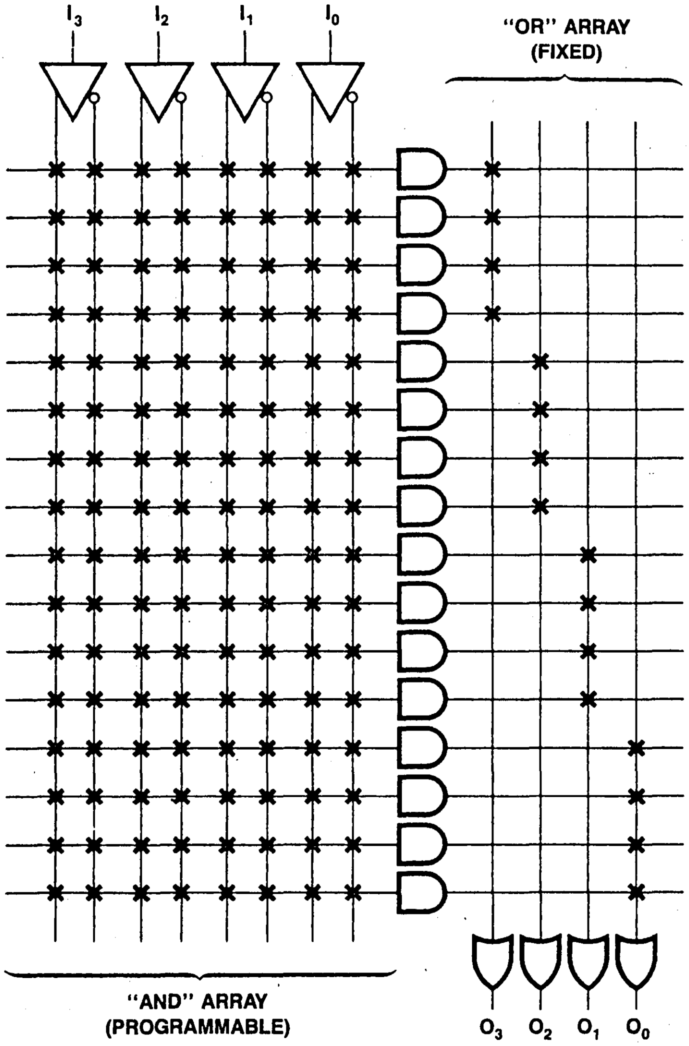


Figure 2.3.5 PAL Device Having 4 Inputs, 4 Ouputs, and 16 Products

Comparison

To illustrate the difference among the three most popular field-programmable logic concepts, the same four logic expressions will be solved with each, as shown in Figure 2.3.6(a). For comparison, each of the approaches is shown as an AND matrix, followed by an OR matrix. The PROM solution shown in Figure 2.3.6(b) requires a 16-fuse

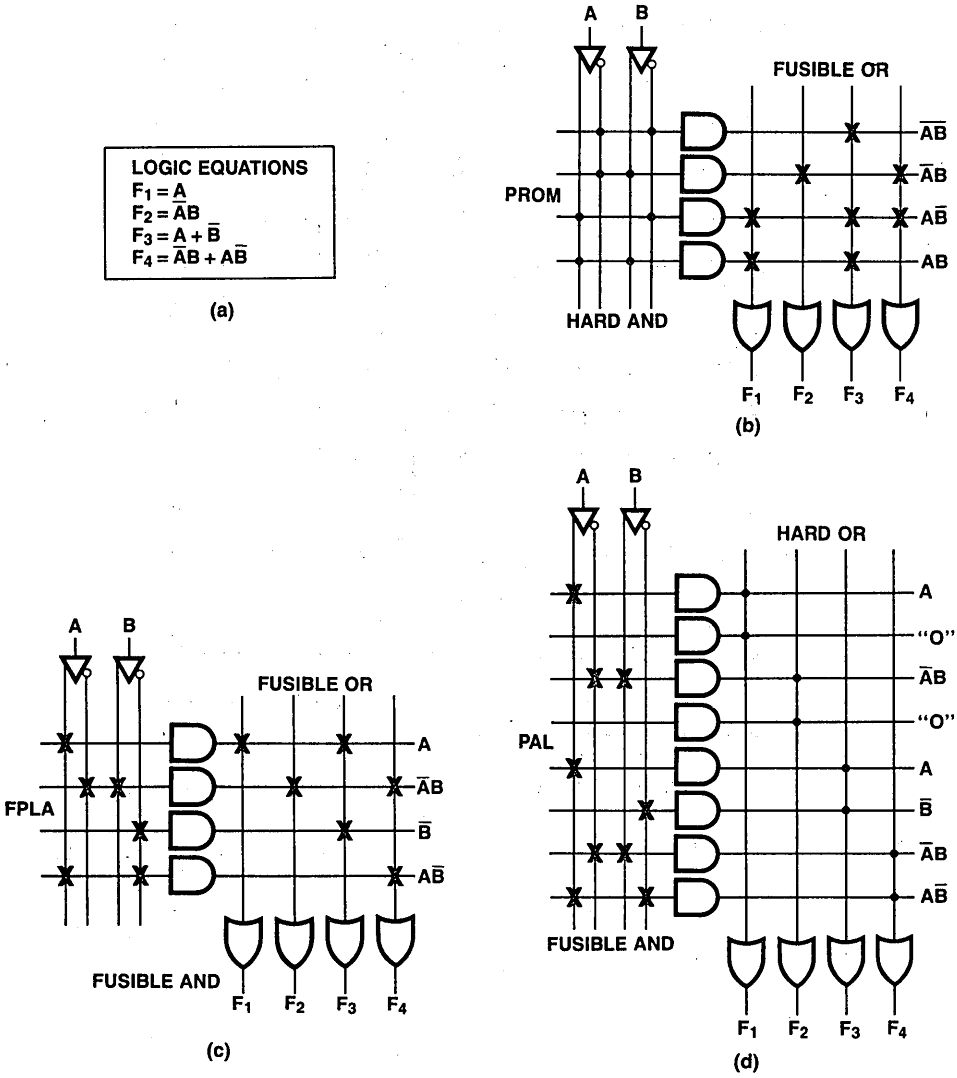


Figure 2.3.6 (a) Logic Equation, (b) PROM Solution, (c) FPLA Solution and (d) PAL Device Solution

matrix, whereas the FPLA and PAL device require 32 fuses each. If we were to add another input variable, the number of fuses in a PROM increases to 32, while the FPLA needs only 8 more and the PAL device needs 16 more. A fourth input again doubles the number of PROM fuses to 64, but adds only 8 to the FPLA and 16 to the PAL device. This example illustrates the previous statement that as the number of inputs increases, PROMs consume more fuses than either FPLAs or PAL devices.

2.4 PROGRAMMABLE LOGIC VERSUS OTHER LSI, SEMICUSTOM AND CUSTOM ALTERNATIVES

Logic designers are noticing an apparent "complexity gap" between TTL and LSI. Products designed with discrete TTL devices would consume unacceptable amounts of physical space and electrical power. Software-programmable LSI devices (microprocessors) offer high density and need relatively little power to operate. But the designer pays a high price in software development and still has to use discretes to interface them to the outside world. Until recently, there has been no device that provides a really effective way of bridging this gap. National has seen this need, and now offers the designer a family of PAL (Programmable Array Logic) devices to fill it. PAL devices offer powerful capabilities for creating cost-effective new products or for improving the effectiveness of existing logic designs. PAL devices save time and money by solving many of the system partitioning and interface problems not otherwise effectively solved by today's semiconductor device technology.

Standardized LSI

LSI (Large Scale Integration) offers many advantages, but advances have been made at the expense of either device flexibility or software complexity. LSI technology has been and still is leading to larger and larger standard logic functions. LSI offers high functional density and low power consumption; single ICs now perform functions that formerly required complete circuit cards. However, most LSI devices don't interface with user systems without large numbers of support devices. Designers are still forced to turn to random logic for many applications. LSI is slow, and it is rigidly partitioned. For all its capability to perform varied and complex tasks, the microprocessor is a slow and expensive way of doing simple, repetitive tasks when the necessary interface and other support devices are added. And, when the time, money, and memory required for software development are considered it is even more expensive.

Full Custom IC's

Custom IC's can be effective design solutions if the product is of low-to-medium complexity, its logic function is well-defined, and its market is high-volume. Its design cycle is typically long, and its cost can be prohibitive. This tends to discourage its use.

Gate Arrays

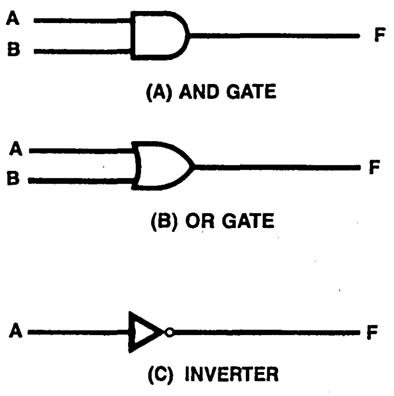
A close relative of the custom circuit is the gate array. With gate arrays, the total logic capability of the chip, its pinouts, and its performance are predefined by the semiconductor manufacturer. The user specifies only the logic interconnection pattern, a process much the same as interconnecting standard small-scale integration (SSI) logic circuits. Since only a metallization pattern is required, the setup costs and turnaround time for gate arrays are lower than for custom circuits, but because the designer can seldom utilize the entire logic capability of the chip, the unit cost per active element is often higher. The setup costs and turnaround time for gate arrays are considerably higher than that for programmable logic, which has practically no turnaround delay.

3

Boolean Logic Review

3.1 BASIC OPERATORS AND THEOREMS

A gate is an electronic circuit which operates on one or more input signals to produce an output signal. There are three basic gates from which all other logic can be realized: AND, OR, and INVERTER gates. Figure 3.1.1 shows these three basic gates and their truth table.



(A) AND GATE

(B) OR GATE

(C) INVERTER

INPUT		OUTPUT
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

INPUT	OUTPUT
A	F
0	1
1	0

Figure 3.1.1 Basic Gates

To express the function of these gates by Boolean* algebra, we need to define Boolean operators as follows:

=	Logical Equality
—	Negate (not, invert, complement)
+	OR (sum)
•	AND (product)
: + :	Exclusive OR
:*:	Exclusive NOR

The function of an AND gate in Figure 3.1.1 can be expressed as:

$$F = A \circ B$$

The function of an OR gate and INVERTER can be expressed as:

$$F = A + B$$

and $$F = \overline{A}$$

Boolean operators are logical operators, which are different from arithmetic operators. For example, + is logical addition, $\circ$ is logical multiplication. We call such equations Boolean equations or logic equations.

A number of logic theorems and laws will be used to manipulate and reduce logical equations. These theorems and laws are as follows:

Theorem 1	$A + 0$	$= A$
Theorem 2	$A \circ 0$	$= 0$
Theorem 3	$A + 1$	$= 1$
Theorem 4	$A \circ 1$	$= A$
Theorem 5	$A + A$	$= A$
Theorem 6	$A \circ A$	$= A$
Theorem 7	$A + \overline{A}$	$= 1$
Theorem 8	$\underline{A} \circ \overline{A}$	$= 0$
Theorem 9	$\overline{A}$	$= A$
Theorem 10	$A + A \circ B$	$= A$
Theorem 11	$A(A + B)$	$= A$
Theorem 12	$(A + B) \bullet (A + C)$	$= A + B \bullet C$
Theorem 13	$A + \overline{A} \circ B$	$= A + B$

Commutative Law

$$A + B = B + A$$
$$A \circ B = B \circ A$$

Associative Law

$$A + B + C = (A + B) + C = A + (B + C)$$
$$A \circ B \circ C = (A \circ B) \circ C = A \bullet (B \bullet C)$$

Distributive Law

$$A + (B \circ C \circ D) = (A + B) \circ (A + C) \bullet (A + D)$$
$$A \circ (B + C + D) = A \bullet B + A \bullet C + A \bullet D$$

DeMorgan's Theorem

$$\overline{(A + B + C)} = \overline{A} \bullet \overline{B} \bullet \overline{C}$$
$$\overline{(A \circ B \circ C)} = \overline{A} + \overline{B} + \overline{C}$$

**George Boole was the son of a shoemaker. His formal education ended in the third grade. Despite this, he was a brilliant scholar, teaching Greek and Latin in his own school, and an accepted mathematician who made lasting contributions in the areas of differential and difference equations as well as in algebra.*

The complement of any Boolean expression, or a part of any expression, may be found by means of DeMorgan's theorem. Two steps are used to form a complement in this theorem:

1. OR symbols are replaced with AND symbols or AND symbols with OR symbols.
2. Each of the terms in the expression is complemented.

DeMorgan's theorem is one of the most powerful tools for engineering applications. It is very useful for designing with programmable logic devices because it provides a quick and simple conversion method between PRODUCT-OF-SUMS and SUM-OF-PRODUCTS expressions, which will be defined later.

3.2 DERIVATION OF A BOOLEAN EXPRESSION

Any logic expression can be reduced to a two-level form and expressed as either a SUM-OF-PRODUCTS (SOP) or PRODUCT-OF-SUMS (POS). Before we define SOP or POS, we need to define "terms."

1. Product Term — A product term is a single variable or the logical product of several variables. The variable may or may not be complemented.
2. Sum Term — A sum term is a single variable or the sum of several variables. The variables may or may not be complemented.
3. Normal Term — A normal term is a product or sum term in which no variable appears more than once.
4. Minterm — A minterm is a product term containing every variable once and only once (either true or complemented).
5. Maxterm — A maxterm is a sum term containing every variable once and only once (either true or complemented).

For example, the term $A \bullet B \bullet C$ is a product term; $A + B$ is a sum term; A is both a product term and a sum term; $A + B \bullet C$ is neither a product term nor a sum term; $A + \overline{B}$ is a sum term; $A \bullet \overline{B} \bullet \overline{C}$ is a product term; $\overline{B}$ is both a sum term and a product term. We now define two most important forms:

1. SUM-OF-PRODUCTS Expression — A sum-of-products expression is a product term or several product terms logically added together.
2. PRODUCT-OF-SUMS Expression — A product-of-sums expression is a sum term or several sum terms logically multiplied together.

For example, the expression $\overline{A} \bullet B + A \bullet \overline{B}$ is a sum-of-products expression; $(A + B) \bullet (\overline{A} + \overline{B})$ is a product-of-sums expression.

One prime reason for using sum-of-products or product-of-sums expressions is their straightforward conversion to very simple gating networks. In their purest, simplest form they go into two-level networks, which are networks for which the longest path through which a signal must pass from input to output is two gates long.

When designing a logic circuit, the logic designer works from two sets of known values; the various states which the inputs to the logical network can take, and the desired outputs for each input condition. The logic expression is derived from these sets of values and the procedure is as follows:

1. Construct a table of the input and output values (Table 3.2.1 left half).

2a. To derive a SUM-OF-PRODUCTS (SOP) expression:
A product term column is added listing the inputs A, B, and C according to their value in the input columns (Table 3.2.1). Then the product terms from each row in which the output is a "1" are collected.

Therefore:

$$F = \overline{A} \bullet B \bullet \overline{C} + \overline{A} \bullet B \bullet C + A \bullet B \bullet \overline{C} \qquad \text{(Eq. 3.2.1)}$$

2b. To derive a PRODUCT-OF-SUMS (POS) expression:
A sum term column is added listing the inputs A, B and C according to their *complement* value in the input columns (Table 3.2.1). Then the sum terms from each row in which the output is "0" are collected.

Therefore:

$$F = (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})$$
(Eq. 3.2.2)

Inputs			Outputs		
A	B	C	F	Product Terms	Sum Terms
0	0	0	0	$\overline{A}\ \overline{B}\ \overline{C}$	$A+B+C$
0	0	1	0	$\overline{A}\ \overline{B}\ C$	$A+B+\overline{C}$
0	1	0	1	$\overline{A}\ B\ \overline{C}$	$A+\overline{B}+C$
0	1	1	1	$\overline{A}\ B\ C$	$A+\overline{B}+\overline{C}$
1	0	0	0	$A\ \overline{B}\ \overline{C}$	$\overline{A}+B+C$
1	0	1	0	$A\ \overline{B}\ C$	$\overline{A}+B+\overline{C}$
1	1	0	1	$A\ B\ \overline{C}$	$\overline{A}+\overline{B}+C$
1	1	1	0	$A\ B\ C$	$\overline{A}+\overline{B}+\overline{C}$

Table 3.2.1 Truth Table of Eq. 3.2.1 and Eq. 3.2.2

Figure 3.2.1 is the logic circuit which direct derived from Eq. 3.2.1. Figure 3.2.2 is derived from Eq. 3.2.2.

Eq. 3.2.1 can be simplified as shown below:

$$
\begin{aligned}
F &= \overline{A} \bullet B \bullet \overline{C} + \overline{A} \bullet B \bullet C + A \bullet B \bullet \overline{C} \\
&= \overline{A} \bullet B\,(\overline{C} + C) + A \bullet B \bullet \overline{C} \\
&= \overline{A} \bullet B + A \bullet B \bullet \overline{C} \\
&= B\,(\overline{A} + A \bullet \overline{C}) \\
&= B\,(\overline{A} + \overline{C}) \\
&= \overline{A} \bullet B + B \bullet \overline{C}
\end{aligned}
$$

Eq 3.2.2 can be simplified as shown:

$$
\begin{aligned}
F &= (A + B + C)\,(A + B + \overline{C})\,(\overline{A} + B + C)\,(\overline{A} + B + \overline{C})\,(\overline{A} + \overline{B} + \overline{C}) \\
&= (A + B)\,(\overline{A} + B)\,(\overline{A} + \overline{C}) \\
&= B\,(\overline{A} + \overline{C}) \\
&= \overline{A} \bullet B + B \bullet \overline{C}
\end{aligned}
$$

The two final expressions obtained are identical and can be implemented by the circuit shown in Figure 3.2.3. This is much simpler than the circuits in Figures 3.2.1 and 3.2.2. This simplified procedure is called minimization.

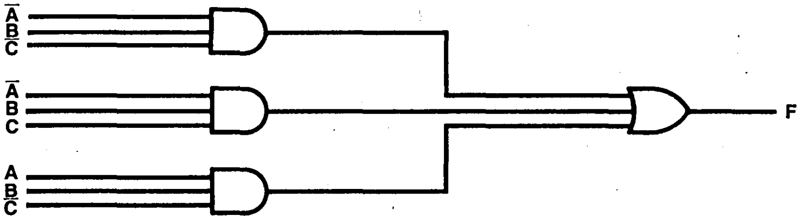


Figure 3.2.1 Logic Circuits of Eq. 3.2.1

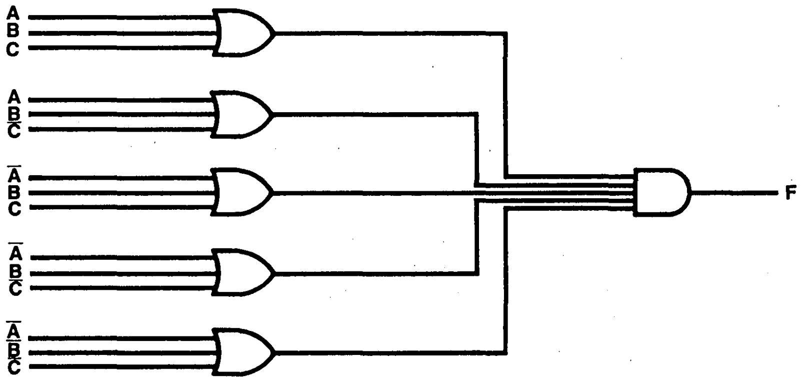


Figure 3.2.2 Logic Circuits of Eq. 3.2.2

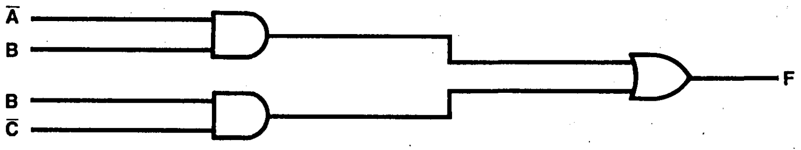


Figure 3.2.3 Simplified Logic Circuits

3.3 MINIMIZATION

Logic circuits can be represented by logic expressions or so called logic equations. As discussed, we can minimize the logic circuit through logic equations minimization. For example, Figure 3.3.1 can be expressed by Eq. 3.3.1.

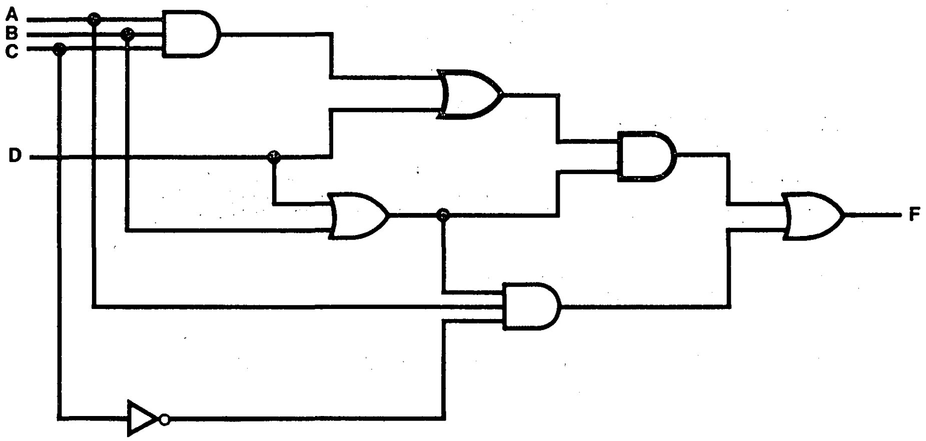


Figure 3.3.1 A Random Logic Circuit

$$F = (A \bullet B \bullet C + D) \bullet (B + D) + A \bullet \overline{C} \bullet (B + D) \qquad \text{(Eq. 3.3.1)}$$

By using the theorems and laws mentioned in 3.1, we minimize Eq. 3.3.1 as follows:

$$\begin{aligned} F &= A \bullet B \bullet C + B \bullet D + A \bullet B \bullet C \bullet D + D + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D \\ &= A \bullet B \bullet C(1 + D) + D(B + 1) + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D && \text{Distributive Law} \\ &= A \bullet B \bullet C + D + A \bullet \overline{C} \bullet B + A \bullet \overline{C} \bullet D && \text{Theory 3} \\ &= A \bullet B(C + \overline{C}) + D(1 + A \bullet \overline{C}) && \text{Distributive Law} \\ &= A \bullet B + D. \end{aligned}$$

The minimum SOP expression can now be implemented as the simple AND-OR logic circuits as shown in Figure 3.3.2.

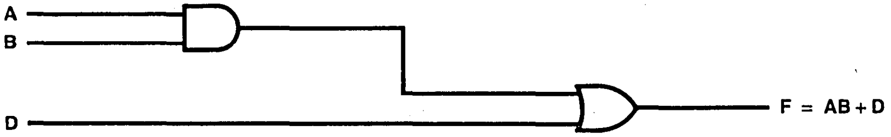


Figure 3.3.2 Minimized Logic Circuit

We can use Boolean Algebra to reduce the number of product terms. However, Karnaugh Mapping and the Quine-McCluskey method are two other powerful tools to minimize the logic equations. We'll discuss Karnaugh Mapping method in the next section.

3.4 K-MAP METHOD

A Karnaugh map, hereafter called a K-map, is a graphical method for representing a Boolean function. It is similar to a truth table in that the K-map supplies the TRUE or FALSE value of a Boolean function for all possible combinations of its logical argument. There are many ways in which a K-map can be arranged. The most important considerations of the arrangement are:

1. There must be a unique location on the K-map for entering the TRUE/FALSE value of the function that corresponds to each combination of input variables.
2. The locations should be arranged so, with minimization mentioned in Section 3.3, that they are readily apparent to the trained observer.

The second consideration implies that a successful K-mapping arrangement should point to groups of minterms or maxterms that can be combined into reduced forms. K-maps are also useful in expanding partially reduced expressions into standard forms prior to the minimization process.

The K-map is one of the most powerful tools at the hands of the logic designer. The power of the K-map does not lie in its application of any marvelous new theorems, but rather in its utilization of the remarkable ability of the human mind to perceive patterns in pictorial representations of data. This is not a new idea. Anytime we use a graph instead of a table of numerical data, we are utilizing the human ability to recognize complex patterns and relationships in a graphical representation far more rapidly and surely than in a tabular representation. A few examples of how to create a K-map follow.

First, consider a truth table for two variables. We list all four possible input combinations and the corresponding function values, i.e., the truth tables for AND and OR. (Figure 3.4.1)

A	B	A•B
0	0	0
0	1	0
1	1	1
1	0	0

A	B	A+B
0	0	0
0	1	1
1	1	1
1	0	1

Figure 3.4.1 Truth Tables for AND and OR

As an alternative approach, set up a diagram consisting of four small boxes, one for each combination of variables. Place a "1" in any box representing a combination of variables for which the function has the value 1. There is no logical objection to putting "0's" in the other boxes, but they are usually omitted for clarity.

The diagrams in Figure 3.4.2(a) are perfectly valid K-maps, but it is more common to arrange the four boxes in a square, as shown in Figure 3.4.2(b).

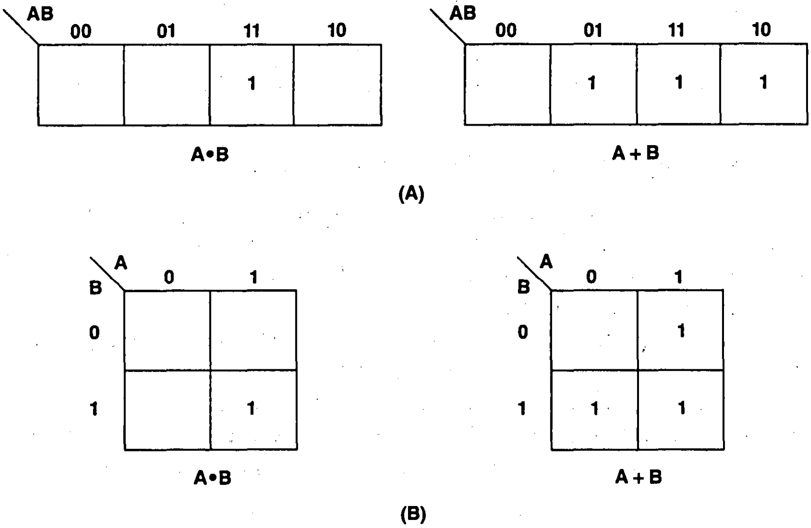


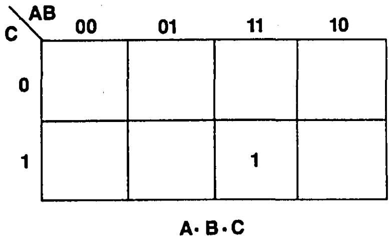
Figure 3.4.2 K-maps for AND and OR

Since there must be one square for each input combination, there must be 2^n squares in a K-map for n-variables. Whatever the number of variables, we may interpret the map in terms of a graphical form of the truth table (Figure 3.4.3(a)) or in terms of union and intersection of areas (Figure 3.4.3(b)).

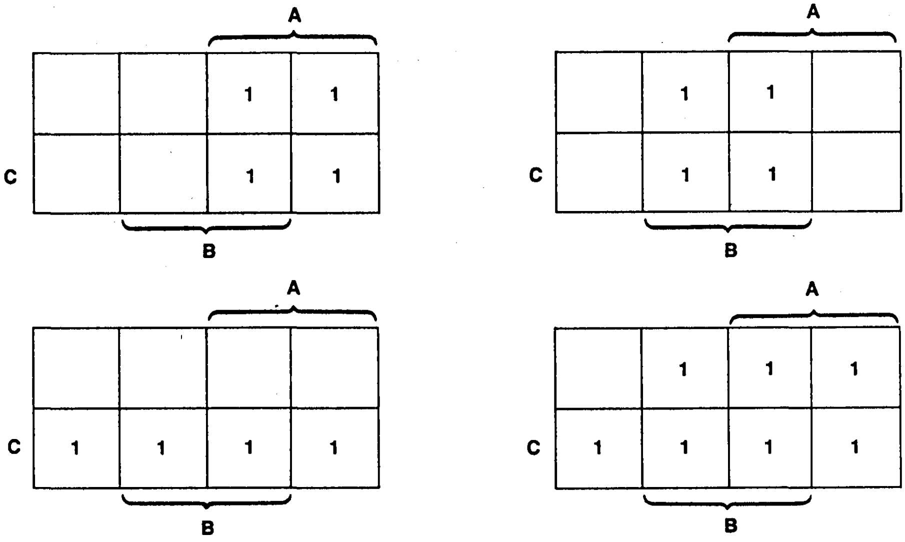
The K-maps for some other three-variable functions are shown in Figure 3.4.4.

Particularly note the functions mapped in Figure 3.4.3(a) and 3.4.4(b). These are both minterms. Each is represented by one square, obviously, and each one of the eight squares corresponds to one of the eight minterms of three variables. This is the origin of the name minterm. A minterm is the form of Boolean function corresponding to the minimum possible area, other than 0, on a K-map. A maxterm, on the other hand, is the form of Boolean function corresponding to the maximum possible area, other than 1, on a K-map. Figure 3.4.3 (b) and 3.4.4 (c) are two examples.

A	B	C	A·B·C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



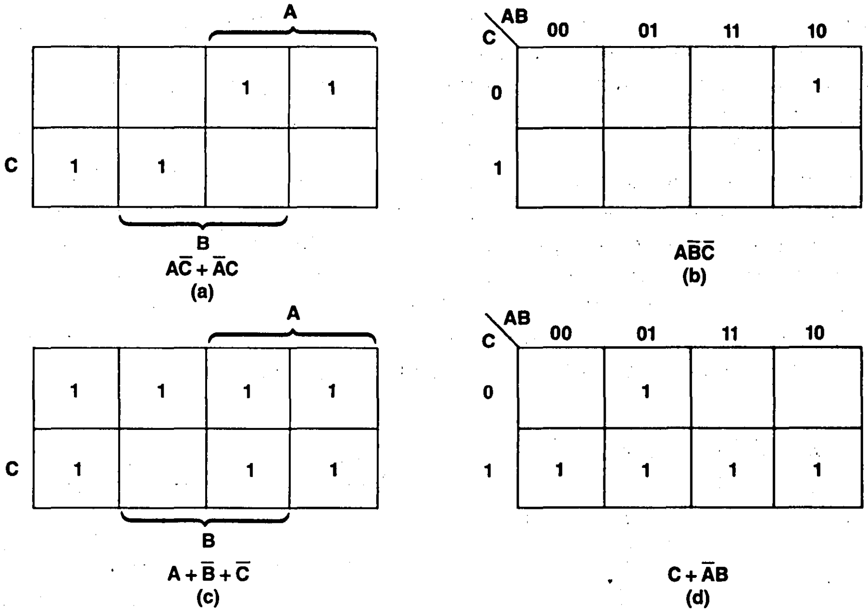
(a)



A + B + C = A+B+C

(b)

Figure 3.4.3 K-Maps for 3-variable AND and OR



$A\bar{C}+\bar{A}C$
(a)

$A\bar{B}\bar{C}$
(b)

$A+\bar{B}+\bar{C}$
(c)

$C+\bar{A}B$
(d)

Figure 3.4.4 Sample 3-variable K-maps

Since each square on a K-map corresponds to a row in a truth table, it is appropriate to number the squares just as we numbered the row. These standard K-maps are shown in Figure 3.4.5 for two and three variables. Now, if a function is stated in the form of the minterm list, all we need to do is enter 1's in the corresponding squares to produce the K-map.

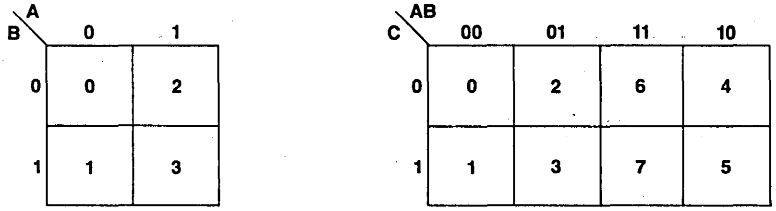


Figure 3.4.5 K-maps for Two and Three Variables

If a function is stated as a maxterm list, we can enter 0's in the squares listed or 1's in those not listed.

A map showing the 0's of a function is a perfectly valid K-map, although it is more common to show the 1's.

For example, the K-map of f(A, B, C) = m(0, 2, 3, 7) is shown in Figure 3.4.6 and the K-map of f(A, B, C) = M(0, 1, 5, 6) is shown in Figure 3.4.7. where m means minterm, M means maxterm.

C \ AB	00	01	11	10
0	1	1		
1		1	1	

Figure 3.4.6 K-map of m(0, 2, 3, 7)

C \ AB	00	01	11	10
0	0		0	
1	0			0

OR

C \ AB	00	01	11	10
0		1		1
1		1	1	

Figure 3.4.7 K-map of M(0, 1, 5, 6)

As shown, the K-map can be generated from the truth table on minterm expression or maxterm expression. For the remainder of this section, we will learn how to minimize the minterm expression by using the K-map.

The general principle of this minimization technique is "Any pair of n-variable minterms which are adjacent on a K-map may be combined into a single product term of n – 1 literals." The definition of "adjacent" should include opposite edges of the K-map, for instance, Figure 3.4.8(a) and 3.4.8(b) both have a pair of adjacent minterms.

	1		
	1		

(a)

1			1

(b)

Figure 3.4.8 Adjacent Minterms on a K-map

Consider this function

$$f(A, B, C) = m(0, 1, 4, 6)$$
$$= \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}\overline{C} + AB\overline{C}$$

Which results on the K-map, on the pattern shown in Figure 3.4.9

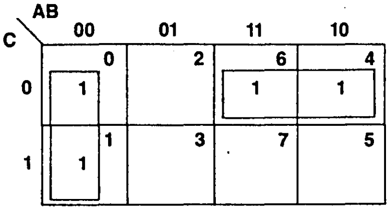


Figure 3.4.9 Minimization

Therefore, combine minterms 0 and 1, 4 and 6 to get a minimal expression:

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

Figure 3.4.10 shows some examples. Notice that it is permissible to include a minterm in several terms if it helps make the term shorter.

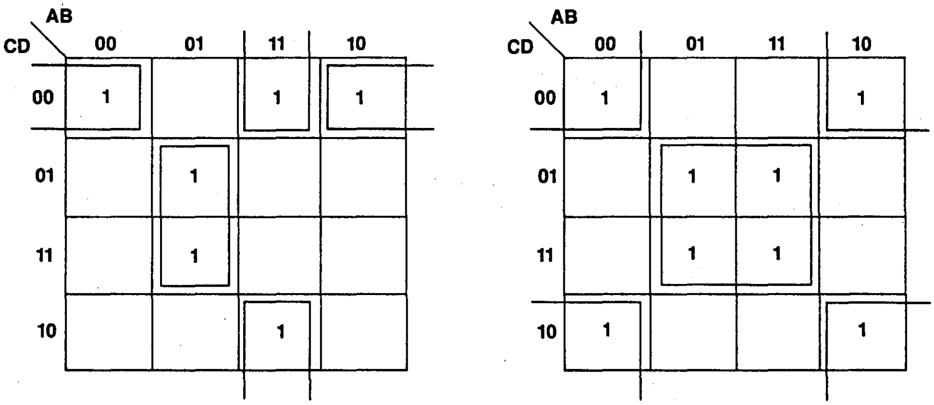
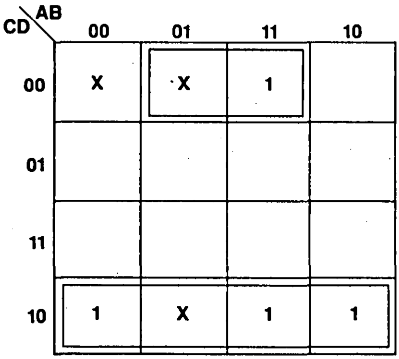


Figure 3.4.10 Minimization

Quite often, some of the possible combinations of input values never occur. In this case, we "don't care" what the function does if these input combinations appear. The K-map makes it easy to take advantage of these "don't care" conditions by letting the "don't care" minterms be 1 or 0, depending on which value results in a simpler expression. Figure 3.4.11 shows an example of the use of "don't cares" (redundancies) to simplify the terms.



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

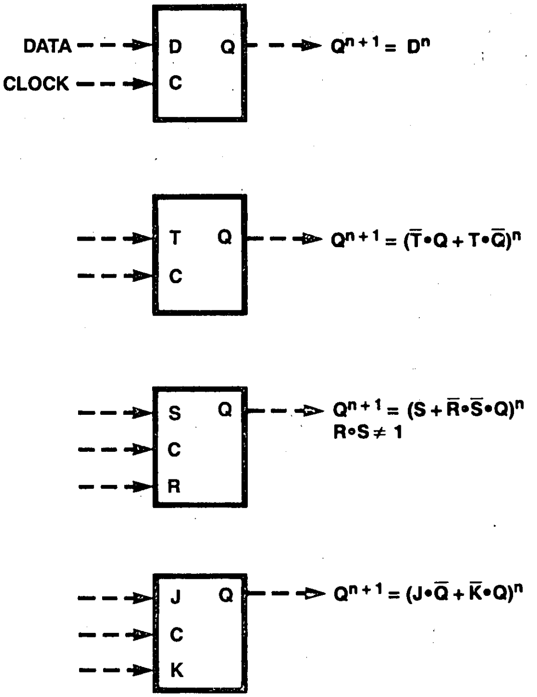
Figure 3.4.11 Minimization

When working with larger functions, the tabular reduction developed by Quine and modified by McCluskey is an alternative to the K-map method. The Quine-McCluskey minimization method involves simple, repetitive operations that compare each minterm that is present in a sun-of-minterms expression for a Boolean functions to all other minterms with which it may form a combinable grouping.

The reader can refer to "Introduction to Switching Theory and Logic Design" by Hill and Peterson to understand the Quine-McCluskey method.

3.5 SEQUENTIAL CIRCUIT ELEMENTS

Usually the subject of logic design is subdivided into two types: sequential and combinational. A purely combinational logic subsystem has no memory. Its outputs are completely defined by its present inputs. The analysis and design of combinational logic is much easier. A sequential logic subsystem has memory and its outputs are functions of not only present inputs but the previous outputs. Circuits of multiplexer/selector, decoder/encoder, adder, and comparator are examples of combinational circuits. Shift register, counter, state machine, and memory controller are examples of sequential circuits.


DATA
CLOCK
D
Q
C
$Q^{n+1} = D^n$
T
Q
C
$Q^{n+1} = (\bar{T} \bullet Q + T \bullet \bar{Q})^n$
S
Q
C
R
$Q^{n+1} = (S + \bar{R} \bullet \bar{S} \bullet Q)^n$
$R \bullet S \neq 1$
J
Q
C
K
$Q^{n+1} = (J \bullet \bar{Q} + \bar{K} \bullet Q)^n$

D^n	Q^{n+1}
0	0
1	1

T^n	Q^{n+1}
0	Q^n
1	$(\bar{Q})^n$

R	S	Q^{n+1}
0	0	Q^n
0	1	1
1	0	0
1	1	X

J	K	Q^{n+1}
0	0	Q^n
0	1	0
1	0	1
1	1	$(\bar{Q})^n$

Figure 3.5.1 Basic Flip-Flops

Just as we have a logic gate as the basic combinational circuit element, we have a flip-flop as a basic sequential circuit element. A flip-flop is a memory device which can remember, or store, a binary bit of information. There are four basic flip-flop types: (1) D flip-flop, (2) T flip-flop, (3) RS flip-flop, and (4) JK flip-flop. Figure 3.5.1 shows these elements and their truth table.

With the memory elements, the output does not change as a function of the inputs until the clock transition. Therefore, a superscript notation is used to indicate that the output during clock period n + 1 is a function of the inputs during the previous clock period n.

The D (delay) flip-flop means the input (D) is "stored" in the flip flop when the clock occurs and will appear on the output (Q) during the next (n + 1) clock time. The D flip-flop is thus very much like a single-bit RAM. It is very useful for data storage and other special applications.

The other three types of flip-flops defined in Figure 3.5.1 are also one-bit storage elements, but instead of simply storing the input, they change state in response to the inputs by various logical rules. Since they hold their previous state in spite of the clock, unless an input goes true, they often simplify the combinational logic functions required to control them in control applications.

The T (toggle) flip-flop, for example, stays in its previous state if the T input is false before the clock. If the T input is true, the output changes to the opposite state (toggle) on the clock. The T flip-flop is thus useful, for example, in binary counters where we want each bit to invert every time there is a carry from the lower order bits.

The R-S flip-flop sets after the S input is true and resets after the R input is true. Its output is undefined if both R and S are true. It is possible to define a Set Overrides Reset (SOR) or a Reset Overrides Set (ROS) flip-flop. It will set or reset respectively if both the R and the S inputs are true.

The J-K flip-flop sets after J is true and resets after K is true. It is similar to an R-S flip-flop except that if J and K are both true, the output changes to the opposite state (toggle). It can be used as a T flip-flop by tying the J and K inputs together.

Since the J-K flip-flop can essentially do the job of both the R-S and the T flip-flop, the R-S and the T flip-flops are seldom seen. The choice is between J-K flip-flops for small counters and control or D flip-flops for data storage applications. Actually the J-K flip-flop can even do the job of the D flip-flop with the addition of a single inverter, as shown in Figure 3.5.2.

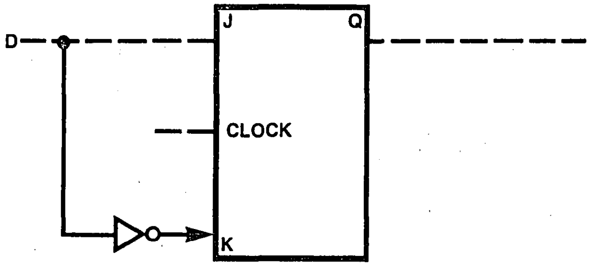


Figure 3.5.2 Implement D Flip-Flop by Using J-K

Another memory element type, called a latch, is often described on data sheets with a truth table like the one for the D flip-flop in Figure 3.5.1. It is definitely not like a D flip-flop, however because the output changes as soon as the clock goes high and does not "latch" until the clock falls (if the input changes while the clock is high, the output follow it). Because of this characteristic, a latch is not usable in the synchronous logic.

3.6 STATE MACHINE FUNDAMENTALS

The relationships among present-state variables, primary input variables, next-state (or excitation) variables, and primary output variables that describe the behavior of a sequential system can be specified in several ways. As an example, consider the simple sequential system that is shown in Figure 3.6.1.

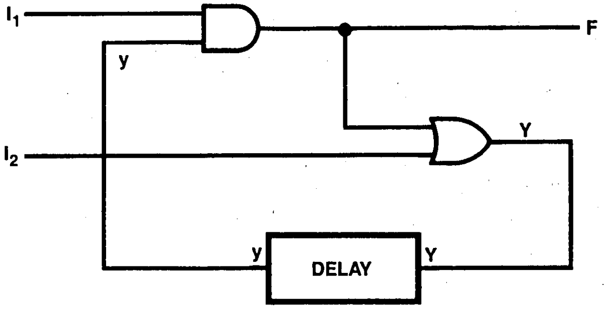


Figure 3.6.1 A Typical Sequential Circuit

This system has two primary input variables, having four different combinations of values. There is one primary output variable and one state variable. It uses delay for memory. There are only two possible present states: y = 0 and y = 1. When combined with the four input combinations, these give eight different total present states. The values of the next-state variable, Y, and the primary output variable, F, must be specified for each total present state. The tabular arrangement shown in Table 3.6.1 is a common method for presenting this information. This descriptive tool is called a state table.

PRESENT-STATE	NEXT-STATE Y	OUTPUT F
y	I_1I_2 = 00 01 10 11	I_1 I_2 = 00 01 10 11
0	0 1 0 1	0 0 0 0
1	0 1 1 1	0 0 1 1

Table 3.6.1 State Table

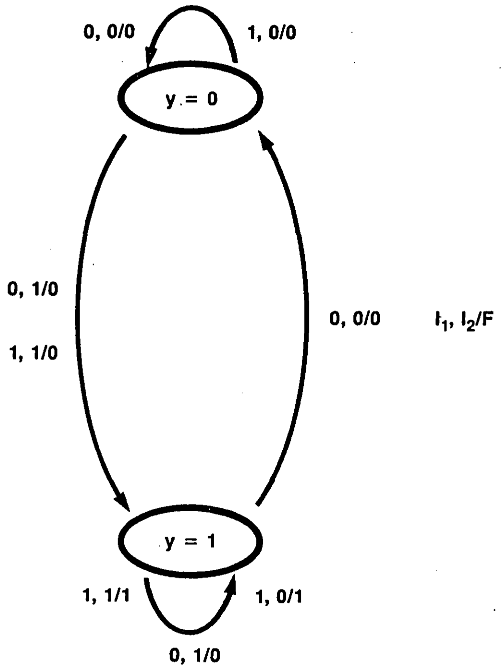


Figure 3.6.2 State Diagram

A second method for describing the behavior of a sequential system is the use of a state diagram. This method presents a pictorial representation of the present-state/next-state sequences that apply to the sequential device. State changes are marked with directed arrows, with the primary input and output conditions that apply to each state transfer given beside the arrows. The state diagram for the system of Figure 3.6.1 is shown in Figure 3.6.2. A slash separates the input information from the output information.

State tables and state diagrams are essential tools in the analysis and design of sequential digital systems. The reader should be familiar with these two tools by reading the references listed in the end of this section.

Because a sequential system has feedback from its outputs to its input, certain types of instabilities and uncertainties can occur. When present, these conditions make the operation of circuit difficult or impossible to describe. They may even render the circuit useless, since its behavior may not be predictable or consistent. Several of these types of problems are listed below.

1) The input or output conditions of the system may be indeterminant. For example, the circuit in Figure 3.6.3.

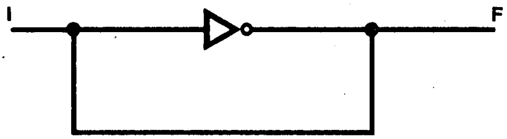


Figure 3.6.3 Example of Hazard Circuit

2) The output condition of the system may be unstable, changing even though the external inputs do not change. Figure 3.6.4. illustrates an example.

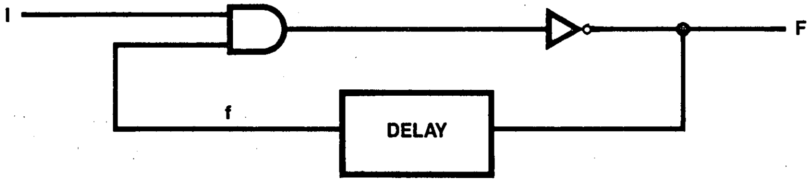


Figure 3.6.4 Example of Unstable Circuit

3) The output condition of the system, even though stable, may not be predictable depending upon the primary input conditions. Figure 3.6.5 is an example.

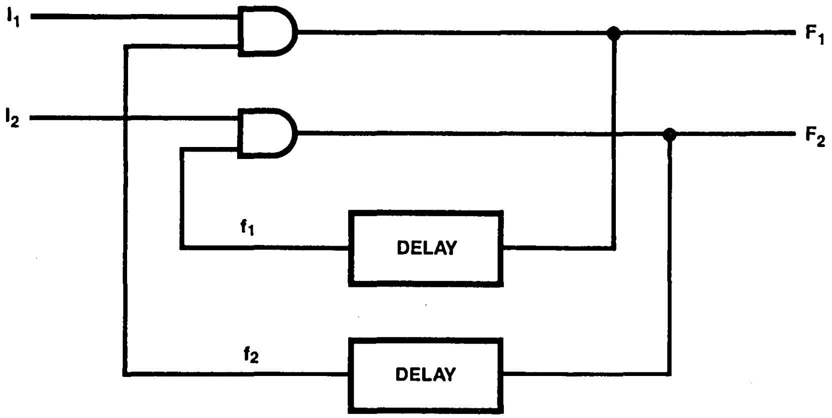


Figure 3.6.5 Example of Circuit with Unpredictable Output States

However, these problems mentioned above can be avoided by making certain restrictions in the way sequential systems are designed and used. For instance, the following are some restrictions:

1. Avoiding continuing instabilities (oscillations).
2. Allowing only fundamental-mode operation.
3. Allowing only pulse-mode operation.

References

Hill & Peterson "Introduction to Switching Theory and Logical Design"
Kohavi "Switching and Finite Automata Theory"
Rhyne "Fundamentals of Digital Systems Designs"
Krieger "Basic Switching Circuit Theory."

4

The Programmable Logic Family

National's programmable logic family consists of PAL devices and PROMs that come in a variety of gate densities, pin-counts, architectures, speed and power specifications. The following sections describe and tabulate these various options in addition to displaying the logic schematics.

4.1 BASIC GROUPS

The programmable logic devices are divided into two sections: one to address PAL devices and the other to address PROMs.

4.2 THE PAL DEVICE FAMILY

The PAL device family is separated by pin-count and by architecture. There is a 20-pin family and a 24-pin family. Each family contains simple combinational logic devices and more complex devices which have on-chip feedback options and output registers. The 20-pin small PAL devices and the 20-pin medium PAL devices are listed in Table 4.2.1.

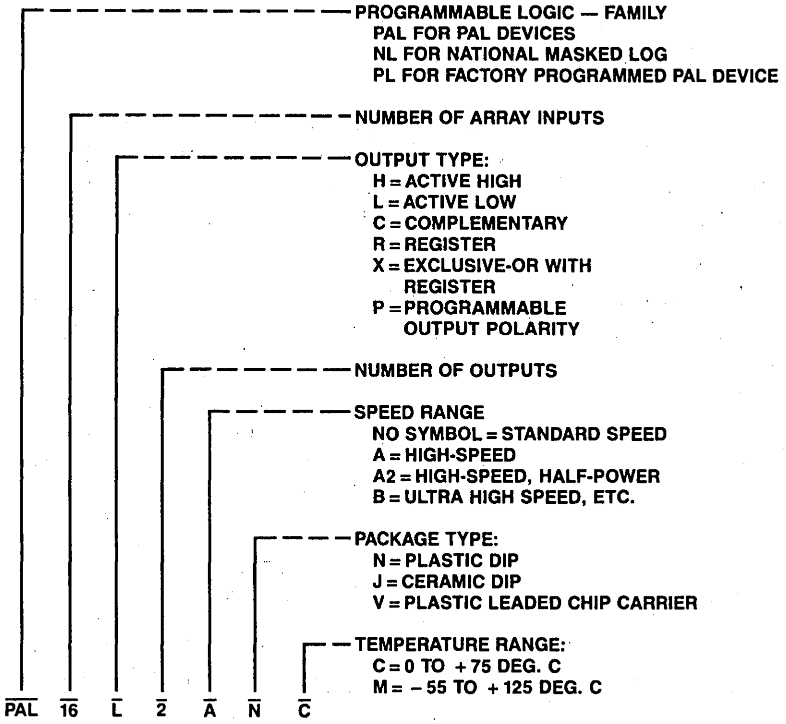
Part No.	No. of Inputs	No. of Outputs	No. of I/Os	No. of Registers	Output Polarity	Functions
10H8	10	8			AND-OR	AND-OR Array
12H6	12	6			AND-OR	AND-OR Array
14H4	14	4			AND-OR	AND-OR Array
16H2	16	2			AND-OR	AND-OR Array
10L8	10	8			AND-NOR	AND-OR-Invert Array
12L6	12	6			AND-NOR	AND-OR-Invert Array
14L4	14	4			AND-NOR	AND-OR-Invert Array
16L2	16	2			AND-NOR	AND-OR-Invert Array
16C1	16	1			AND-OR/NOR	AND-OR/AND-OR-Invert Array
16L8	10	8	6		AND-NOR	AND-OR-Invert Array
16R8	8	8		8	AND-OR	AND-OR-Invert Register
16R6	8	8	2	6	AND-OR	AND-OR-Invert Register
16R4	8	8	4	4	AND-OR	AND-OR-Invert Register

Table 4.2.1 Members of the 20-Pin PAL Device Family

The 24-pin PAL devices are listed in Table 4.2.2 and Table 4.2.3 shows how to read the part numbers.

Part No.	No. of Inputs	No. of Outputs	No. of I/Os	No. of Registers	Output Polarity	Functions
12L10	12	10			AND-NOR	AND-OR Invert Gate Array
14L8	14	8			AND-NOR	AND-OR Invert Gate Array
16L6	16	6			AND-NOR	AND-OR Invert Gate Array
18L4	18	4			AND-NOR	AND-OR Invert Gate Array
20L2	20	2			AND-NOR	AND-OR Invert Gate Array
20L8	14	2	6		AND-NOR	AND-OR Invert Gate Array
20L10	12	2	8		AND-NOR	AND-OR Invert Gate Array
20R8	12	8		8	AND-NOR	AND-OR Invert w/Registers
20R6	12	6	2	6	AND-NOR	AND-OR Invert w/Registers
20R4	12	4	4	4	AND-NOR	AND-OR Invert w/Registers
20X10	10	10		10	AND-NOR	AND-OR-XOR Invert w/Registers
20X8	10	8	2	8	AND-NOR	AND-OR-XOR Invert w/Registers
20X4	10	4	6	4	AND-NOR	AND-OR-XOR Invert w/Registers

Table 4.2.2 Members of the 24-Pin PAL Device Family



PROGRAMMABLE LOGIC — FAMILY
PAL FOR PAL DEVICES
NL FOR NATIONAL MASKED LOG
PL FOR FACTORY PROGRAMMED PAL DEVICE

NUMBER OF ARRAY INPUTS

OUTPUT TYPE:
H = ACTIVE HIGH
L = ACTIVE LOW
C = COMPLEMENTARY
R = REGISTER
X = EXCLUSIVE-OR WITH REGISTER
P = PROGRAMMABLE OUTPUT POLARITY

NUMBER OF OUTPUTS

SPEED RANGE
NO SYMBOL = STANDARD SPEED
A = HIGH-SPEED
A2 = HIGH-SPEED, HALF-POWER
B = ULTRA HIGH SPEED, ETC.

PACKAGE TYPE:
N = PLASTIC DIP
J = CERAMIC DIP
V = PLASTIC LEADED CHIP CARRIER

TEMPERATURE RANGE:
C = 0 TO + 75 DEG. C
M = − 55 TO + 125 DEG. C

Table 4.2.3 PAL Device Part Number Interpretation

PAL Devices For Every Task

The members of the PAL device family are listed in Tables 4.2.1 and 4.2.2. They are designed to cover the spectrum of logic functions at lower cost and lower package count than SSI/MSI logic. This allows you to select the PAL device that best fits your application. PAL devices come in three basic configurations: Gates, Register Outputs With Feedback, and Programmable I/O.

Gates

PALs are available in sizes from 12 × 10 (12 inputs, 10 outputs) to 20 × 2, with either active-high or active-low output configurations. One part has complimentary outputs. This wide variety of input/output formats allows the PAL to replace many different-sized blocks of combinational logic with single packages.

Register Options With Feedback

High-end members of the PAL device family feature latched data outputs with register feedback. Each Sum-Of-Product term is stored in a D flip-flop on the rising edge of the system clock. (See Figure 4.2.1) The Q-output of the flip-flop can then be gated to the output pin by enabling the active low TRI-STATE© buffer.

In addition to being available to transmission, the Q-output is also fed back into the PAL array as an input term. This feedback allows the PAL device to "remember" its prior state. And, it can alter its function based upon that state. This allows one to configure the PAL device as a state machine that can be programmed to execute elementary functions such as count up, count down, skip, shift, and branch.

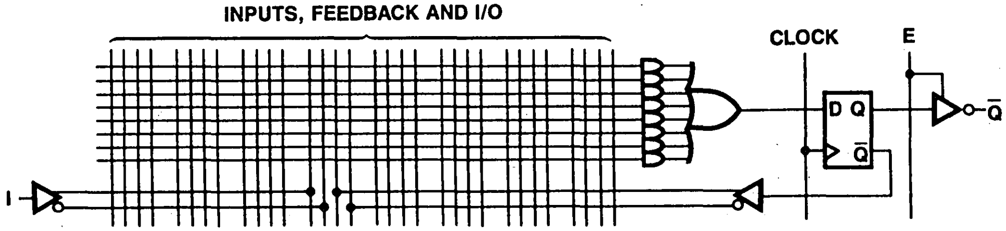


Figure 4.2.1 PAL Device Output Register Circuit, Simplified Logic Diagram

Programmable I/O

Another feature of the high-end members of the PAL family is programmable input/output. This allows the product terms to directly control the outputs of the PAL device. (Figure 4.2.2) One product term is used to enable the TRI-STATE buffer, which in turn gates the summation term to the output pin. The output is also fed back into the PAL device array as an input. Thus, the PAL drives the I/O pin when the TRI-STATE gate is enabled. The I/O pin is an input to the PAL device array when the TRI-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.

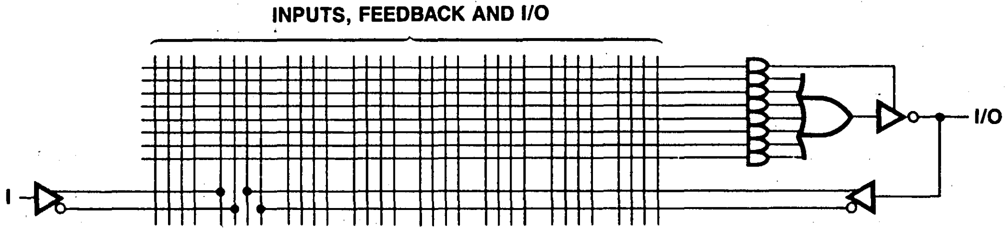


Figure 4.2.2 PAL Device Bidirectional Circuit, Logic Diagram

PAL Device – Speed/Power Groups

PAL devices are available with various speed/power specifications. For easy reference, these are summarized in Tables 4.2.4 and 4.2.5.

	20-Pin Small PAL Devices		20-Pin Medium PAL Devices			
	10H8, 12H6, 14H4, 16H2, 10L8, 12L6, 14L4, 16L2, 16C1		16L8, 16R8, 16R6, 16R4			
	T_{AA} Max (ns)	I_{CC} Max (mA)	T_{AA} Max (ns)	T_{SU} Min (ns)	T_{CLK} Max (ns)	I_{CC} Max (mA)
Standard	35	90	35	35	25	180
A Series	25	90	25	25	15	180
B Series	—	—	15	15	12	180
A-2 Series	35	45	35	35	25	90
B-2 Series	—	—	*25	*25	*15	*90

*Preliminary information.

Table 4.2.4 20-Pin PAL Device Speed/Power Groups

	20L10		20X10, 20X8, 20X4			20C1, 20L2, 18L4, 16L6, 14L8, 12L10		20L8A, 20R8A, 20R6A, 20R4A			
	T_{AA} Max (ns)	I_{CC} Max (mA)	T_{SU} Min (ns)	T_{CLK} Max (ns)	I_{CC} Max (mA)	T_{AA} Max (ns)	I_{CC} Max (mA)	T_{AA} Max (ns)	T_{SU} Min (ns)	T_{CLK} Max (ns)	I_{CC} Max (mA)
Standard	50	165	50	30	180	40	100	—	—	—	—
A Series	—	—	—	—	—	—	—	25	25	15	210

Table 4.2.5 24-Pin Speed/Power Groups

PAL Device Logic Symbols

The logic symbols for each of the individual PAL devices gives a concise functional description of that device. Figure 4.2.3 shows a typical logic symbol, that of the 10H8 gate array.

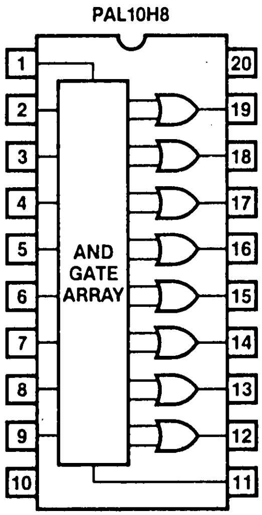
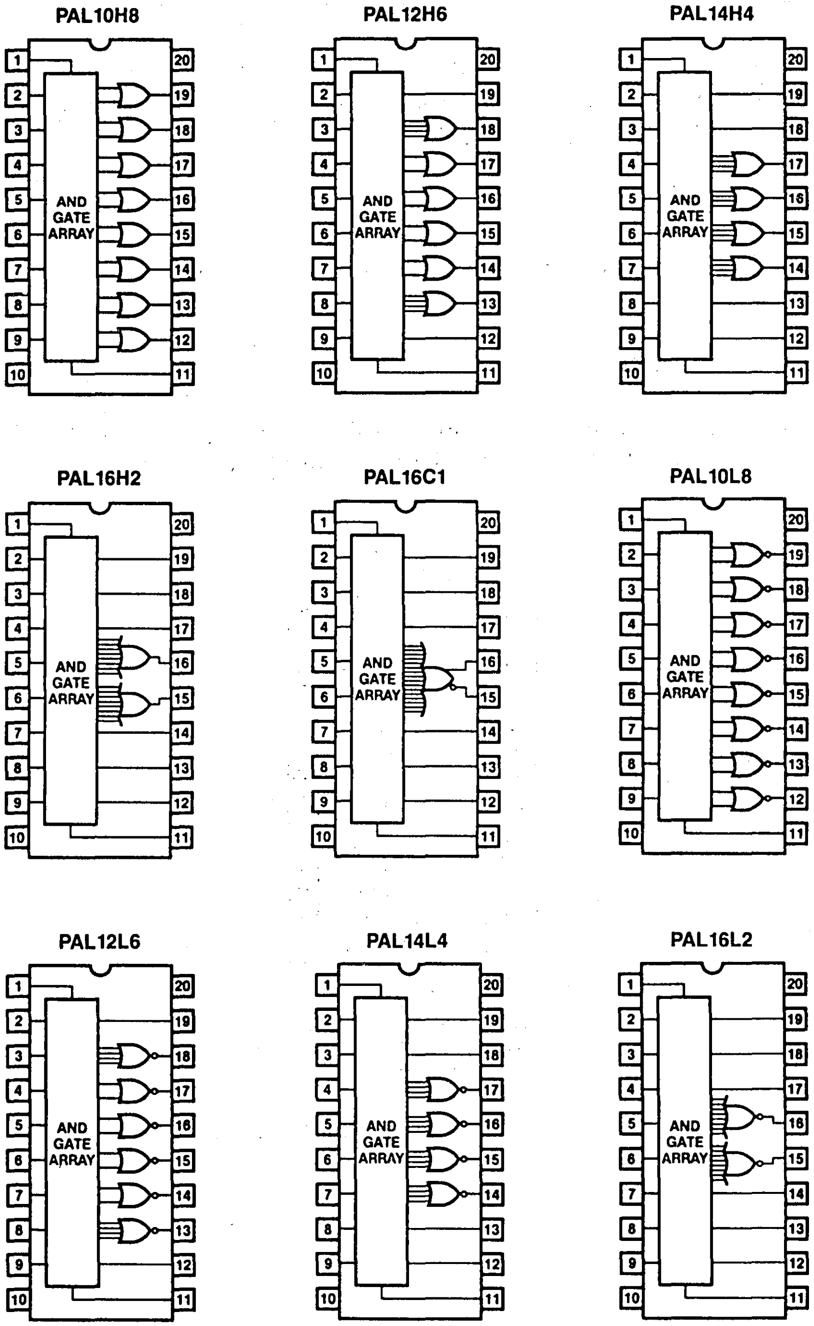


Figure 4.2.3 Logic Symbol, PAL10H8



PAL10H8

PAL12H6

PAL14H4

PAL16H2

PAL16C1

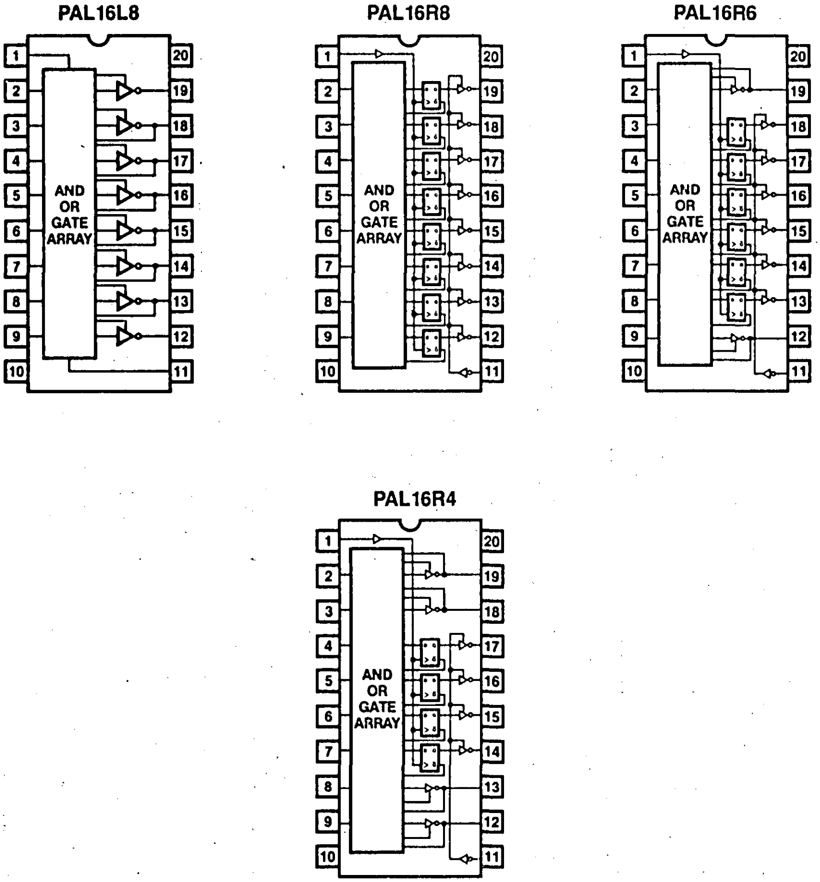
PAL10L8

PAL12L6

PAL14L4

PAL16L2

Figure 4.2.4 PAL Device Logic Symbols — Series 20


PAL16L8

PAL16R8

PAL16R6

PAL16R4

Figure 4.2.4 PAL Device Logic Symbols — Series 20 (Contd.)

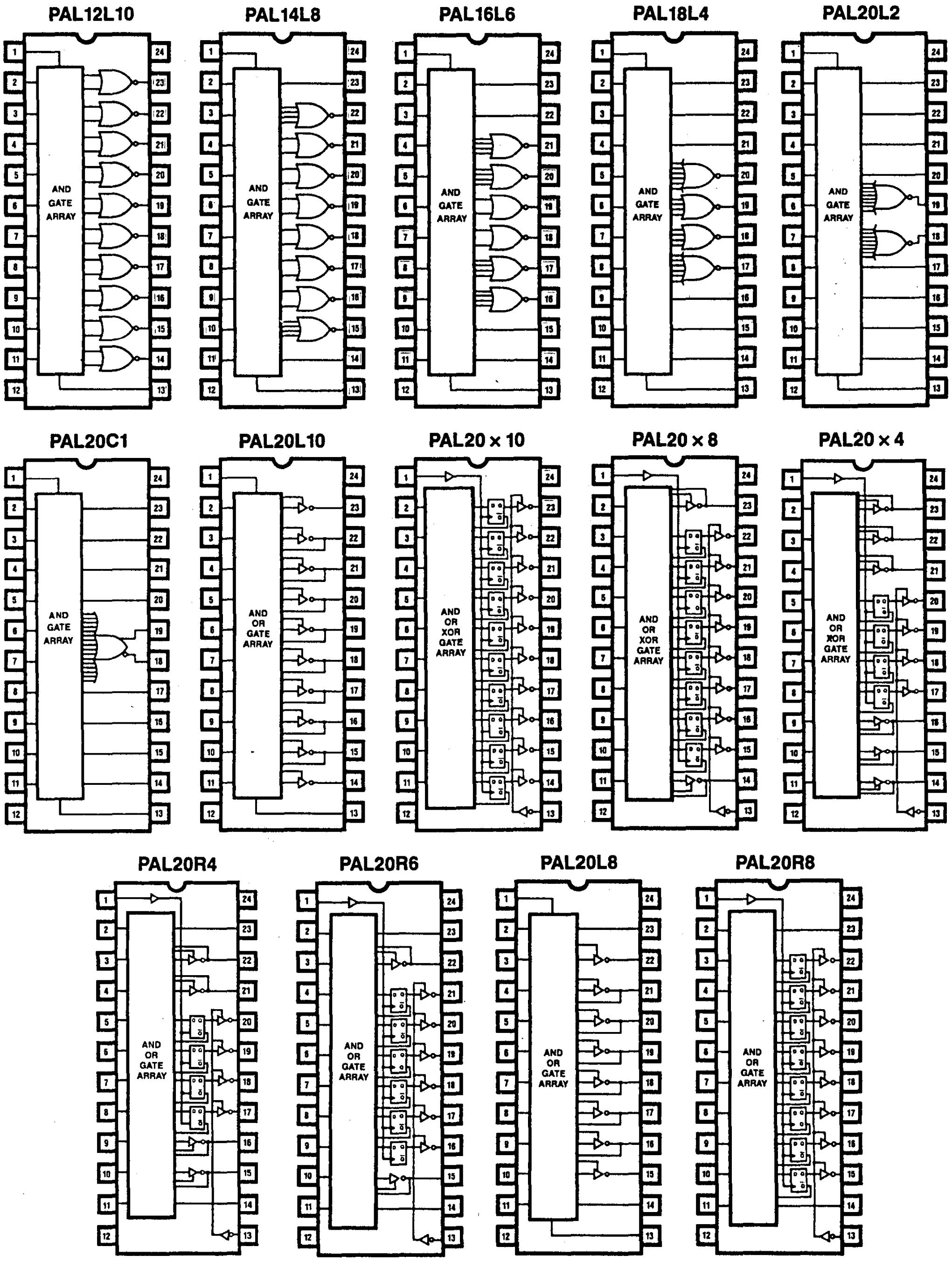


Figure 4.2.5 PAL Device Logic Symbols — Series 24

4.3 THE PROM FAMILY

National's broad PROM family extends from a 32 × 8 bit (256 bit) PROM to a 4096 × 8 bit (32K) PROM. Only the low density byte-wide PROMs are considered here for programmable logic applications. The products in this category are shown in Table 4.3.1.

Part No.	Density	No. of Inputs	No. of Outputs	No. of Product Terms/ Output	No. of Pins	T_{AA} Max (ns)	I_{CC} Max (mA)
74S288	256 Bit (32 × 8)	5	8	32	16	35	110
87X288B	256 Bit (32 × 8)	5	8	32	16	15	140
74LS471	2K (256 × 8)	8	8	256	20	60	100
74LS472	4K (512 × 8)	9	8	512	20	60	155
74S472A	4K (512 × 8)	9	8	512	20	50	155
74S472B	4K (512 × 8)	9	8	512	20	35	155
74S474	4K (512 × 8)	9	8	512	24	65	170
74S474A	4K (512 × 8)	9	8	512	24	45	125
74S474B	4K (512 × 8)	9	8	512	24	35	170
87SR474	4K (12 × 8)	9	8	512	24*	35	185
87SR476	4K (512 × 8)	9	8	512	24*	35	185
87SR25	4K (512 × 8)	9	8	512	24*	35	185

Military versions are also available. Above data is commercial.
**24 Pin Narrow Dual-In-Line Package*

Table 4.3.1 PROM Configurations

Size			DIP	Part	TAA	TEA	ICC	Temperature
(Bits)	Organization		(Pins)	Number	(Max.) in nS	(Max.) in nS	(Max.) in mA	Celsius
32 x 8 Standard PROMs								
256	32x8	OC	16	DM54S188	45	30	110	–55 to +125
	32x8	OC	16	DM74S188	35	20	110	0 to +70
	32x8	TS	16	DM54S288	45	30	110	–55 to +125
	32x8	TS	16	DM74S288	35	20	110	0 to +70
32 x 8 Ultra High-Speed PROMs								
256	32x8	TS	16	PL77X288	20	15	140	–55 to +125
	32x8	TS	16	PL87X288	15	12	140	0 to +70
256 x 8 Standard PROMs								
2048	256x8	TS	20	DM54LS471	70	35	100	–55 to +125
	256x8	TS	20	DM74LS471	60	30	100	0 to +70
512 x 8 Standard PROMs								
4096	512x8	OC	20	DM54S473	75	35	155	–55 to +125
	512x8	OC	20	DM74S473	60	30	155	0 to +70
	512x8	TS	20	DM54S472	75	35	155	–55 to +125
	512x8	TS	20	DM74S472	60	30	155	0 to +70
	512x8	OC	20	DM54S473A	60	35	155	–55 to +125
	512x8	OC	20	DM74S473A	45	25	155	0 to +70
	512x8	TS	20	DM54S472A	60	35	155	–55 to +125
	512x8	TS	20	DM74S472A	45	25	155	0 to +70
	512x8	TS	20	DM54S472B	50	35	155	–55 to +125
	512x8	TS	20	DM74S472B	35	25	155	0 to +70
	512x8	OC	24	DM54S475	75	40	170	–55 to +125
	512x8	OC	24	DM74S475	65	35	170	0 to +70
	512x8	TS	24	DM54S474	75	40	170	–55 to +125
	512x8	TS	24	DM74S474	65	35	170	0 to +70
	512x8	OC	24	DM54S475A	60	35	170	–55 to +125
	512x8	OC	24	DM74S475A	45	25	170	0 to +70
	512x8	TS	24	DM54S474A	60	35	170	–55 to +125
	512x8	TS	24	DM74S474A	45	25	170	0 to +70
	512x8	TS	24	DM54S474B	50	35	170	–55 to +125
	512x8	TS	24	DM74S474B	35	25	170	0 to +70
512 x 8 Registered PROMs								
4076	512x8	REG	24*	DM77SR474	40**	30	185	–55 to +125
	512x8	REG	24*	DM87SR474	35**	25	185	0 to +70
	512x8	REG	24*	DM77SR476	40**	30	185	–55 to +125
	512x8	REG	24*	DM77SR25	40**	30	185	–55 to +125
	512x8	REG	24*	DM87SR476	35**	25	185	0 to +70
	512x8	REG	24*	DM87SR25	35**	25	185	0 to +70

* 300 mil wide package.

** Set-up time.

Table 4.3.2 PROM Products for Logic

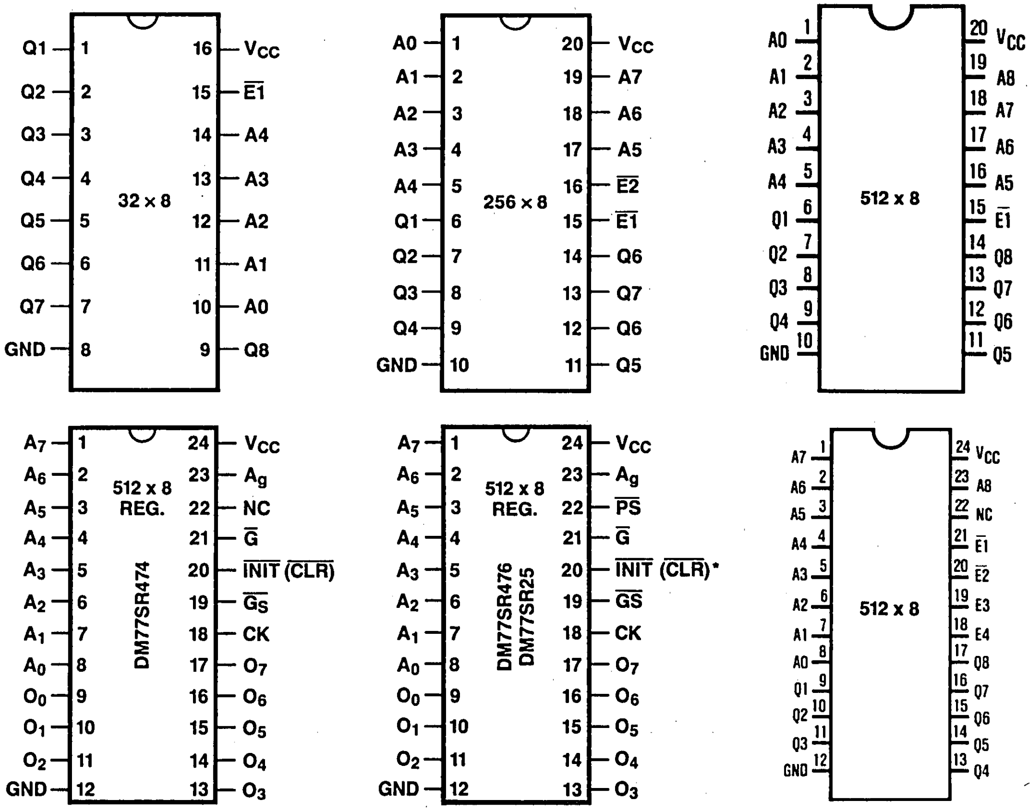


Figure 4.3.1 PROM Logic Symbols

Note:

All of the virgin devices come with their fuses intact. But for the sake of simplicity, the fuse-linked crosspoints in the array are shown unconnected.

4.4 LOGIC DIAGRAMS

The following pages show the logic diagrams of the PAL device and PROM family of programmable logic devices. The logic diagrams are ordered in the following sequence:

PAL Devices:

Figures 4.4.1—4.4.13 (20-pin PAL devices)
Figures 4.4.14—4.4.27 (24-pin PAL devices)

PROMs:

Figures 4.4.28—4.4.32

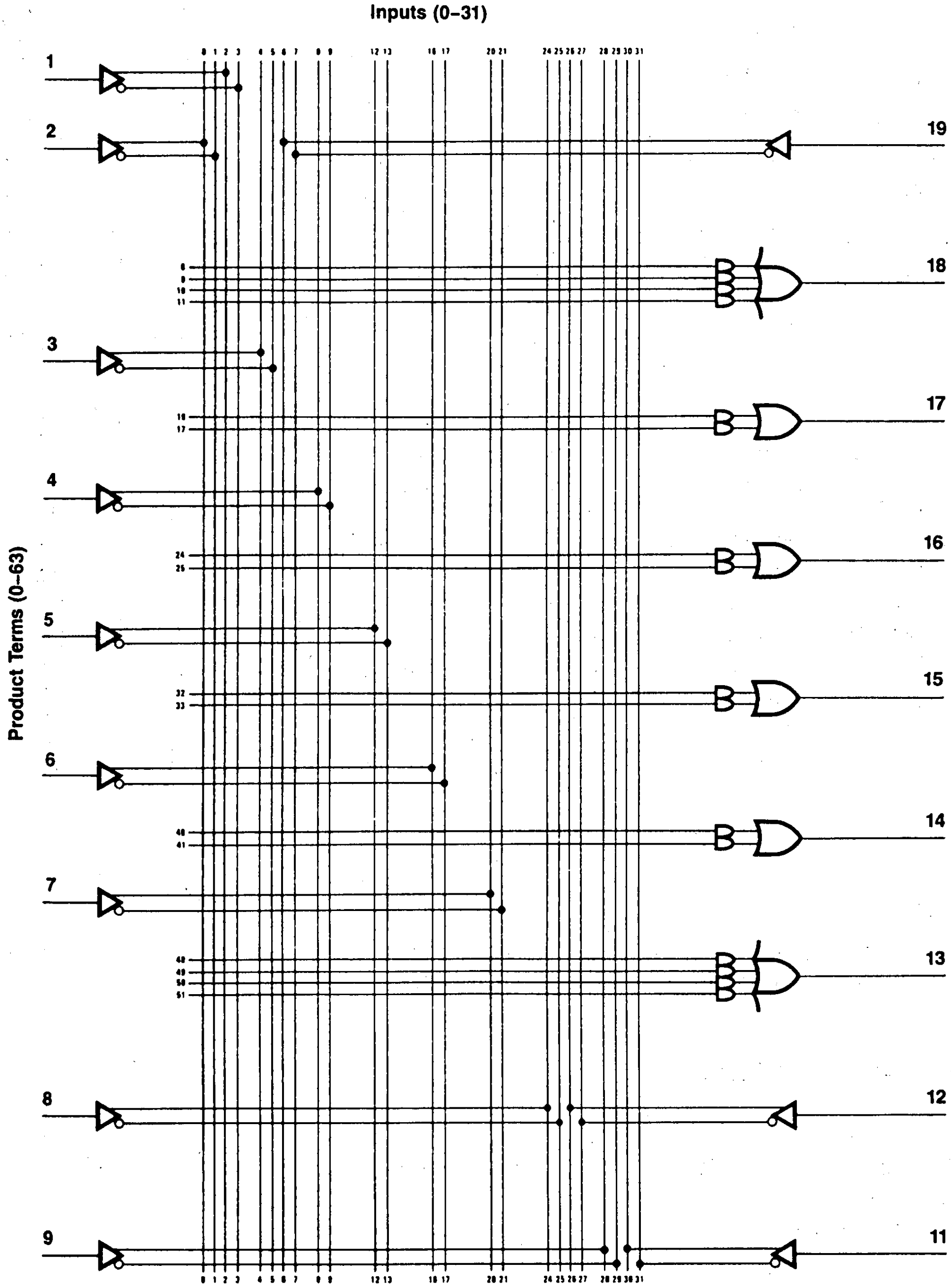


Figure 4.4.1 Logic Diagram PAL10H8

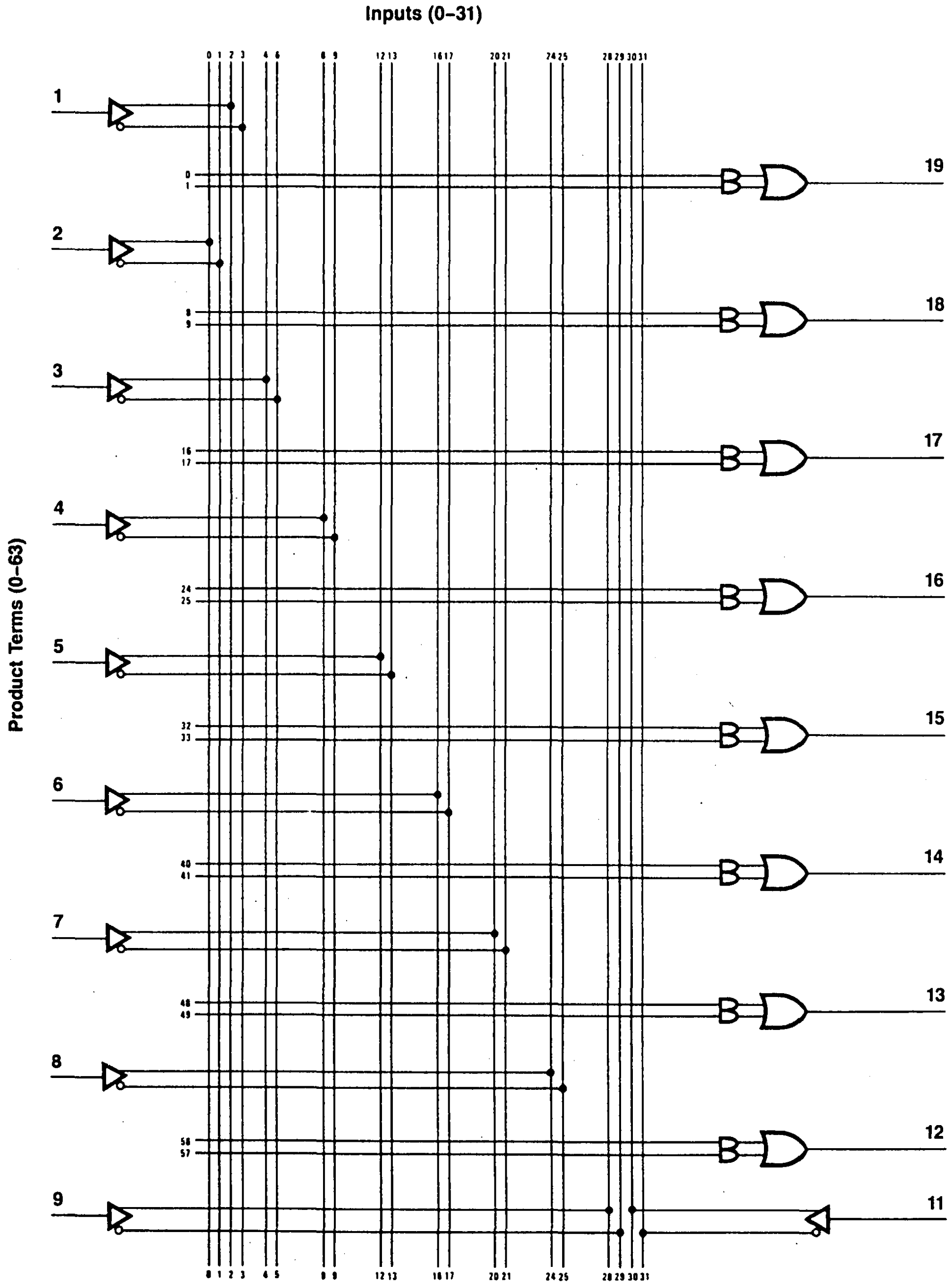


Figure 4.4.2 Logic Diagram PAL12H6

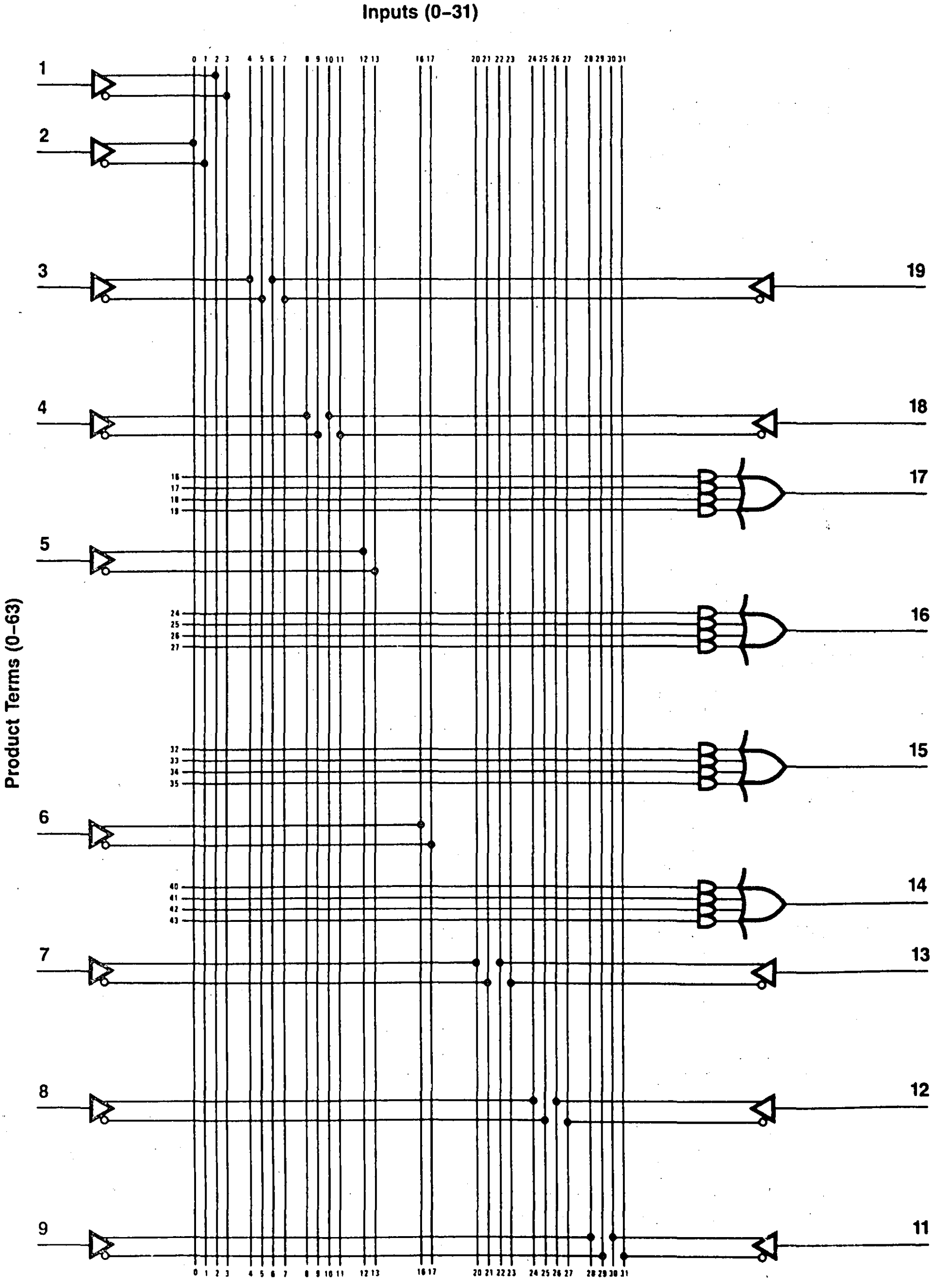


Figure 4.4.3 Logic Diagram PAL14H4

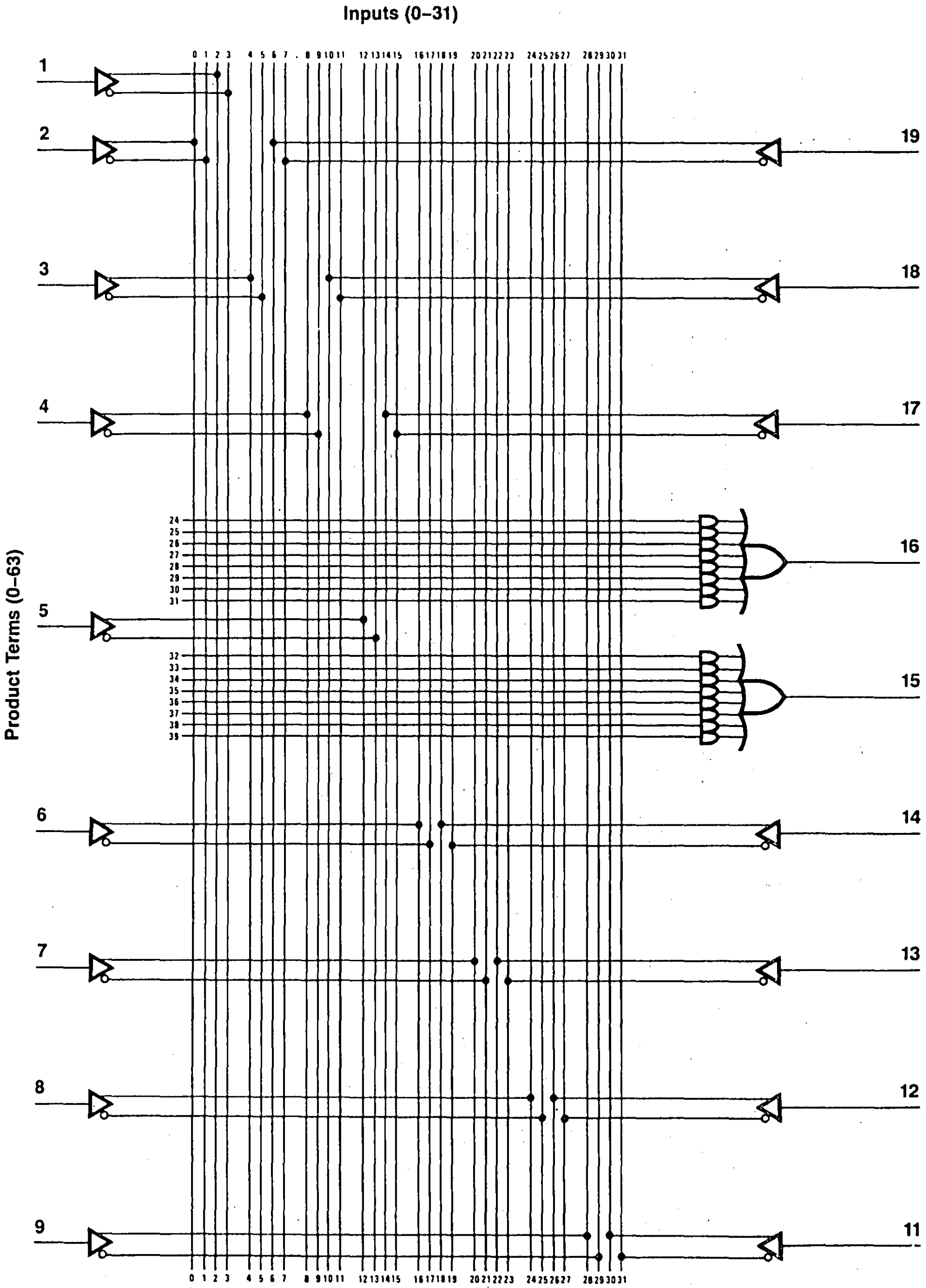


Figure 4.4.4 Logic Diagram PAL16H2

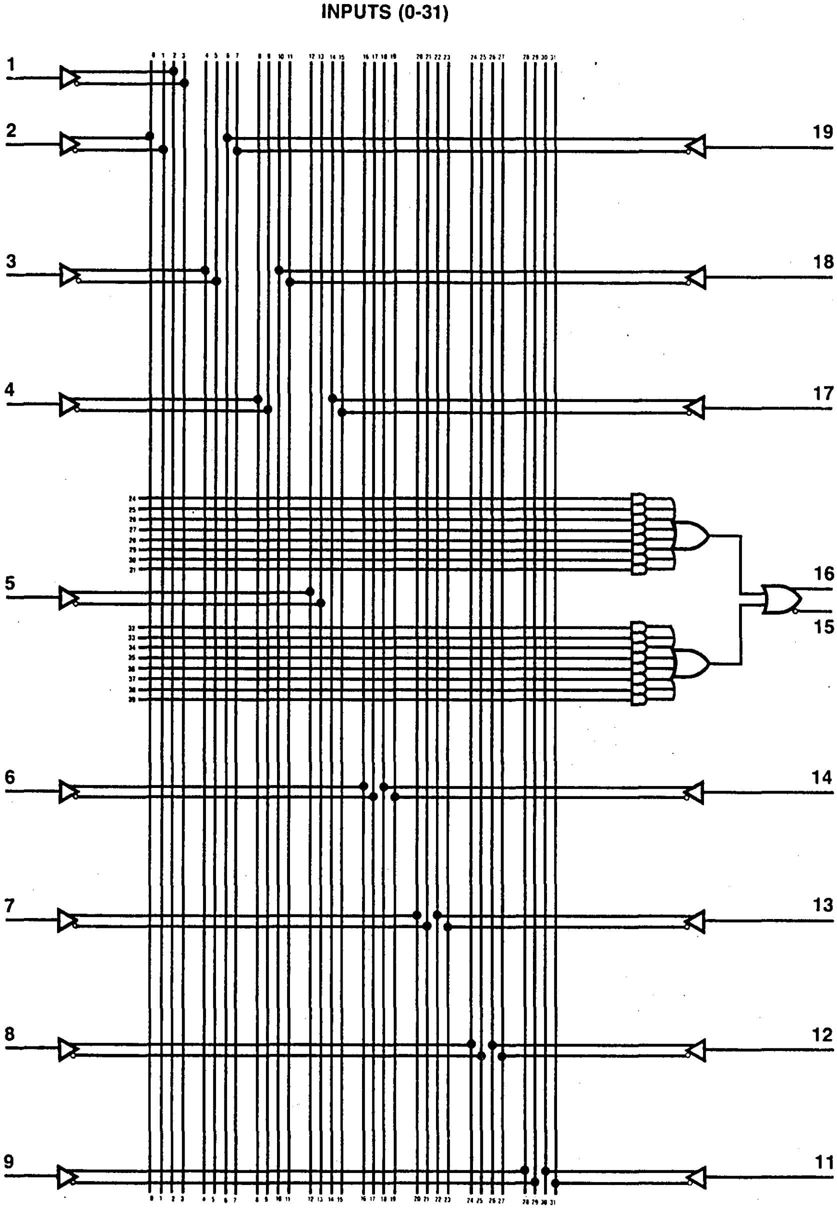


Figure 4.4.5 Logic Diagram PAL16C1

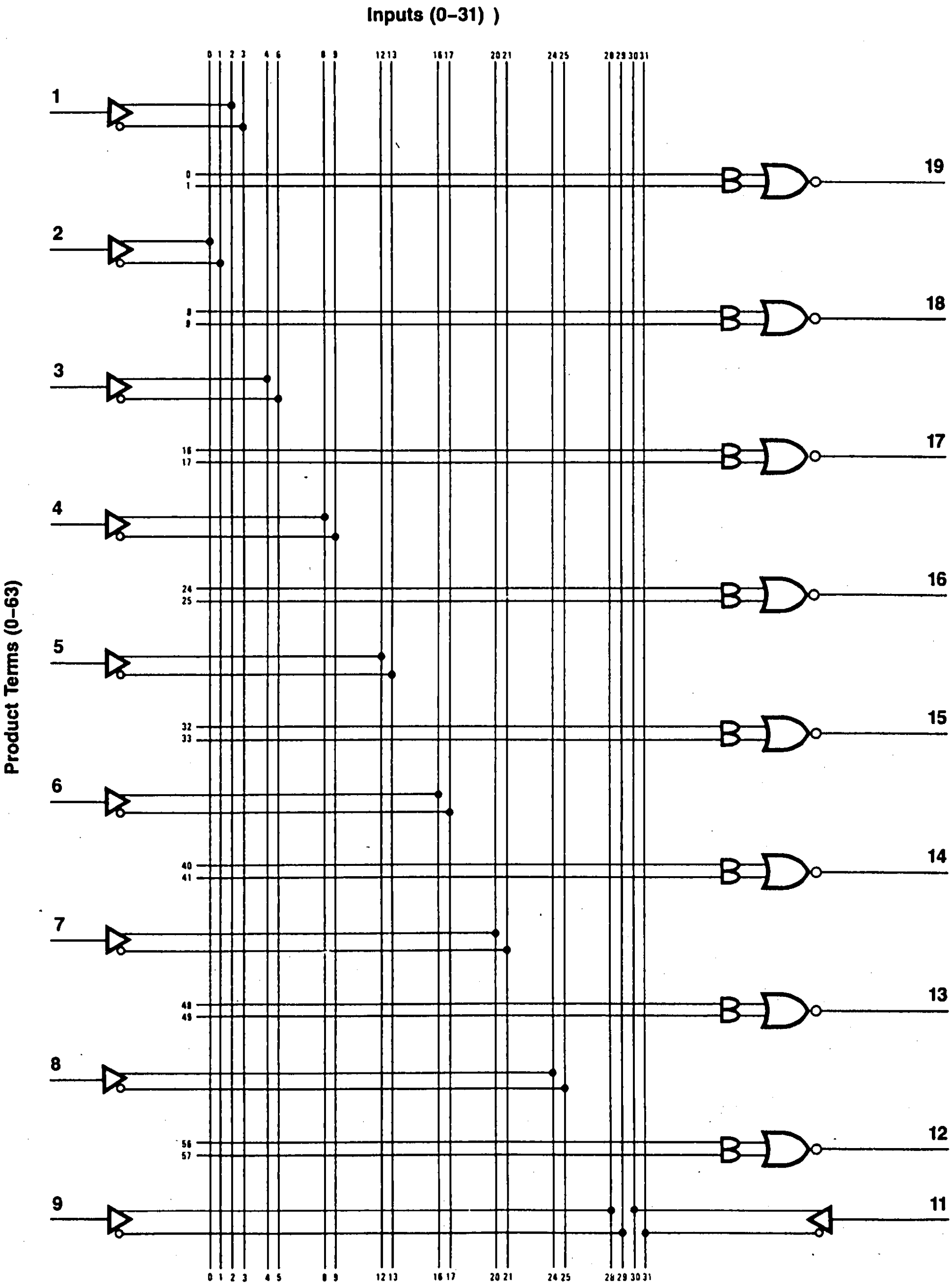


Figure 4.4.6 Logic Diagram PAL10L8

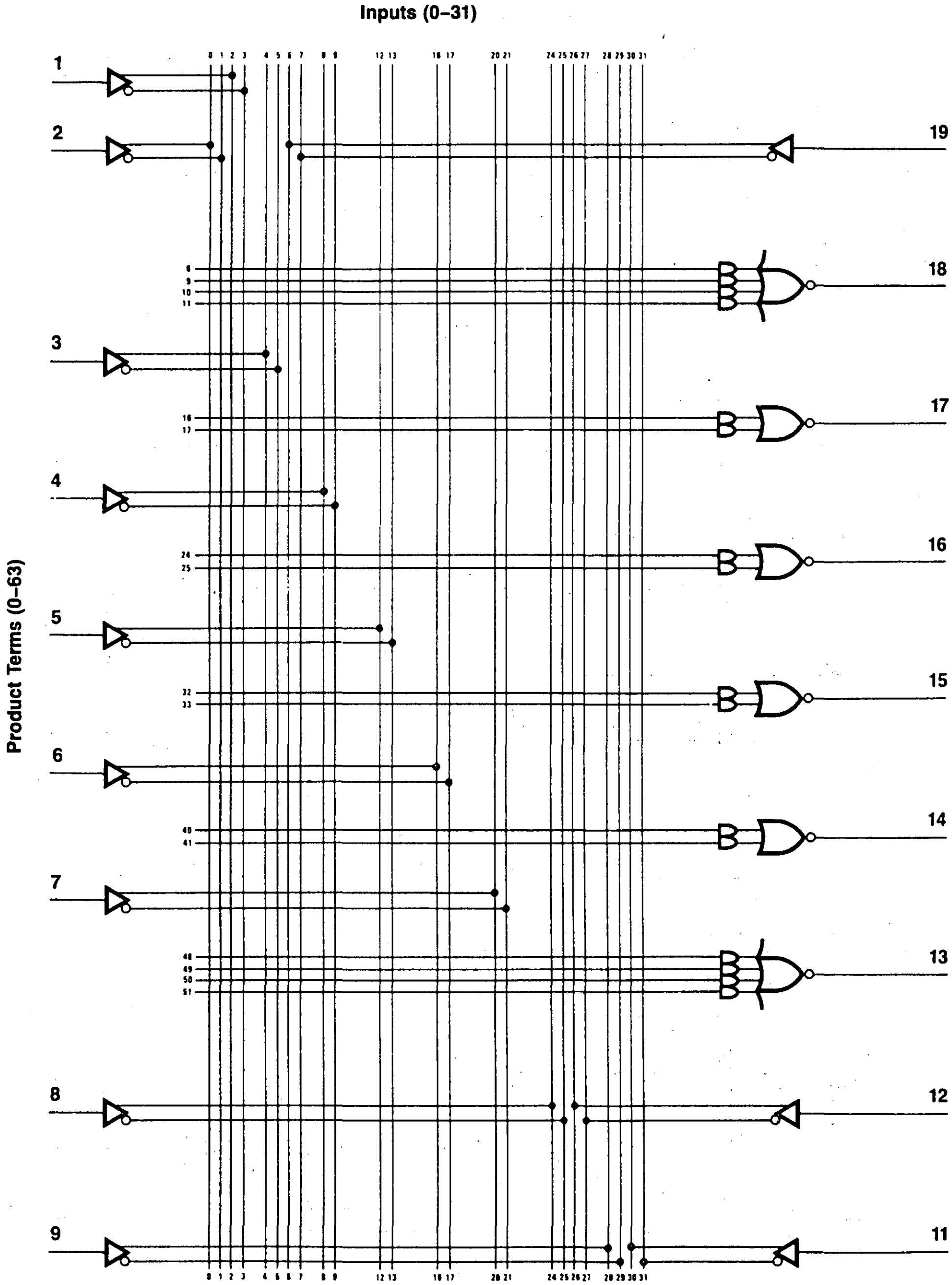


Figure 4.4.7 Logic Diagram PAL12L6

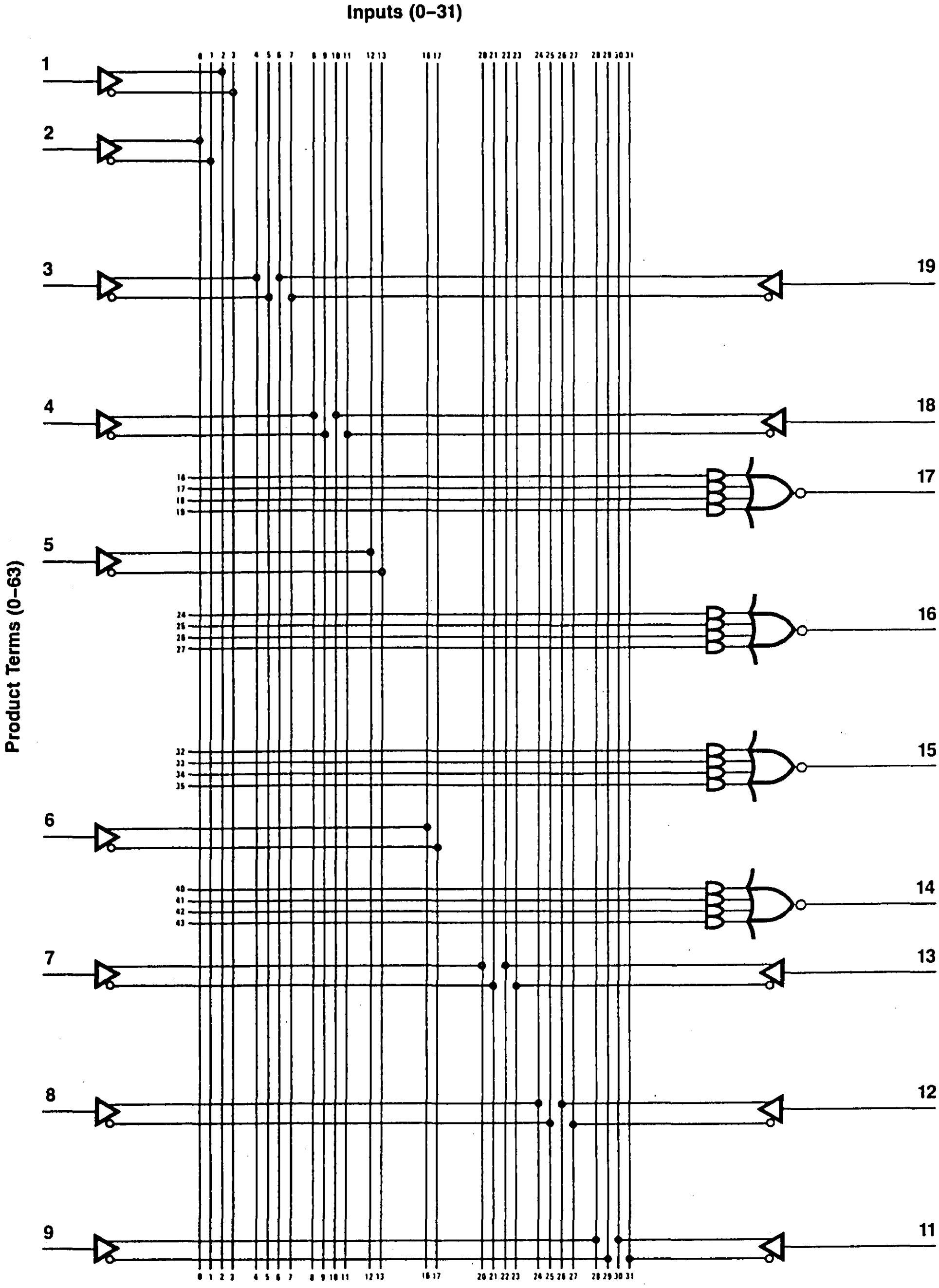


Figure 4.4.8 Logic Diagram PAL14L4

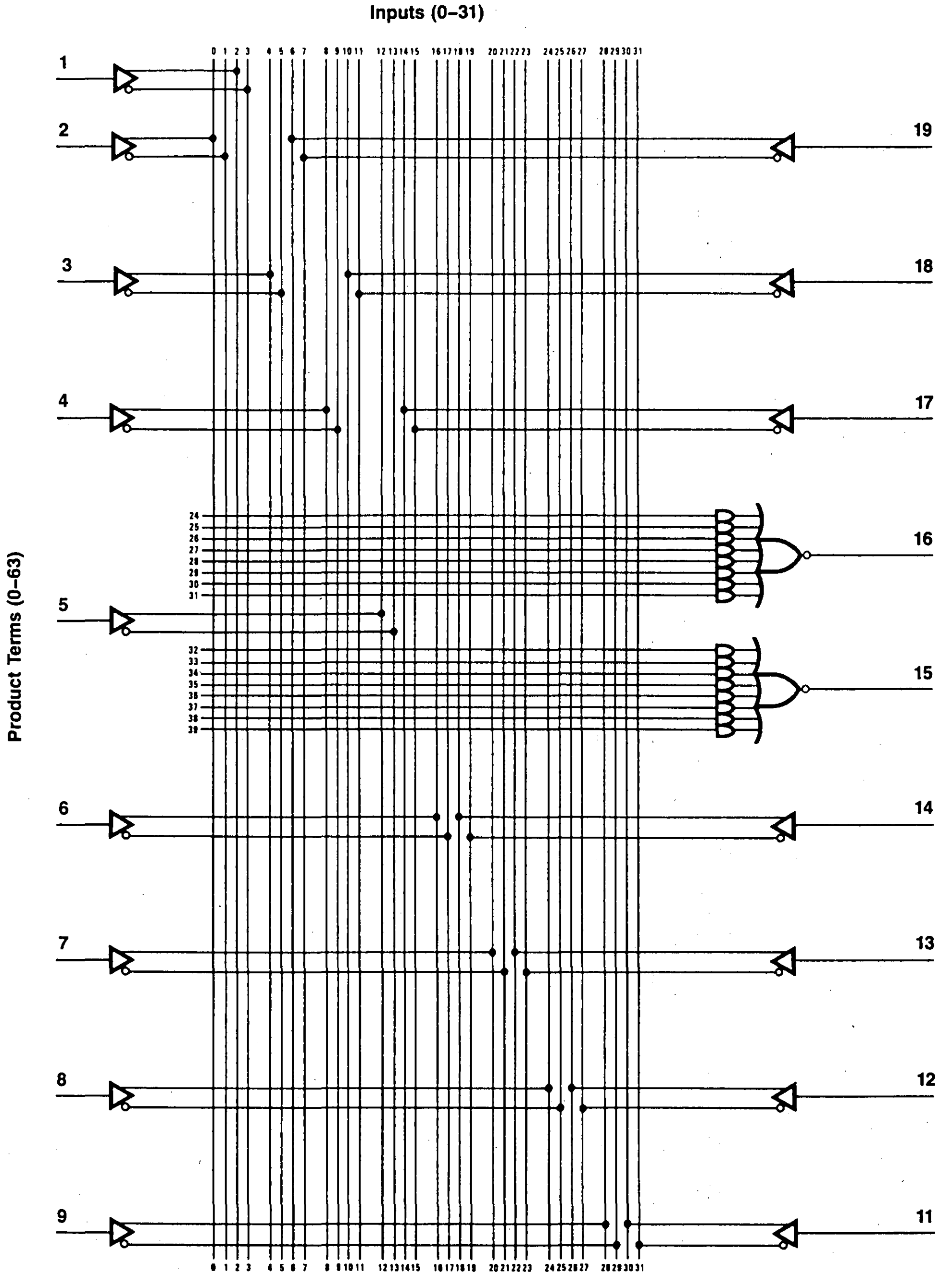


Figure 4.4.9 Logic Diagram PAL16L2

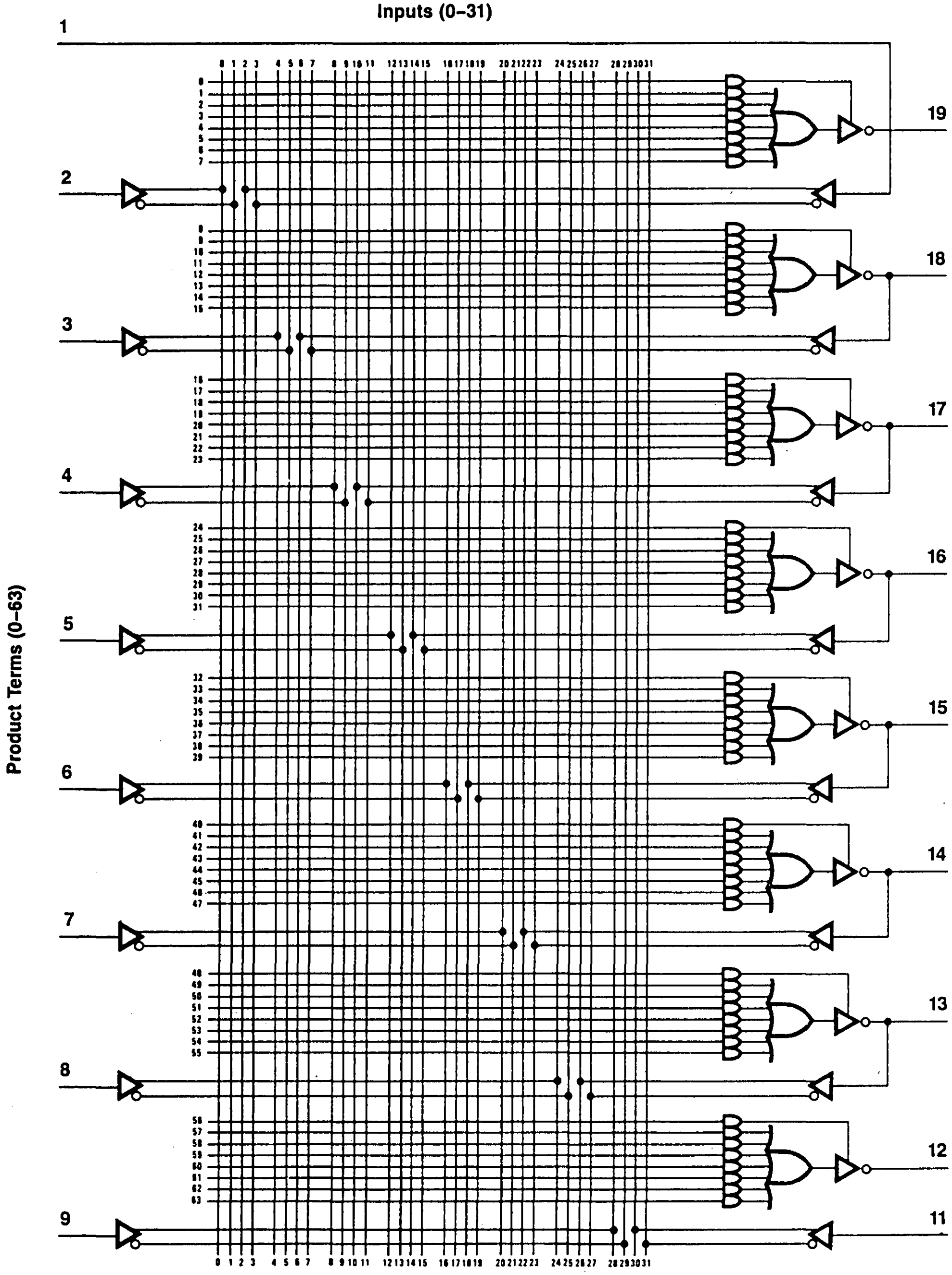


Figure 4.4.10 Logic Diagram PAL16L8

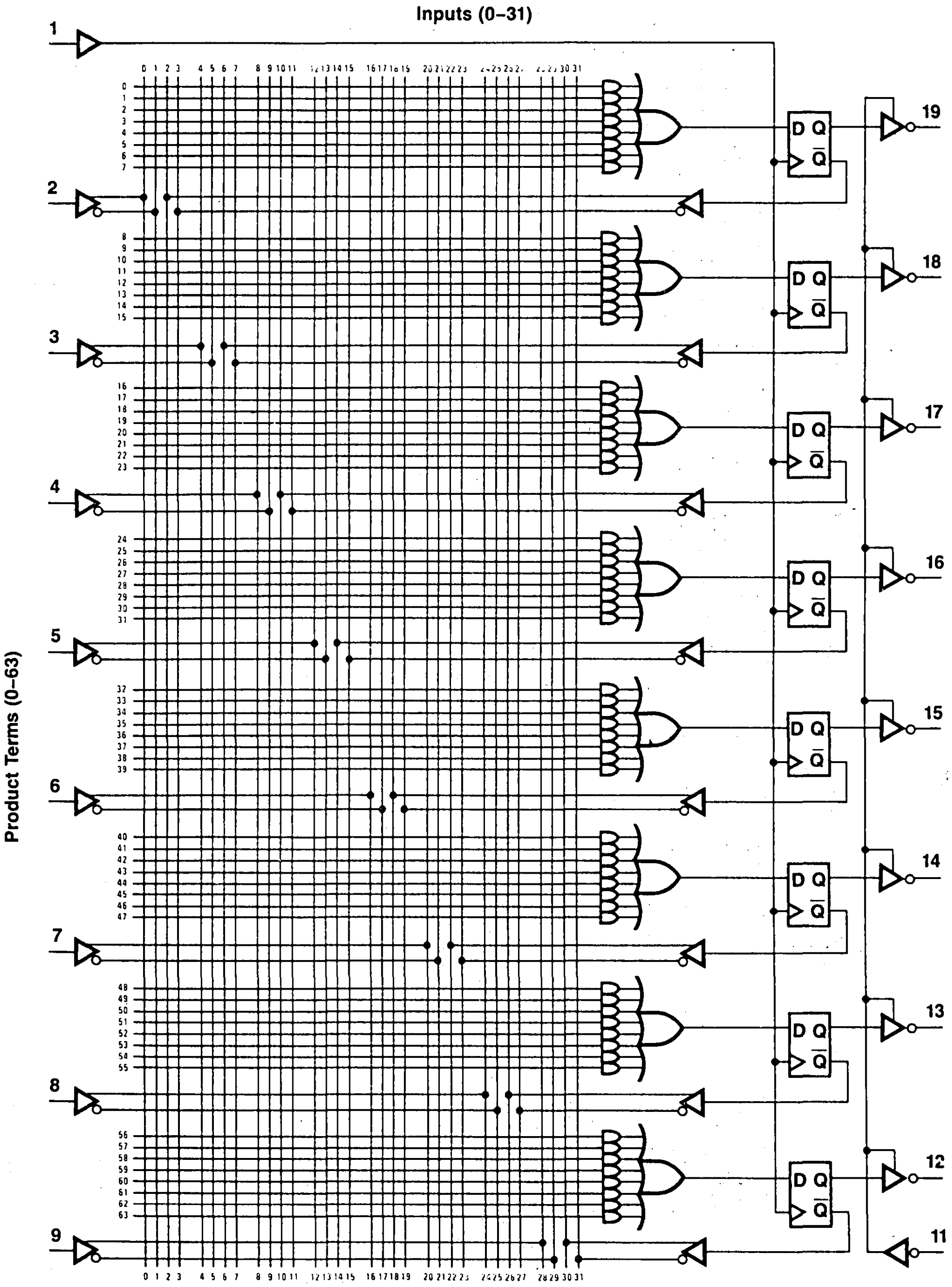


Figure 4.4.11 Logic Diagram PAL16R8

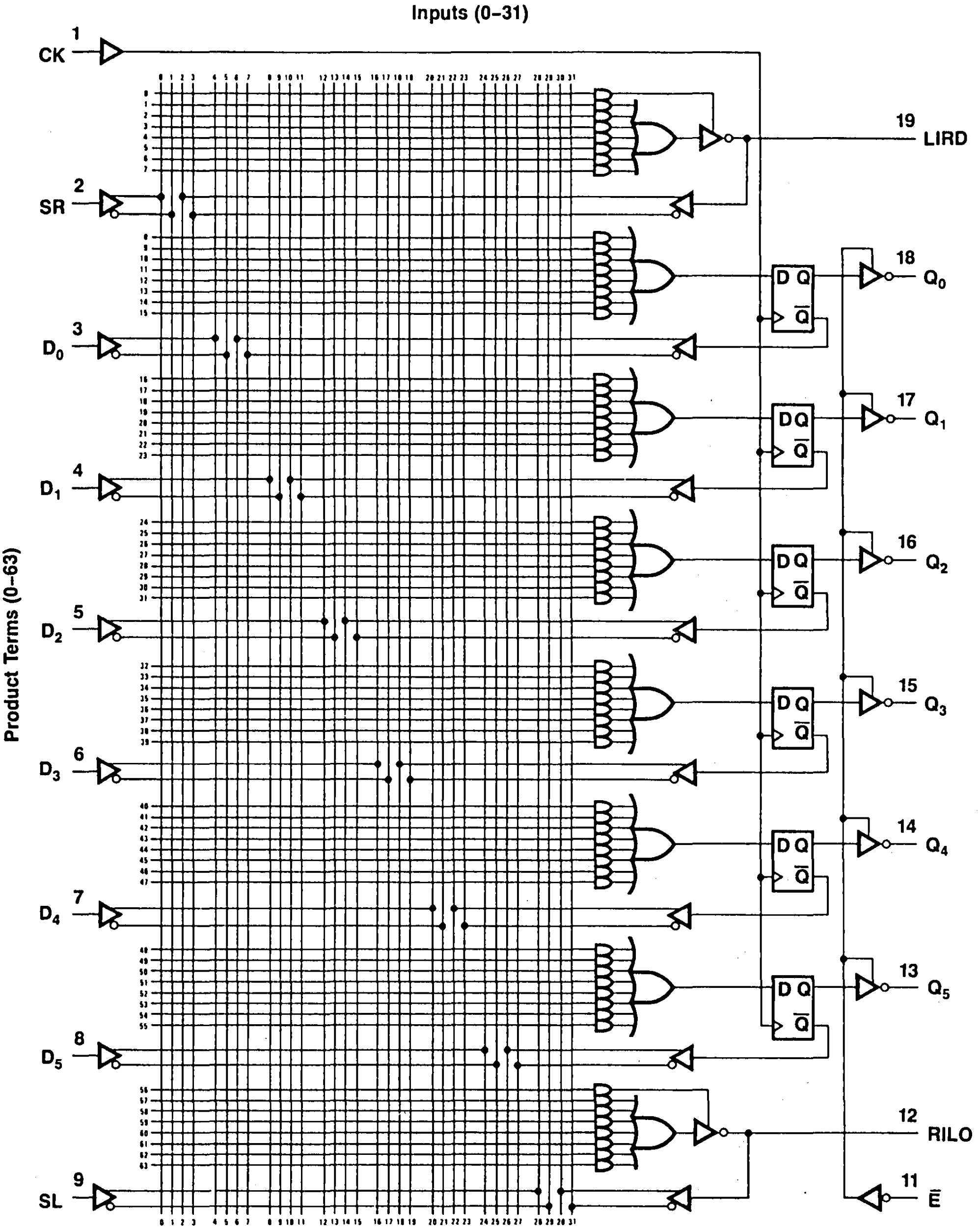


Figure 4.4.12 Logic Diagram PAL16R6

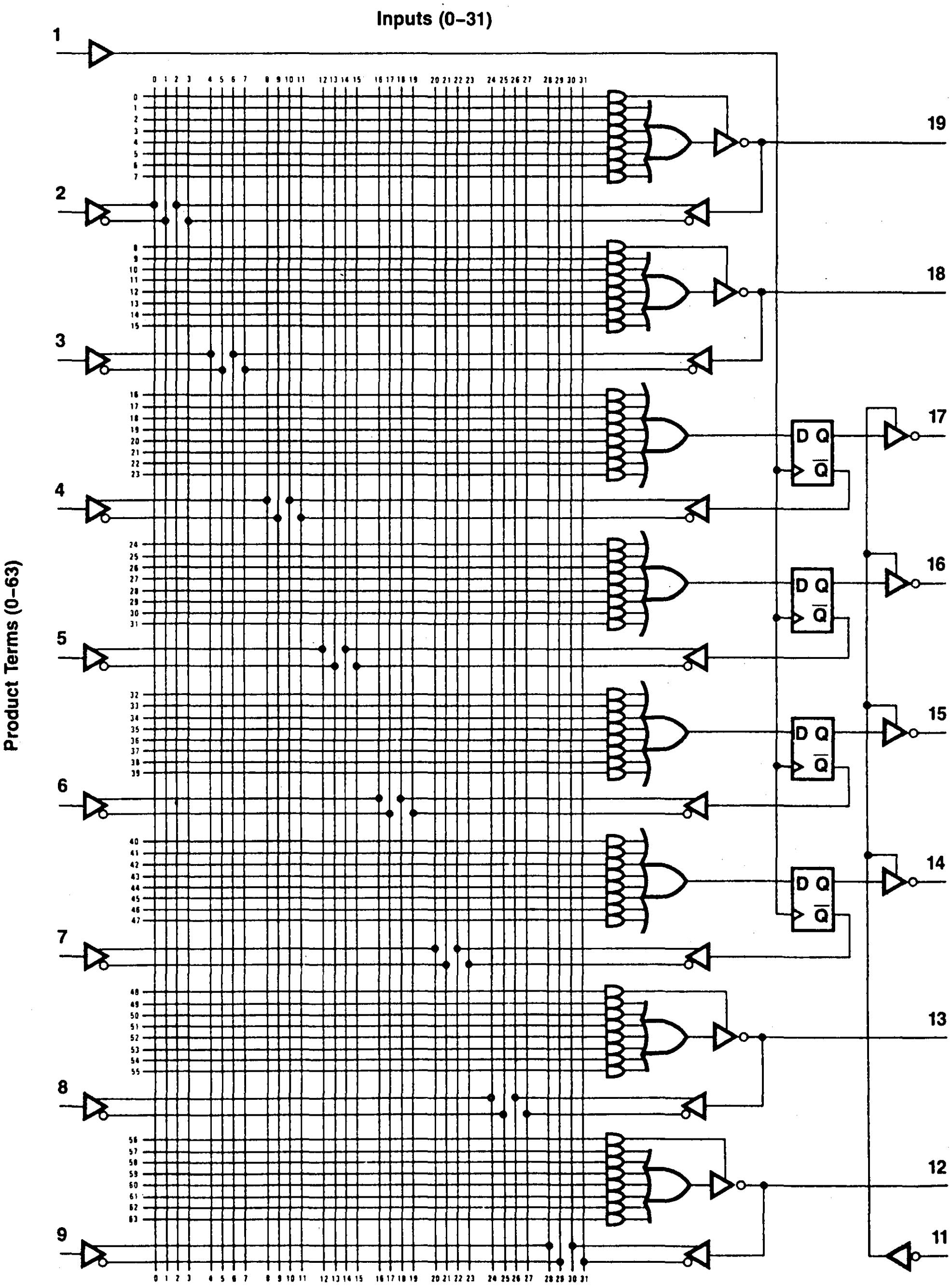


Figure 4.4.13 Logic Diagram PAL16R4

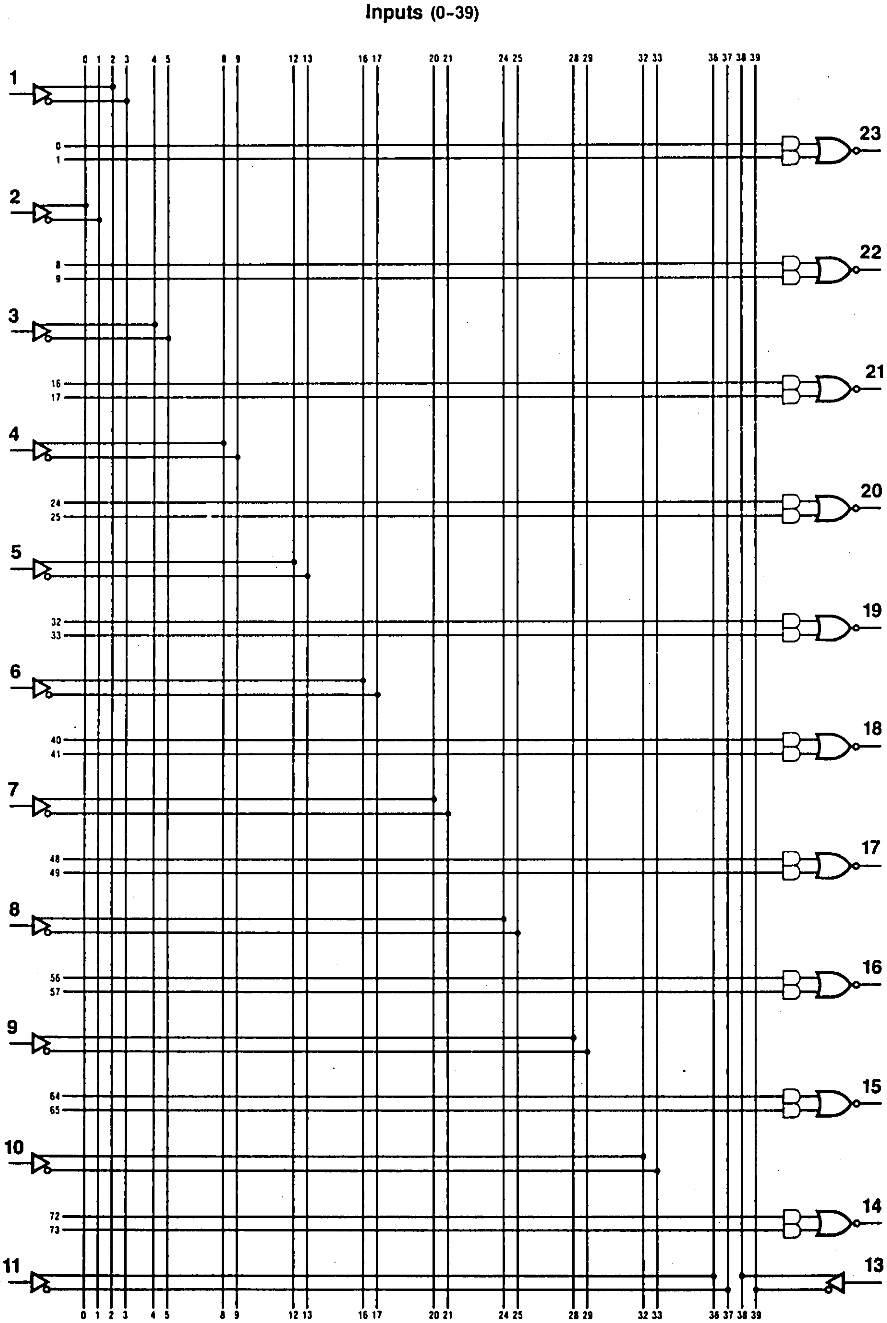


Figure 4.4.14 Logic Diagram PAL12L10

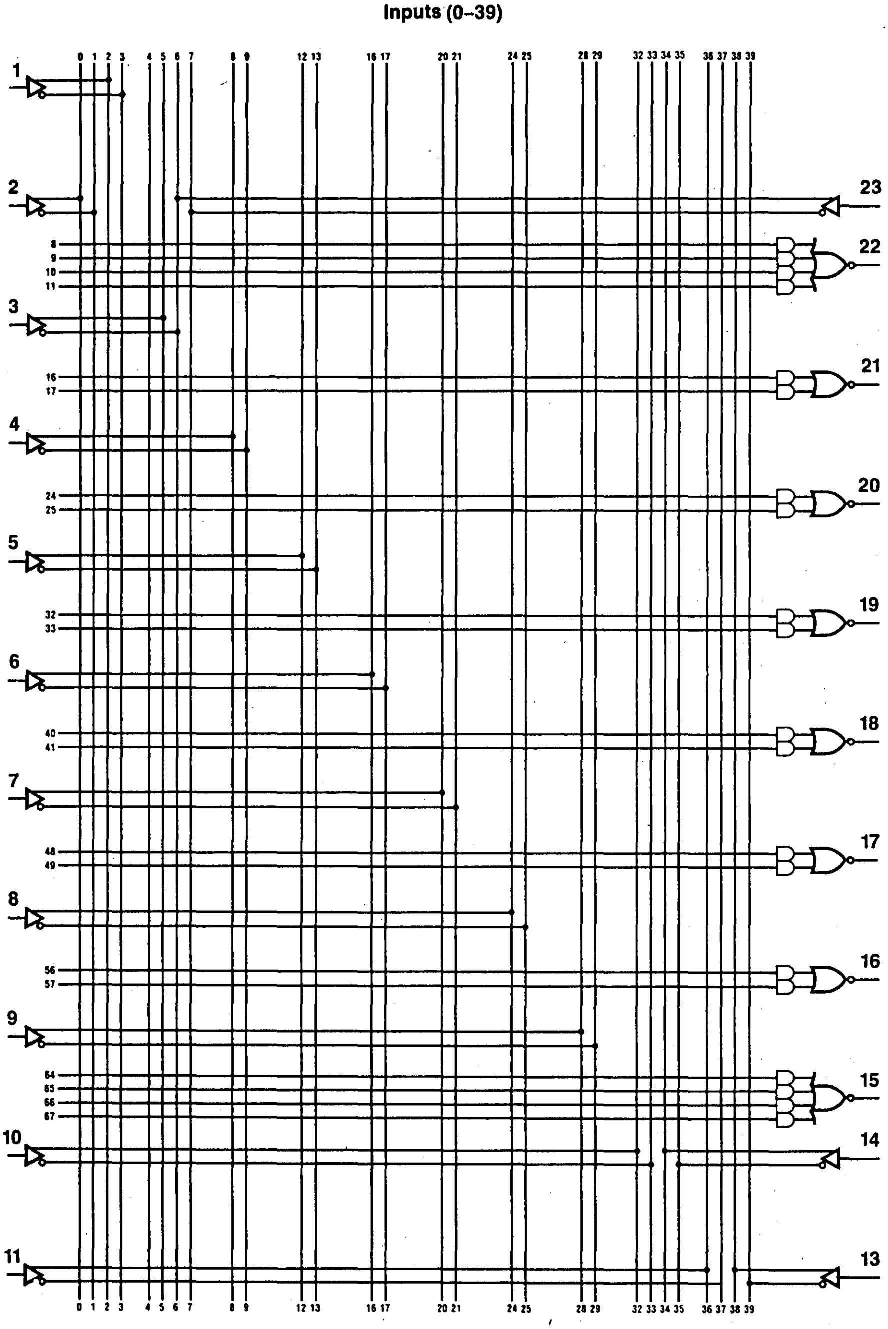


Figure 4.4.15 Logic Diagram PAL14L8

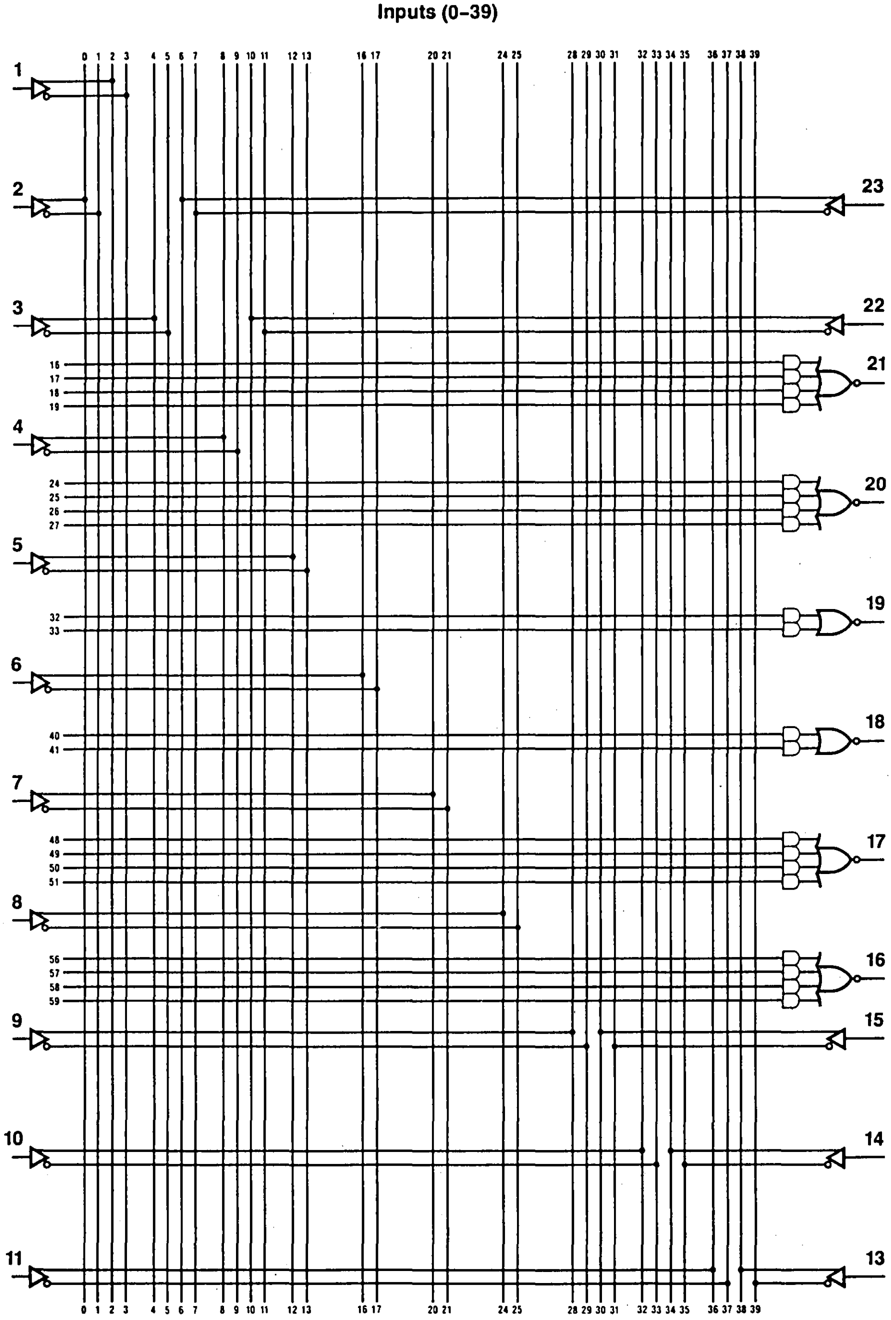


Figure 4.4.16 Logic Diagram PAL16L6

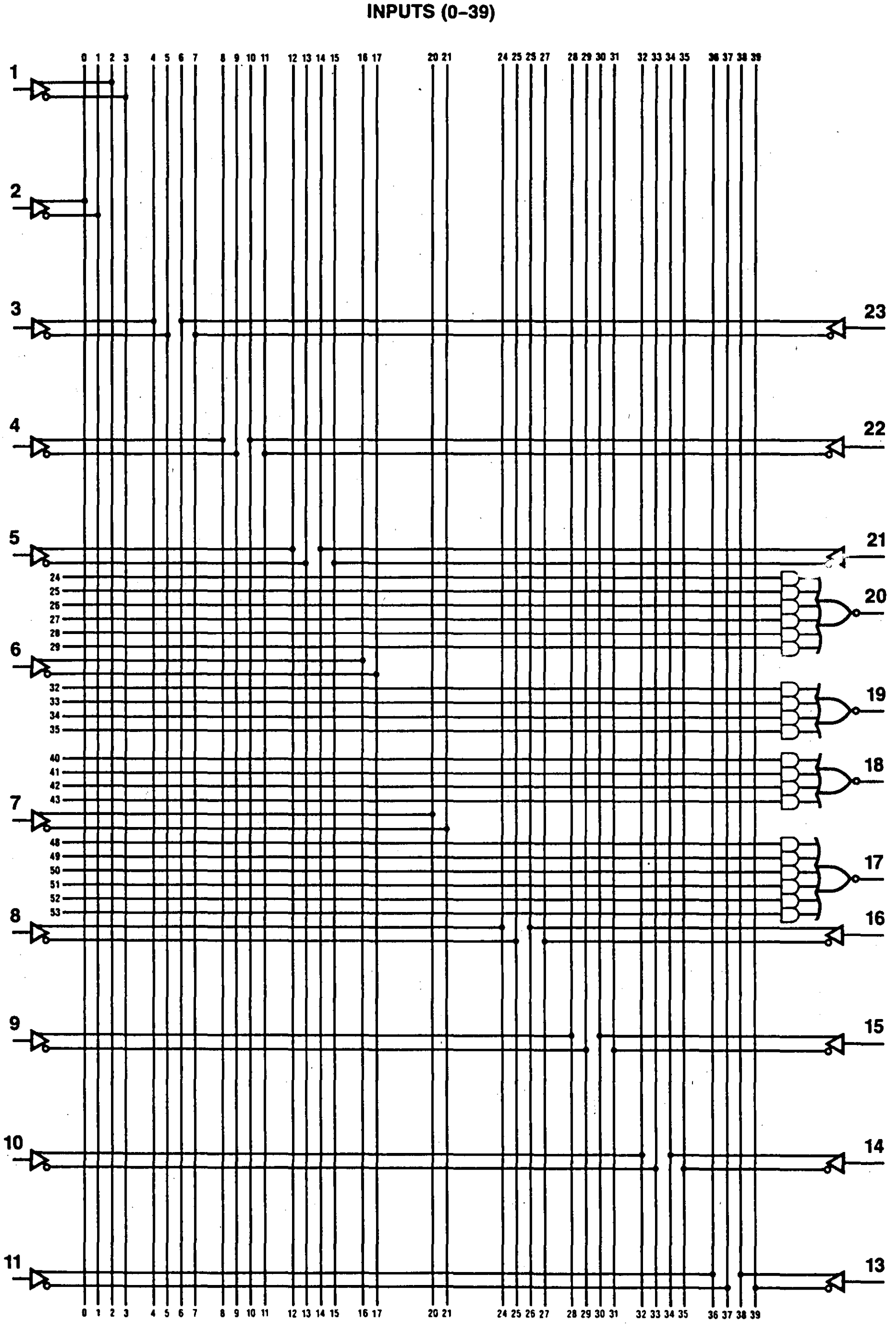


Figure 4.4.17 Logic Diagram PAL18L4

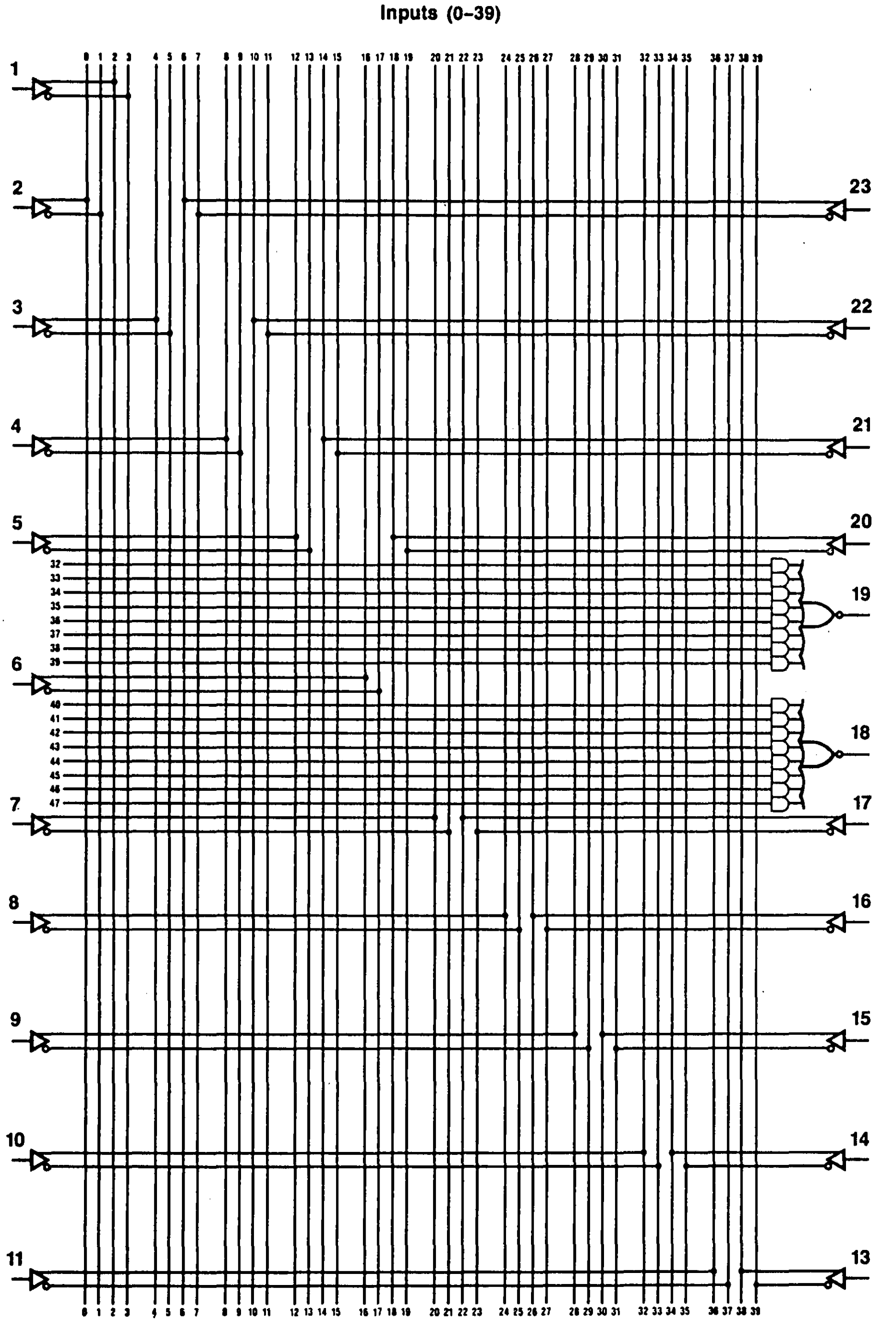


Figure 4.4.18 Logic Diagram PAL20L2

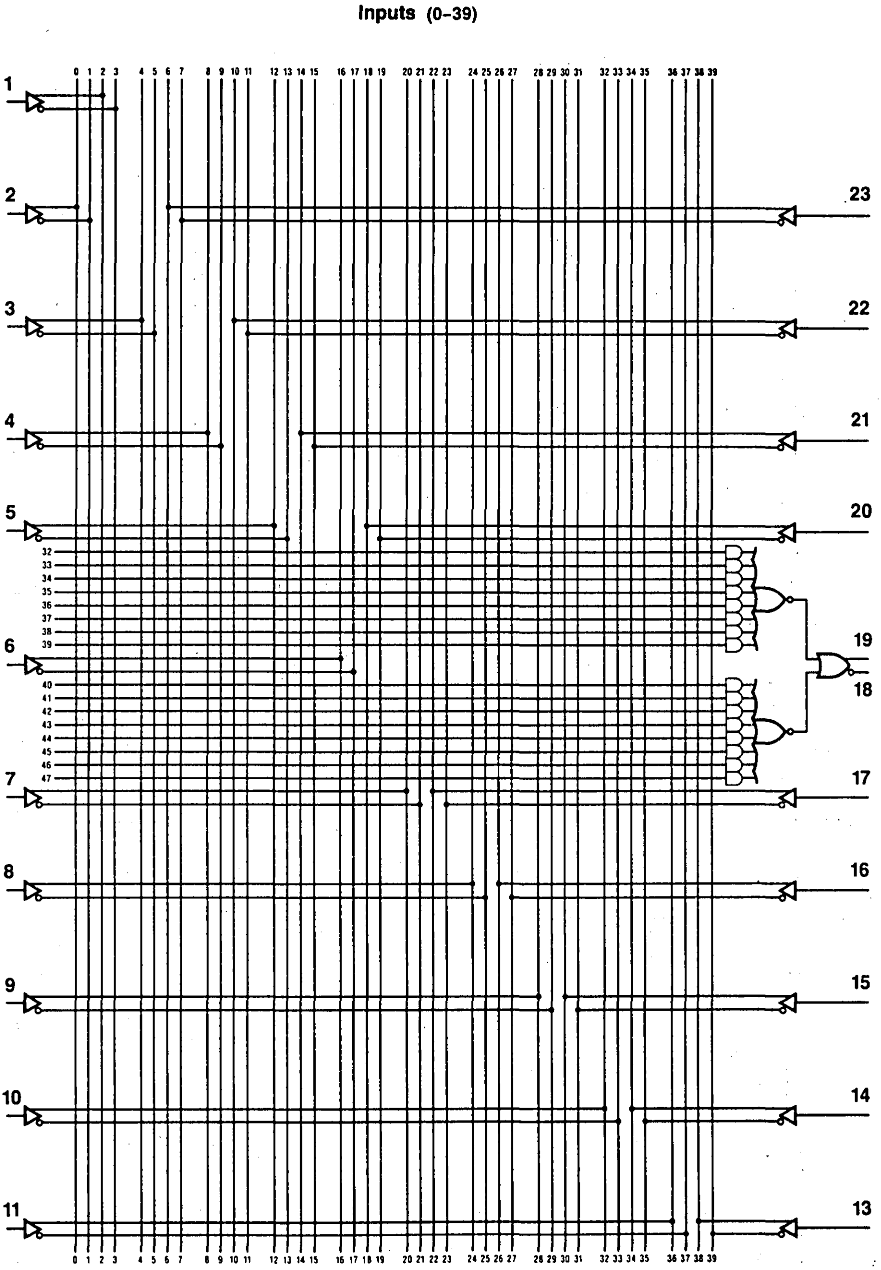


Figure 4.4.19 Logic Diagram PAL20C1

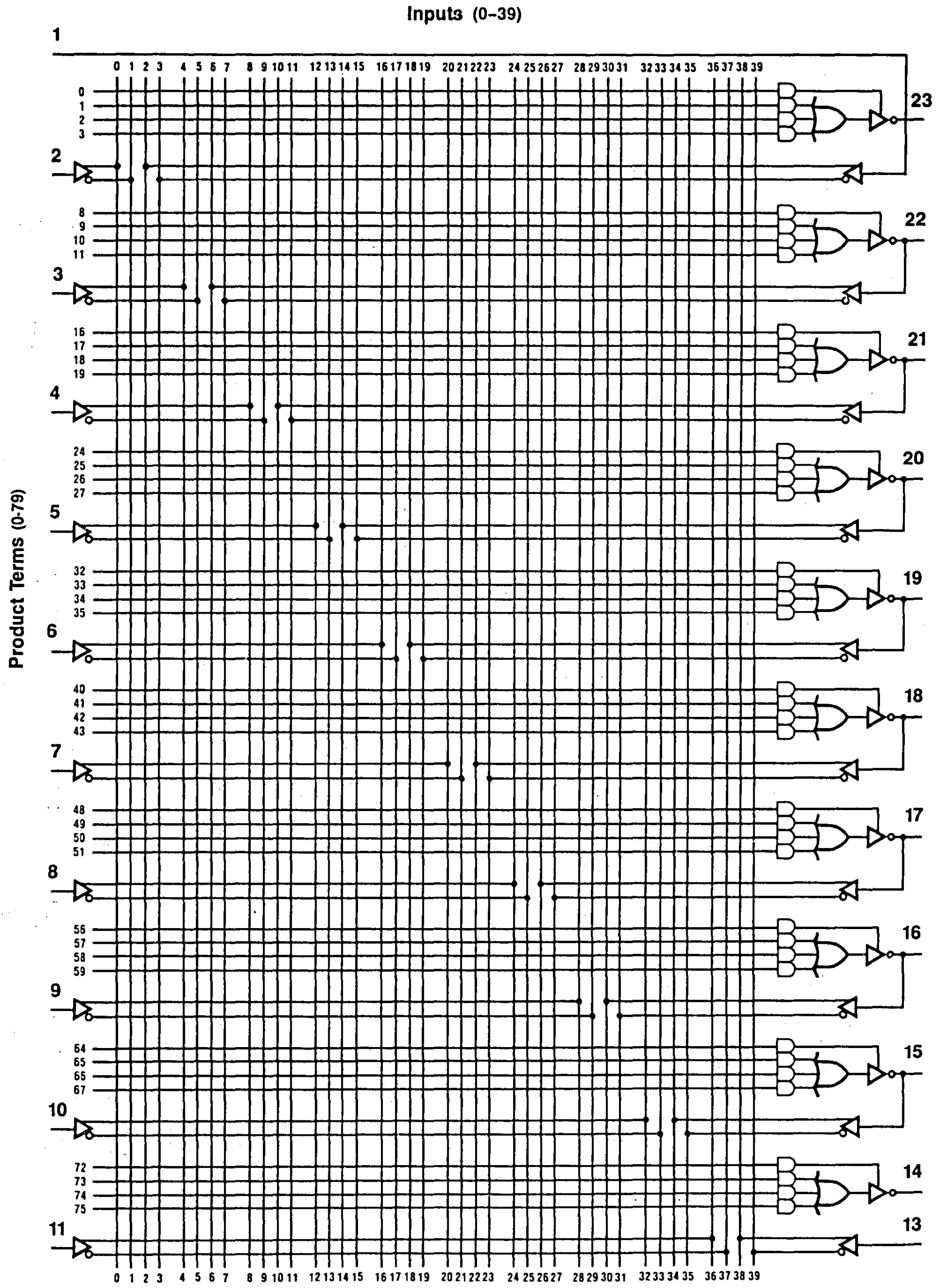


Figure 4.4.20 Logic Diagram PAL20L10

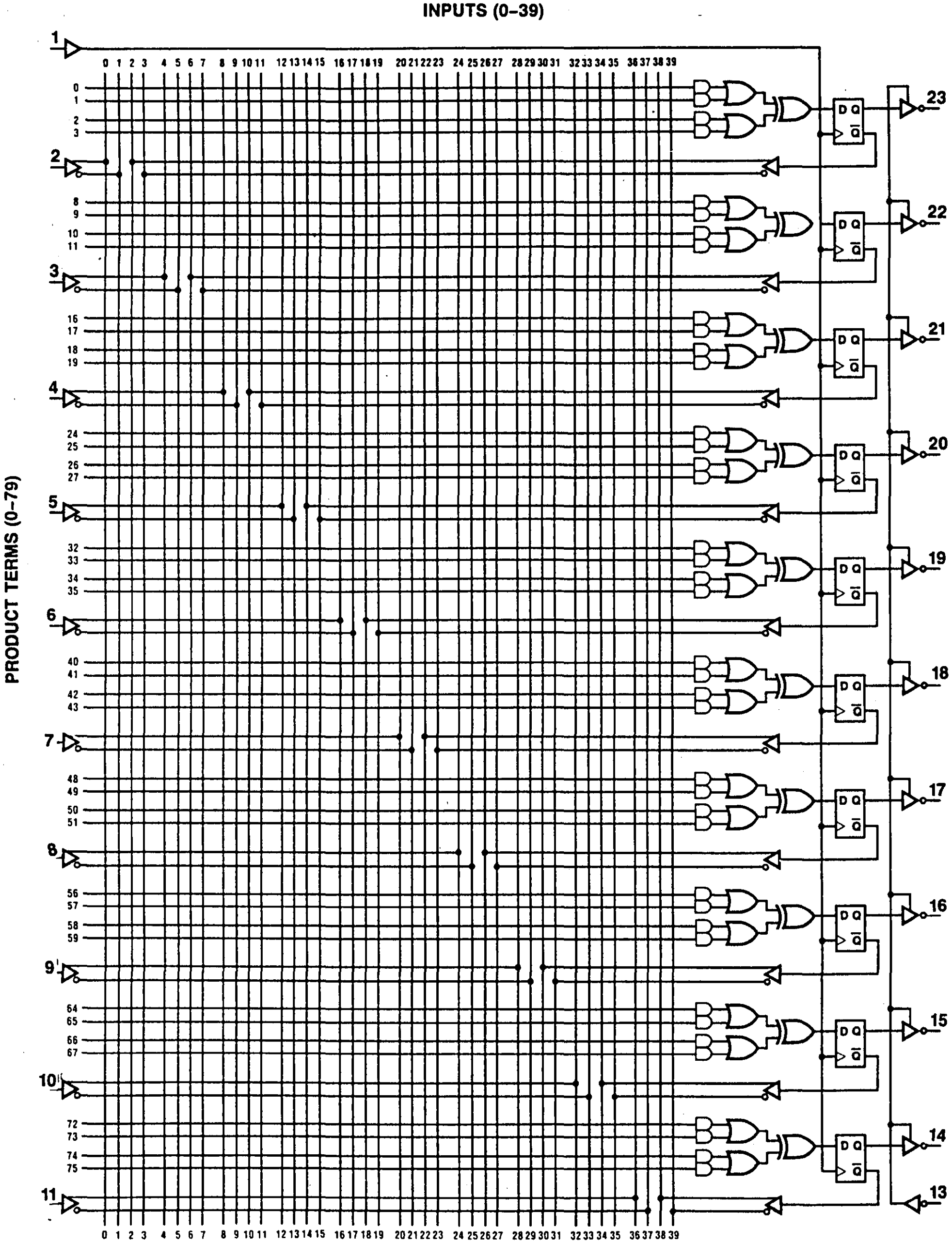


Figure 4.4.21 Logic Diagram PAL20X10

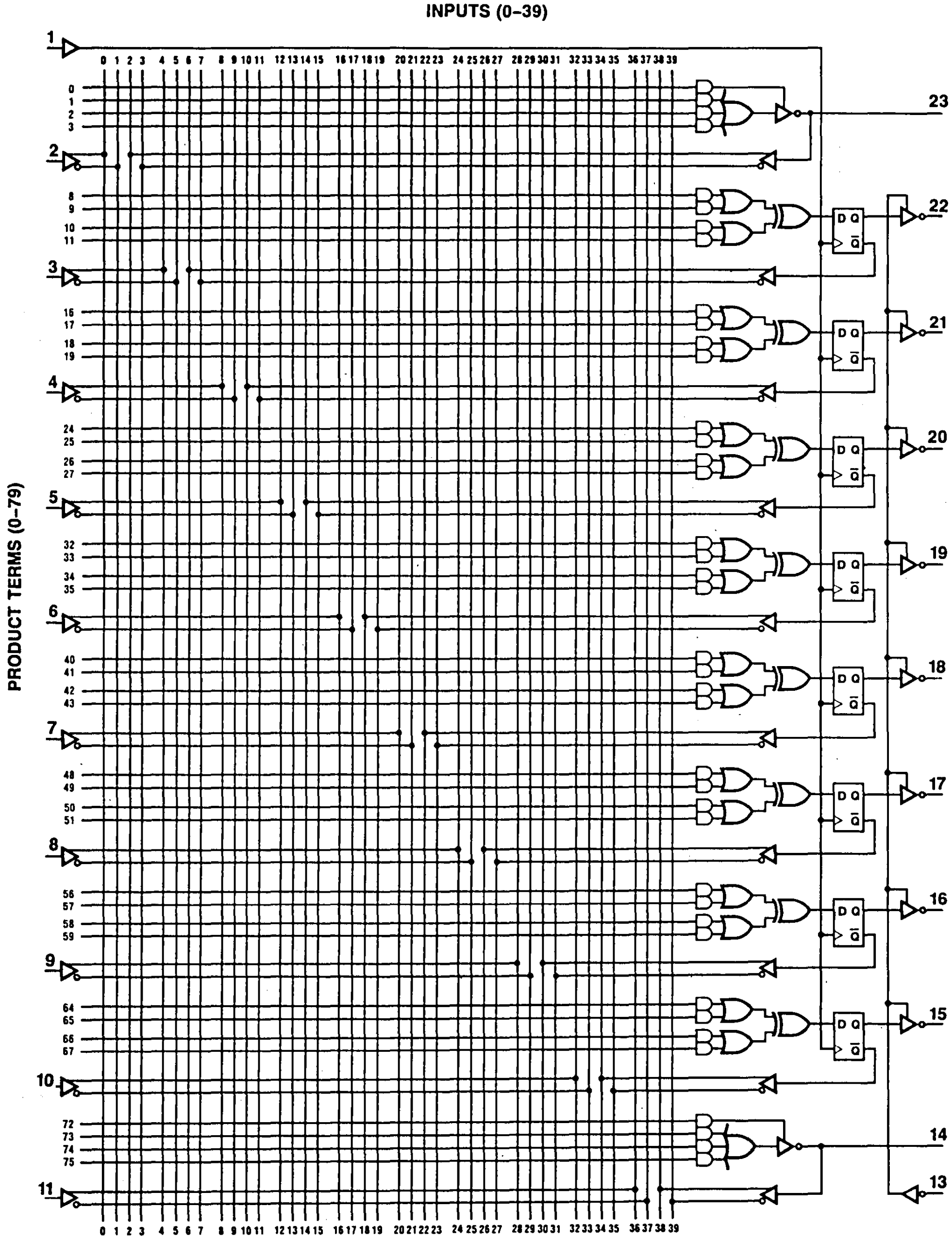


Figure 4.4.22 Logic Diagram PAL20X8

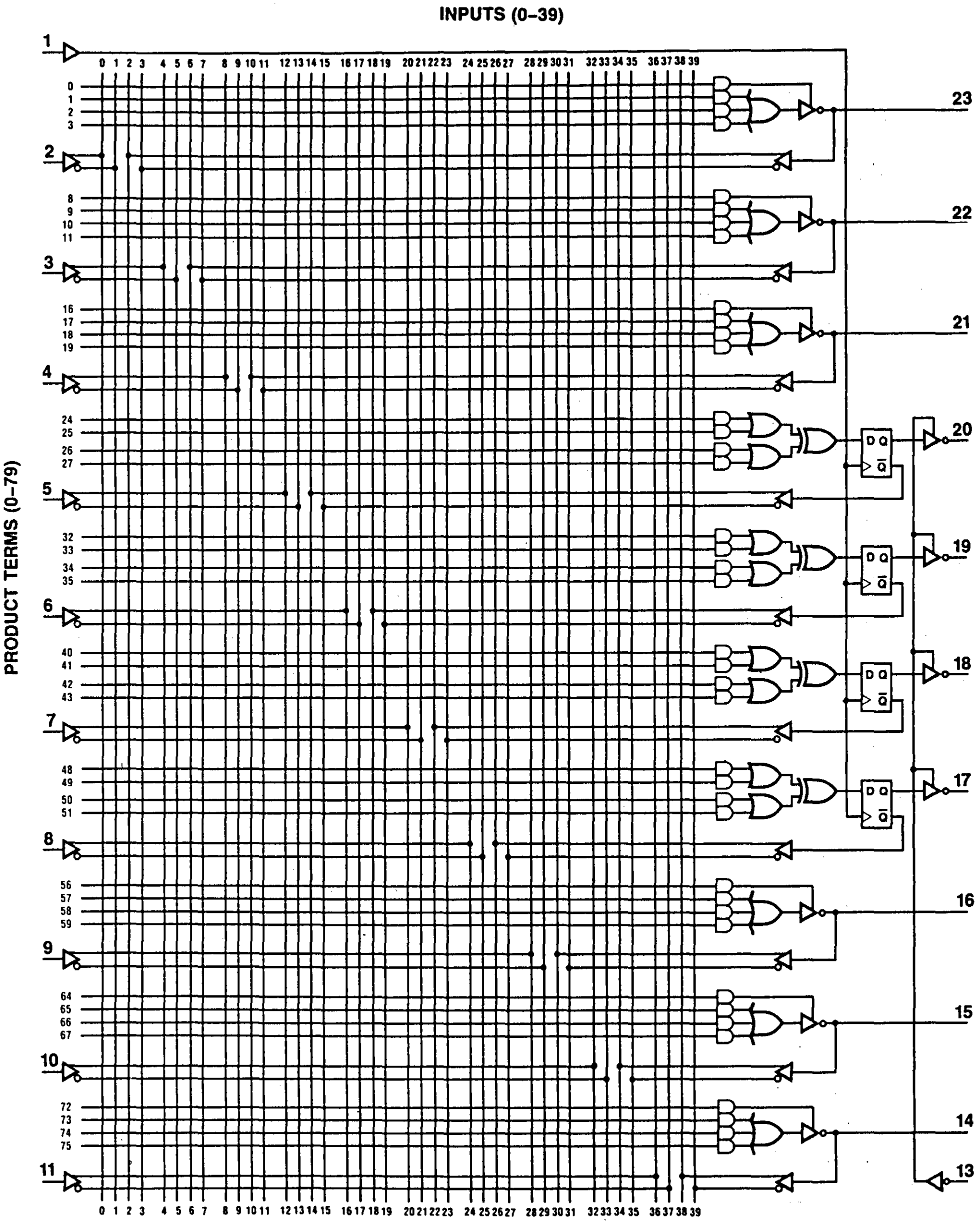


Figure 4.4.23 Logic Diagram PAL20X4

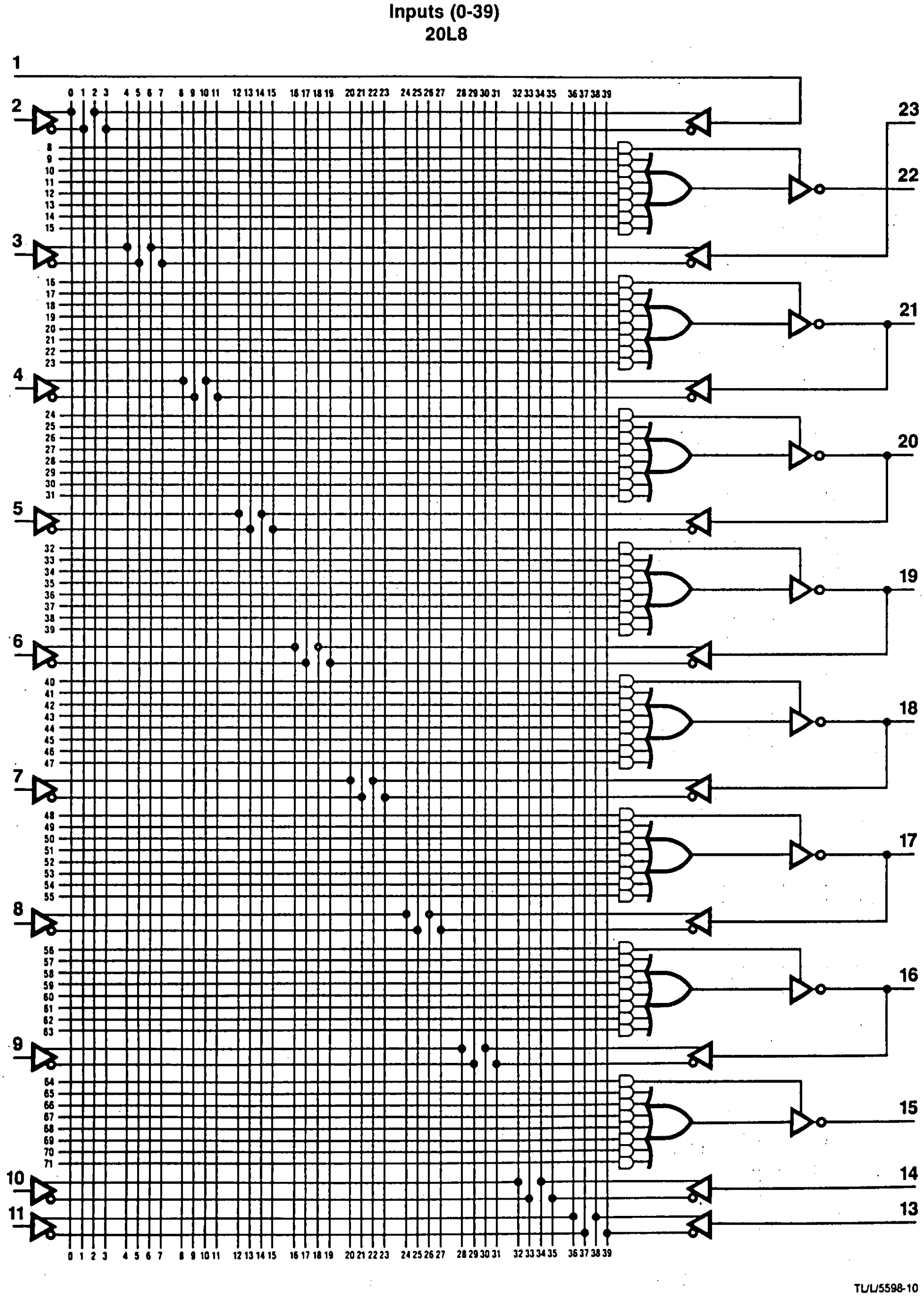


Figure 4.4.24 Logic Diagram PAL20L8

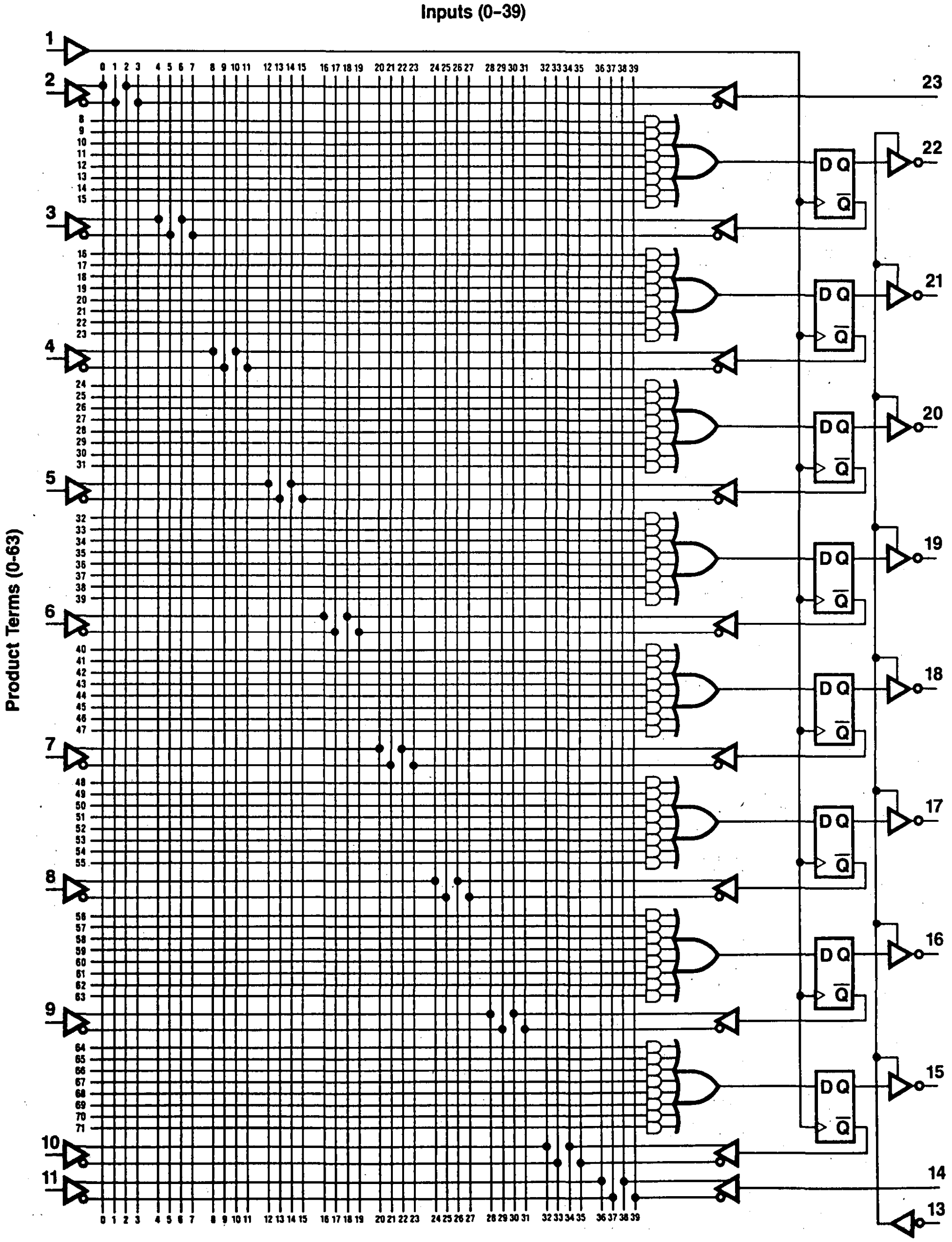


Figure 4.4.25 Logic Diagram PAL20R8

TL/L/5598-13

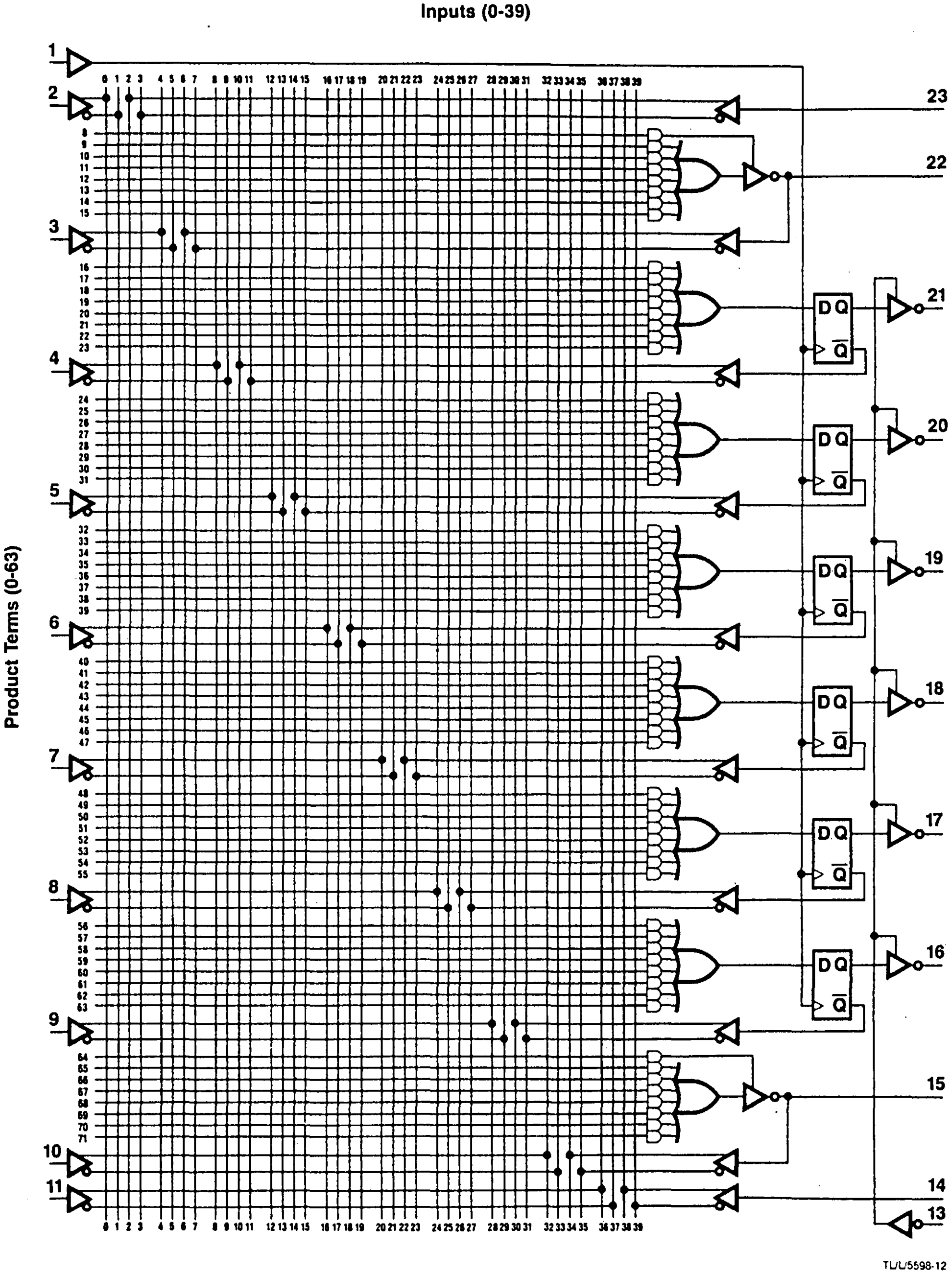


Figure 4.4.26 Logic Diagram PAL20R6

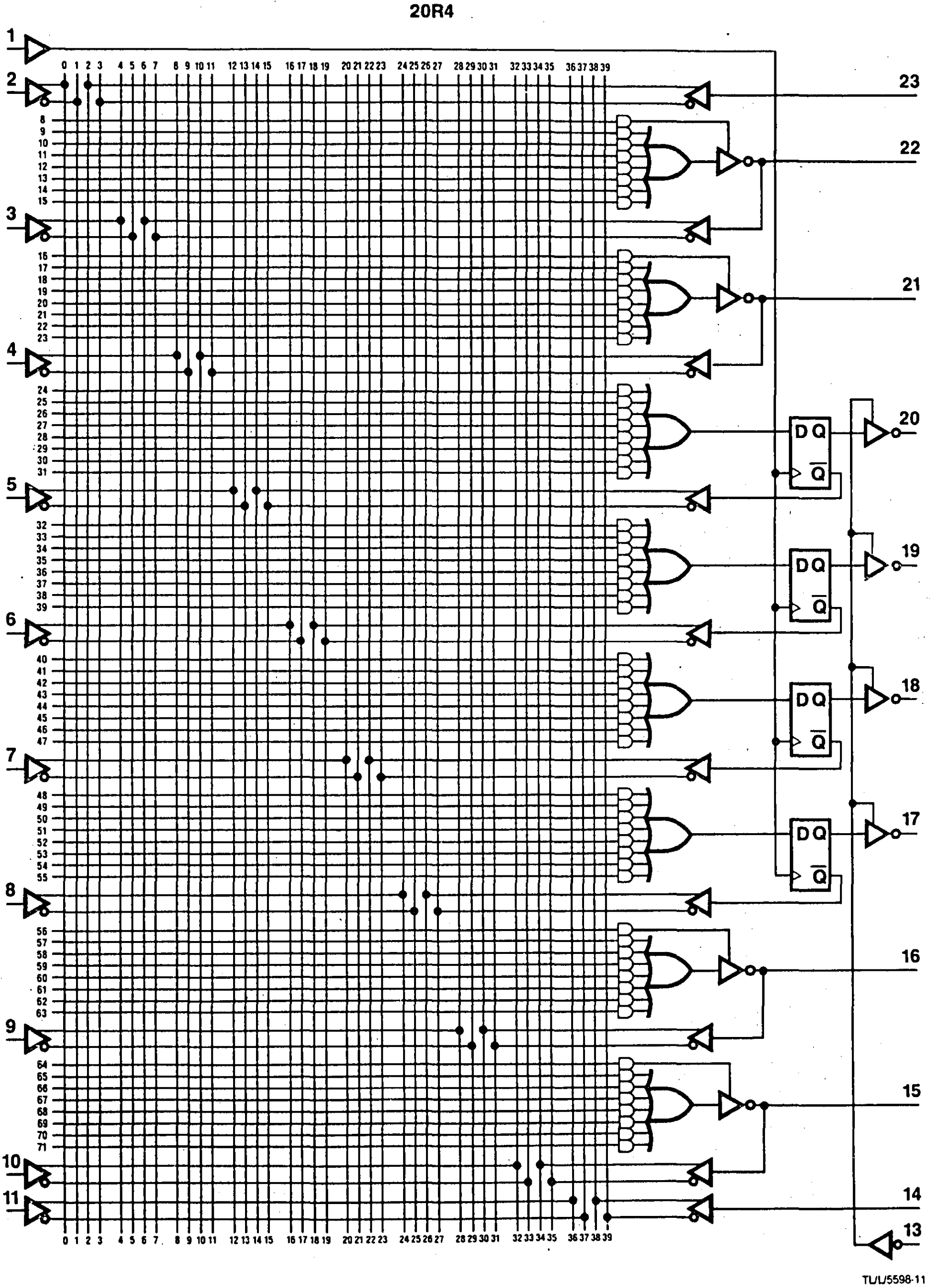
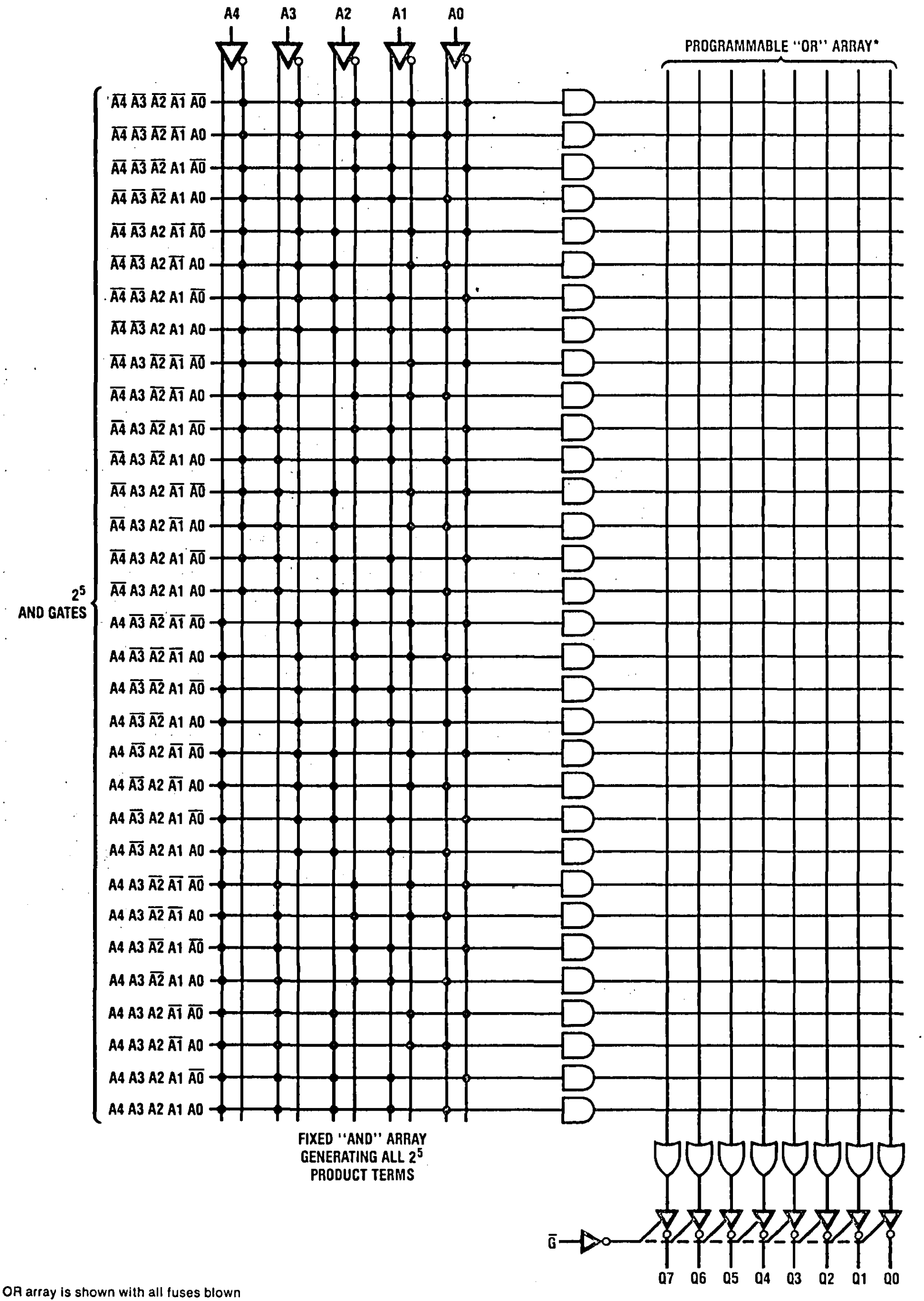


Figure 4.4.27 Logic Diagram PAL20R4


A4
A3
A2
A1
A0
PROGRAMMABLE "OR" ARRAY*
2^5 AND GATES
$\overline{A4}\,\overline{A3}\,\overline{A2}\,\overline{A1}\,\overline{A0}$
$\overline{A4}\,\overline{A3}\,\overline{A2}\,\overline{A1}\,A0$
$\overline{A4}\,\overline{A3}\,\overline{A2}\,A1\,\overline{A0}$
$\overline{A4}\,\overline{A3}\,\overline{A2}\,A1\,A0$
$\overline{A4}\,\overline{A3}\,A2\,\overline{A1}\,\overline{A0}$
$\overline{A4}\,\overline{A3}\,A2\,\overline{A1}\,A0$
$\overline{A4}\,\overline{A3}\,A2\,A1\,\overline{A0}$
$\overline{A4}\,\overline{A3}\,A2\,A1\,A0$
$\overline{A4}\,A3\,\overline{A2}\,\overline{A1}\,\overline{A0}$
$\overline{A4}\,A3\,\overline{A2}\,\overline{A1}\,A0$
$\overline{A4}\,A3\,\overline{A2}\,A1\,\overline{A0}$
$\overline{A4}\,A3\,\overline{A2}\,A1\,A0$
$\overline{A4}\,A3\,A2\,\overline{A1}\,\overline{A0}$
$\overline{A4}\,A3\,A2\,\overline{A1}\,A0$
$\overline{A4}\,A3\,A2\,A1\,\overline{A0}$
$\overline{A4}\,A3\,A2\,A1\,A0$
$A4\,\overline{A3}\,\overline{A2}\,\overline{A1}\,\overline{A0}$
$A4\,\overline{A3}\,\overline{A2}\,\overline{A1}\,A0$
$A4\,\overline{A3}\,\overline{A2}\,A1\,\overline{A0}$
$A4\,\overline{A3}\,\overline{A2}\,A1\,A0$
$A4\,\overline{A3}\,A2\,\overline{A1}\,\overline{A0}$
$A4\,\overline{A3}\,A2\,\overline{A1}\,A0$
$A4\,\overline{A3}\,A2\,A1\,\overline{A0}$
$A4\,\overline{A3}\,A2\,A1\,A0$
$A4\,A3\,\overline{A2}\,\overline{A1}\,\overline{A0}$
$A4\,A3\,\overline{A2}\,\overline{A1}\,A0$
$A4\,A3\,\overline{A2}\,A1\,\overline{A0}$
$A4\,A3\,\overline{A2}\,A1\,A0$
$A4\,A3\,A2\,\overline{A1}\,\overline{A0}$
$A4\,A3\,A2\,\overline{A1}\,A0$
$A4\,A3\,A2\,A1\,\overline{A0}$
$A4\,A3\,A2\,A1\,A0$
FIXED "AND" ARRAY GENERATING ALL 2^5 PRODUCT TERMS
$\overline{G}$
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0

*OR array is shown with all fuses blown

TL/L/6747-3

Figure 4.4.28 32 × 8 PROM Logic Diagram

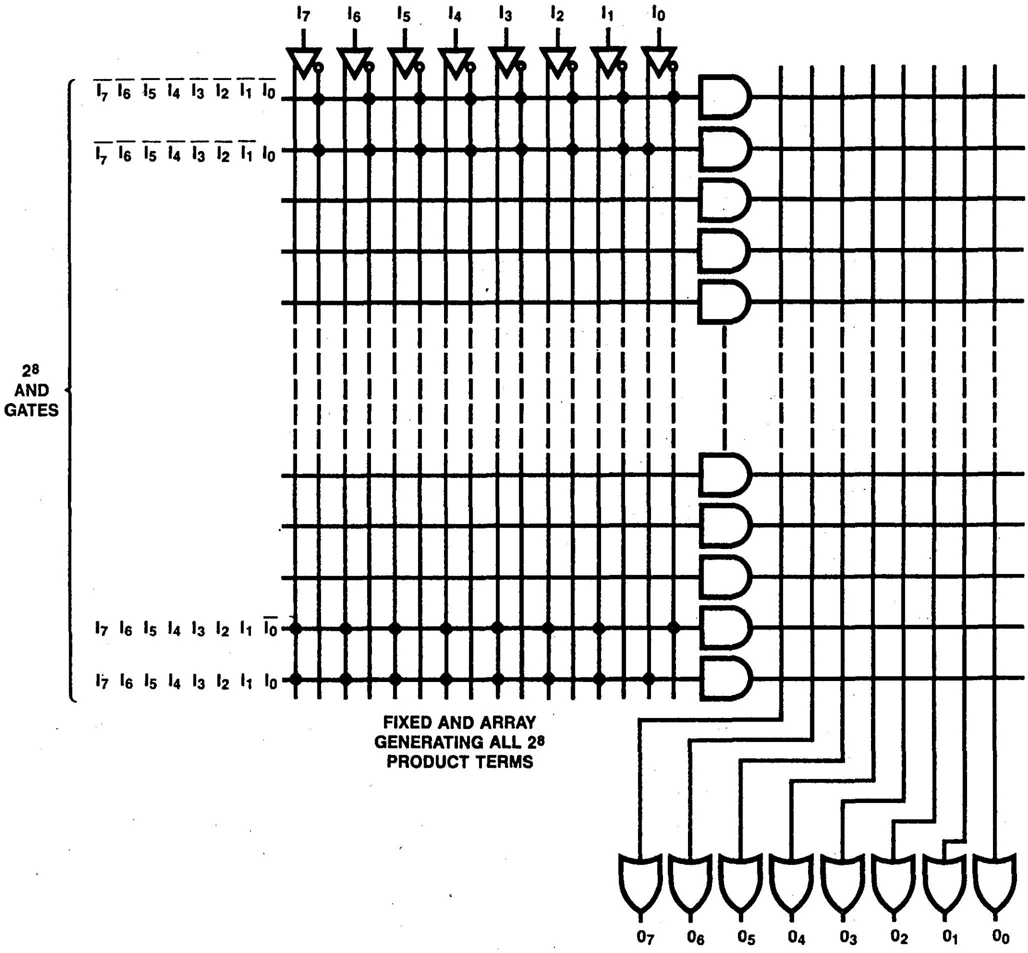

I7 I6 I5 I4 I3 I2 I1 I0
$\overline{I_7}\ \overline{I_6}\ \overline{I_5}\ \overline{I_4}\ \overline{I_3}\ \overline{I_2}\ \overline{I_1}\ \overline{I_0}$
$\overline{I_7}\ \overline{I_6}\ \overline{I_5}\ \overline{I_4}\ \overline{I_3}\ \overline{I_2}\ \overline{I_1}\ I_0$
2^8 AND GATES
$I_7\ I_6\ I_5\ I_4\ I_3\ I_2\ I_1\ \overline{I_0}$
$I_7\ I_6\ I_5\ I_4\ I_3\ I_2\ I_1\ I_0$
FIXED AND ARRAY GENERATING ALL 2^8 PRODUCT TERMS
O7 O6 O5 O4 O3 O2 O1 O0

Figure 4.4.29 256 × 8 PROM Logic Diagram

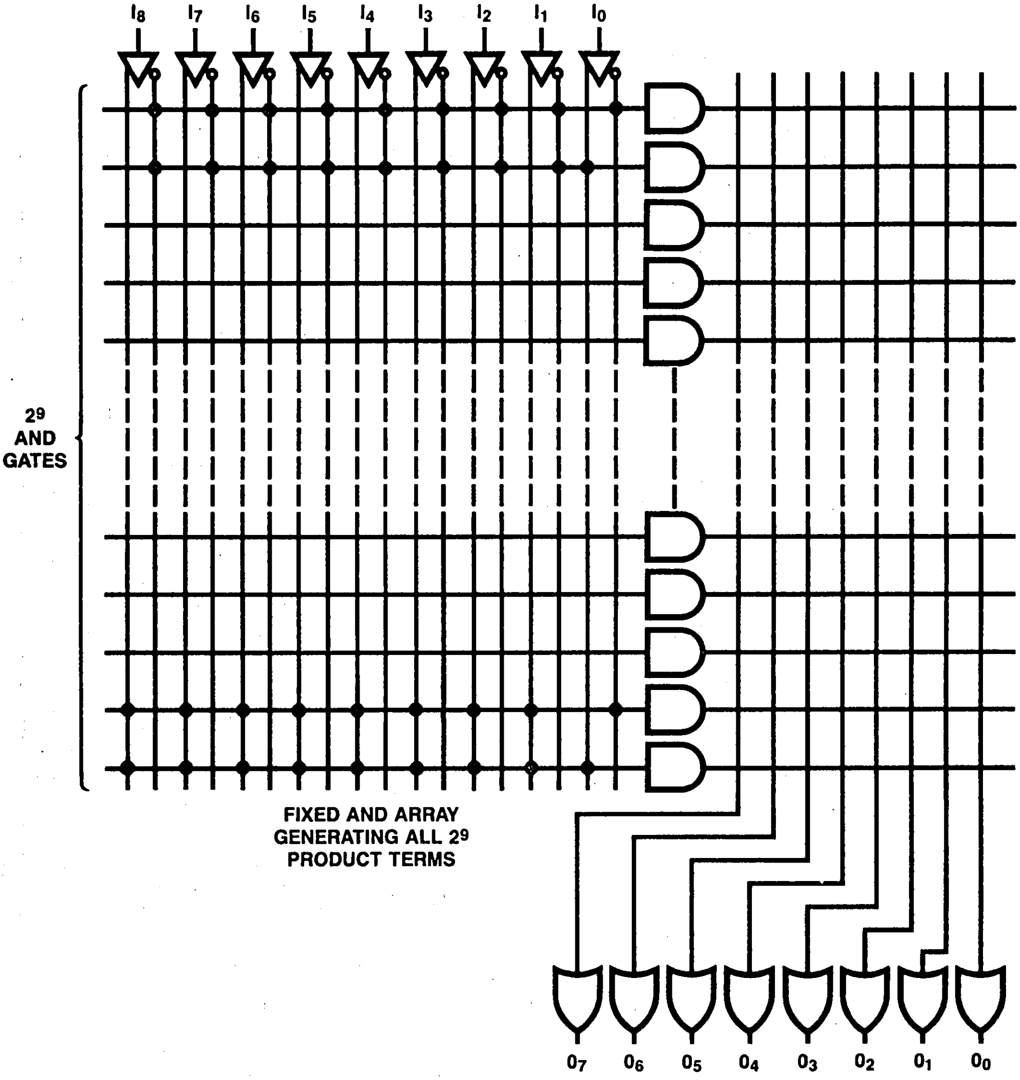


Figure 4.4.30 512 × 8 PROM Logic Diagram

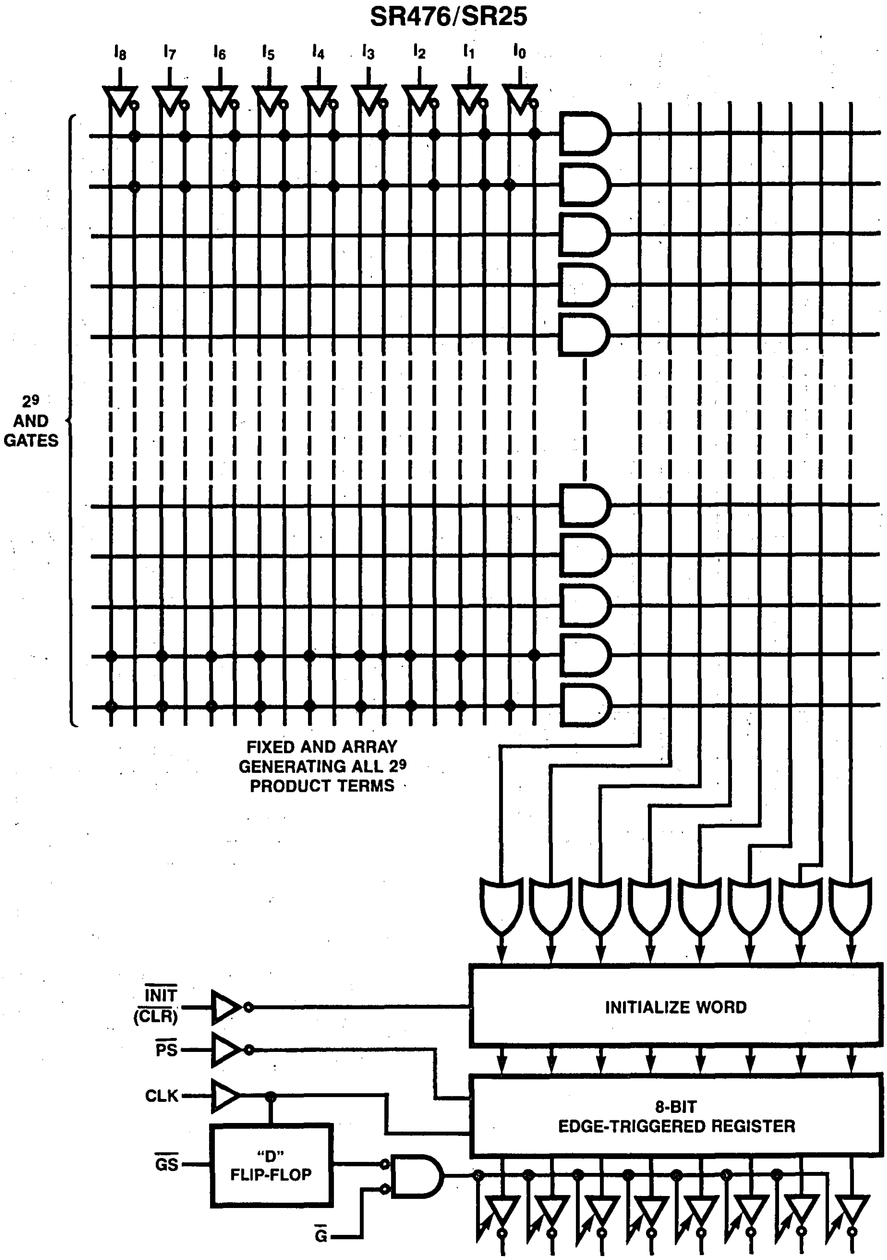


Figure 4.4.31 512 × 8 Registered PROM Logic Diagram

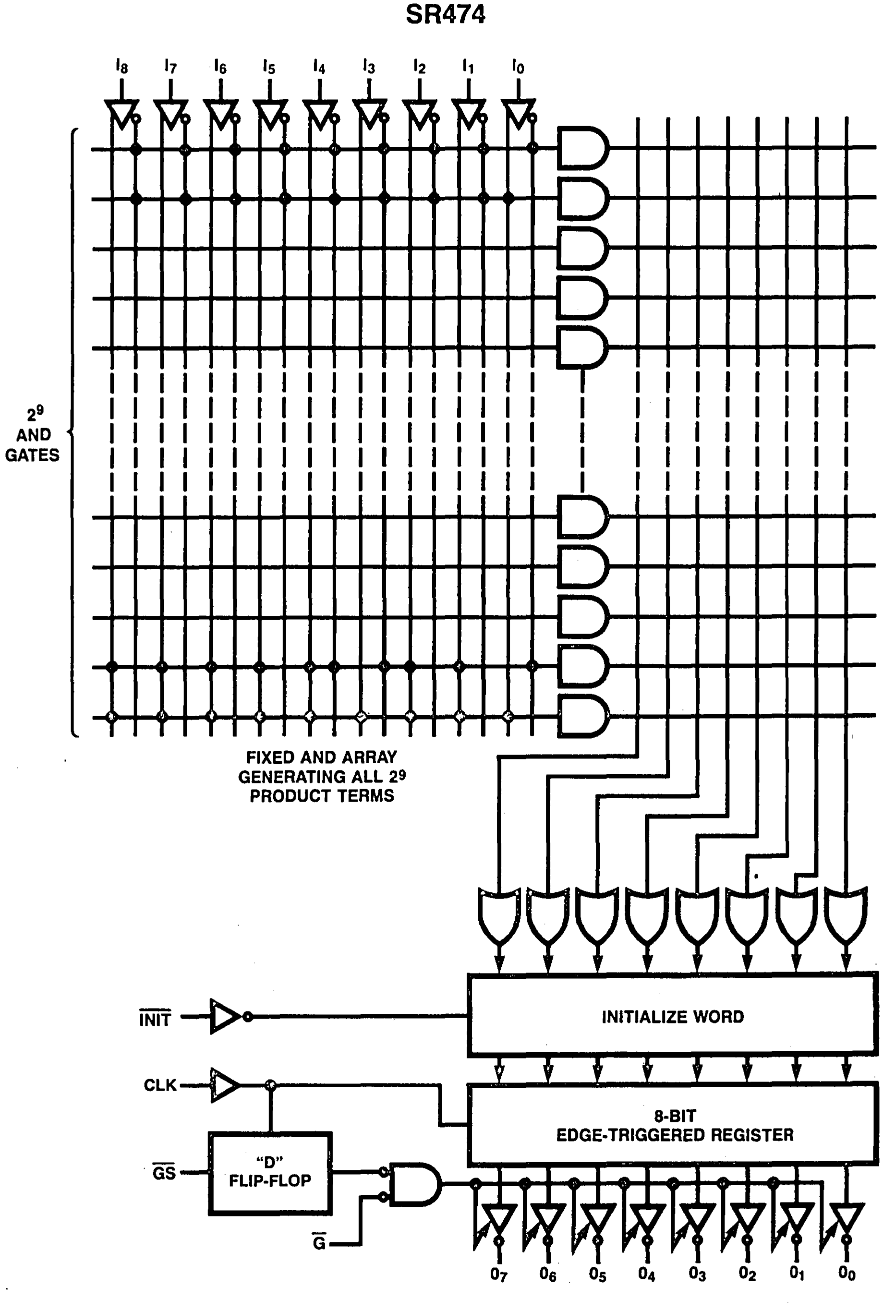


Figure 4.4.32 512 × 8 Registered PROM Logic Diagram

5

How to Design with Programmable Logic

There are two design objectives to keep in mind when using programmable logic devices. The first objective is to use the programmable logic device to replace discrete chips in the existing product. Each device will be able to replace 3 to 8 TTL chips. The second objective is to design the programmable logic device into the new/next generation product.

Each design is different. But the procedures are similiar. Figure 5.0 shows a typical design sequence.

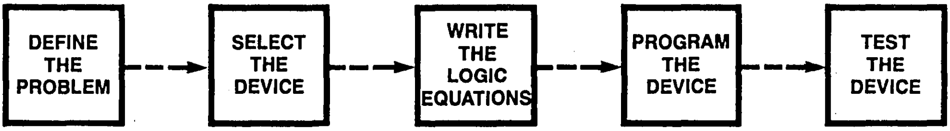


Figure 5.1.1 Design Sequence of the Programmable Logic Device

The design sequence can also be viewed as a set of five questions: (1) How do I define the problem? (2) How do I select the logic device? (3) How do I write the logic equations? (4) How do I program the device? (5) How do I test the device?

5.1 PROBLEM DEFINITION

First, we need to know the function of the logic circuit. Is it used for generating combinational control signals, decoding addresses/operation codes, or multiplexing/demultiplexing signals? Is it used for counting or shifting bits, generating different control sequences, or implementing a state machine for any usage?

Then we can decide on the type of logic circuit. Is it combinational, sequential or mixed? Table 5.1.1 shows the typical combinational and sequential circuits and the PAL devices that can be adapted.

	Typical Circuits	PAL Devices Used For
COMBINATIONAL	Decoder/encoder, multiplexer, adder, memory mapped I/O, strictly signal combination (no latch).	10H8, 12H6, 14H4, 16H2, 10L8, 12L6, 14L4, 16L2, 16C1, 12L10, 14L8, 16L6, 18L4, 20L2, 16L8
SEQUENTIAL	Counter, shift registers, accumulator, Control sequence generator	16L8, 16R8, 16R6, 16R4, 20L10, 20X10, 20X8, 20X4, 20L8, 20R8, 20R6, 20R4

Table 5.1.1 Typical PAL Circuits

5.2 DEVICE SELECTION

The next question is, which PAL device should we choose to optimize space and cost? To answer this, we first need to calculate the number of inputs and outputs of the logic circuits being designed and decide on the outputs' polarity: active-low or active-high. For example, if there are 10 input and 7 output signals and the majority of outputs are active-low, then the best choice is the 10L8. If the number of outputs are six, then we can use either the 10L8 or 12L6. Since each PAL device has limited product terms, we need to know how many product terms each output uses. The number of product terms each output will use can be viewed from logic equations. For instance, the logic equation of O1 = P1 + P2 + P3 + P4 + P5 will use five product terms for the output O1. Fortunately, National's software, PLAN, will help the user to select the right PAL device. See chapter 6 for a discussion of PLAN.

Table 5.2.1 shows National's 20 pin PAL device configurations and Table 5.2.2 shows the 24 pin PAL devices.

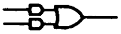
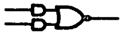
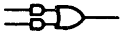
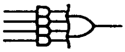
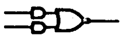
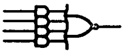
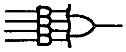
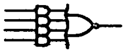
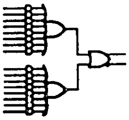
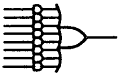
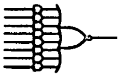
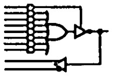
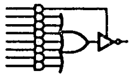
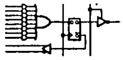
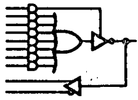
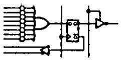
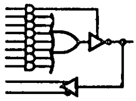
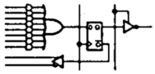
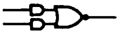
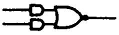
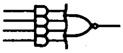
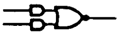
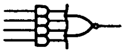
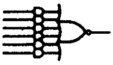
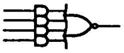
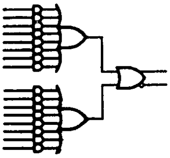
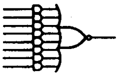
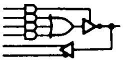
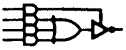
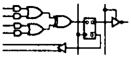
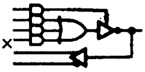
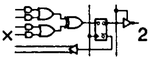
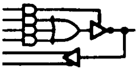
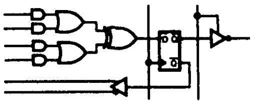
		Max Propagation Delay (ns) I/O (and CLK to Output)					
PAL	Complexity (1)	Standard	Series A	Series B	I_{CC} Max (mA)	No. of Data Inputs	No. of Outputs and Configurations
10H8	20S	35	25		90	10	8 x 
10L8	20S	35	25		90	10	8 x 
12H6	20S	35	25		90	12	4 x  2 x 
12L6	20S	35	25		90	12	4 x  2 x 
14H4	20S	35	25		90	14	4 x 
14L4	20S	35	25		90	14	4 x 
16C1	20S	35	25		90	16	1 x 
16H2	20S	35	25		90	16	2 x 
16L2	20S	35	25		90	16	2 x 
16L8	20M	35	25	15	180	16–10	6 x  2 x 
16R4	20M	35/25	25/15	15/12	180	12–8	4 x  4 x 
16R6	20M	35/25	25/15	15/12	180	10–8	6 x  2 x 
16R8	20M	35/25	25/15	15/12	180	8	8 x 

Table 5.2.1 20 Pin PAL Device Configuration

PAL	Complexity (1)	Max Propagation Delay (ns) I/O (and CLK to Output)			I_{CC} Max (mA)	No. of Data Inputs	No. of Outputs and Configurations
		Standard	Series A	Series B			
12L10	24S	40			100	12	10 × 
14L8	24S	40			100	14	6 ×  2 × 
16L6	24S	40			100	16	2 ×  4 × 
18L4	24S	40			100	18	2 ×  2 × 
20C1	24S	40			100	20	1 × 
20L2	24S	40			100	20	2 × 
20L10	24M	50			165	20–12	8 ×  2 × 
20X4	24M	50/30			180	16–10	4 ×  6× 
20X8	24M	50/30			180	12–10	8 ×  2 × 
20X10	24M	50/30			180	10	10 × 

(1) Complexity:

20 = 20-Pin PAL	S = Small PAL
24 = 24 Pin PAL	M = Medium PAL

Table 5.2.2 24 Pin PAL Device Configuration

5.3 WRITING LOGIC EQUATIONS

Writing logic equations from an existing combinational circuit is straightforward. Examples are given in Chapter 3. Also, the generation of logic equations for a new design combinational circuit is quite simple. The procedures are as follows:

1. Define the inputs and outputs.
2. Generate the Truth Table.
3. Use the techniques mentioned in Section 3.2 to get the SOP expression for each output.
4. Use the minimization techniques mentioned in Section 3.3, i.e., Boolean Algebra, K-Map or the Quine-McCluskey method to minimize every SOP expression.
5. These four steps result in the logic equations.

Figure 5.3.1 shows these steps:

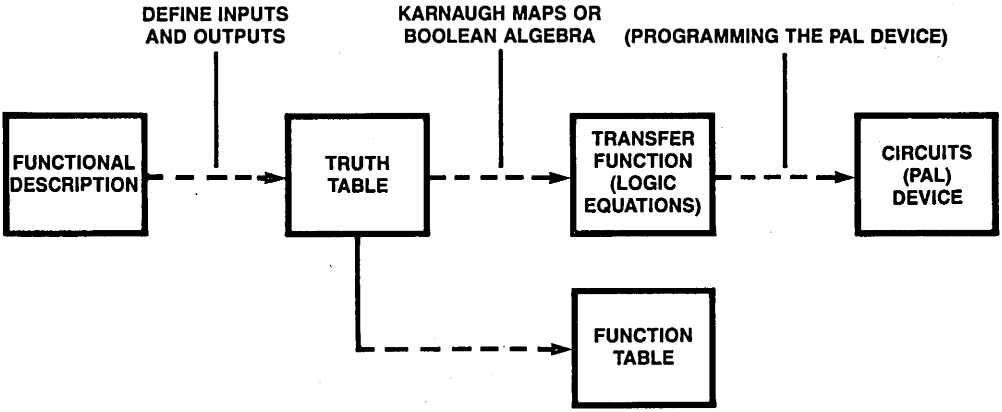

DEFINE INPUTS AND OUTPUTS
KARNAUGH MAPS OR BOOLEAN ALGEBRA
(PROGRAMMING THE PAL DEVICE)
FUNCTIONAL DESCRIPTION
TRUTH TABLE
TRANSFER FUNCTION (LOGIC EQUATIONS)
CIRCUITS (PAL) DEVICE
FUNCTION TABLE

Figure 5.3.1 Combinational PAL Device Design Steps

It is much more complicated to generate logic equations for a sequential circuit. Generally, the procedures are as follows:

1. Define the inputs and outputs, different states and variables.
2. Generate the state diagram.
3. Generate the state table.
4. Minimize the state table.

5. Assign the new state.
6. Generate the transition table.
7. Use the minimization technique to minimize transition table.
8. These seven steps result in the logic equations.

Figure 5.3.2 shows these seven steps.

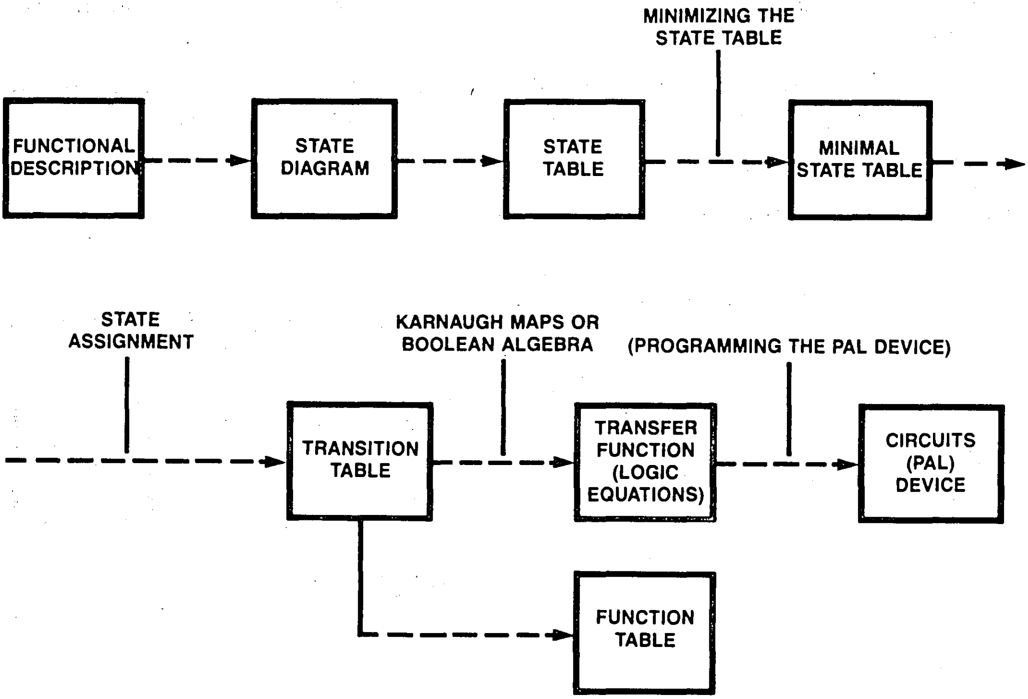


Figure 5.3.2 Sequential PAL Device Design Steps

5.4 PROGRAMMING THE DEVICE

Given the logic equations, the PAL device programmer will manage the programming job for us. All we need to do is to enter those logic equations into the terminal. The programming procedures are shown in Figure 5.4.1.

After programming, the fuse status should be verified. Most programmers will provide this fuse verification capability.

Manually coding the programming format sheet, which has appeared in National's 1983 PAL Device Data Book will not be discussed in this Design Guide.

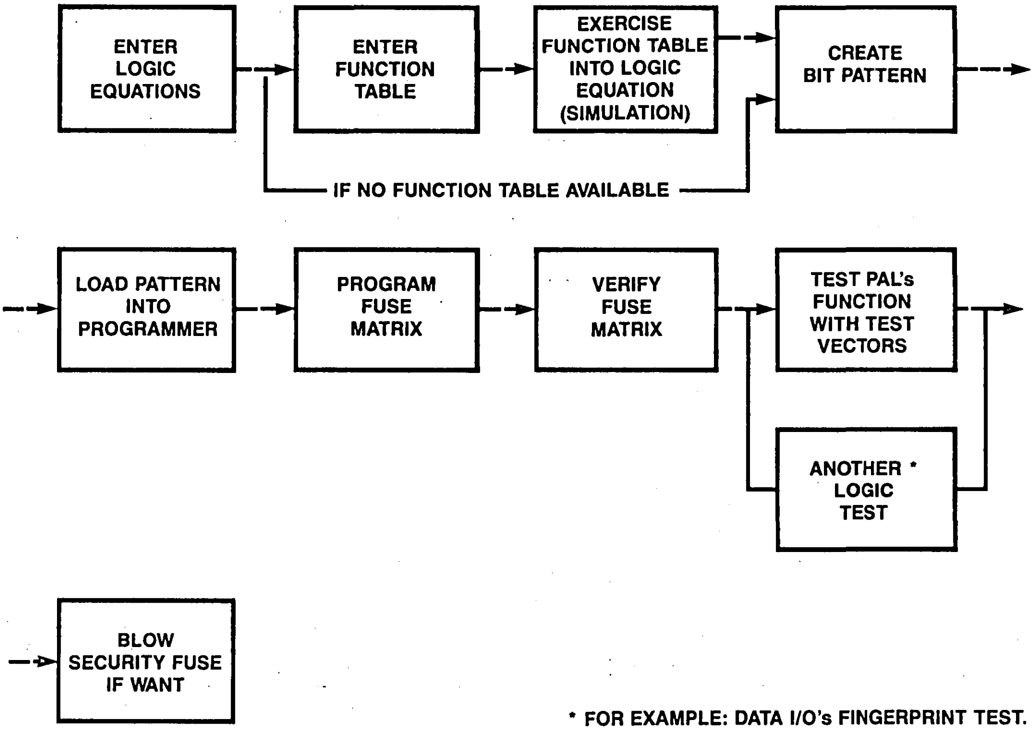

ENTER LOGIC EQUATIONS
ENTER FUNCTION TABLE
EXERCISE FUNCTION TABLE INTO LOGIC EQUATION (SIMULATION)
CREATE BIT PATTERN
IF NO FUNCTION TABLE AVAILABLE
LOAD PATTERN INTO PROGRAMMER
PROGRAM FUSE MATRIX
VERIFY FUSE MATRIX
TEST PAL's FUNCTION WITH TEST VECTORS
ANOTHER * LOGIC TEST
BLOW SECURITY FUSE IF WANT
* FOR EXAMPLE: DATA I/O's FINGERPRINT TEST.

Figure 5.4.1 PAL Device Programming Procedures

5.5 TESTING THE DEVICE*

Fuse verification tells us if the fuse was blown correctly or not; but it doesn't tell us if the PAL device functions properly. Therefore, we also need to do functional testing. There are two ways to do functional testing. One method uses function tables. Another method uses test vectors. Each of these methods may give a different result.

Function tables are generated without reference to the logic equations. The function table tells what the PAL device should do. Function tables are used to determine if the device functions as intended. If it does not, we have to go back to the equations, since there may be a problem there.

Test vectors are generated directly from the logic equations. They are used to verify the internal operation of the PAL device. If a problem is detected, it implies that something is internally wrong with the device. However, a device may pass the test vector screening and still not function properly if the logic equations were derrived incorrectly.

It is the logic designer's responsibility to generate the function table. This is the person who best knows the design. After the design is released, the test engineer will

**Also see Chapter 7 for details about testing.*

take the responsibility for testing incoming devices. As mentioned before, the function table can't catch all the interior bugs. Therefore, the test engineer needs to write the test vectors. It is a large and sophisticated job to create test vectors. Figure 5.5.1 shows these steps and will be explained in chapter 7. There are a few software packages available for generating test vectors, for example; HILO[1], and TEGAS[2], LOGCAP[3], LAZAR[4].

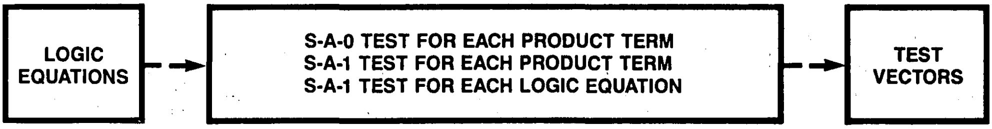

LOGIC EQUATIONS
S-A-0 TEST FOR EACH PRODUCT TERM
S-A-1 TEST FOR EACH PRODUCT TERM
S-A-1 TEST FOR EACH LOGIC EQUATION
TEST VECTORS

Figure 5.5.1 Test Vectors Creating Steps

5.6 PROGRAMMER VENDOR LIST

						Storage Media for		Programs		
Mfgr.	Basic Equipment	PAL-Device Module	Adapters	PAL Device Design-Software Included	Performs Logic Simulation	Bit Pattern	Test Vectors	20-Pin	24-Pin	Blows Security Fuses
Data I/O	Model 19, 19A or 100A	1427	1428-1 -2 -3	No	No	Master PAL	—	Yes	No	No
Digelec	μP 803	FAM 51	20 + 24 Pin Socket	Yes	No	Master PAL	+	Yes	Yes	Yes
Kontron	EPP 80 or MPP 80S	MOD 21	SA 27 + SA 27-1	No	No	Master PAL	—	Yes	Yes	Yes
Stag	PPX	PM 202 + BRAL	AM10H8 • • • AM16C1	Yes	No	Master PAL	—	Yes	No	Yes
Citel	System 47		PL1	No	No	Master 7 PAL, PROM, EPROM	PROM	Yes	Yes	Yes

All these systems program and verify the PAL in the PROM mode. They *do not perform a logic simulation* in the PAL device mode. Additional (external) circuitry for logic simulation should be used if PAL devices go into volume production — otherwise, a small percentage of the PAL devices will show failures when testing the complete PC board. OK for prototype-making.

Table 5.6.1 PAL Device Programmers

[1] *HILO is a registered trademark of Gen Rad.*
[2] *TEGAS is a registered trademark of CDC.*
[3] *LOGCAP is a registered trademark of Phoenix Data Systems.*
[4] *LAZAR is a registered trademark of Teledyne.*

				PAL Device		Storage Media for Programs				
		PAL-		Design-	Performs					Blows
Mfgr.	Basic Equipment	Device Module	Adapters	Software Included	Logic Simulation	Bit Pattern	Test Vectors	20-Pin	24-Pin	Security Fuses
Data IO	Model 19, 29A or 100 and Any Terminal	Logic-Pack	Design Ad. and Progr. Ad.	Yes	Yes, Automatic or Manual Generation of Test Vectors	Master PAL or EPROM	External or	Yes	Yes	Yes
Digelec	μP 803	FAM 52	20- and 24-Pin Adapter	Yes	Yes, Automatic or Manual Generation of Test Vectors	Master PAL	External	Yes	Yes	Yes
Stag	—	ZL 30	—	Yes	Yes, Automatic or Manual Generation of Test Vectors	Master PAL	External	Yes	Yes	Yes
Structured Design	Any Terminal	SD20/24	—	Yes	Yes, Manual Generation of Test Vectors	Master PAL or On Wafertape	External or	Yes	Yes	No
Structured Design	Any Terminal	SD1000	—	Yes	Yes, Manual Generation of Test Vectors	Master PAL or EPROM	External or	Yes	Yes	Yes

All these systems allow software supported PAL device design. They perform a *fuse-verify* in the PROM mode and can do a *logic simulation* in the PAL device mode. All 5 programmers and 5 development systems can be connected with a host computer to run more sophisticated design software and/or for storage use.

Table 5.6.2 PAL Device Development Systems

5.7 EXAMPLES

Example 1: Replace the existing logic circuit in Figure 5.7.1 by a PAL device.

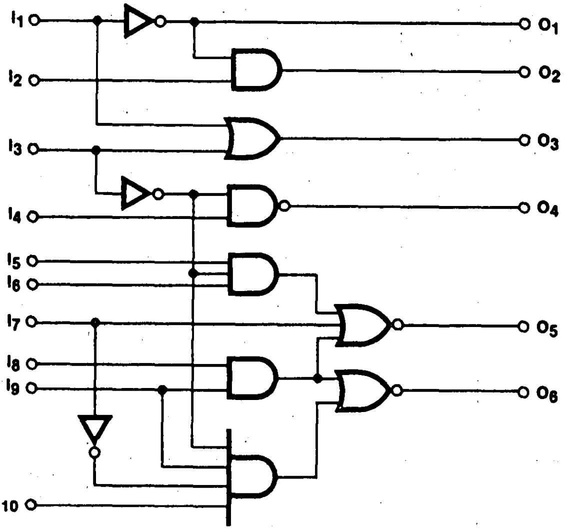


Figure 5.7.1 Design Example, Logic Diagram

We will follow the procedure discussed in this chapter. We know the first step is to understand the function of this circuit. There is no register and latch involved. By experience, we understand that this circuit is used to manipulate different input signals and generate different outputs. We should select the combinational PAL device (i.e., PAL10H8, PAL10L8, PAL12H6, etc.).

The second step is to choose the specific device. Because the number of inputs is 10 and the number of outputs is 6, we limit our choice to be 10H8, 10L8, 12H6 and 12L6. Three outputs have AND-OR functions and 3 outputs have AND-OR-INVERT functions. We could still select from either active-high or active-low (H or L) parts. Since the more complex functions are AND-OR-INVERT, the active LOW (L) series is most likely. Therefore, we now limit our choice to the 10L8 and 12L6 devices. A review of the 10L8's logic diagram shows that all of its NOR gates are two-input gates, and the design example requires a three-input gate. On the other hand, the 12L6 has two 4-input gates which will accommodate the 3-input requirement. It, therefore, is selected.

The third step is to write the logic equation. It is very straightforward for this example.

We get:

$0_1 = /I_1$
$0_2 = /I_1 * I_2$
$0_3 = I_1 + I_3$
$0_4 = /(/I_3 * I_4)$
$0_5 = /(/I_3 * I_5 * I_6 + I_7 + I_8 * I_9)$
$0_6 = /(I_8 * I_9 + /I_3 * /I_7 * I_9 * I_{10})$

Since we have selected a PAL12L6 (which has inverting outputs) we need to apply DeMorgan's theorem to convert these equations from active-high to active-low outputs. DeMorgan's theorem can be used to convert any logic form to the AND-OR or AND-NOR structure used in PALs. Applying DeMorgan's theorem gives the active LOW form of the equation:

$/0_1 = I_1$
$/0_2 = I_1 + /I_2$
$/0_3 = /I_1 * /I_3$
$/0_4 = /I_3 * I_4$
$/0_5 = /I_3 * I_5 * I_6 + I_7 + I_8 * I_9$
$/0_6 = I_8 * I_9 + /I_3 * /I_7 * I_9 * I_{10}$

Assuming that there are no board layout constraints, input I_1 through I_{10} may be assigned to pins 1 through 11 (pin 10 is ground). The only constraint on output pin assignment is that 0_5 must be assigned to pin 13 or 18 to take advantage of one of the 4-input NOR gates.

The fourth step is to program the PAL device. To do this we must enter the logic equations into the computer or the PAL device programmer. National's PLAN software allows users to enter logic equations in any format. But PALASM requires the program shown in Figure 5.7.2 in its host computer to be used as follows:

```
PAL12L6
PAT201
PAL DESIGN EXAMPLE

I1 I2 I3 I4 I5 I6 I7 I8 I9 GND I10 NC 05
06 04 03 02 01 NC VCC

/01 = I1
/02 = I1 + /I2
/03 = /I1 • /I3
/04 = /I3 • I4
/06 = I8 • I9 + /I3 • /I7 • I9 • I10
/05 = /I3 • I5 • I6 + I7 + I8 • I9

DESCRIPTION

THIS PROGRAM IS A DESIGN SAMPLE DESCRIBING
THE USE OF PALASM AS A PAL DESIGN AID.
```

Figure 5.7.2 Example of PALASM Program Input

Line 1: At the left margin, the PAL device is specified. For this example, the 12L6 remains the best solution, therefore entering PAL12L6 at the left margin.

Line 2: A unique pattern number for this PAL device design is entered at the left margin on Line 2, followed by designer's name and date.

Line 3: The name or description of the device or function is entered. If this runs over one line, Line 4 may be used to complete it.

Line 4: If not used to complete Line 3, this line is skipped.

Lines 5, 6, and 7: These lines are used for pin assignments. All 20 of the pins on the PAL are assigned symbolic names, usually corresponding to the symbols used on the logic diagram. (Note that GND and V_{CC} must be included.) Assignment starts at pin 1 and proceeds sequentially, through pin 20.

Line 8: Beginning on Line 8 or Line 6, if only Line 5 is needed for the pin assignments, the logic equations that describe the required functions are written using the symbols defined in Lines 5, 6 and 7, in the format applicable to the PAL device selected. For example, the output of the 12L6 is low for the selected product term; therefore, the logic equations must be of the form $/O_X = f(I_1, I_2, \ldots)$. The symbology used must be that shown in Figure 5.7.3.

=	EQUAL
:=	REPLACED BY, FOLLOWING CLOCK
/	COMPLEMENT
*	AND, PRODUCT
+	OR, SUM
:+:	XOR
:*:	XNOR
( )	CONDITION TRI-STATE IF STATEMENT, ARITHMETIC

Figure 5.7.3 PALASM Operators

Then the PAL device software will generate the fuse map and bit pattern shown in Table 5.7.1, load pattern into programmer, program the device and verify the fuse matrix. Since there is no function table in this example, we need to do another logic test to guarantee it works properly. For example, we can do the fingerprint test if we already have a known good device, or we can generate a few (or whole) test vectors to do the structure test in a DATA I/O programmer.

8	--X-	----	--	--	--	--	----	----
9	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX
10	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX
11	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX
16	--X-	----	--	--	--	--	----	----
17	-X--	----	--	--	--	--	----	----
24	---X	-X--	--	--	--	--	----	----
25	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX
32	----	-X--	X-	--	--	--	----	----
33	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX
40	----	----	--	--	--	--	X---	X---
41	----	-X--	--	--	--	-X	----	X-X-
48	----	-X--	--	X-	X-	--	----	----
49	----	----	--	--	--	X-	----	----
50	----	----	--	--	--	--	X---	X---
51	XXXX	XXXX	XX	XX	XX	XX	XXXX	XXXX

Table 5.7.1 Fuse Map

Figure 5.7.4 is the logic diagram of this PAL device and Figure 5.7.5 shows the PAL device legend.

Example 2: Design a multiplexer to select one of three input data buses which contain 4 data lines, as shown in Figure 5.7.6. The output should be high if we don't select any data bus.

From Figure 5.7.6 we know there are 14 input lines and 4 outputs. Since we select one out of three, we need 3 product terms in each output. In addition, we need another product term to implement diselection which will cause all output-high. From the PAL device select chart (Table 5.2.1) we find 14H4 is the best fit.

The logic equation is very easily derived from intuition or we can get from the truth table shown in Table 5.7.2.

PLAN software will help us to select the device, assign pinouts, and generate a fusemap. All we need to do is enter the logic equations.

Y1 = /SELA * /SELB * A1 + SELA * /SELB * B1 + /SELA * SELB * C1 + SELA * SELB

Y2 = /SELA * /SELB * A2 + SELA * /SELB * B2 + /SELA * SELB * C2 + SELA * SELB

Y3 = /SELA * /SELB * A3 + SELA * /SELB * B3 + /SELA * SELB * C3 + SELA * SELB

Y4 = /SELA * /SELB * A4 + SELA * /SELB * B4 + /SELA * SELB * C4 + SELA * SELB

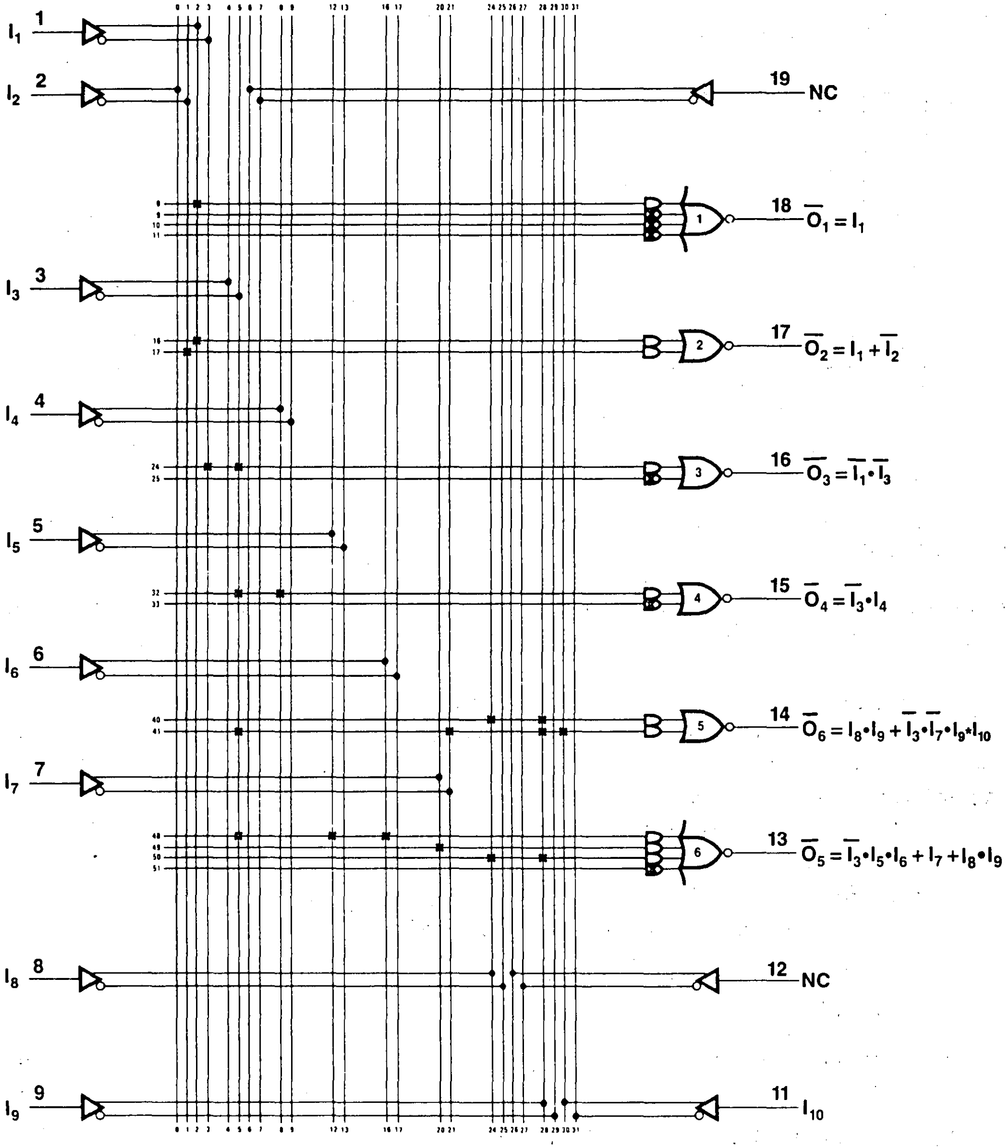

I_1 1
I_2 2
19 NC
18 $\overline{O}_1 = I_1$
I_3 3
17 $\overline{O}_2 = I_1 + \overline{I}_2$
I_4 4
16 $\overline{O}_3 = \overline{I}_1 \cdot \overline{I}_3$
I_5 5
15 $\overline{O}_4 = \overline{I}_3 \cdot I_4$
I_6 6
14 $\overline{O}_6 = I_8 \cdot I_9 + \overline{I}_3 \cdot \overline{I}_7 \cdot I_9 \cdot I_{10}$
I_7 7
13 $\overline{O}_5 = \overline{I}_3 \cdot I_5 \cdot I_6 + I_7 + I_8 \cdot I_9$
I_8 8
12 NC
I_9 9
11 I_{10}
1
2
3
4
5
6

Figure 5.7.4 Logic Diagram of the National Type 12L6 PAL®

PAL Legend

Constants

LOW (L)	NEGATIVE (N)	ZERO (0)	GND	FALSE	x	FUSE NOT BLOWN
HIGH (H)	POSITIVE (P)	ONE (1)	V_{CC}	TRUE	-	FUSE BLOWN

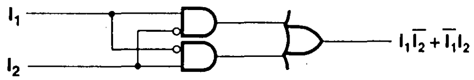
Operators

```
  =   EQUAL
 :=   REPLACED BY FOLLOWING CLOCK
  /   COMPLEMENT
  *   AND, PRODUCT
  +   OR, SUM
:+:   XOR
:*:   XNOR
( )   CONDITIONAL THREE STATE, IF STATEMENT, ARITHMETIC
```

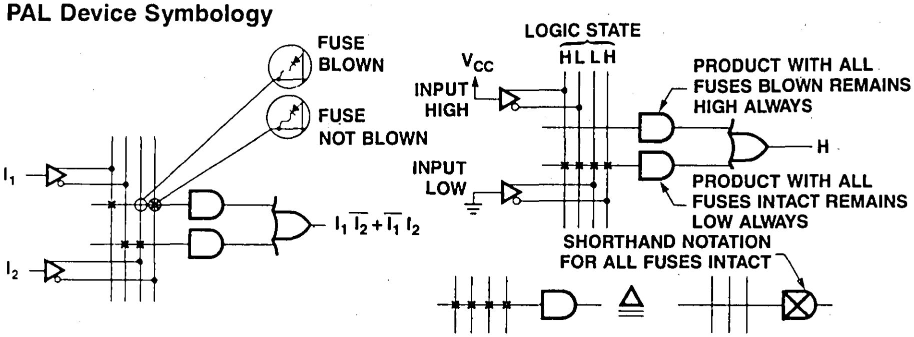
Equations

Standard	Q_1	$= I_1\overline{I_2}+\overline{I_1}I_2$
PALASM	O1	= I1*/I2 + /I1*I2

Conventional Symbology



PAL Device Symbology


FUSE BLOWN
FUSE NOT BLOWN
I_1
I_2
$I_1\overline{I_2}+\overline{I_1}I_2$
LOGIC STATE
H L L H
V_{CC}
INPUT HIGH
INPUT LOW
PRODUCT WITH ALL FUSES BLOWN REMAINS HIGH ALWAYS
H
PRODUCT WITH ALL FUSES INTACT REMAINS LOW ALWAYS
SHORTHAND NOTATION FOR ALL FUSES INTACT

PAL Logic Diagram

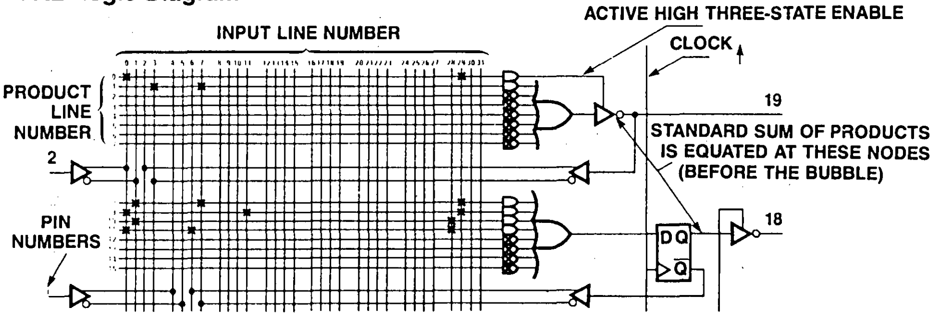

INPUT LINE NUMBER
PRODUCT LINE NUMBER
2
PIN NUMBERS
ACTIVE HIGH THREE-STATE ENABLE
CLOCK
19
STANDARD SUM OF PRODUCTS IS EQUATED AT THESE NODES (BEFORE THE BUBBLE)
D Q
$\overline{Q}$
18

Figure 5.7.5 PAL Legend

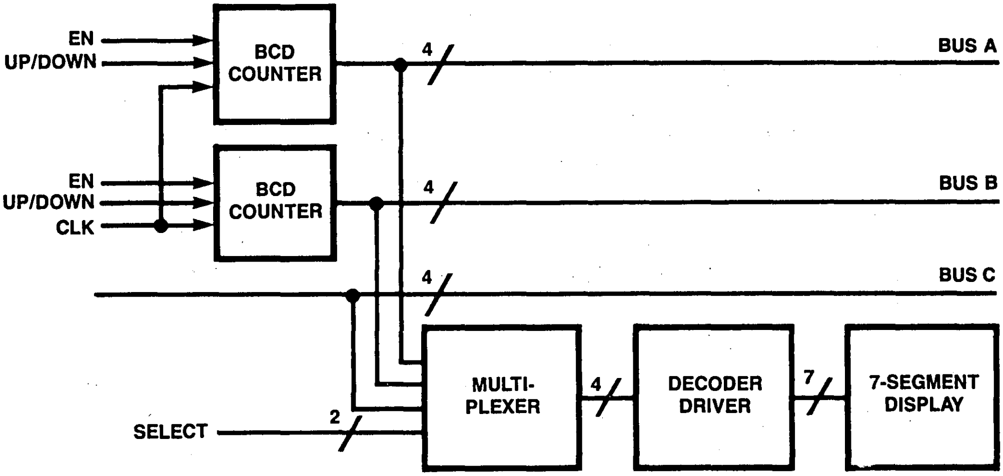


Figure 5.7.6 Block Diagram of a Multiplexer

A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4	SELA	SELB	Y1	Y2	Y3	Y4
A1	A2	A3	A4	X	X	X	X	X	X	X	X	L	L	A1	A2	A3	A4
X	X	X	X	B1	B2	B3	B4	X	X	X	X	H	L	B1	B2	B3	B4
X	X	X	X	X	X	X	X	C1	C2	C3	C4	L	H	C1	C2	C3	C4
X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H

Table 5.7.2 Truth Table

We can replace 2 of 74S153 in this application.
The Function Table and logic diagram are shown in Table 5.7.3 and Figure 5.7.7.

A1	A2	A3	A4	B1	B2	B3	B4	C1	C2	C3	C4	SELA	SELB	Y1	Y2	Y3	Y4
L	L	L	L	X	X	X	X	X	X	X	X	L	L	L	L	L	L
X	X	X	X	X	X	X	X	X	X	X	X	H	H	H	H	H	H
H	H	H	H	X	X	X	X	X	X	X	X	L	L	H	H	H	H
X	X	X	X	H	H	H	H	X	X	X	X	H	L	H	H	H	H
X	X	X	X	X	X	X	X	L	L	L	H	L	H	L	L	L	H
X	X	X	X	X	X	X	X	L	L	H	L	L	H	L	L	H	L

Table 5.7.3 Function Table

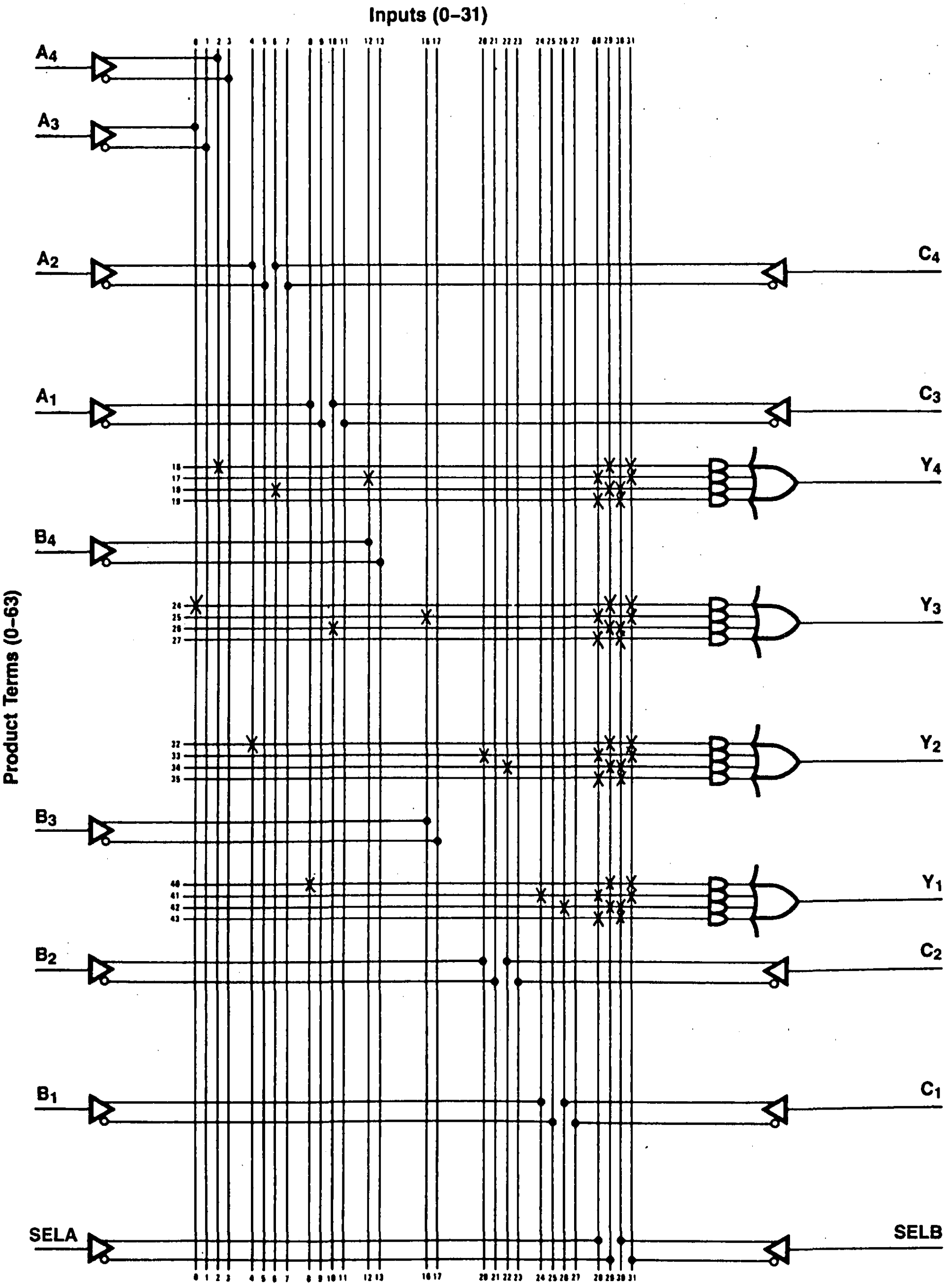


Figure 5.7.7 Logic Diagram of the National Type 14H4 PAL Device

Example 3: Design a 3-bit counter which causes only one bit change for each change of state shown in Figure 5.7.8. A RESET input will initialize the counter to 000.

The PAL device under design is used for a 3-bit counter with only one input line, RESET. When active, it will reset all three flip-flops. Obviously we can use a 16R4 to implement this application.

A	B	C
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0
0	0	0
0	0	1

REPEAT

Figure 5.7.8 3-Bit Counter

Q^n ——————→ Q^{n+1}*	D	J	K	S	R	T
0 ——————→ 0	0	0	X	0	X	0
0 ——————→ 1	1	1	X	1	0	1
1 ——————→ 0	0	X	1	0	1	1
1 ——————→ 1	1	X	0	X	0	0

*Q^n, Q^{n+1} STAND FOR PRESENT AND NEXT STATE; X IS DON'T CARE.

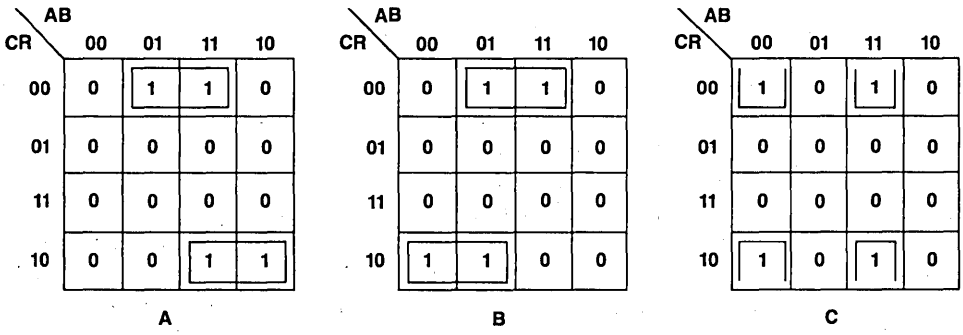
Table 5.7.4 Transition Lists

We can easily write the transition table for this simple example as shown in Table 5.7.5.

CLK	R (RESET)	A^n	B^n	C^n	A^{n+1}	B^{n+1}	C^{n+1}
↑	0	0	0	0	0	0	1
↑	0	0	0	1	0	1	1
↑	0	0	1	1	0	1	0
↑	0	0	1	0	1	1	0
↑	0	1	1	0	1	1	1
↑	0	1	1	1	1	0	1
↑	0	1	0	1	1	0	0
↑	0	1	0	0	0	0	0
↑	1	X	X	X	0	0	0

Table 5.7.5 Transition Table

We can get the logic equation from Table 5.7.5 by K-map minimization technology as shown in Figure 5.7.9.



A

CR \ AB	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	0	0	0	0
10	0	0	1	1

B

CR \ AB	00	01	11	10
00	0	1	1	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

C

CR \ AB	00	01	11	10
00	1	0	1	0
01	0	0	0	0
11	0	0	0	0
10	1	0	1	0

Figure 5.7.9 K-map

$A := B\overline{C}\overline{R} + AC\overline{R}$
$B := B\overline{C}\overline{R} + \overline{A}C\overline{R}$
$C := \overline{A}\overline{B}\overline{R} + AB\overline{R}$

We can also get the Function Table from Table 5.7.5. In this case, we replace 2 of 74S00 and 1 of 74S175.

Example 4: Design a video-telephone sync pulse detector.

The video-telephone set contains a CRT for displaying the received picture from another video-telephone, and a vidicon camera for generating the picture to be transmitted.

The vidicon sweeps across the head and shoulders view of the person talking, starting at the upper left of the picture and moving right as shown in Figure 5.7.10.

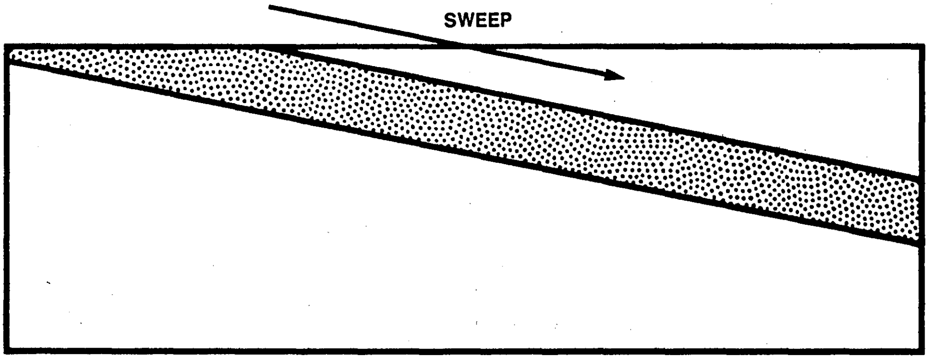


Figure 5.7.10 Sweep Generation

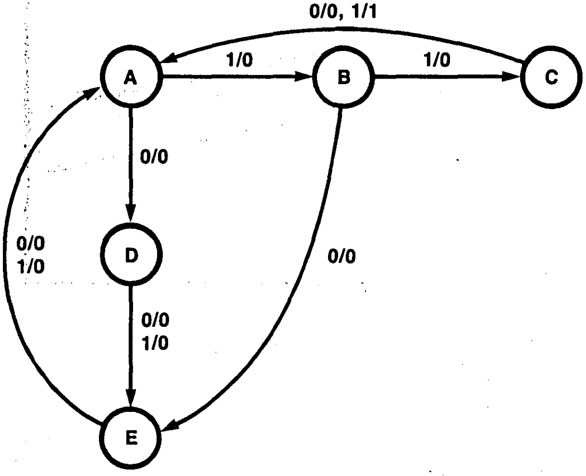
The dots shown in the figure represent samples taken by the vidicon. The vidicon produces a voltage that is proportional to the light intensity for each sample taken. The voltage is then quantized into seven levels. These seven levels correspond to light levels from white to black with intermediate levels of gray. Because there are seven quantized levels, a 3-bit quantizer is employed. These seven levels are then channel-encoded such that where the code 1 1 1 is reserved for the line sync pulse. The data are transmitted in a bit-serial manner. When the sync pulse is detected, the receiver camera flies back to start a new line, as shown in Figure 5.7.10. The use of the line sync pulse ensures that

0	0	0	WHITE
0	0	1	LEVELS OF GRAY
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	BLACK

all the lines start at a well-defined left edge. This prevents the occurrence of skewed lines which will distort the picture.

The PAL device under design is used as a sync pulse detector which will trigger the flyback circuit. There is another feature we need to design into this PAL device which automatically resets to the initial state after three input pulses. This reset procedure will ensure that no false output occurs due to consecutive sequences which produce an overlapping 1 1 1 sequence.

From the function description above, we can generate the State Diagram and State Table as shown in Figure 5.7.11 (a) and (b).



(A) STATE DIAGRAM

	X 0	X 1
A	D/0	B/0
B	E/0	C/0
C	A/0	A/1
D	E/0	E/0
E	A/0	A/0

(B) STATE TABLE

Figure 5.7.11 (A) State Diagram (B) State Table

Where A is the initial state, the sequence $A \xrightarrow{1/0} B \xrightarrow{1/0} C \xrightarrow{1/1} A$ will detect the sync pulse (1 1 1) and generate a "1" output. Note that the state diagram is arranged so that every sequence of length 3 returns the machine to the initial state A.

Since we have 5 different states (3 registers are enough), 1 input for serial data, 1 non-register output for sync pulse detecting, we may use the 16R4 to implement this application.

Let's assign these 5 different states as in Table 5.7.6.

STATE	STATE ASSIGNMENT Y_1, Y_2, Y_3
A	0 0 0
B	0 0 1
C	0 1 0
D	1 0 1
E	1 1 0

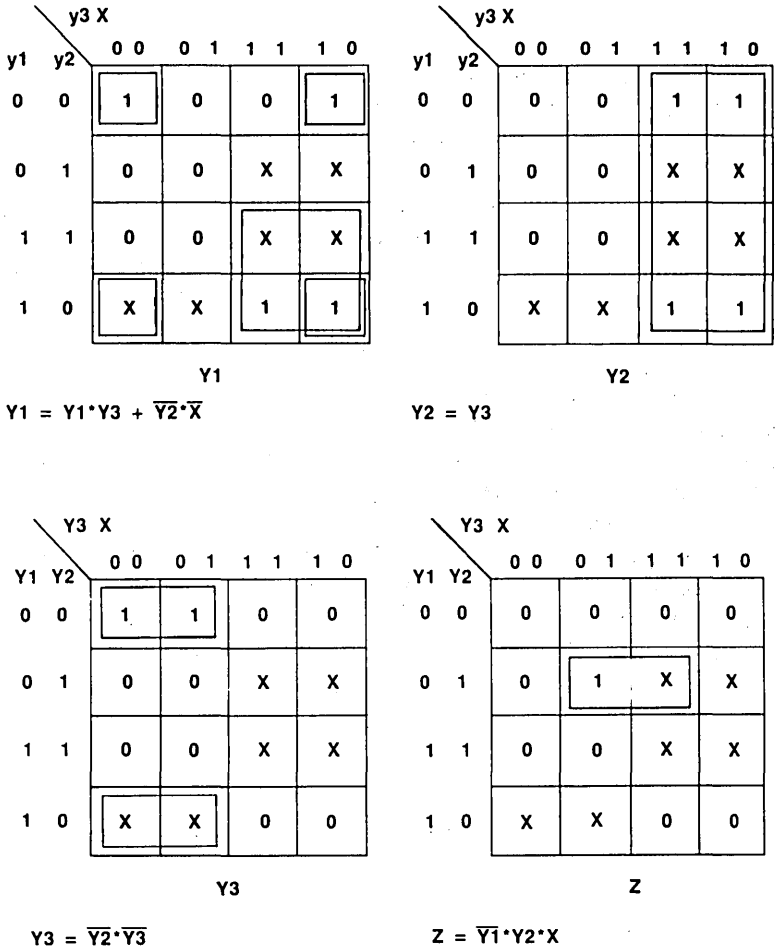
Table 5.7.6 State Assignment

Then from the State table Figure 5.7.11 (B) we get the Transition table shown in Table 5.7.7.

	X		X	
y1 y2 y3	0	1	0	1
0 0 0	1 0 1	0 0 1	0	0
0 0 1	1 1 0	0 1 0	0	0
0 1 0	0 0 0	0 0 0	0	1
0 1 1	X X X	X X X	X	X
1 0 0	X X X	X X X	X	X
1 0 1	1 1 0	1 1 0	0	0
1 1 0	0 0 0	0 0 0	0	0
1 1 1	X X X	X X X	X	X
	Y1 Y2 Y3		Z	

Table 5.7.7 Transition Table

From Table 5.7.7 Transition Table we can draw the K-map of each register output Y1, Y2, Y3 and the non-register output Z as shown in Figure 5.7.12.



y1 y2 \ y3 X	0 0	0 1	1 1	1 0
0 0	1	0	0	1
0 1	0	0	X	X
1 1	0	0	X	X
1 0	X	X	1	1

Y1

$Y1 = Y1 \cdot Y3 + \overline{Y2} \cdot \overline{X}$

y1 y2 \ y3 X	0 0	0 1	1 1	1 0
0 0	0	0	1	1
0 1	0	0	X	X
1 1	0	0	X	X
1 0	X	X	1	1

Y2

$Y2 = Y3$

Y1 Y2 \ Y3 X	0 0	0 1	1 1	1 0
0 0	1	1	0	0
0 1	0	0	X	X
1 1	0	0	X	X
1 0	X	X	0	0

Y3

$Y3 = \overline{Y2} \cdot \overline{Y3}$

Y1 Y2 \ Y3 X	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	1	X	X
1 1	0	0	X	X
1 0	X	X	0	0

Z

$Z = \overline{Y1} \cdot Y2 \cdot X$

Figure 5.7.12 K-map

Therefore, we get the logic equations as:

$$\begin{aligned}
Y1 &:= Y1 \star Y3 + \overline{Y2} \star \overline{X} \\
Y2 &:= Y3 \\
Y3 &:= \overline{Y2} \star \overline{Y3} \\
Z &= \overline{Y1} \star Y2 \star X
\end{aligned}$$

Summary

The four design examples are quite simple for purposes of illustration. The author has attempted to give the reader a very clear idea and to encourage the reader to use PAL devices. The reader can find other examples in the applications section of Chapter 8.

Here the author would like to point out one thing; "There are many different approaches to designing a PAL device circuit." Some users like to directly code the PAL device logic diagram (coding "x"). In this case, users may not need logic equations. But if circuits become more complicated, then the user will find that the logic equations are much easier to get than directly coding "x" in the PAL device logic diagram. There are many ways to develop logic equations. One approach is to use truth tables or transition tables. Another way, which is widely used, is from timing waveforms.

The user can draw the timing diagram for each output, then derive his logic equations from these timing waveforms. But no matter what method is used, the user still needs to know the K-map or other techniques (the Quine-McCluskey method is frequently used) to minimize his logic gates.

The author strongly recommends deriving the logic equations for PAL devices rather than coding "X" in the PAL device logic diagram. Then the user can take advantage of PAL device software (PLAN, PALASM, etc.) instead of manually coding the PAL device programming format sheet.

6

Software Support

Today a variety of software products makes the logic design engineer's task much easier. The designer can now focus on the intricacies of logic design at the Boolean level instead of filling in tedious fuse map charts, or worrying whether a standard logic part exists to implement the logic. Some of the traditional programmer vendors are now marketing full-fledged development systems or CAD systems that include the terminal, software and the hardware for fuse blowing, and logic verification. Other vendors market software only or programmer/verifier only. The key part of any development system is the software and this section describes the attributes of these products.

6.1 ADVANTAGES OF SOFTWARE-BASED PROGRAMMABLE LOGIC DESIGN

When programmable logic devices were first introduced, the only method for specifying the logic to be implemented was to manually code the status of each fuse on a form and then enter this information into a programmer. With a device like the PAL16L8 which has 2048 fuses, this manual method is clearly time-consuming and error-prone. Furthermore, these early programmers could not verify if the programmed device was functional. They could only check if the correct fuses were blown. Information about testing is found in Chapter 7.

The first phase in software development was the development of tools to eliminate the manual fuse-map entry. Users could enter Boolean equations in Sum-Of-Products format on a computer and the program would generate the fuse-map information which could be downloaded to a programmer unit (Figure 6.1.1).

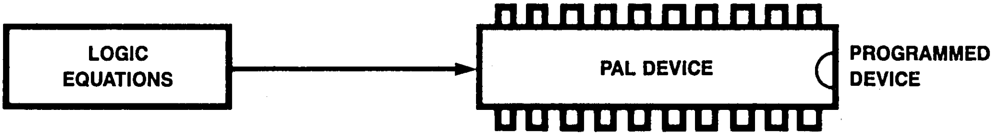


Figure 6.1.1 Early Role of Software

Subsequent developments in software goes further in providing two additional capabilities. The first area of improvement is logic design. Recent developments are emphasizing design tools for logic circuit design with features like high level logic design options and plans for logic minimization, and state-machine synthesis. The second area being addressed is that of functional testing of programmed devices. Most of the current software has features to perform simulation for design verification, i.e., verify if the user supplied test vectors match the logic conditions described by the equations for the logic being implemented. These test vectors can also be downloaded to a programmer which will perform a functional test on the programmed device (Figure 6.1.2).

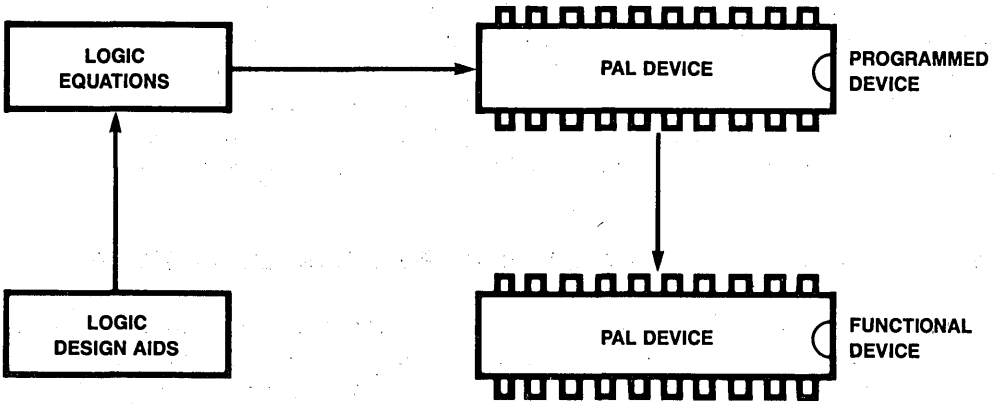


Figure 6.1.2 Expanded Role of Software

The next section describes National's contribution to advanced programmable logic design software called Programmable Logic Analysis by National (PLAN).

6.2 PROGRAMMABLE LOGIC ANALYSIS BY NATIONAL (PLAN)

PLAN is a set of interactive software tools for logic designers who will be using programmable logic devices in their circuits. The advantages of PLAN are that: (1) it is easy to use; and (2) it comes with clear and simple documentation that explains the numerous features of PLAN and the methods of accessing and using these features. PLAN also has a liberal sprinkling of error messages to help the user. PLAN does not have PALASM type input format constraints and is available on more than one operating system. The package actually contains three programs: PLUS, SERV, and PROG.

PLUS allows the user to define logic via Boolean equations and also selects an appropriate device and assigns pin-outs. The resulting equations, device, and pin-outs are stored in a file.

The next program, called SERV, can then be used to access the logic defined by PLUS for possible reassignment of the device and pin-out. When the device and pin-outs are finalized, SERV also displays the pin-out diagrams, fuse-maps and equations. For documentation purposes, the above data can also be printed out.

The third program, called PROG, takes the logic and pin assignment data and provides it to a programmer in a format that the user selects. This program can also acquire a previously defined file containing test vectors and download it to a programmer for functional verification.

The software package is available on 8-inch SSSD (Single Side Single Density) floppy disks to run under CP/M-80 and 5 1/4-inch SSSD floppy disks for operation under MS-DOS and APPLE-DOS. Future revisions will include other operating systems.

Boolean Entry

The Boolean entry operators that PLAN supports are shown in Table 6.2.1

=	EQUALITY
*	AND, PRODUCT
+	OR, SUM
/	COMPLEMENT
:=	REPLACED BY (AFTER CLOCK)
()	CONDITIONAL TRI-STATE
:+:	EXCLUSIVE OR

Table 6.2.1 Boolean Operators

An example of a logic equation using these operators is:

(/INP1 * INP2) OUT2 = /INP3 * INP4

A useful feature that PLAN offers during Boolean logic entry is the definition and inclusion of logic macros. Table 6.2.2 is an example of the use of the macro feature in PLAN.

MACRO IS EN1*/CK2	
INPUT	RESULTING EQUATION
OUT1 = INP1*/INP2 + /INP1*INP2	OUT1 = INP1*/INP*1EN1*/CK2 + /INP1*INP2*EN1*/CK2
OUT2 = INP3 + INP4 *INP5*INP6	OUT2 = INP3 + INP4EN1*/CK2 *INP5*INP6

Table 6.2.2 Macro Entry with PLAN

PLAN allows the user to edit the Boolean equations after entry. When the equations are finalized, the program will automatically select a device that can implement the defined logic and assign pin-outs to that device. This process is shown in Figure 6.2.3.

The information can also be stored in a file and the data in the file is essentially the information in Figure 6.2.3.

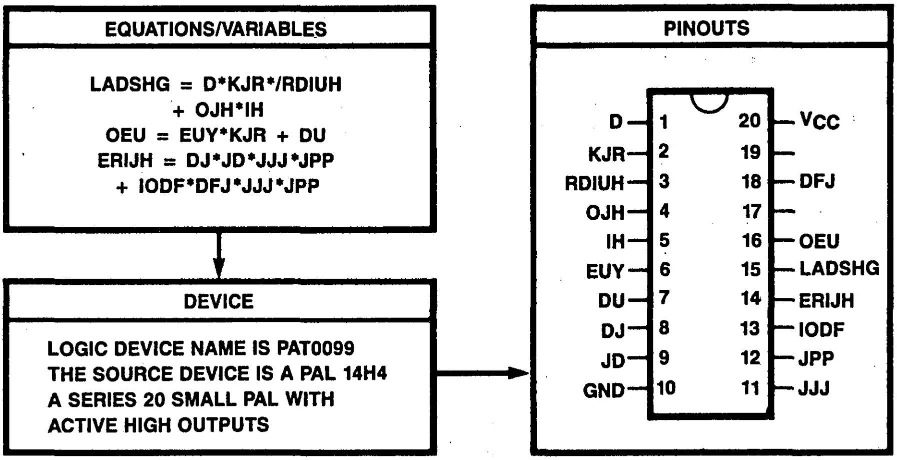


Figure 6.2.1 PLAN File Information

File Editing and Documentation

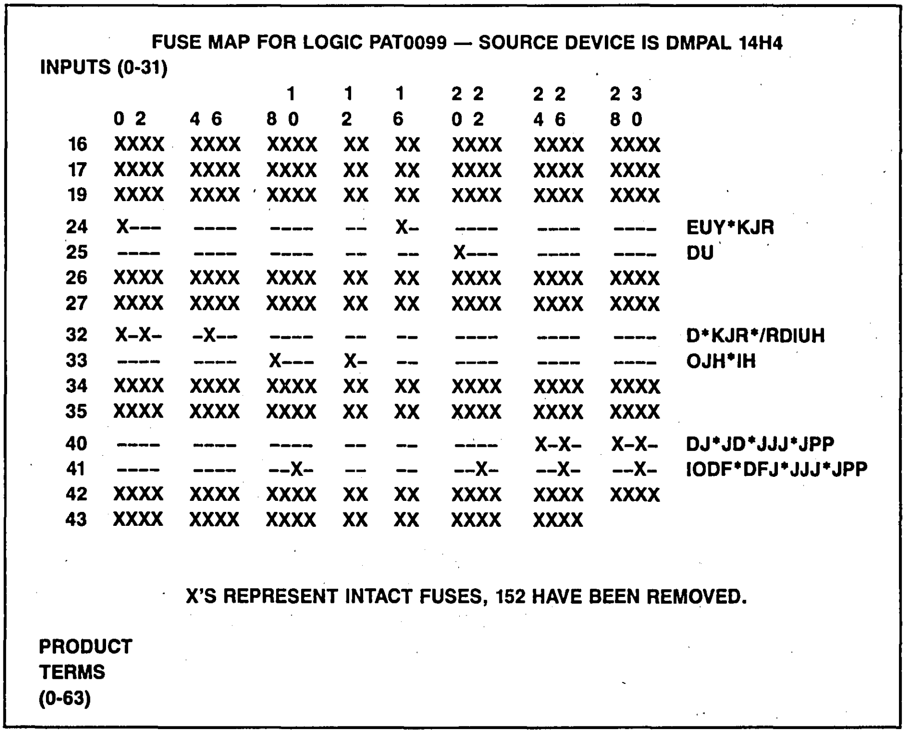
The program SERV can be used to change the selected device and also to change the pin-out assignment. When the device and pin-outs have been finalized, the device diagram with pin-out, the equations or the fuse-map of the programmed device can be printed out or viewed on the screen. Figure 6.2.4 is an example of the fuse-map display.

Programming and Testing

In order for a programmer to function, it has to receive the fuse-map information in a specified format. The third program in PLAN, called PROG, will provide the fuse-map information, at the users option, in any of the five formats listed in Table 6.2.3.

The programmer fuse-map data can be saved in a file for later use. PROG can also access a file containing test vectors and download them to a programmer for functional verification of a programmed device.

Because of its ability to support the various data formats, many programmers are supported by PLAN and most are physically interfaced through a standard RS-232 cable.



```
          FUSE MAP FOR LOGIC PAT0099 — SOURCE DEVICE IS DMPAL 14H4
INPUTS (0-31)
                        1      1    1     2 2     2 2     2 3
        0 2     4 6     8 0    2    6     0 2     4 6     8 0
  16    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX
  17    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX
  19    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX

  24    X---    ----    ----   --   X-    ----    ----    ----    EUY*KJR
  25    ----    ----    ----   --   --    X---    ----    ----    DU
  26    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX
  27    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX

  32    X-X-    -X--    ----   --   --    ----    ----    ----    D*KJR*/RDIUH
  33    ----    ----    X---   X-   --    ----    ----    ----    OJH*IH
  34    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX
  35    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX

  40    ----    ----    ----   --   --    ----    X-X-    X-X-    DJ*JD*JJJ*JPP
  41    ----    ----    --X-   --   --    --X-    --X-    --X-    IODF*DFJ*JJJ*JPP
  42    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX    XXXX
  43    XXXX    XXXX    XXXX   XX   XX    XXXX    XXXX


          X'S REPRESENT INTACT FUSES, 152 HAVE BEEN REMOVED.

PRODUCT
TERMS
(0-63)
```

Figure 6.2.2 Fuse-Map Display from PLAN

MMI Hex
JEDEC
Intel Hex
Standard Hex
PALASM Format

Table 6.2.3 Fuse-Map File Formats in PLAN

Order from: National Semiconductor Corporation PLAN
2900 Semiconductor Drive
M/S D3698
Santa Clara, CA. 95057
(408) 721-4107

6.3 OTHER SOFTWARE

CUPL™ by Assisted Technology

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

Universal

- PRODUCT SUPPORT: CUPL supports products from every manufacturer of of programmable logic. With CUPL you are free to use not only programmable logic. With CUPL you are free to use not only PALS, but also other programmable logic devices.
- PALASM CONVERSIONS: CUPL has a PALASM to CUPL language translator which allows for an easy conversion from your previous PALASM designs to CUPL.
- LOGIC PROGRAMMER COMPATIBILITY: CUPL produces a standard JEDEC download file and is compatible with any logic programmer that JEDEC files.

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:

- FLEXIBLE INPUT: CUPL gives the engineer complete freedom in entering logic descriptions for their design.
  - Equations
  - Truth Tables
  - State Machine Syntax
- 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.

- SHORTHAND FEATURES: Instead of writing out fully expanded equations CUPL provides varous shorthand capabilities such as:
  - List Notation: Rather than [A6,A5,A4,A3,A2,A1,A0]
    CUPL only requires [A7..0]
  - Bit Fields: A group of bits may be assigned to a name,
    as in 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 Distributive Property:
    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

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 throughout your work so there can be detailed explanations included in each part of the project.

Error Checking
CUPL includes a comprehensive error check capability with detailed error messages designed to lead you to the source of the problem.

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.

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.

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.

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.

CUPL-GTS™

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.

Many hardware designers, however, are most comfortable with the traditional logic schematic as a logic description format.

CUPL-GTS is a powerful combination of hardware and software which turns an IBM-PC type computer into a programmable logic workstation allowing the user to draw logic schematics for the function of a PAL. A basic premise in creating GTS was to provide a friendly environment where the user is isolated from the traditional keyboard as much as possible. 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. 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 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 into 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. 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 wndow according to the printer configuration file (PRINTCAP). The PRINTCAP file allows the user to configure the GTS print function for any dot matrix printer they might have.

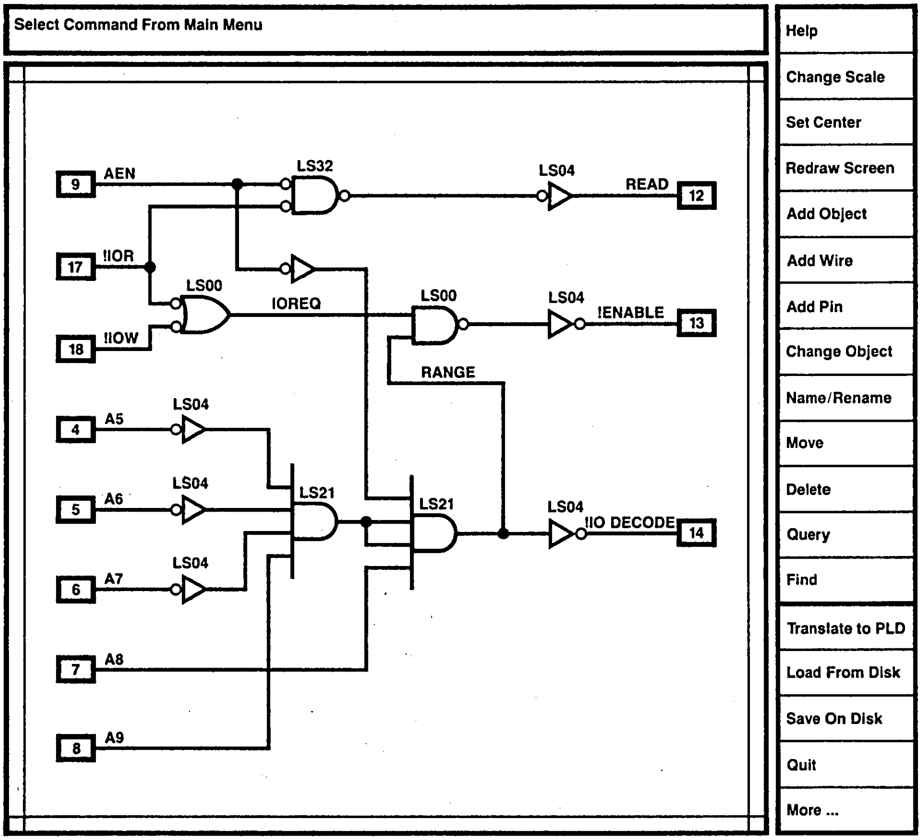
Often a logic description does not fit in a particular PAL due to a logic capacity (product-term) limitation. When this occurs, the universal capability of GTS will easily allow the user to try placing this same logic in a different PAL of a similar architecture.

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 designed 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 a simple I/O decoder design would appear on the CUPL-GTS screen prior to translation to a PAL16L8 or PAL16P8.



Select Command From Main Menu

Figure 6.3.1 CUPL-GTS Screen Display Example

PALASM

The oldest design aid for PAL devices is PALASM, which is a FORTRAN IV-based software package. PALASM accepts logic equations in a rigid format and assembles them into fuse-map data for programmers. In addition, PALASM also accepts user input test vectors, performs simulation and formats them to be programmer compatible. Table 6.3.2 lists the PALASM operators.

:	Comment follows
/	Complement, prefix to a pin name.
*	AND (product)
+	OR (sum)
:+:	XOR (exclusive OR)
:*:	XNOR (exclusive NOR)
( )	Conditional three-state
=	Equality
:=:	Replaced by after the low to high transition of the clock.

Table 6.3.2 PALASM Operators

ABEL™ by Data I/O

As the use of PALs and PLEs (PROMs) increases, high level design tools become 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 PALs, PLEs, and FPLAs 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. ABEL™ was developed by Data I/O Corporation, Redmond, WA.

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 created 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 is a design described in the ABEL™ design language. This design would be processed to verify its correctness and to reduce the number of terms required to implement it. The design is implemented in a PAL.

6809 Memory Address Decoder

Address decoding is a typical application of programmable logic devices, and the following describes the ABEL™ implementation of such a desing.

Design Specification

Figure 6.3.2 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.

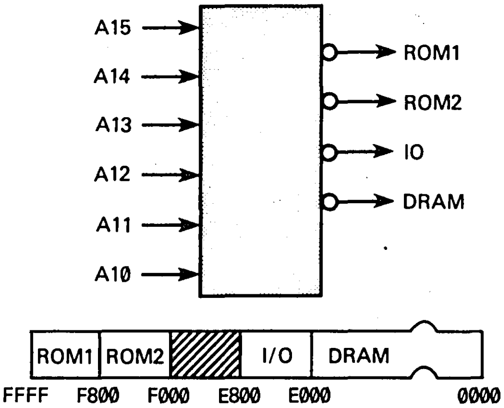


Figure 6.3.2 Block Diagram: 6809 Memory Address Decoder

Table 6.3.1 shows the address ranges associated with each section of memory. These address ranges can also be seen in figure 6.3.2.

Memory Section	Address Range (hex)
DRAM	0000-DFFF
I/O	E000-E7FF
ROM2	F000-F7FF
ROM1	F800-FFFF

Table 6.3.1 Address Ranges for 6809 Controller

Design Method

Figure 6.3.3 shows a simplified block diagram for the address decoder. The address decoder is implemented with simple Boolean equations employing both relational and logical operators as shown in figure 6.3.4. A significant amount of 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 6.3.1.

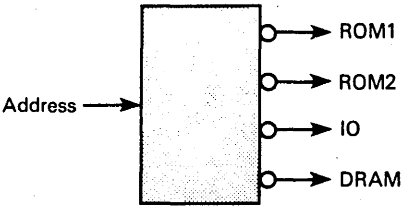
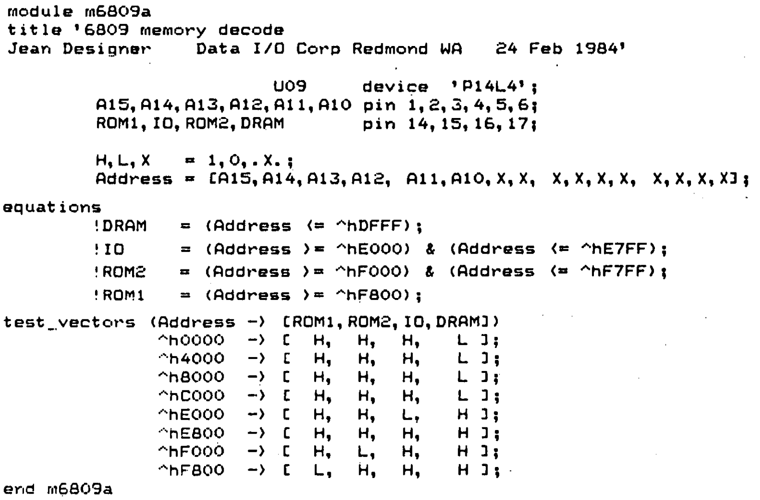


Figure 6.3.3 Simplified Block Diagram: 6809 Memory Address Decoder



```
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 6.3.4 Source File: 6809 Memory Address Decoder

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 sife of the "->" symbol. Inputs 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.*

Summary

A design described with the ABEL™ design language has been shown. This design shows how Boolean equations with logical and relational operators are used to describe an address decoder. Test vectors were written to test the function of the design using ABEL™'s simulator. In addition to the Boolean equations shown in this example, ABEL™ features truth tables and state diagrams. State diagrams allow the designer to fully describe state machines in terms of their states and state transitions. Truth tables specify designs in terms of their inputs and outputs, much like test vectors.

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.

6.4 SOFTWARE FOR TESTING PROGRAMMABLE LOGIC

Some of the test equipment vendors also have software that can be used for testing programmed devices in a production environment. These software packages do not have any design aids but have automatic test vector generation and simulation tools and are generally written to run on powerful mini-computers.

6.5 SOFTWARE VENDOR LIST

Listed below are the major software vendors for Programmable Logic.

NATIONAL SEMICONDUCTOR CORPORATION
PLAN
2900 Semiconductor Drive
M/S 16-198
P.O. Box 58090
Santa Clara, CA 95052-8090
(408) 721-4107

ASSISTED TECHNOLOGIES, INC.
2381 Zanker Road, Suite 150
San Jose, CA 95131

DATA I/O CORPORATION
10525 Willows Road N.E.
C-46
Redmond, WA 98052

A vendor who supplies software for production testing of Programmable Logic is provided below.

GENRAD
170 Tracer Lane
Waltham, MA 02254

7

Testing and Reliability

7.1 NATIONAL FACTORY TESTING

National's PAL devices include special test circuitry designed to permit thorough AC and DC testing to be accomplished on an unprogrammed unit. This test circuitry is used to ensure good programming yield and to verify that devices will meet all parametric and switching specifications after programming.

Each PAL device has special test fuses. These test fuses are blown during factory testing and demonstrate beyond reasonable doubt that the device is capable of opening all fuses when programmed by the user. They also increase the confidence level in unique addressing.

Table 7.1.1 shows the total number of fuses and test fuses for each device. Figure 7.1.1 shows the PAL test flow in National's factory.

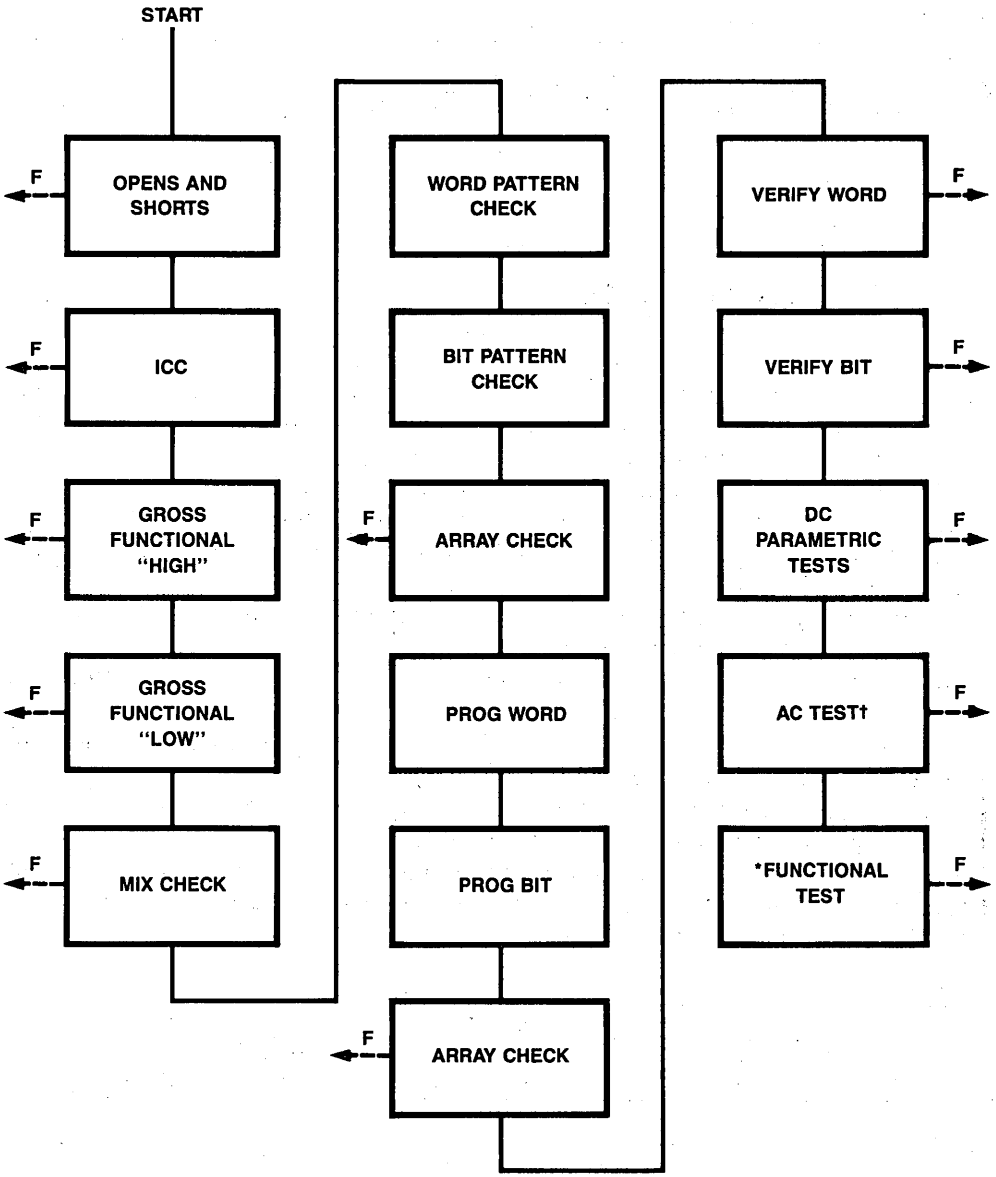
Since PAL devices are logic devices, in addition to testing the fuses blown their logic function should be tested after programming. This can be performed on a National tester, or on some PAL device programmers, using user defined test vectors or by comparison against a known good unit (fingerprint test).

Test vectors are relatively easy to generate for combinational designs using PAL devices. Sequential function testing is more difficult.

National's application Note #351 by Tom Wang tells the user how to generate these test vectors. National also supports customer test vectors and fully tests its custom order NML or programmed PAL devices.

	AND Array Organization							
Device Number	Input Lines	×	T/C	×	Product Lines	=	Number of Fuses	Number of Test Fuses
PAL10H8	10		2		16		320	42
PAL12H6	12		2		16		384	44
PAL14H4	14		2		16		448	46
PAL16H2	16		2		16		512	48
PAL16C1	16		2		16		512	48
PAL16L8	16		2		64		2048	98
PAL16R8	16		2		64		2048	98
PAL16R6	16		2		64		2048	98
PAL16R4	16		2		64		2048	98

Table 7.1.1 Test Fuses



† FOR SAMPLE ONLY
* FOR NML/PROGRAMMED PAL

Figure 7.1.1 PAL Device Test Flow

7.2 LOGIC VERIFICATION

PAL devices are not only memory devices, but also logic devices. Therefore, in addition to verifying the fuses blown after programming, we also need to verify the logic operation before it is put in a system. Logic verification provides assurance that a device will function in a board. Figure 7.2.1 shows the PAL device's architecture which will clarify the difference between fuse programming/verification and logic verification. The programming/verification circuit is required to allow custom configuration by the user. This circuit is operational only when a super voltage is applied to V_{CC}. Under normal 5.0 volt operation, this circuit is invisible and the logic circuit will take over. Therefore the skills we use to check the PAL device under normal 5.0 volt operation are called logic verification. The most important skill we use now is called functional test.

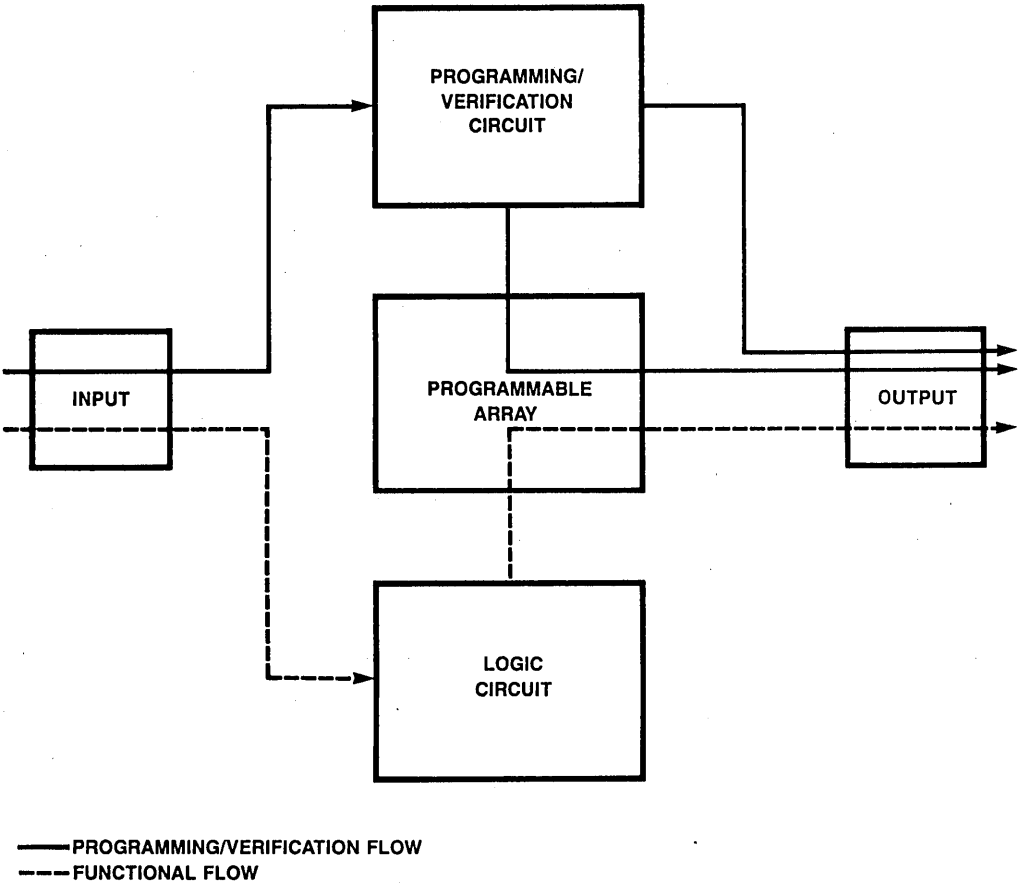

PROGRAMMING/ VERIFICATION CIRCUIT
PROGRAMMABLE ARRAY
LOGIC CIRCUIT
INPUT
OUTPUT
PROGRAMMING/VERIFICATION FLOW
FUNCTIONAL FLOW

Figure 7.2.1 PAL Device's Architecture

Functional testing must accomplish two purposes:

1) It must verify that the PAL device, after programming, performs the function intended.

2) It must verify the circuit removed through programming does not affect the PAL device's operation.

The functional testing technique relies on the test vectors. A test vector means a combination of desired input variable values and expected output variable values. The PAL device will be exercised by the desired input values. Then, the received outputs will be compared with the expected output values. The device is considered a "malfunction" if the comparison does not match. Figure 7.2.2 shows an example.

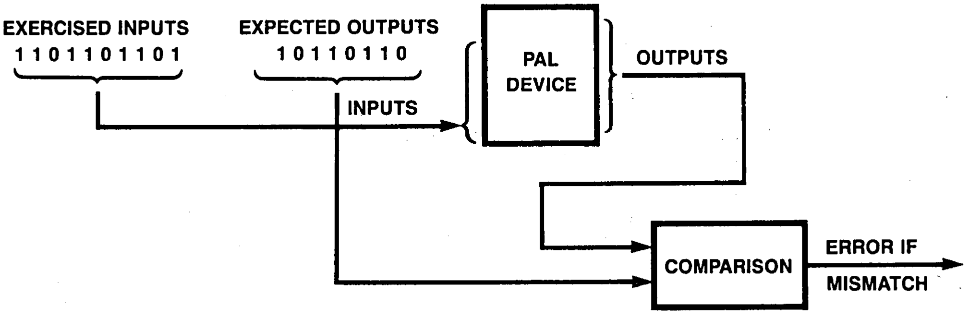


Figure 7.2.2 Function of Test Vector

There are many methods of generating test vectors:

1. Exhaustive — generate the whole different input combination and the expected output values. For instance, for 3-input AND gate in Figure 7.2.3, we get eight test vectors as in Table 7.2.1. For an n-inputs device, we get 2^n test vectors.

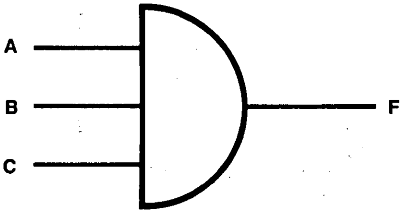


Figure 7.2.3 3-Input AND Gate

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 7.2.1 Test Vectors Generated by Exhaustive Methods

2. Fault modeling — Use the stuck at 0 and stuck at 1 technique to sensitize the different logic path. For instance, in Figure 7.2.3, there are three different paths, i.e. AF, BF and CF. Therefore we get six test vectors shown in Table 7.2.2 (a). Due to vector 1,3 and 5 being the same, we can reduce to four test vectors as in Table 7.2.2 (b).

A	B	C	F
1	1	1	1
0	1	1	0
1	1	1	1
1	0	1	0
1	1	1	1
1	1	0	0

(A)

A	B	C	F
1	1	1	1
0	1	1	0
1	0	1	0
1	1	0	0

(B)

Table 7.2.2 Test Vectors Generated by Fault Modeling

3. Structure Test — Only pick up the possible existing input states and their corresponding output states.

There is another skill to do the logic verification. It uses the signature analysis technique. This technique uses random input values exercising on a good device to generate different outputs. The outputs are manipulated in certain ways to get a "test sum" called a "signature." Then, using the same sequence of input values to another device we get its signature which is compared with the known good one. Some PAL device programmer vendors offer user fingerprint tests which are based on signature analysis techniques such as DATA I/O, Digital Media.

7.3 CUSTOMER'S RESPONSIBILITIES

The number of parts that are non-functional after programming is generally less than 2% and may be picked up during board-level check. However, the author strongly recommends that the user do the logic verification before putting PAL device components into the system.

Since the user defines the function of the PAL device, it is impossible for the supplier to perform full functional testing prior to shipment unless the user orders an NML or programmed PAL device from National.

It is the user's responsibility to generate test vectors or do the fingerprint test. The methods for generating test vectors was discussed in Section 7.2.

7.4 RELIABILITY DATA

Following is sample reliability data on National's PAL devices. For additional information please contact your National representative or distributor.

Product: Bipolar PALs (DM3300)
Package: Molded (N) and Hermetic (J)

Test Method: Dynamic (DHTL)/Static (SHTL) High Temperature Operating Life
Conditions: Continuous Operation at Rated Supply Voltage, and 125°C
Duration: 1000 Hours

File I.D.	Device	Package	Test	Sample	168	500	1000	Failure Mode
	Type	Type		Size	Hours	—	—	
RMB75131	16R4	J	DHTL	77	0	0	0	
RMB75133	16L8			77	0	0	0	
RMB75101	16R6			77	0	0	0	
RMB75137	16R6			77	0	1	0	Fuse verify and functional
RMB75096	16R4		SHTL	77	0	0	0	
RMB75132	16R4			77	0	0	0	
RMB75097	16L8			77	0	0	0	
RMB75142	16R8			77	0	0	0	
RMB75143	16L8	N	DHTL	77	0	0	0	
RMB75144	16R8			77	0	0	0	
RMB75190	16R4			77	0	0	0	
RMB75144	16R8		SHTL	77	0	0	0	
RMB75154	16L8			77	0	0	0	

Total Devices: 1001

Total Device Hours at 125°C: $1001 * 10^3$

Failure Rate at Stress = 0.2%/1000 Hours

Total Device Hours at 55°C, and 0.4EV = $12.012*10^6$

Failure Rate at 55°C, 0.4EV and 60% Confidence Level:

%/1000 Hours: 0.0168; PPM Hours: 0.168; Fits: 168; MTBF: $5.9*10^6$

Test Method: Temperature Humidity Bias Test
Conditions: Continuous Operation at Rated Supply Voltage, 85°C, and 85%RH
Duration: 1000 Hours

File I.D.	Device	Package	Sample	168	500	1000	Failure Mode
	Type	Type	Size	Hours	—	—	
RMB75143	16L8	N	77	0	0	0	
RMB75144	16R8		77	0	0	0	
RMB75199	16R4		77	0	0	0	

Total Devices: 231

Failure Rate at Stress: 0.4%/1000 Hours

7.5 PAL DEVICE FUNCTIONAL TESTING

Combinational and Sequential Circuits

Digital circuits can be classified as either combinational or sequential. Combinational circuits (e.g., decoder, multiplexer, adder, etc.) whose present value of the outputs at any time are functions of only the present circuit inputs at that time can be described as:

$$Y = F(X)$$

where F is Boolean sum of products transfer function (Figure 7.5.1).

INPUTS X
F(X)
OUTPUTS Y

Figure 7.5.1 Combinational Circuit

Sequential circuits (e.g., counter, shift register, accumulator, etc.) whose present value of the outputs at any given time will be the functions not only of the present circuit inputs at that time, but also the previous value of the outputs can be described as:

$$Y = F(X, Y)$$

where F is the Boolean Sum-of-Product transfer function. See (Figure 7.5.2).

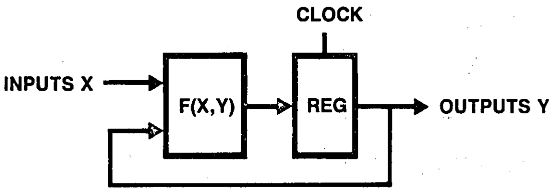


Figure 7.5.2 Sequential Circuit

Description of PAL (Programmable Array Logic) Device

Due to rapidly increasing integrated circuit technology, logic circuit designers face a difficult decision: should they use conventional TTL gates or custom LSI to implement desired combinational/sequential circuits.

Use of conventional TTL gates does not take advantage of the increased integration available. However, expensive and complicated software often makes custom LSI unsatisfactory. There is a big void between these two solutions. This void is now being addressed by semicustom approaches (e.g., PAL devices or gate array, etc). Since PAL devices have advantages over other semicustom chips in many areas (for instance, cost effectiveness, quick turnaround, complete software support, multi-source, etc.), it may be the best approach for the logic designer designing combinational/sequential circuits.

National offers the designer a family of PAL devices. See Table 7.5.1 for a broad overview of National's products.

PAL Device Design Procedure

Designing combinational circuits is straightforward. The first step is to define the circuit's function. The second step is to build a truth table. The third step is to minimize the truth table by using Karnaugh maps or Boolean algebra, in order to get the transfer function (i.e., logic equations). Step four is programming the circuits. Figure 7.5.3 is a flow diagram which applies to designing combinational PAL devices.

It is much more complicated to design a sequential circuit, as discussed in many textbooks and articles. Figure 7.5.4 is a flow diagram which applies to designing sequential PAL devices.

The last step in both Figures 7.5.3 and 7.5.4 is programming the PAL device. The entire procedure for programming a PAL device is shown in Figure 7.5.5. The first step is to generate the logic equations and function table. The second step is, using PAL device software tools (e.g., PALASM®, PLAN™, etc.), to create a bit pattern and exercise the function table, if any, in the logic equations. The third step is to load the bit pattern into a PAL device programmer to program and verify the fuse matrix. The fourth step is to functionally test the PAL device. The last step is to blow the security fuse. This last step is optional.

	High Speed	Ultra-High	Low Power	Package	
Standard	(25 ns)	Speed (15 ns)	(35 ns)	(Pins)	Description
(35 ns)					
10H8	10H8A		10H8A2	20	10 Input, 8 Output AND-OR Array
12H6	12H6A		12H6A2	20	12 Input, 6 Output AND-OR Array
14H4	14H4A		14H4A2	20	14 Input, 4 output AND-OR Array
16H2	16H2A		16H2A2	20	16 Input, 4 Output AND-OR Array
10L8	10L8A		10L8A2	20	10 Input, 8 Ouptut AND-OR Array
12L6	12L6A		12L6A2	20	12 Input, 6 Output AND-OR Array
14L4	14L4A		14L4A2	20	14 Input, 4 Output AND-OR Array
16L2	16L2A		16L2A2	20	16 Input, 2 Output AND-OR Array
16C1	16C1A		16L1A2	20	16 Input, 1 Output AND-OR/NOR Array
16L8	16L8A	16L8B	16L8A2	20	16 Input, 8 Output AND-OR-Inv Array
16R8	16R8A	16R8B	16R8A2	20	16 Input, 8 Output AND-OR-Reg Array
16R6	16R6A	16R6B	16R6A2	20	16 Input, 6 Output AND-OR Reg Array
16R4	16R4A	16R4B	16R4A2	20	16 Input, 4 Output AND-OR-Reg Array
(40 ns)					
12L10				24	12 Input, 10 Output AND-OR Array
14L8				24	14 Input, 8 Output AND-OR Array
16L6				24	16 Input, 6 Output AND-OR Array
18L4				24	18 Input, 4 Output AND-OR Array
20L2				24	20 Input, 2 Output AND-OR Array
20C1				24	20 Input, 1 Output AND-OR/NOR Array
	20L8A			24	20 Input, 8 Output AND-OR-Inv Array
	20R8A			24	20 Input, 8 Output AND-OR-Reg Array
	20R6A			24	20 Input, 6 Output AND-OR-Reg Array
	20R4A			24	20 Input, 4 Output AND-OR-Reg Array
(50 ns)					
20L10				24	20 Input, 10 Output AND-OR-Inv Array
20X10				24	20 Input, 10 Output AND-OR-XOR-Reg Array
20X8				24	20 Input, 8 Output AND-OR-XOR-Reg Array
20X4				24	20 Input, 4 Output AND-OR-XOR-Reg Array

Table 7.5.1 National's PAL Device Family

Description of Functional Table

In Figures 7.5.3, 7.5.4 and 7.5.5 we encounter a step called "generating function table." However, what is the meaning of a function table and why do we need it? A function table is a sequence of test conditions which are representative of the device in actual circuit operation. When we derive the logic equations by using Karnaugh maps or Boolean algebra, it is possible to introduce errors that may not be obvious. The function table is a means of expressing what we expect the PAL device to do in the system. PALASM or other software simulators will exercise the function table in the logic equations and report simulation errors. Then, we can correct the function table and/or the logic equations until no simulation error occurs.

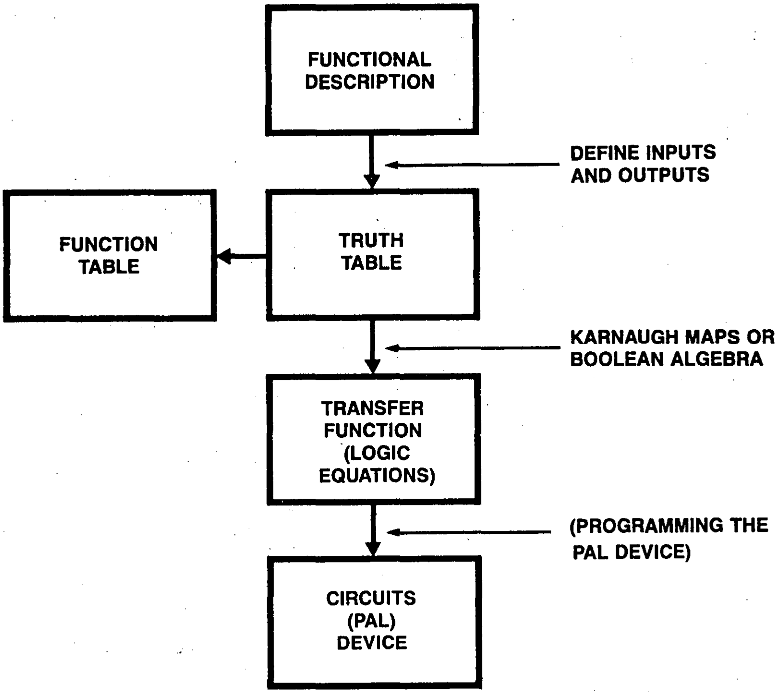

FUNCTIONAL DESCRIPTION
DEFINE INPUTS AND OUTPUTS
FUNCTION TABLE
TRUTH TABLE
KARNAUGH MAPS OR BOOLEAN ALGEBRA
TRANSFER FUNCTION (LOGIC EQUATIONS)
(PROGRAMMING THE PAL DEVICE)
CIRCUITS (PAL) DEVICE

Figure 7.5.3 Combinational PAL Device Design Steps

Even if both the logic equations and blown fuses are correct, there is no guarantee that the PAL device will function properly. PALASM or other software tools can generate test vectors from the function table entries and exercise these test vectors in the PAL device after it has been programmed. Even though the functional verification fallout is very small (typically less than 2%), it is necessary to perform this test at the device level. Ten devices on a board with a 2% device fallout translates into 18% fallout at the board level if these devices are not individually tested.

Thus, we can see that a good function table will provide a high degree of confidence that the design is correct. It will also help ensure that the PAL device will work properly the first time it is plugged into the system.

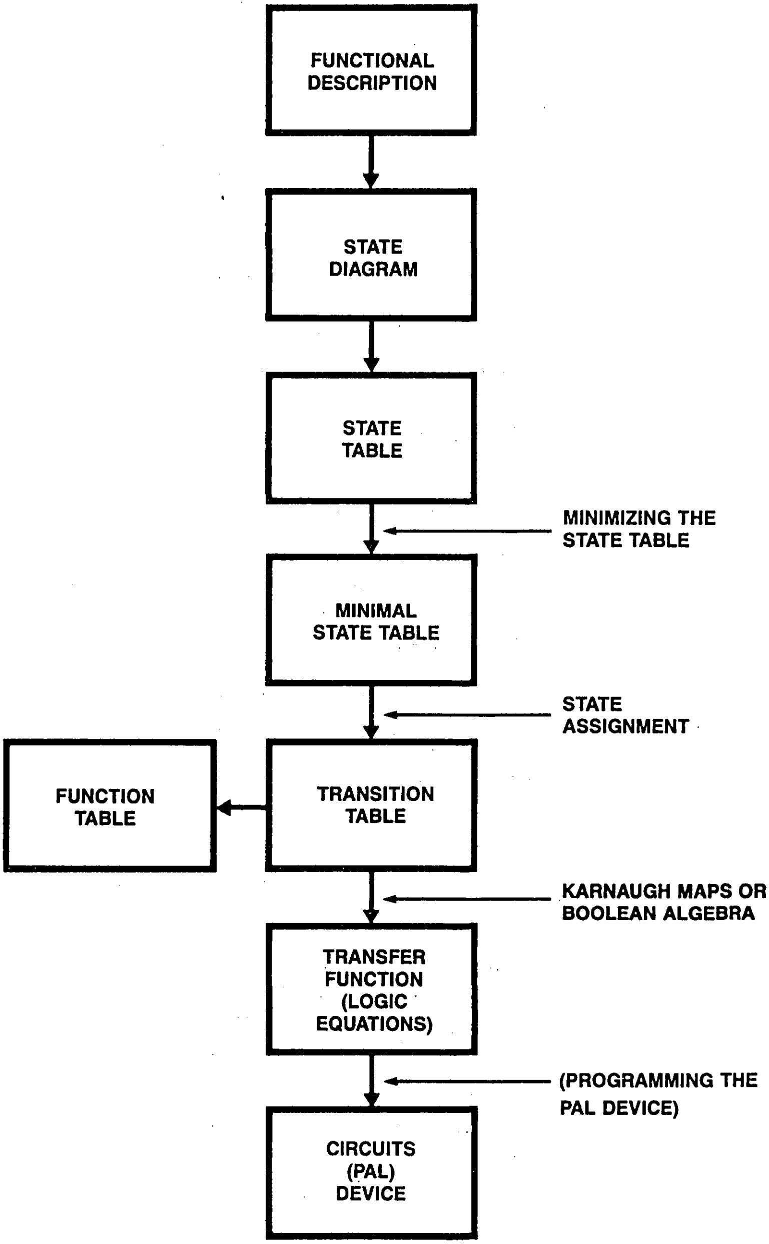


Figure 7.5.4 Sequential PAL Device Design Steps

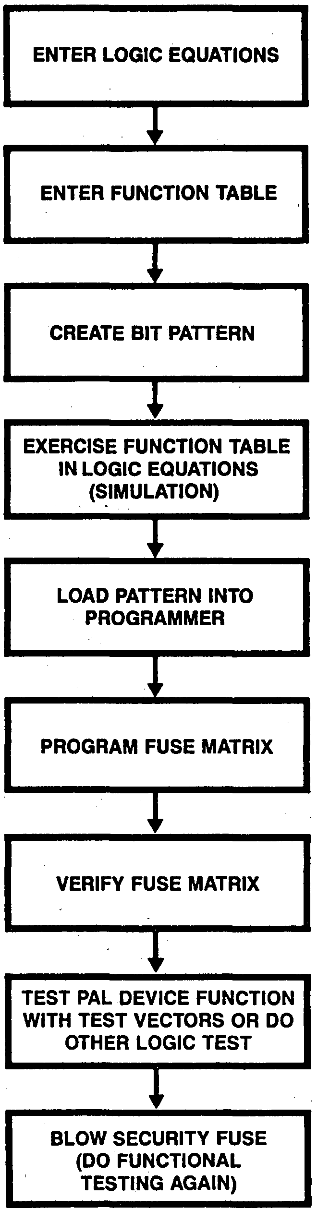

ENTER LOGIC EQUATIONS
ENTER FUNCTION TABLE
CREATE BIT PATTERN
EXERCISE FUNCTION TABLE IN LOGIC EQUATIONS (SIMULATION)
LOAD PATTERN INTO PROGRAMMER
PROGRAM FUSE MATRIX
VERIFY FUSE MATRIX
TEST PAL DEVICE FUNCTION WITH TEST VECTORS OR DO OTHER LOGIC TEST
BLOW SECURITY FUSE (DO FUNCTIONAL TESTING AGAIN)

Figure 7.5.5 PAL Device Programming Procedures

How to Generate Test Vectors and the Function Table from Logic Equations

It is the PAL device designer's responsibility to generate the function table since he/she knows the operation of the design best. However, if this is not possible, we can generate the function table manually from the existing logic equations. To do this, the correct logic equations are needed. Figure 7.5.6 outlines the procedure which will be detailed by examples in the next section. The "optimization" procedure is sometimes difficult and may need intuition. (Notice the different procedure between combinational and sequential PAL in the last step.)

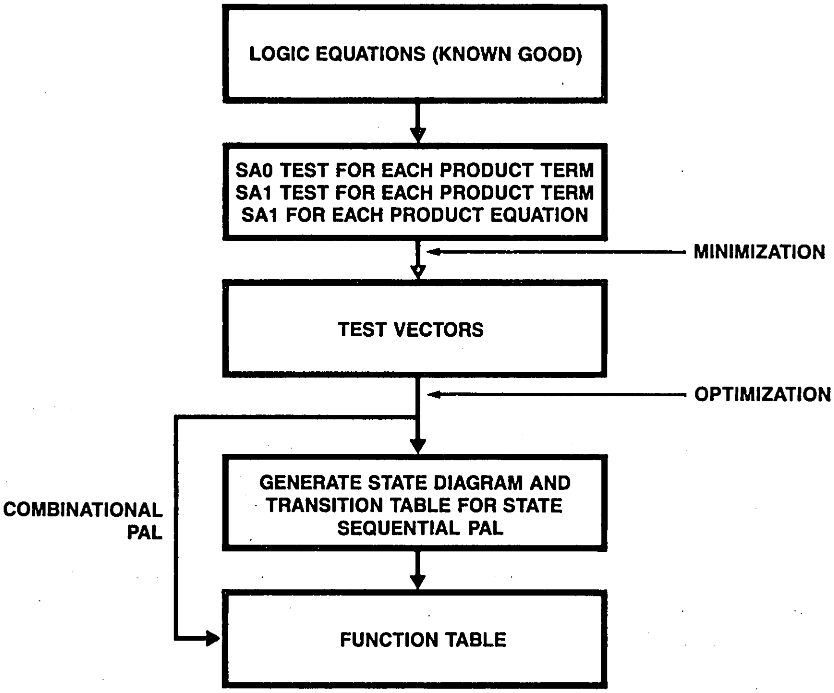

LOGIC EQUATIONS (KNOWN GOOD)
SA0 TEST FOR EACH PRODUCT TERM
SA1 TEST FOR EACH PRODUCT TERM
SA1 FOR EACH PRODUCT EQUATION
MINIMIZATION
TEST VECTORS
OPTIMIZATION
COMBINATIONAL PAL
GENERATE STATE DIAGRAM AND TRANSITION TABLE FOR STATE SEQUENTIAL PAL
FUNCTION TABLE

Figure 7.5.6 Test Vector and Function Table Creating Steps

Before going to the next section, a few conventions are defined. First, only the following symbols can be accepted in the test vectors or function table:

H—Logic High
L—Logic Low
X—Irrelevant "Don't Care"
Z—High Impedance
C—Clock
?—Undetermined
0 and 1 can be treated as Low and High.

Second, let's consider a general logic equation (or product equation)

$$O_1 = P1 + P2 + P3$$

where O_1 is the output; P1, P2 and P3 are the product terms.

$$\begin{aligned} \text{If } P1 &= I_1 * I_2 * /I_3 \\ P2 &= /I_2 * I_3 * I_5 \\ P3 &= I_6 * /I_8 * /I_9 \end{aligned}$$

where I_1, I_2, I_3, I_5, I_6, I_8 and I_9 are inputs.

Then the output O_1 will be

$$O_1 = I_1 * I_2 * /I_3 + /I_2 * I_3 * I_5 + I_6 * /I_8 * /I_9$$

where, I_1, I_2, $/I_3$, I_5, I_6, $/I_8$, $/I_9$ are called factors.

Consider a particular test vector, V1, which will cause the product term P1 to be high and the product terms P2 and P3 to be low. In this case the output, O_1, will be high. Now, if a fault is created by the PAL device which causes P1 to be low, then the output, O_1, will be low which is different from the fault-free condition. This fault condition is called "stuck at 0" (SA0) fault. Thus, the vector, V1, is able to detect the product term, P1, for the SA0 fault and we can say that V1 covers P1 for the SA0 fault.

In order to get P1 to be high, all factors of P1 should be high (i.e., I_1, I_2 and $/I_3$ are high). Both I_2 = high and $/I_3$ = high will cause P2 to be low no matter what I_5 is. Therefore, the vector of:

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	O_1	O_2	O_3	O_4	O_5	O_6
H	H	L	X	X	L	X	X	X	X	X	X	H	X	X	X	X	X

will cover P1 for the SA0 fault.

Similarly, if there is another vector, V2, which causes P1 to be low† (only one factor of P1 is low, the other factors of P1 are high) provided that P2 and P3 are low, then the output, O_1, is low. Now if a fault is created by the PAL device which causes P1 to be high then the output, O_1, will be high which is different from the fault-free condition.

† *To talk about letting a product term which is under test be low means that we only force one factor of this term to be low and the other factors should remain high.*

This fault condition is called "stuck at 1" (SA1) fault. Thus, the vector, V2, is able to detect the product term, P1, for SA1 fault and we can say that V2 covers P1 for SA1 fault.

For example, if I_1 is low, I_2 and $/I_3$ are high, the P1 is low. Therefore the vector of

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	O_1	O_2	O_3	O_4	O_5	O_6
L	H	L	X	X	L	X	X	X	X	X	X	L	X	X	X	X	X

will cover P1 for the SA1 fault.

Similarly, the following vectors will cover P1 for the SA1 fault, too.

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	O_1	O_2	O_3	O_4	O_5	O_6
H	L	L	X	X	L	X	X	X	X	X	X	L	X	X	X	X	X
H	H	H	X	X	L	X	X	X	X	X	X	L	X	X	X	X	X

To get an SA1 fault test for a product equation, generate a vector which sets all the factors in each product term to be low. The output of this product equation will then be low. If a fault is created by an AND or OR gate of the PAL device which causes the product term to be high, then the output will be high, which is different from the fault-free condition. For example, if I_1, I_2, $/I_3$, I_5, I_6, $/I_8$, $/I_9$ are low, then the following vector will cover equation O_1 for an SA1 fault.

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	O_1	O_2	O_3	O_4	O_5	O_6
L	L	H	X	L	L	X	H	H	X	X	X	L	X	X	X	X	X

A good function table should cover all of the product terms for the SA0 and SA1 faults. The Product Term Coverage (PTC) is calculated as:

$$\text{PTC} = \frac{\text{Total \# of SA0 Faults Tested} + \text{Total \# of SA1 Faults Tested}}{2 \times \text{Total Number of Product Terms}} \times 100\ (\%)$$

To achieve 100% PTC is the goal of generating a function table. PALASM version 1.5 and up will inform the user of:

- Total number of SA1 faults tested
- Total number of SA0 faults tested
- Product term coverage (PTC)

In case all the product terms are not covered, the user receives a message which tells him the product term and the type of fault for which it was not tested (e.g., "Product P_2 of EQN 1 Untested (SA0) Fault"). This implies that the user must update the function table by including vectors which will cover product terms for the faults.

7.6 EXAMPLES OF TESTING

Example 1: Combinational PAL12H6

```
PAL12H6
PTAN301
Tom Wang
Portion of random control logic for 8086 CPU board
PD EN ED EA S1 SA E1 DO DE GND SO NC3 NO C3 HA SS LA MW PW VCC
MW = /SO + PW * DE              ;   (1)
LA = /SA * /DO                  ;   (2)
SS = S1 * PD * /SA              ;   (3)
HA = S1 * PD * /SA * EA * E1    ;   (4)
C3 = PD * ED * EA               ;   (5)
NO = PD * /EN
```

Description

This is a portion of random control logic for 8086 CPU board. See (Figure 7.5.7).

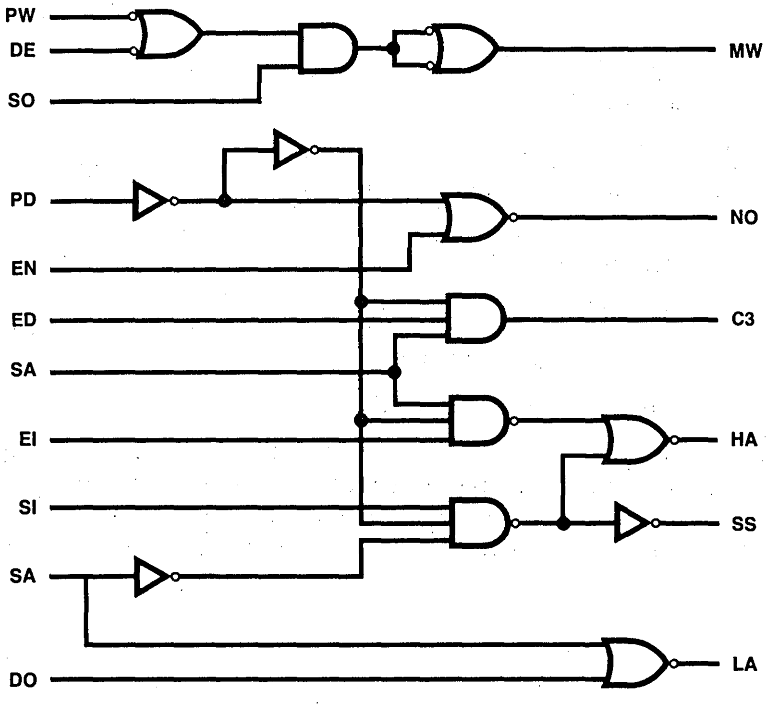


Figure 7.5.7 Logic Circuit of Example 1

The generation of function table is described in the following steps:

Step 1: Get Test Vector Coding Form; Fill in the input and output names.

Step 2: Exercise the product term 1 (/SO) of equation 1.
SA0 Fault Testing: Let PT1 be high and PT2 be low, then the output of equation 1, MW, should be high; so, we get vector 1.
SA1 Fault Testing: Let PT1 and PT2 be low, then the output of equation 1, MW should be low; so we get vector 2.

Step 3: Exercise product term 2 (PW * DE) of equation 1.
SA0 Fault Testing: Let PT1 be low and PT2 be high, then the output of equation 1, MW, should be high (i.e., vector 3).
SA1 Fault Testing: Let PT1 and PT2 be low, then the output of equation 1, MW, should be low.
Since PT2 consists of two factors, PW and DE, we create two SA1 test vectors (i.e., vectors 4 and 5).

Step 4: SA1 Fault Testing for product equation 1.
Let PT1 and PT2 be low, then the output of equation 1, MW, should be low (i.e., vector 6).
This step is similar to the SA1 test in step 3 but is different, since all the factors in this equation were set to be low.

Step 5: Exercise product term 1 (/SA * /DO) of equation 2.
SA0 Fault Testing: Let PT1 be high, then the output LA should be high.
SA1 Fault Testing: Let PT1 be low, then the output LA should be low.
So, we get vectors 7, 8, and 9 in Table 7.5.2

Step 6: SA1 fault test for product equation 2, we get vector 10.

Step 7: Continue to exercise the rest of the product terms, completing all 31 test vectors (Table 7.5.2).

Step 8: Optimize the test vectors to get the function table.

1) Because of vector 2, we don't need vectors 4 and 6.
2) Combine vectors 7–10 with vectors 1–6.
3) Rearrange vectors 11–15, then combine with the preceding vectors.
4) Merge vectors 28–31 with vectors 23–27.
5) This results in only 17 vectors (Table 7.5.3).
6) These 17 vectors can still be minimized by comparison and intuition to get only 7 vectors (Table 7.5.4).
7) By inserting "X" into unused spaces, the result is Table 7.5.5, which is the function table.

Inputs																Outputs									
	PD	EN	ED	EA	SI	SA	EI	DO	DE	SO	NC3	PW				NO	C3	HA	SS	LA	MW				
1									X	L		L									H				
2									X	H		L									L				
3									H	H		H									H				
4									H	H		L									L				
5									L	H		H									L				
6									L	H		L									L				
7						L		L												H					
8						H		L												L					
9						L		H												L					
10						H		H												L					
11	H				H	L																			
12	H				L	L													H						
13	L				H	L													L						
14	H				H	H													L						
15	L				L	H													L						
16	H			H	H	L	H											H							
17	H			H	L	L	H											L							
18	L			H	H	L	H											L							
19	H			H	H	H	H											L							
20	H			L	H	L	H											L							
21	H			H	H	L	L											L							
22	L			L	L	H	L											L							
23	H		H	H													H								
24	L		H	H													L								
25	H		L	H													L								
26	H		H	L													L								
27	L		L	L													L								
28	H	L														H									
29	L	L														L									
30	H	H														L									
31	L	H														L									

Table 7.5.2 Test Vectors

	Inputs															Outputs									
	PD	EN	ED	EA	SI	SA	EI	DO	DE	SO	NC3	PW				NO	C3	HA	SS	LA	MW				
1	H				H	L		L	X	L		L							H	H	H				
2	H				H	H		L	X	H		L							L	L	L				
3	H				L	L		H	H	H		H							L	L	H				
4	L				L	H		H	L	H		H							L	L	L				
5	L				H	L													L						
6	H			H	H	L	H											H							
7	H			H	L	L	H											L							
8	L			H	H	L	H											L							
9	H			H	H	H	H											L							
10	H			L	H	L	H											L							
11	H			H	H	L	L											L							
12	L			L	L	H	L											L							
13	H	L	H	H												H	H								
14	L	L	H	H												L	L								
15	H	H	L	H												L	L								
16	H		H	L													L								
17	L	H	L	L												L	L								

Table 7.5.3 Test Vectors

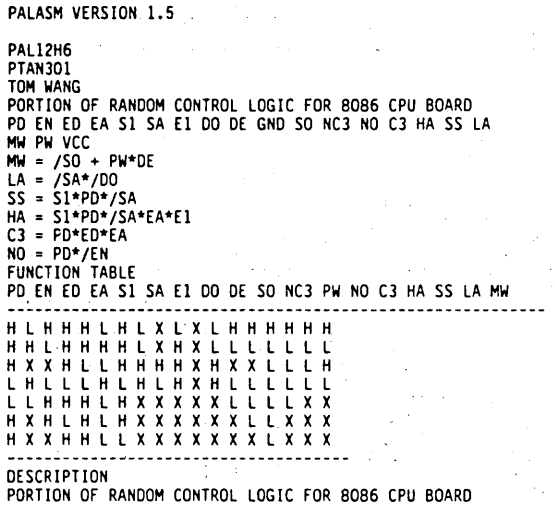
	Inputs															Outputs									
	PD	EN	ED	EA	SI	SA	EI	DO	DE	SO	NC3	PW				NO	C3	HA	SS	LA	MW				
1	H	L	H	H	H	L	H	L	X	L		L				H	H	H	H	H	H				
2	H	H	L	H	H	H	H	L	X	H		L				L	L	L	L	L	L				
3	H			H	L	L	H	H	H	H		H						L	L	L	H				
4	L	H	L	L	L	H	L	H	L	H		H				L	L	L	L	L	L				
5	L	L	H	H	H	L	H									L	L	L	L						
6	H		H	L	H	L	H										L	L							
7	H			H	H	L	L											L							

Table 7.5.4 Final Test Vectors

	Inputs															Outputs									
	PD	EN	ED	EA	SI	SA	EI	DO	DE	SO	NC3	PW				NO	C3	HA	SS	LA	MW				
1	H	L	H	H	H	L	H	L	X	L	X	L				H	H	H	H	H	H				
2	H	H	L	H	H	H	H	L	X	H	X	L				L	L	L	L	L	L				
3	H	X	X	H	L	L	H	H	H	H	X	H				X	X	L	L	L	H				
4	L	H	L	L	L	H	L	H	L	H	X	H				L	L	L	L	L	L				
5	L	L	H	H	H	L	H	X	X	X	X	X				L	L	L	L	X	X				
6	H	X	H	L	H	L	H	X	X	X	X	X				X	L	L	X	X	X				
7	H	X	X	H	H	L	L	X	X	X	X	X				X	X	L	X	X	X				

Table 7.5.5 Final Function Table

The following are printouts of PAL device design specifications, function table, pinout list, fuse map, simulation result, and fault testing result. We get 100% PTC!



```
PALASM VERSION 1.5

PAL12H6
PTAN301
TOM WANG
PORTION OF RANDOM CONTROL LOGIC FOR 8086 CPU BOARD
PD EN ED EA S1 SA E1 DO DE GND SO NC3 NO C3 HA SS LA
MW PW VCC
MW = /SO + PW*DE
LA = /SA*/DO
SS = S1*PD*/SA
HA = S1*PD*/SA*EA*E1
C3 = PD*ED*EA
NO = PD*/EN
FUNCTION TABLE
PD EN ED EA S1 SA E1 DO DE SO NC3 PW NO C3 HA SS LA MW
-----------------------------------------------------------
H L H H H L H L X L X L H H H H H H
H H L H H H H L X H X L L L L L L L
H X X H L L H H H H X H X X L L L H
L H L L L H L H L H X H L L L L L L
L L H H H L H X X X X X L L L L X X
H X H L H L H X X X X X X L L X X X
H X X H H L L X X X X X X X L X X X
-------------------------------------
DESCRIPTION
PORTION OF RANDOM CONTROL LOGIC FOR 8086 CPU BOARD
```

```
TOM WANG

                 ***************   ***************
                 *               * *              *
              ****                              ****
      PD      * 1*              P A L           *20*   VCC
              ****                              ****
                 *             1 2 H 6          *
              ****                              ****
      EN      * 2*                              *19*   PW
              ****                              ****
                 *                              *
              ****                              ****
      ED      * 3*                              *18*   MW
              ****                              ****
                 *                              *
              ****                              ****
      EA      * 4*                              *17*   LA
              ****                              ****
                 *                              *
              ****                              ****
      S1      * 5*                              *16*   SS
              ****                              ****
                 *                              *
              ****                              ****
      SA      * 6*                              *15*   HA
              ****                              ****
                 *                              *
              ****                              ****
      E1      * 7*                              *14*   C3
              ****                              ****
                 *                              *
              ****                              ****
      DO      * 8*                              *13*   NO
              ****                              ****
                 *                              *
              ****                              ****
      DE      * 9*                              *12*   NC3
              ****                              ****
                 *                              *
              ****                              ****
     GND      *10*                              *11*   SO
              ****                              ****
                 *                              *
                 ********************************
```

```
TOM WANG

  1 10111010XXOXHHHHHH01
  2 11011110XX1XLLLLLL01
  3 1XX100111X1XXXLLLH11
  4 010001010X1XLLLLLL11
  5 0011101XXXXXLLLLXXX1
  6 1X10101XXXXXXLLXXXX1
  7 1XX1100XXXXXXXLXXXX1

PASS SIMULATION            49           8

TOM WANG

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

BEG*FPLT PAL12H6   8
  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 ---- ---- --00 --00 --00 --00 ---- ---X /SO
  9 ---- --X- --00 --00 --00 --00 ---- X--- PW*DE
 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 ---- ---- --00 --00 -X00 --00 -X-- ---- /SA*/DO
 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 X-00 -X00 --00 ---- ---- S1*PD*/SA
 25 XXXX XXXX XX00 XX00 XX00 XX00 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 --X- ---- X-00 X-00 -X00 X-00 ---- ---- S1*PD*/SA*EA*E1
 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 --X- X--- X-00 --00 --00 --00 ---- ---- PD*ED*EA
 41 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 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

 48 -XX- ---- --00 --00 --00 --00 ---- ---- PD*/EN
 49 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 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

END*FPLT


LEGEND:  X : FUSE NOT BLOWN (L,N,0)    - : FUSE BLOWN   (H,P,1)
         0 : PHANTOM FUSE   (L,N,0)    0 : PHANTOM FUSE (H,P,1)

NUMBER OF FUSES BLOWN =  206


TOM WANG

  1 10111010XX0XHHHHHH01
  2 11011110XX1XLLLLLL01
  3 1XX100111X1XXXLLLH11
  4 010001010X1XLLLLLL11
  5 0011101XXXXXLLLLXXX1
  6 1X10101XXXXXXLLXXXX1
  7 1XX1100XXXXXXXLXXXX1


PASS SIMULATION          49           8

NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE =  7

NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE =  7

PRODUCT  TERM   COVERAGE                   =100%
```

The differences between sequential and combinational circuits have been discussed. The output of sequential circuits is a function not only of the present inputs, but the previous outputs.

There are two kinds of outputs in the sequential PAL device: registered output, and non-registered output. For example, pin 14 of the PAL16R4 is a registered output; pin 13 is a non-registered output. Different combinations of registered outputs are defined as different states. Each present-state is related to the present inputs and previous state, so the function table vectors need to be arranged in proper sequential order.

Furthermore, since the previous state is obtained from the previous vector, it is necessary to "initialize" the registers to a "known state". (Output is a function of the inputs but is independent of the previous state, similar to a clear or preset function).

The following is an example of the sequential PAL16R4. Referring to Figure 7.5.6, generate the state diagram and state transition table to derive the proper function table.

Example 2: Sequential PAL16R4

```
PAL16R4
PTAN302
Tom Wang
Op code analyzer

CLK  /2B12  /2B23  /B2B1  /B2B3  /3B  /B3B  /B1B  GND  /EN  F1ST  /ILLOP
/C    /B     /A     /17    /RD    F23    VCC
If (VCC)  /F1ST = F23                                          ;  (1)
If (VCC)  ILLOP = /A * /B  *  /C                               ;  (2)
C: = A * /B * /C * /B3B + /A * /B * C * /B2B2 + RD + A * B * C * /B1B + A * /B * C *
     /B2B3 * /3B + /A * B * /B2B1                              ;  (3)
B: = A * /B * /C * /B3B + /A * /B * C * /B2B2 + RD + A * B * C * /B1B * /2B23 +
     A * /B * C * /B2B3 + /A * B * /B2B1                       ;  (4)
A: = A * /B * /C * /B3B + /A * /B * C * /B2B2 + RD + A * B * C * /B1B * /2B12 +
     A * /B * C * /B2B3 + /A * B * /B2B1 + B * /C              ;  (5)
17: = A * B * C                                                ;  (6)
If (VCC)  /F23 = /A * /B * /C + A * B * C                      ;  (7)
```

Description

The function of this PAL device is to analyze the incoming op code.
The generation of the function table is described in the following steps:

Step 1: Get test vector coding form. Fill in the input and output names. Since the outputs C, B and A act as inputs as well, they appear on both sides and are considered first because they feed back to themselves. Therefore, equations 3, 4, and 5 are exercised first.

Step 2: Exercise product term 1 of equation 3.

SA0 Fault Testing: Let PT1 (A * /B * /C * /B3B) be high and PT2, 3, 4, 5, and 6 be low; the output of equation 3 should be high; so, we get vector 1 in Table 7.5.6.

SA1 Fault Testing: Let PT1, 2, 3, 4, 5, and 6 be low; the output of equation 3 should be low; so, we get vectors 2, 3, 4, and 5 in Table 7.5.6.

Step 3: Exercise product term 2 of equation 3.

SA0 Fault Testing: Let PT2 be high and PT1, 3, 4, 5, and 6 be low; the output of equation 3 should be high; so, we get vector 6 in Table 7.5.6.

SA1 Fault Testing: Let PT1 2, 3, 4, 5, and 6 be low; the output of equation 3 should be low; so, we get vectors 7, 8, 9, and 10 in Table 7.5.6.

Step 4: Exercise product term 3 of equation 3 (only SA0 fault testing is needed).

SA0 Fault Testing: Let PT3 be high and PT1, 2, 4, 5, and 6 be low; the output of equation 3 should be high; so, we get vector 11 in Table 7.5.6.

Step 5: Continue to exercise the rest of the product terms, completing all of equation 3.

Step 6: SA1 fault test for product equation 3; so, we get vector 25.

Step 7: Repeat step 2 through step 6 for equation 4; i.e.,
SA0 Fault Testing: Let PT1 be high and PT2, 3, 4, 5, and 6 be low; the output of equation 4 should be high.
SA1 Fault Testing: Let PT1, 2, 3, 4, 5, and 6 of equation 4 be low, the output of equation 4 should be low.
SA0 Fault Testing for PT2, SA1 Fault Testing for PT2.
SA0 Fault Testing for PT3, SA1 Fault Testing for PT3.
SA0 Fault Testing for PT4, SA1 Fault Testing for PT4.
SA0 Fault Testing for PT5, SA1 Fault Testing for PT5.
SA0 Fault Testing for PT6, SA1 Fault Testing for PT6.
SA0 Fault Testing for equation 4.
So, we get vectors 26 to 50.

Step 8: Repeat step 2 through step 6 for equation 5: i.e.,
SA0 Fault Testing: Let PT1 be high and PT2, 3, 4, 5, 6, and 7 be low; the output of equation 5 should be high.
SA1 Fault Testing: Let PT1, 2, 3, 4, 5, 6, and 7 of equation 5 be low; the output of equation 5 should be low.
SA0 Fault Testing for PT2, SA1 Fault Testing for PT2.
SA0 Fault Testing for PT3, SA1 Fault Testing for PT3.
SA0 Fault Testing for PT4, SA1 Fault Testing for PT4.

Inputs															Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
1								L			L	L	H	L				H							
2								L			L	L	L	L				L							
3								L			L	H	H	L				L							
4								L			H	L	H	L				L							
5								H			L	L	H	L				L							
6					L						H	L	L	L				H							
7					L						H	L	H	L				L							
8					L						H	H	L	L				L							
9					L						L	L	L	L				L							
10					H						H	L	L	L				L							
11														H				H							
12									L		H	H	H	L				H							
13									L		H	H	L	L				L							
14									L		H	L	H	L				L							
15									L		L	H	H	L				L							
16									H		H	H	H	L				L							
17						L	L				H	L	H	L				H							
18						L	L				H	L	L	L				L							
19						L	L				H	H	H	L				L							
20						L	L				L	L	H	L				L							
21						L	H				H	L	H	L				L							
22						H	L				H	L	H	L				L							
23				L								H	L	L				H							
24				H								H	L	L				L							
25				H	H	H	H	H	H		H	H	L	L				L							
26								L			L	L	H	L					H						
27								L			L	L	L	L					L						
28								L			L	H	H	L					L						
29								L			H	L	H	L					L						
30								H			L	L	H	L					L						
31					L						H	L	L	L					H						
32					L						H	L	H	L					L						
33					L						H	H	L	L					L						
34					L						L	L	L	L					L						
35					H						H	L	L	L					L						
37														H					H						
37			L						L		H	H	H	L					H						
38			L						L		H	H	L	L					L						
39			L						L		H	L	H	L					L						

Table 7.5.6 Test Vectors

Inputs																Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
40			L						L		L	H	H	L					L						
41			L						H		H	H	H	L					L						
42			H						L		H	H	H	L					L						
43						L					H	L	H	L					H						
44						L					H	L	L	L					L						
45						L					H	H	H	L					L						
46						L					L	L	H	L					L						
47						H					H	L	H	L					L						
48				L								H	L	L					H						
49				H								H	L	L					L						
50			H	H	H	H		H	H		H	H	L	L					L						
51								L			L	L	H	L						H					
52								L			L	L	L	L						L					
53								L			H	L	H	L						L					
54								H			L	L	H	L						L					
55					L						H	L	L	L						H					
56					L						H	L	H	L						L					
57					L						H	H	L	L						L					
58					L						L	L	L	L						L					
59					H						H	L	L	L						L					
60														H						H					
61		L							L		H	H	H	L						H					
62		L							L		H	H	L	L						L					
63		L							L		H	L	H	L						L					
64		L							H		H	H	H	L						L					
65		H							L		H	H	H	L						L					
66						L					H	L	H	L						H					
67						L					H	L	L	L						L					
68						L					H	H	H	L						L					
69						L					L	L	H	L						L					
70						H					H	L	H	L						L					
71				L								H	L	L						H					
72				H								H	L	L						L					
73											L	H		L						H					
74		H		H	H	H		H	H		H	H	L	L						L					

Table 7.5.6 Test Vectors Continued

SA0 Fault Testing for PT5, SA1 Fault Testing for PT5.
SA0 Fault Testing for PT6, SA1 Fault Testing for PT6.
SA0 Fault Testing for equation 5.
So, we get vectors 51 to 74.

Step 9: Minimize the vectors following these rules:

1) Vectors which have same inputs can be combined to be one vector.

2) If the inputs of a vector are subsets of another vector's inputs, then they can be combined to form one vector.

So, vectors 1, 26, and 51 can be combined to one vector 1 in Table 7.5.7; vectors 12 and 37 can be combined to one vector 21 in Table 7.5.7, etc.

3) Decide the "?" (undetermined) state in the output by using the inputs and logic equations (inserting the known values into logic equations).

Therefore, we get Table 7.5.8.

Step 10: Assign the state numbers. See Table 7.5.9, then we get Table 7.5.10.

Step 11: Build the state diagram and transition path (Figure 7.5.8) from the vector Table 7.5.10.

Step 12: Generate the function table from the state diagram.

1) Be aware of two rules:
a) Generate the initial state first.
b) Generate the function table in sequential order and cover all possible paths.

2) The value of outputs F1ST, ILLOP, 17 and F23 in each test vector can be derived easily by inserting the previous values of outputs C, B, and A and the present values of inputs (none in this example) into their corresponding logic equations.

3) We can quickly identify that the RD signal in this example is the initialize or reset signal, so RD is set high as the first vector in the function table.

4) Finally, insert an "X" into the unused space. We get the function table as shown in Table 7.5.11.

Inputs																Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
1								L			L	L	H	L				H	H	H					
2								L			L	L	L	L				L	L	L					
3								L			L	H	H	L				L	L	H					
4								L			H	L	H	L				L	L	L					
5								H			L	L	H	L				L	L	L					
6					L						H	L	L	L				H	H	H					
7					L						H	L	H	L				L	L	L					
8					L						H	H	L	L				L	L	L					
9					L						L	L	L	L				L	L	L					
10					H						H	L	L	L				L	L	L					
11											X	X	X	H				H	H	H					
12						L	L				H	L	H	L				H	H	H					
13						L	L				H	L	L	L				L	L	L					
14						L	L				H	H	H	L				L	L	L					
15						L	L				L	L	H	L				L	L	L					
16						L	H				H	L	H	L				L	L	L					
17						H	L				H	L	H	L				L	L	L					
18				L							X	H	L	L				H	H	H					
19				H							X	H	L	L				L	L	L					
20		H	H	H	H	H	H	H	H		H	H	L	L				L	L	L					
21			L						L		H	H	H	L				H	H	?					
22			L						L		H	H	L	L				L	L	?					
23			L						L		H	L	H	L				L	L	?					
24			L						L		L	H	H	L				L	L	?					
25			L						H		H	H	H	L				L	L	?					
26			H						L		H	H	H	L				H	L	?					
27		L							L		H	H	H	L				H	?	H					
28		L							L		H	H	L	L				L	?	L					
29		L							L		H	L	H	L				L	?	L					
30		L							H		H	H	H	L				L	?	L					
31		H							L		H	H	H	L				H	?	L					
32											L	H	X	L				?	?	H					

Table 7.5.7 Test Vectors

Inputs																Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
1								L			L	L	H	L				H	H	H					
2								L			L	L	L	L				L	L	L					
3								L			L	H	H	L				L	L	H					
4								L			H	L	H	L				L	L	L					
5								H			L	L	H	L				L	L	L					
6					L						H	L	L	L				H	H	H					
7					L						H	L	H	L				L	L	L					
8					L						H	H	L	L				L	L	L					
9					L						L	L	L	L				L	L	L					
10					H						H	L	L	L				L	L	L					
11														H				H	H	H					
12						L	L				H	L	H	L				H	H	H					
13						L	L				H	L	L	L				L	L	L					
14						L	L				H	H	H	L				L	L	L					
15						L	L				L	L	H	L				L	L	L					
16						L	H				H	L	H	L				L	L	L					
17						H	L				H	L	H	L				L	L	L					
18				L							X	H	L	L				H	H	H					
19				H							X	H	L	L				L	L	L					
20		H	H	H	H	H	H	H	H		H	H	L	L				L	L	L					
21		L	L						L		H	H	H	L				H	H	H					
22		H	L						L		H	H	H	L				H	H	L					
23			L						L		H	H	L	L				L	L	L					
24			L			L			L		H	L	H	L				L	L	H					
25			L			H			L		H	L	H	L				L	L	L					
26			L	L					L		L	H	H	L				L	L	H					
27			L	H					L		L	H	H	L				L	L	L					
28			L						H		H	H	H	L				L	L	L					
29		L	H						L		H	H	H	L				H	L	H					
30		H	H						L		H	H	H	L				H	L	L					
31		L	L						L		H	H	H	L				H	H	H					
32		L	H						L		H	H	H	L				H	L	H					
33		L				L			L		H	H	L	L				L	L	L					
34		L							L		H	L	H	L				L	H	L					
35		L				H			L		H	L	H	L				L	L	L					
36		L							H		H	H	H					L	L	L					
37		H	L						L		H	H	H					H	H	L					
38		H	H						L		H	H	H					H	L	L					
39											L	H	H					L	L	H					
40				H							L	H	X					L	L	H					
41				L							L	H	L					H	H	H					

Table 7.5.8 Test Vectors

C	B	A	State #
H	H	H	1
H	H	L	2
L	L	L	3
H	L	L	4
H	L	H	5
L	H	H	6
L	L	H	7
L	H	L	8

Table 7.5.9 State Assignment

Inputs																Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
1	C							L		H	7	7	7	L				1	1	1					
2	C							L		H	3	3	3	L				3	3	3					
3	C							L		H	6	6	6	L				7	7	7					
4	C							L		H	5	5	5	L				3	3	3					
5	C							H		H	7	7	7	L				3	3	3					
6	C				L					H	4	4	4	L				1	1	1					
7	C				L					H	5	5	5	L				3	3	3					
8	C				L					H	2	2	2	L				3	3	3					
9	C				L					H	3	3	3	L				3	3	3					
10	C				H					H	4	4	4	L				3	3	3					
11	C									H				H				1	1	1					
12	C					L	L			H	5	5	5	L				1	1	1					
13	C					L	L			H	4	4	4	L				3	3	3					
14	C					L	L			H	1	1	1	L				3	3	3					
15	C					L	L			H	7	7	7	L				3	3	3					
16	C					L	H			H	5	5	5	L				3	3	3					
17	C					H	L			H	5	5	5	L				3	3	3					
18	C			L						H	2 or 8	2 or 8	2 or 8	L				1	1	1					
19	C			H						H	2 or 8	2 or 8	2 or 8	L				3	3	3					
20	C	H	H	H	H	H	H	H	H	H	2	2	2	L				3	3	3					
21	C	L	L						L	H	1	1	1	L				1	1	1					
22	C	H	L						L	H	1	1	1	L				2	2	2					
23	C		L						L	H	2	2	2	L				3	3	3					
24	C		L			L			L	H	5	5	5	L				7	7	7					
25	C		L			H			L	H	5	5	5	L				3	3	3					
26	C		L	L					L	H	6	6	6	L				7	7	7					
27	C		L	H					L	H	6	6	6	L				3	3	3					
28	C		L						H	H	1	1	1	L				3	3	3					

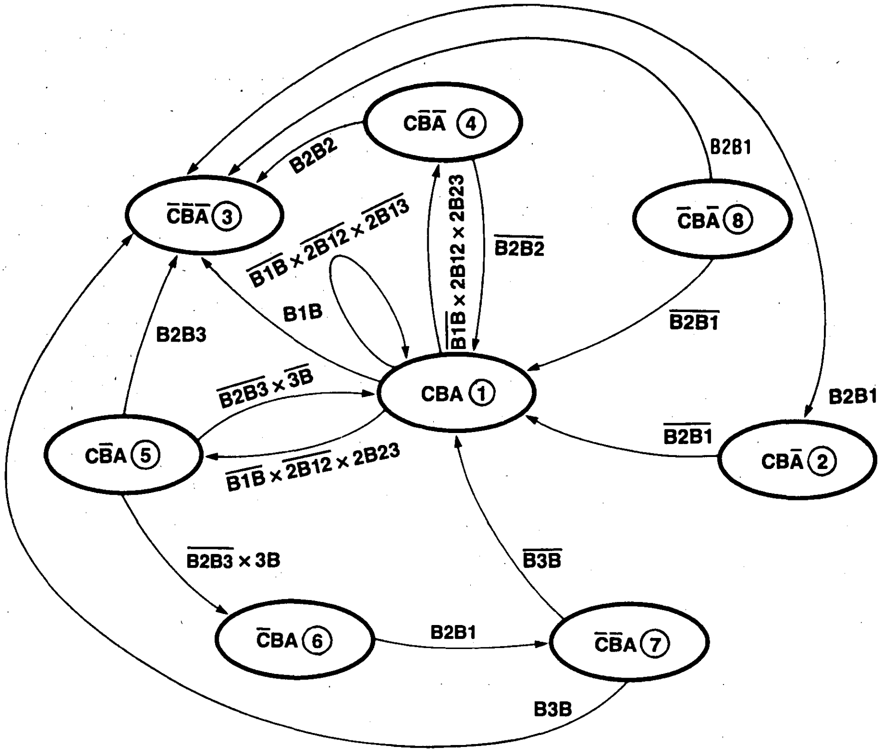
Table 7.5.10 Transition Table

Inputs																Outputs									
	CLK	2B12	2B23	B2B1	B2B2	B2B3	3B	B3B	B1B	EN	C	B	A	RD		RIST	ILLOP	C	B	A	17	F23			
29	C	L	H						L	H	1	1	1	L				5	5	5					
30	C	H	H						L	H	1	1	1	L				4	4	4					
31	C	L	L						L	H	1	1	1	L				1	1	1					
32	C	L	H						L	H	1	1	1	L				5	5	5					
33	C	L							L	H	2	2	2	L				3	3	3					
34	C	L				L			L	H	5	5	5	L				8	8	8					
35	C	L				H			L	H	5	5	5	L				3	3	3					
36	C	L							H	H	1	1	1	L				3	3	3					
37	C	H	L						L	H	1	1	1	L				2	2	2					
38	C	H	H						L	H	1	1	1	L				4	4	4					
39	C									H	6	6	6	L				7	7	7					
40	C			H						H	8 or 6	8 or 6	8 or 6	L				7	7	7					
41	C			L						H	8	8	8	L				1	1	1					
42	C									L				L				Z	Z	Z					

Table 7.5.10 Transition Table Continued

```
FUNCTION TABLE
CLK /2B12 /2B23 /B2B1 /B2B2 /B2B3 /3B /B3B /B1B  /EN F1ST
/ILLOP /C /B /A /17 /RD F23
-----------------------------------------------------
C X X X X X X X X L H H L L L L L L
C L H X X X X X H L L H L L H L H H
C X X H X X X X X L H H L L L H H L
C L H X X X X X H L L H L L H L H H
C X X L X X X X X L H L H H H H H L
C X X X X X X X X L H H L L L H L L
C L L X X X X X H L L H L H H L H H
C X X X H X X X X L H H L L L H H L
C L L X X X X X H L L H L H H L H H
C X X X L X X X X L H L H H H H H L
C X X X X X X X X L H H L L L H L L
C H L X X X X X H L L H L H L L H H
C X X X X H H X X L H H L L L H H L
C H L X X X X X H L L H L H L L H H
C X X X X L X X X L H L H H H H H L
C X X X X X X X X L H H L L L H L L
C H L X X X X X H L L H L H L L H H
C X X X X H L X X L L H H L L H H H
C X X X X X X X X L L H H H L H H H
C X X X X X X L X L H L H H H H H L
C X X X X X X X X L H H L L L H L L
C H L X X X X X H L L H L H L L H H
C X X X X H L X X L L H H L L H H H
C X X X X X X X X L L H H H L H H H
C X X X X X X H X L H H L L L H H L
C X X X X X X X L L H L H H H L H L
C X X X X X X X X L H H L L L H L L
C H H X X X X X H L H H L L L L L L
-----------------------------------------
DESCRIPTION
OP CODE ANALYZER
```

Table 7.5.11 Final Function Table


$C\overline{B}\overline{A}$ 4
B2B2
$\overline{C}\overline{B}\overline{A}$ 3
$\overline{B1B} \times \overline{2B12} \times \overline{2B13}$
$\overline{B1B} \times 2B12 \times 2B23$
$\overline{B2B2}$
B2B1
$\overline{C}B\overline{A}$ 8
B1B
B2B3
$\overline{B2B1}$
$\overline{B2B3} \times \overline{3B}$
CBA 1
B2B1
$\overline{B2B1}$
$C\overline{B}A$ 5
$\overline{B1B} \times \overline{2B12} \times 2B23$
$CB\overline{A}$ 2
$\overline{B2B3} \times 3B$
$\overline{B3B}$
$\overline{C}BA$ 6
B2B1
$\overline{C}\overline{B}A$ 7
B3B

Now we can get any test sequence we like just by following the state transition. The first vector should be the initialize vector and, by intuition, we know state ① is the initialize state.

Figure 7.5.8 State Diagram

The following are printouts of PAL device design specifications, function table, pinout list, fuse map, simulation result, and fault testing result. We get 100% PTC!

```
PALASM VERSION 1.5

PAL16R4
PTAN302
TOM WANG
OP CODE ANALYZER
CLK /2B12 /2B23 /B2B1 /B2B2 /B2B3 /3B /B3B /B1B GND
/EN F1ST /ILLOP /C /B /A /17 /RD F23 VCC
IF (VCC)/F1ST = F23
IF (VCC)ILLOP = /A*/B*/C
C:=A*/B*/C*/B3B + /A*/B*C*/B2B2 + RD + A*B*C*/B1B +
A*/B*C*/B2B3*/3B + /A*B*/B2B1
B:=A*/B*/C*/B3B + /A*/B*C*/B2B2 + RD + A*B*C*/B1B*/2B23 +
A*/B*C*/B2B3 + /A*B*/B2B1
A:=A*/B*/C*/B3B + /A*/B*C*/B2B2 + RD + A*B*C*/B1B*/2B12 +
A*/B*C*/B2B3 + /A*B*/B2B1 + B*/C
17:= A*B*C
IF(VCC)/F23 =/A*/B*/C + A*B*C
```

```
TOM WANG

                ***************   ***************
                      *              * *              *
                   ****                            ****
         CLK       * 1*             P A L          *20*    VCC
                   ****                            ****
                      *            1 6 R 4          *
                   ****                            ****
       /2B12       * 2*                            *19*    F23
                   ****                            ****
                      *                             *
                   ****                            ****
       /2B23       * 3*                            *18*    /RD
                   ****                            ****
                      *                             *
                   ****                            ****
       /B2B1       * 4*                            *17*    /17
                   ****                            ****
                      *                             *
                   ****                            ****
       /B2B2       * 5*                            *16*    /A
                   ****                            ****
                      *                             *
                   ****                            ****
       /B2B3       * 6*                            *15*    /B
                   ****                            ****
                      *                             *
                   ****                            ****
         /3B       * 7*                            *14*    /C
                   ****                            ****
                      *                             *
                   ****                            ****
        /B3B       * 8*                            *13*    /ILLOP
                   ****                            ****
                      *                             *
                   ****                            ****
        /B1B       * 9*                            *12*    F1ST
                   ****                            ****
                      *                             *
                   ****                            ****
         GND       *10*                            *11*    /EN
                   ****                            ****
                      *                             *
                      *********************************
```

```
TOM WANG

  1 CXXXXXXXXX0HHLLLL0L1
  2 C01XXXXX1X0LHLLHL1H1
  3 CXX1XXXXXX0HHLLLH1L1
  4 C01XXXXX1X0LHLLHL1H1
  5 CXX0XXXXXX0HLHHHH1L1
  6 CXXXXXXXXX0HHLLLH0L1
  7 C00XXXXX1X0LHLHHL1H1
  8 CXXX1XXXXX0HHLLLH1L1
  9 C00XXXXX1X0LHLHHL1H1
 10 CXXX0XXXXX0HLHHHH1L1
 11 CXXXXXXXXX0HHLLLH0L1
 12 C10XXXXX1X0LHLHLL1H1
 13 CXXXX11XXX0HHLLLH1L1
 14 C10XXXXX1X0LHLHLL1H1
 15 CXXXX0XXXX0HLHHHH1L1
 16 CXXXXXXXXX0HHLLLH0L1
```

```
    17 C10XXXXX1X0LHLHLL1H1
    18 CXXXX10XXX0LHHLLH1H1
    19 CXXXXXXXXX0LHHHLH1H1
    20 CXXXXXX0XX0HLHHHH1L1
    21 CXXXXXXXXX0HHLLLH0L1
    22 C10XXXXX1X0LHLHLL1H1
    23 CXXXX10XXX0LHHLLH1H1
    24 CXXXXXXXXX0LHHHLH1H1
    25 CXXXXXX1XX0HHLLLH1L1
    26 CXXXXXXX0X0HLHHHL1L1
    27 CXXXXXXXXX0HHLLLH0L1
    28 C11XXXXX1X0HHLLLL0L1

   PASS SIMULATION          672           29


 TOM WANG

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

BEG*FPLT PAL16R4   8
  0 ---- ---- ---- ---- ---- ---- ---- ----
  1 ---- ---- ---- --X- --X- --X- ---- ---- /A*/B*/C
  2 ---- ---- ---- ---X ---X ---X ---- ---- A*B*C
  3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  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 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  9 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 10 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 11 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 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 ---X ---X ---- ---- A*B*C
 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 ---- ---- ---- ---X --X- --X- X--- ---- A*/B*/C*/B3B
 25 ---- ---- ---- X-X- --X- ---X ---- ---- /A*/B*C*/B2B2
 26 ---- ---X ---- ---- ---- ---- ---- ---- RD
 27 X--- ---- ---- ---X ---X ---X ---- X--- A*B*C*/B1B*/2B12
 28 ---- ---- ---- ---X X-X- ---X ---- ---- A*/B*C*/B2B3
 29 ---- ---- X--- --X- ---X ---- ---- ---- /A*B*/B2B1
 30 ---- ---- ---- ---- ---X --X- ---- ---- B*/C
 31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 32 ---- ---- ---- ---X --X- --X- X--- ---- A*/B*/C*/B3B
 33 ---- ---- ---- X-X- --X- ---X ---- ---- /A*/B*C*/B2B2
 34 ---- ---X ---- ---- ---- ---- ---- ---- RD
 35 ---- X--- ---- ---X ---X ---X ---- X--- A*B*C*/B1B*/2B23
 36 ---- ---- ---- ---X X-X- ---X ---- ---- A*/B*C*/B2B3
 37 ---- ---- X--- --X- ---X ---- ---- ---- /A*B*/B2B1
 38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

```
  40 ---- ---- ---- ---X --X- --X- X--- ---- A*/B*/C*/B3B
  41 ---- ---- ---- X-X- --X- ---X ---- ---- /A*/B*C*/B2B2
  42 ---- ---X ---- ---- ---- ---- ---- ---- RD
  43 ---- ---- ---- ---X ---X ---X ---- X--- A*B*C*/B1B
  44 ---- ---- ---- ---X X-X- X--X ---- ---- A*/B*C*/B2B3*/3B
  45 ---- ---- X--- --X- ---X ---- ---- ---- /A*B*/B2B1
  46 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  47 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

  48 ---- ---- ---- ---- ---- ---- ---- ----
  49 ---- ---- ---- --X- --X- --X- ---- ---- /A*/B*/C
  50 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

  56 ---- ---- ---- ---- ---- ---- ---- ----
  57 --X- ---- ---- ---- ---- ---- ---- ---- F23
  58 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

 NUMBER OF FUSES BLOWN =  786

 TOM WANG
```

```
FILE: PTAN302  FUSEPLOT A    <<<  NATIONAL SEMICONDUCTOR TIMESHARING SERVICES SYST

   1 CXXXXXXXXXOHHLLLLOL1
   2 C01XXXXX1XOLHLLHL1H1
   3 CXX1XXXXXXOHHLLLH1L1
   4 C01XXXXX1XOLHLLHL1H1
   5 CXX0XXXXXXOHLHHHH1L1
   6 CXXXXXXXXXOHHLLLHOL1
   7 C00XXXXX1XOLHLHHL1H1
   8 CXXX1XXXXXOHHLLLH1L1
   9 C00XXXXX1XOLHLHHL1H1
  10 CXXX0XXXXXOHLHHHH1L1
  11 CXXXXXXXXXOHHLLLHOL1
  12 C10XXXXX1XOLHLHLL1H1
  13 CXXXX11XXXOHHLLLH1L1
  14 C10XXXXX1XOLHLHLL1H1
  15 CXXXX0XXXXOHLHHHH1L1
  16 CXXXXXXXXXOHHLLLHOL1
  17 C10XXXXX1XOLHLHLL1H1
  18 CXXXX10XXXOLHHLLH1H1
  19 CXXXXXXXXXOLHHHLH1H1
  20 CXXXXXX0XXOHLHHHH1L1
  21 CXXXXXXXXXOHHLLLHOL1
  22 C10XXXXX1XOLHLHLL1H1
  23 CXXXX10XXXOLHHLLH1H1
  24 CXXXXXXXXXOLHHHLH1H1
  25 CXXXXXX1XXOHHLLLH1L1
  26 CXXXXXXX0XOHLHHHL1L1
  27 CXXXXXXXXXOHHLLLHOL1
  28 C11XXXXX1XOHHLLLLOL1

PASS SIMULATION          672          29

NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE = 24

NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE = 24

PRODUCT  TERM   COVERAGE                =100%
```

8

Applications*

8.1 BASIC GATES

This example demonstrates how fusable logic can implement the basic inverter, AND OR, NAND, NOR and exclusive -OR functions. The PAL 12H6 is selected because it has 12 inputs and 6 outputs.

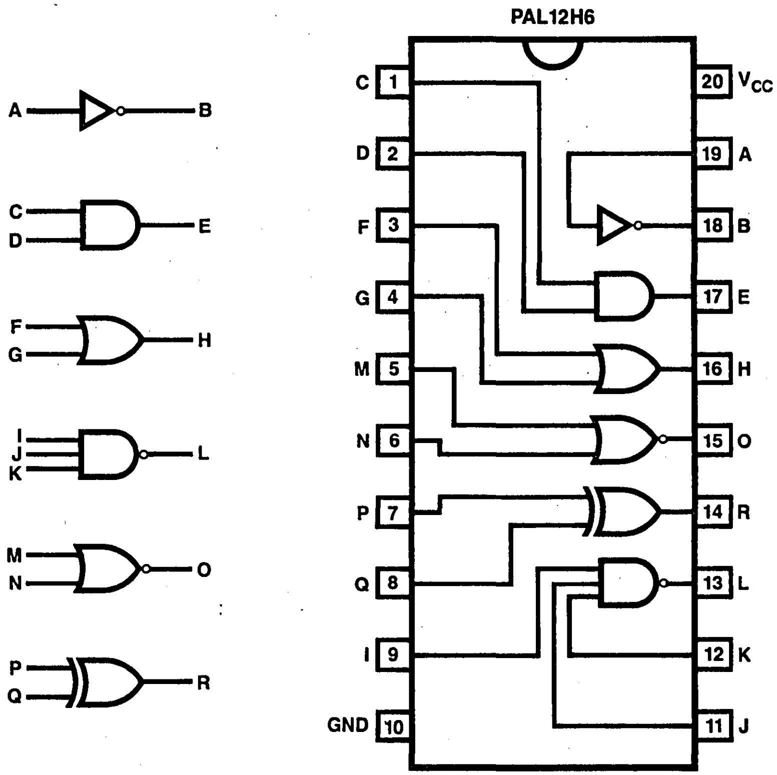


Figure 8.1.1 Basic Gates

* ***Applications contained in this chapter are for illustration purposes only and National makes no representation or warranty that such applications will be suitable for the use specified without further testing or modification.***

```
PALASM VERSION 1.5

PAL12H6
TOM WANG
BASIC GATE
NSC SANTA CLARA
C D F G M N P Q I GND J K L R O H E B A VCC
B = /A
E = C*D
H = F + G
L = /I + /J + /K
O = /M*/N
R = P*/Q + /P*Q
FUNCTION TABLE
A B C D E F G H I J K L M N O P Q R
-------------------------------------------
L H  X X X  X X X  X X X X  X X X  X X X ;TEST INVERTER
H L  X X X  X X X  X X X X  X X X  X X X ;TEST INVERTER
X X  L L L  X X X  X X X X  X X X  X X X ;TEST AND GATE
X X  L H L  X X X  X X X X  X X X  X X X ;TEST AND GATE
X X  H L L  X X X  X X X X  X X X  X X X ;TEST AND GATE
X X  H H H  X X X  X X X X  X X X  X X X ;TEST AND GATE
X X  X X X  L L L  X X X X  X X X  X X X ;TEST OR GATE
X X  X X X  L H H  X X X X  X X X  X X X ;TEST OR GATE
X X  X X X  H L H  X X X X  X X X  X X X ;TEST OR GATE
X X  X X X  H H H  X X X X  X X X  X X X ;TEST OR GATE
X X  X X X  X X X  L L L H  X X X  X X X ;TEST NAND GATE
X X  X X X  X X X  L L H H  X X X  X X X ;TEST NAND GATE
X X  X X X  X X X  L H L H  X X X  X X X ;TEST NAND GATE
X X  X X X  X X X  H L L H  X X X  X X X ;TEST NAND GATE
X X  X X X  X X X  H H H L  X X X  X X X ;TEST NAND GATE
X X  X X X  X X X  X X X X  L L H  X X X ;TEST NOR GATE
X X  X X X  X X X  X X X X  L H L  X X X ;TEST NOR GATE
X X  X X X  X X X  X X X X  H L L  X X X ;TEST NOR GATE
X X  X X X  X X X  X X X X  H H L  X X X ;TEST NOR GATE
X X  X X X  X X X  X X X X  X X X  L L L ;TEST EXCLUSIVE OR GATE
X X  X X X  X X X  X X X X  X X X  L H H ;TEST EXCLUSIVE OR GATE
X X  X X X  X X X  X X X X  X X X  H L H ;TEST EXCLUSIVE OR GATE
X X  X X X  X X X  X X X X  X X X  H H L ;TEST EXCLUSIVE OR GATE
-------------------------------------------
DESCRIPTION

BASIC GATE

                 ***************   ***************
                    *             * *             *
                 ****                          ****
          C      * 1*            P A L         *20*    VCC
                 ****                          ****
                    *           1 2 H 6        *
                 ****                          ****
          D      * 2*                          *19*    A
                 ****                          ****
                    *                          *
                 ****                          ****
          F      * 3*                          *18*    B
                 ****                          ****
                    *                          *
                 ****                          ****
          G      * 4*                          *17*    E
                 ****                          ****
```

```
              *                       *
           ****                       ****
        M  * 5*                       *16*  H
           ****                       ****
              *                       *
           ****                       ****
        N  * 6*                       *15*  O
           ****                       ****
              *                       *
           ****                       ****
        P  * 7*                       *14*  R
           ****                       ****
              *                       *
           ****                       ****
        Q  * 8*                       *13*  L
           ****                       ****
              *                       *
           ****                       ****
        I  * 9*                       *12*  K
           ****                       ****
              *                       *
           ****                       ****
      GND  *10*                       *11*  J
           ****                       ****
              *                       *
              *************************

       BASIC GATE

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

      BEG*FPLT PAL12H6   8

        8 ---- ---X --00 --00 --00 --00 ---- ---- /A
        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

       16 X-X- ---- --00 --00 --00 --00 ---- ---- C*D
       17 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

       24 ---- X--- --00 --00 --00 --00 ---- ---- F
       25 ---- ---- X-00 --00 --00 --00 ---- ---- G

       32 ---- ---- --00 -X00 -X00 --00 ---- ---- /M*/N
       33 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

       40 ---- ---- --00 --00 --00 X-00 -X-- ---- P*/Q
       41 ---- ---- --00 --00 --00 -X00 X--- ---- /P*Q

       48 ---- ---- --00 --00 --00 --00 ---- -X-- /I
       49 ---- ---- --00 --00 --00 --00 ---- ---X /J
       50 ---- ---- --00 --00 --00 --00 ---X ---- /K
       51 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)
          0 : PHANTOM FUSE   (L,N,0)   0 : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOWN =  306
```

```
BASIC GATE                                    BASIC GATE

  1 XXXXXXXXXXXXXXXXXH01                        1 XXXXXXXXXXXXXXXXXH01
  2 XXXXXXXXXXXXXXXXXL11                        2 XXXXXXXXXXXXXXXXXL11
  3 00XXXXXXXXXXXXXXLXX1                        3 00XXXXXXXXXXXXXXLXX1
  4 01XXXXXXXXXXXXXXLXX1                        4 01XXXXXXXXXXXXXXLXX1
  5 10XXXXXXXXXXXXXXLXX1                        5 10XXXXXXXXXXXXXXLXX1
  6 11XXXXXXXXXXXXXXHXX1                        6 11XXXXXXXXXXXXXXHXX1
  7 XX00XXXXXXXXXXXLXXX1                        7 XX00XXXXXXXXXXXLXXX1
  8 XX01XXXXXXXXXXXHXXX1                        8 XX01XXXXXXXXXXXHXXX1
  9 XX10XXXXXXXXXXXHXXX1                        9 XX10XXXXXXXXXXXHXXX1
 10 XX11XXXXXXXXXXXHXXX1                       10 XX11XXXXXXXXXXXHXXX1
 11 XXXXXXXX0X00HXXXXXX1                       11 XXXXXXXX0X00HXXXXXX1
 12 XXXXXXXX0X01HXXXXXX1                       12 XXXXXXXX0X01HXXXXXX1
 13 XXXXXXXX0X10HXXXXXX1                       13 XXXXXXXX0X10HXXXXXX1
 14 XXXXXXXX1X00HXXXXXX1                       14 XXXXXXXX1X00HXXXXXX1
 15 XXXXXXXX1X11LXXXXXX1                       15 XXXXXXXX1X11LXXXXXX1
 16 XXXX00XXXXXXXXHXXXX1                       16 XXXX00XXXXXXXXHXXXX1
 17 XXXX01XXXXXXXXLXXXX1                       17 XXXX01XXXXXXXXLXXXX1
 18 XXXX10XXXXXXXXLXXXX1                       18 XXXX10XXXXXXXXLXXXX1
 19 XXXX11XXXXXXXXLXXXX1                       19 XXXX11XXXXXXXXLXXXX1
 20 XXXXXX00XXXXXLXXXXX1                       20 XXXXXX00XXXXXLXXXXX1
 21 XXXXXX01XXXXXHXXXXX1                       21 XXXXXX01XXXXXHXXXXX1
 22 XXXXXX10XXXXXHXXXXX1                       22 XXXXXX10XXXXXHXXXXX1
 23 XXXXXX11XXXXXLXXXXX1                       23 XXXXXX11XXXXXLXXXXX1

PASS SIMULATION          230           24


PASS SIMULATION          230           24
 PRODUCT:    1 OF EQUATION.  4                 UNTESTED(SA0)FAULT
 PRODUCT:    2 OF EQUATION.  4                 UNTESTED(SA0)FAULT
 PRODUCT:    3 OF EQUATION.  4                 UNTESTED(SA0)FAULT

NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE = 10

NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE =  7

PRODUCT  TERM   COVERAGE                 = 85%
```

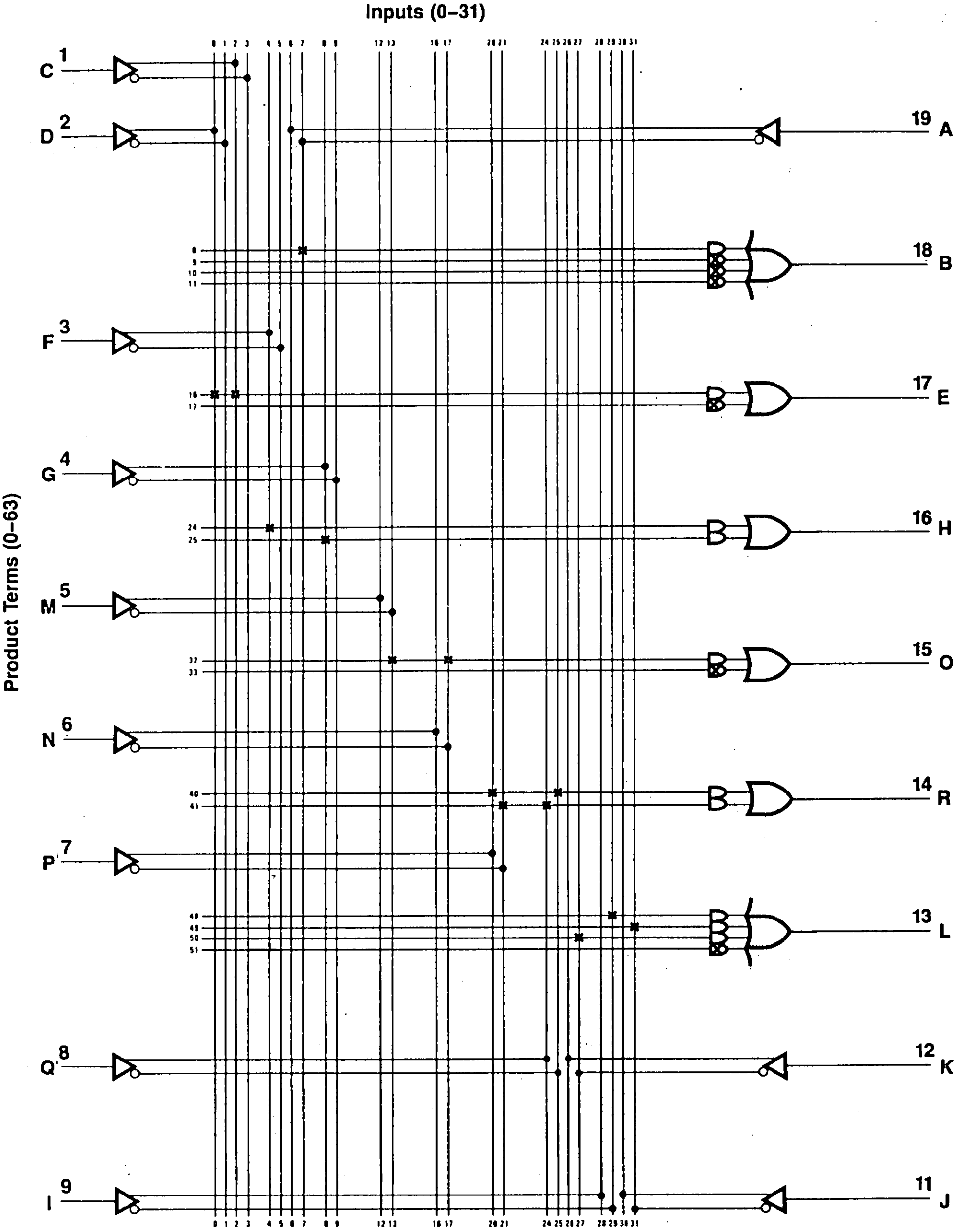


Figure 8.1.2 Logic Diagram PAL12H6

8.2 BASIC CLOCKED FLIP FLOPS

This example demonstrates how fusable logic, PAL16R8, can implement the basic flip-flops; J-K flip-flop; T flip-flop, D flip-flop, and S-R flip-flop. A PAL16L8 can be substituted for this application. Then, the clock input (CLK) would be gated with the data inputs to implement the basic flip-flop.

```
PALASM VERSION 1.5

PAL16R8
BFLIP
BASIC
NSC
CLK J K T PR CLR D S R GND
/OC /SRC /SRT /DC /DT /TC /TT /JKC /JKT VCC
JKT:=J*/JKT*/CLR
    +/K*JKT*/CLR
    +PR
JKC:=/J*K*/PR
    +/J*/JKT*/PR
    +K*JKT*/PR
    +CLR
TT:=T*/TT*/CLR
   +/T*TT*/CLR
   +PR
TC:=/T*/TT*/PR
   +T*TT*/PR
   +CLR
DT:=D*/CLR
   +PR
DC:=/D*/PR
   +CLR
SRT:=S*/CLR
   +/R*SRT*/CLR
   +PR
SRC:=/S*R*/PR
   +/S*/SRT*/PR
   +CLR
FUNCTION TABLE
CLK /OC PR CLR J K JKT JKC T TT TC D DT DC S R SRT SRC
-------------------------------------------------------
X H X X  X X Z  Z  X Z Z  X Z Z  X X Z  Z;HI-Z

C L L H  X X L  H  X X X  X X X  X X X  X;CLEAR
C L L L  L L L  H  X X X  X X X  X X X  X;
C L L L  L H L  H  X X X  X X X  X X X  X;
C L L L  H H H  L  X X X  X X X  X X X  X;TOGGLE
C L L L  H L H  L  X X X  X X X  X X X  X;
C L L L  L L H  L  X X X  X X X  X X X  X;
C L L L  L H L  H  X X X  X X X  X X X  X;
C L H L  X X H  L  X X X  X X X  X X X  X;PRESET
C L L L  H H L  H  X X X  X X X  X X X  X;TOGGLE
C L L L  H L H  L  X X X  X X X  X X X  X;

C L L H  X X X  X  X L H  X X X  X X X  X;CLEAR
C L L L  X X X  X  L L H  X X X  X X X  X;
C L L L  X X X  X  H H L  X X X  X X X  X;TOGGLE
C L L L  X X X  X  H L H  X X X  X X X  X;TOGGLE
C L H L  X X X  X  X H L  X X X  X X X  X;PRESET
```

```
C L L H   X X X   X   X X X   X L H   X X X   X;CLEAR
C L L L   X X X   X   X X X   L L H   X X X   X;
C L L L   X X X   X   X X X   H H L   X X X   X;
C L L L   X X X   X   X X X   L L H   X X X   X;
C L H L   X X X   X   X X X   X H L   X X X   X;PRESET

C L L H   X X X   X   X X X   X X X   X X L   H;CLEAR
C L L L   X X X   X   X X X   X X X   L L L   H;
C L L L   X X X   X   X X X   X X X   H L H   L;SET
C L L L   X X X   X   X X X   X X X   L H L   H;RESET
C L L L   X X X   X   X X X   X X X   L H L   H;HOLD
C L H L   X X X   X   X X X   X X X   X X H   L;PRESET
C L L L   X X X   X   X X X   X X X   L L H   L;
C L L L   X X X   X   X X X   X X X   H L H   L;
------------------------------------------------------
DESCRIPTION

BASIC
           ***************  ***************
           *             * *              *
        ****                              ****
  CLK   * 1*            P A L             *20*   VCC
        ****                              ****
           *           1 6 R 8            *
        ****                              ****
    J   * 2*                              *19*   /JKT
        ****                              ****
           *                              *
        ****                              ****
    K   * 3*                              *18*   /JKC
        ****                              ****
           *                              *
        ****                              ****
    T   * 4*                              *17*   /TT
        ****                              ****
           *                              *
        ****                              ****
   PR   * 5*                              *16*   /TC
        ****                              ****
           *                              *
        ****                              ****
  CLR   * 6*                              *15*   /DT
        ****                              ****
           *                              *
        ****                              ****
    D   * 7*                              *14*   /DC
        ****                              ****
           *                              *
        ****                              ****
    S   * 8*                              *13*   /SRT
        ****                              ****
           *                              *
        ****                              ****
    R   * 9*                              *12*   /SRC
        ****                              ****
           *                              *
        ****                              ****
 GND    *10*                              *11*   /OC
        ****                              ****
           *                              *
           ********************************
```

```
BASIC

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

BEG*FPLT PAL16R8   8
  0 X-X- ---- ---- ---- -X-- ---- ---- ---- J*/JKT*/CLR
  1 ---X -X-- ---- ---- -X-- ---- ---- ---- /K*JKT*/CLR
  2 ---- ---- ---- X--- ---- ---- ---- ---- PR
  3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  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-- X--- ---- -X-- ---- ---- ---- ---- /J*K*/PR
  9 -XX- ---- ---- -X-- ---- ---- ---- ---- /J*/JKT*/PR
 10 ---X X--- ---- -X-- ---- ---- ---- ---- K*JKT*/PR
 11 ---- ---- ---- ---- X--- ---- ---- ---- CLR
 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-X- ---- -X-- ---- ---- ---- T*/TT*/CLR
 17 ---- ---- -X-X ---- -X-- ---- ---- ---- /T*TT*/CLR
 18 ---- ---- ---- X--- ---- ---- ---- ---- PR
 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-- ---- ---- ---- ---- /T*/TT*/PR
 25 ---- ---- X--X -X-- ---- ---- ---- ---- T*TT*/PR
 26 ---- ---- ---- ---- X--- ---- ---- ---- CLR
 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--- ---- ---- D*/CLR
 33 ---- ---- ---- X--- ---- ---- ---- ---- PR
 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-- ---- ---- /D*/PR
 41 ---- ---- ---- ---- X--- ---- ---- ---- CLR
 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
 48 ---- ---- ---- ---- -X-- ---- X--- ---- S*/CLR
 49 ---- ---- ---- ---- -X-- ---- ---X -X-- /R*SRT*/CLR
 50 ---- ---- ---- X--- ---- ---- ---- ---- PR
 51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 56 ---- ---- ---- -X-- ---- ---- -X-- X--- /S*R*/PR
 57 ---- ---- ---- -X-- ---- ---- -XX- ---- /S*/SRT*/PR
 58 ---- ---- ---- ---- X--- ---- ---- ---- CLR
 59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

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

 NUMBER OF FUSES BLOWN =  686

                  BASIC

                    1 XXXXXXXXXX1ZZZZZZZZ1
                    2 CXXX01XXXX0XXXXXXLH1
                    3 CXXX01XXXX0XXXXXXLH1
                    4 C00X00XXXX0XXXXXXLH1
                    5 C01X00XXXX0XXXXXXLH1
                    6 C11X00XXXX0XXXXXXHL1
                    7 C10X00XXXX0XXXXXXHL1
                    8 C00X00XXXX0XXXXXXHL1
                    9 C01X00XXXX0XXXXXXLH1
                   10 CXXX10XXXX0XXXXXXHL1
                   11 C11X00XXXX0XXXXXXLH1
                   12 C10X00XXXX0XXXXXXHL1
                   13 CXXX01XXXX0XXXXLHXX1
                   14 CXXX01XXXX0XXXXLHXX1
                   15 CXX000XXXX0XXXXLHXX1
                   16 CXX100XXXX0XXXXHLXX1
                   17 CXX100XXXX0XXXXLHXX1
                   18 CXXX10XXXX0XXXXHLXX1
                   19 CXXX01XXXX0XXLHXXXX1
                   20 CXXX01XXXX0XXLHXXXX1
                   21 CXXX000XXX0XXLHXXXX1
                   22 CXXX001XXX0XXHLXXXX1
                   23 CXXX000XXX0XXLHXXXX1
                   24 CXXX10XXXX0XXHLXXXX1
                   25 CXXX01XXXX0LHXXXXXX1
                   26 CXXX01XXXX0LHXXXXXX1
                   27 CXXX00X00X0LHXXXXXX1
                   28 CXXX00X10X0HLXXXXXX1
                   29 CXXX00X01X0LHXXXXXX1
                   30 CXXX00X01X0LHXXXXXX1
                   31 CXXX10XXXX0HLXXXXXX1
                   32 CXXX00X00X0HLXXXXXX1
                   33 CXXX00X10X0HLXXXXXX1

                  PASS SIMULATION        759          34
```

```
                    BASIC

                      1 XXXXXXXXXX1ZZZZZZZZ1
                      2 CXXX01XXXX0XXXXXXLH1
                      3 CXXX01XXXX0XXXXXXLH1
                      4 C00X00XXXX0XXXXXXLH1
                      5 C01X00XXXX0XXXXXXLH1
                      6 C11X00XXXX0XXXXXXHL1
                      7 C10X00XXXX0XXXXXXHL1
                      8 C00X00XXXX0XXXXXXHL1
                      9 C01X00XXXX0XXXXXXLH1
                     10 CXXX10XXXX0XXXXXXHL1
                     11 C11X00XXXX0XXXXXXLH1
                     12 C10X00XXXX0XXXXXXHL1
                     13 CXXX01XXXX0XXXXLHXX1
                     14 CXXX01XXXX0XXXXLHXX1
                     15 CXX000XXXX0XXXXLHXX1
                     16 CXX100XXXX0XXXXHLXX1
                     17 CXX100XXXX0XXXXLHXX1
                     18 CXXX10XXXX0XXXXHLXX1
                     19 CXXX01XXXX0XXLHXXXX1
                     20 CXXX01XXXX0XXLHXXXX1
                     21 CXXX000XXX0XXLHXXXX1
                     22 CXXX001XXX0XXHLXXXX1
                     23 CXXX000XXX0XXLHXXXX1
                     24 CXXX10XXXX0XXHLXXXX1
                     25 CXXX01XXXX0LHXXXXXX1
                     26 CXXX01XXXX0LHXXXXXX1
                     27 CXXX00X00X0LHXXXXXX1
                     28 CXXX00X10X0HLXXXXXX1
                     29 CXXX00X01X0LHXXXXXX1
                     30 CXXX00X01X0LHXXXXXX1
                     31 CXXX10XXXX0HLXXXXXX1
                     32 CXXX00X00X0HLXXXXXX1
                     33 CXXX00X10X0HLXXXXXX1

PASS SIMULATION          759            34
 PRODUCT:   1 OF EQUATION.  2                 UNTESTED(SA0)FAULT
 PRODUCT:   4 OF EQUATION.  2                 UNTESTED(SA0)FAULT
 PRODUCT:   2 OF EQUATION.  3                 UNTESTED(SA0)FAULT
 PRODUCT:   3 OF EQUATION.  4                 UNTESTED(SA0)FAULT
 PRODUCT:   2 OF EQUATION.  6                 UNTESTED(SA0)FAULT
 PRODUCT:   3 OF EQUATION.  8                 UNTESTED(SA0)FAULT

NUMBER OF STUCK AT ONE (SA1)  FAULTS ARE = 23

NUMBER OF STUCK AT ZERO (SA0) FAULTS ARE = 17

PRODUCT  TERM   COVERAGE                 = 86%
```

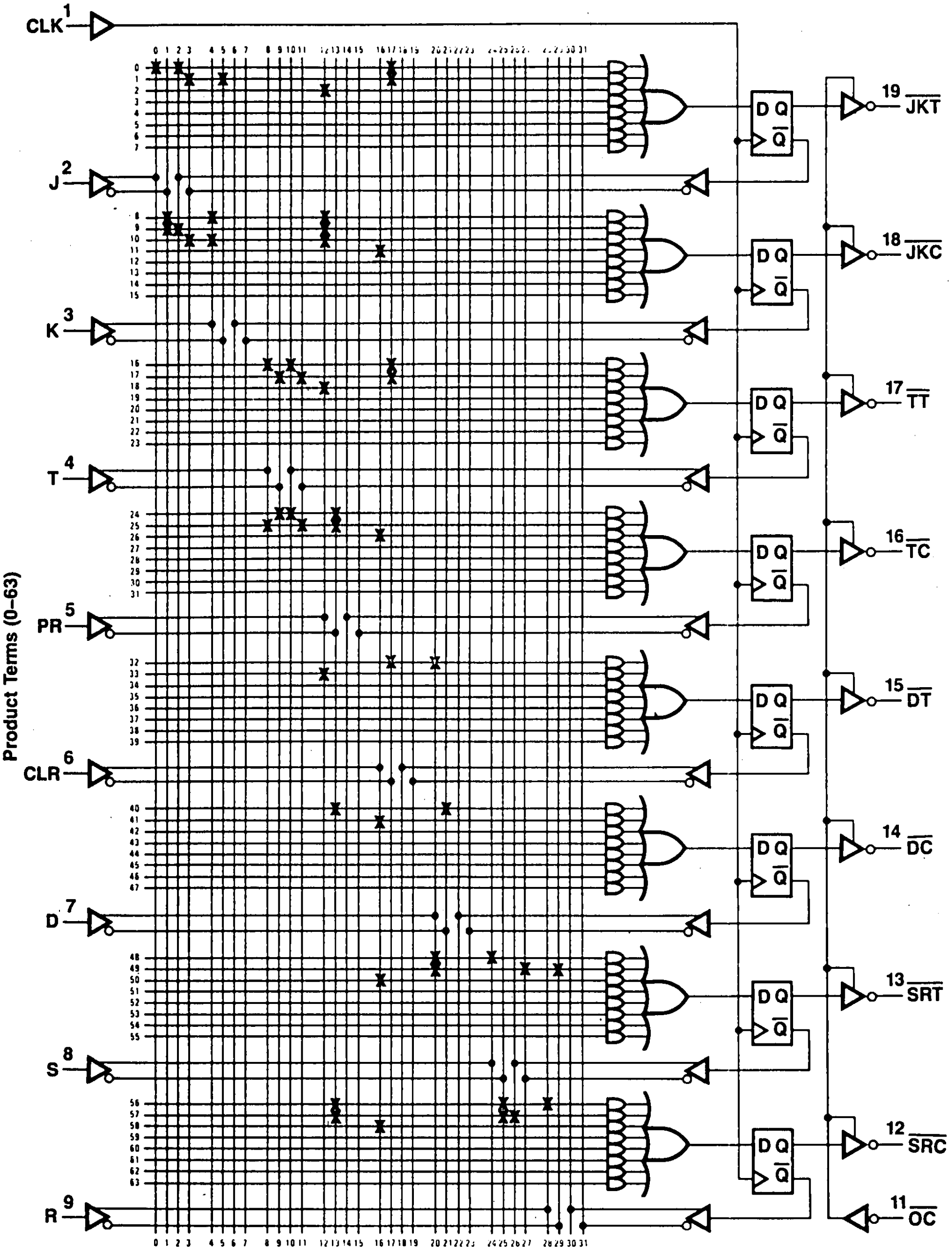


Figure 8.2.1 Logic Diagram PAL16R8

8.3 MEMORY-MAPPED I/O (ADDRESS DECODER)

Memory-mapped I/O is an interface technique that treats I/O devices' physical addresses the same as memory address space. That is, no Memory-I/O decoding is required. Furthermore, most computers have more instructions to manipulate the contents of memory than they have I/O instructions. Therefore, the use of memory mapping can make I/O control much more flexible. PAL devices can be used to make memory-mapped I/O implementation easy, even if changes in memory addresses are required.

Functional Description

Figure 8.3.1 shows a circuit that is typical of those found in memory-mapped I/O applications. The inputs to the decode logic are the system memory address lines, A_0-A_F. The logic shown compares the address on the memory bus with the programmed comparison address. When an address on the bus matches, the corresponding I/O port enable signal is set. In conjunction with other system control signals, this enable can be used to transfer data to and from the system data bus.

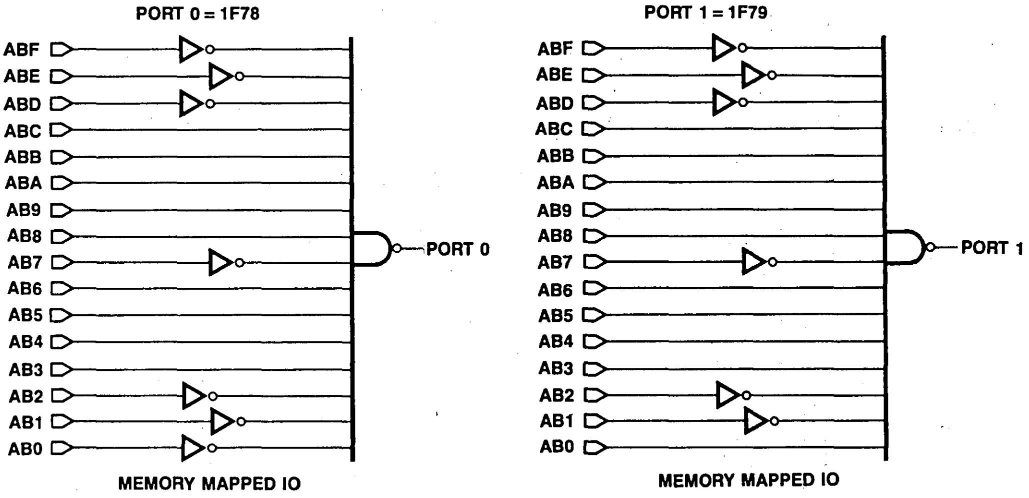


Figure 8.3.1 Memory Mapped I/O Logic Diagram

PAL Device Design

One PAL16L2 can be used to monitor a 16-bit address bus, fully decode addresses, and furnish enables to two ports, each of which can be anywhere within 64K of address space. Partial decoding for a larger number of ports can be done using other members of the PAL device family.

Typical logic equations for the memory-mapped I/O logic are as follows:

Port 0 = /AB0•/AB1•/AB2•AB3•AB4•AB5•AB6•/AB7•
AB8•AB9•ABA•ABB•ABC•/ABD•/ABE•/ABF

Port 1 = AB0•/AB1•/AB2•AB3•AB4•AB5•AB6•/AB7•
AB8•AB9•ABA•ABB•ABC•/ABD•/ABE•/ABF

The above example shows address decoding for memory locations 1F78$_H$ and 1F79$_H$. The equation terms could be changed to accommodate any 16-bit address.

```
PALASM VERSION 1.5

PAL16L2
PAT
MEMORY
MAP
AB0 AB1 AB2 AB3 AB4 AB5 AB6 AB7 AB8 GND
AB9 ABA ABB ABC /PORT1 /PORT0 ABD ABE ABF VCC
PORT0=/AB0*/AB1*/AB2*AB3 *AB4*AB5*AB6*/AB7*AB8*AB9*
     ABA*ABA*ABC*/ABD*/ABE*/ABF
PORT1=AB0*/AB1*/AB2*AB3*AB4*AB5*AB6*/AB7*AB8*AB9*
     ABA*ABB*ABC*/ABD*/ABE*/ABF
DESCRIPTION

MEMORY


          ***************   ***************
          *              * *              *
       ****                               ****
AB0    * 1*             P A L             *20*   VCC
       ****                               ****
          *            1 6 L 2            *
       ****                               ****
AB1    * 2*                               *19*   ABF
       ****                               ****
          *                               *
       ****                               ****
AB2    * 3*                               *18*   ABE
       ****                               ****
          *                               *
       ****                               ****
AB3    * 4*                               *17*   ABD
       ****                               ****
          *                               *
       ****                               ****
AB4    * 5*                               *16*   /PORT0
       ****                               ****
```

```
                      *                              *
                   ****                              ****
             AB5   * 6*                              *15*   /PORT1
                   ****                              ****
                      *                              *
                   ****                              ****
             AB6   * 7*                              *14*   ABC
                   ****                              ****
                      *                              *
                   ****                              ****
             AB7   * 8*                              *13*   ABB
                   ****                              ****
                      *                              *
                   ****                              ****
             AB8   * 9*                              *12*   ABA
                   ****                              ****
                      *                              *
                   ****                              ****
             GND   *10*                              *11*   AB9
                   ****                              ****
                      *                              *
                      *******************************


 MEMORY

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

BEG*FPLT PAL16L2   8


 24 -X-X -X-X X--X X--X X-X- X--- -XX- X-X- /AB0*/AB1*/AB2*AB3*AB4*AB5*AB6*
 25 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 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 -XX- -X-X X--X X--X X-X- X-X- -XX- X-X- AB0*/AB1*/AB2*AB3*AB4*AB5*AB6*/
 33 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 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


END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,O)   - : FUSE BLOWN   (H,P,1)
          0 : PHANTOM FUSE   (L,N,O)   0 : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOWN =   32
```

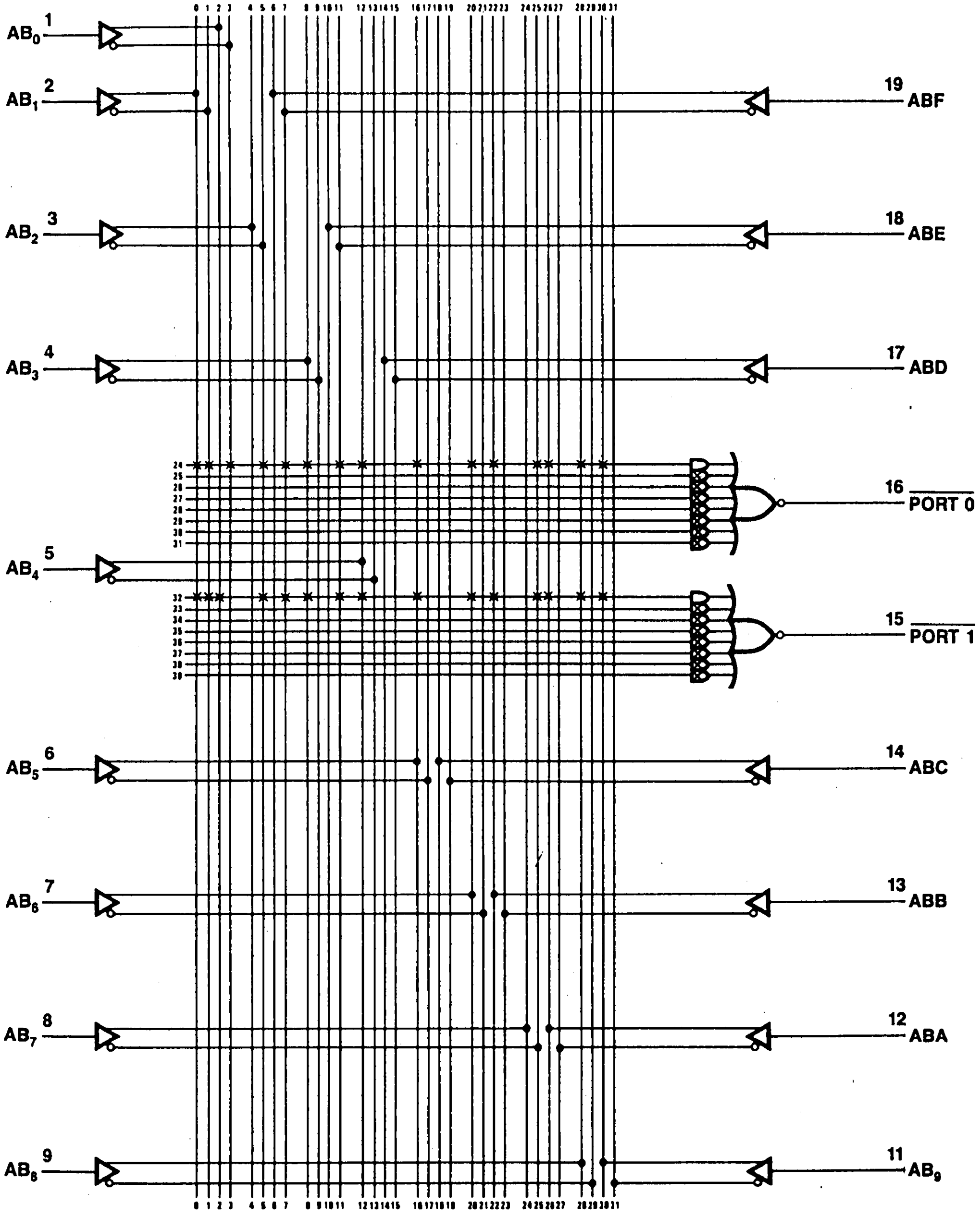


Figure 8.3.2 Logic Diagam PAL16L2

8.4 HEXADECIMAL DECODER/LAMP DRIVER

The increasing use of microcomputers has led to an increased need to display numbers in hexadecimal format (0-9, A-F). Standard drivers for this function are not available, so most applications are forced to use several packages to decode each digit, of the display. Since 6 to 12 digits are often being displayed, this approach can become very expensive. This example demonstrates how the hexadecimal display format can be both decoded and the LED indicators driven using a single PAL for each digit of the display.

Functional Description

A hex decoder/lamp driver accepts a four-bit hex digit, converts it to its corresponding seven-segment display code, and activates the appropriate segments on the display. These drivers can be used in both direct-drive and multiplexed display applications. A single PAL can provide both the basic decode/drive functions, and additional useful features as well.

General Description

Figure 8.4.1 shows three digits of a display system that uses three PALs to implement the complete decoding and display-driving functions. The inputs to each section are a hex code on pins D_0-D_3, a ripple blanking signal, an intensity control signal, and a lamp test signal.

The hex codes are decoded to form the seven-segment patterns shown in Figure 8.4.1. The input codes, digit, represented, and segments driven are as follows:

D_3	D_2	D_1	D_0	Digit	Segments
0	0	0	0	0	ABCDEF
0	0	0	1	1	BC
0	0	1	0	2	ABDEG
0	0	1	1	3	ABCDG
0	1	0	0	4	BCFG
0	1	0	1	5	ACDFG
0	1	1	0	6	ACDEFG
0	1	1	1	7	ABC
1	0	0	0	8	ABCDEFG
1	0	0	1	9	ABCDFG
1	0	1	0	A	ABCEFG
1	0	1	1	B	CDEFG
1	1	0	0	C	ADEF
1	1	0	1	D	BCDEG
1	1	1	0	E	ADEFG
1	1	1	1	F	AEFG

Table 8.4.1 Function Description

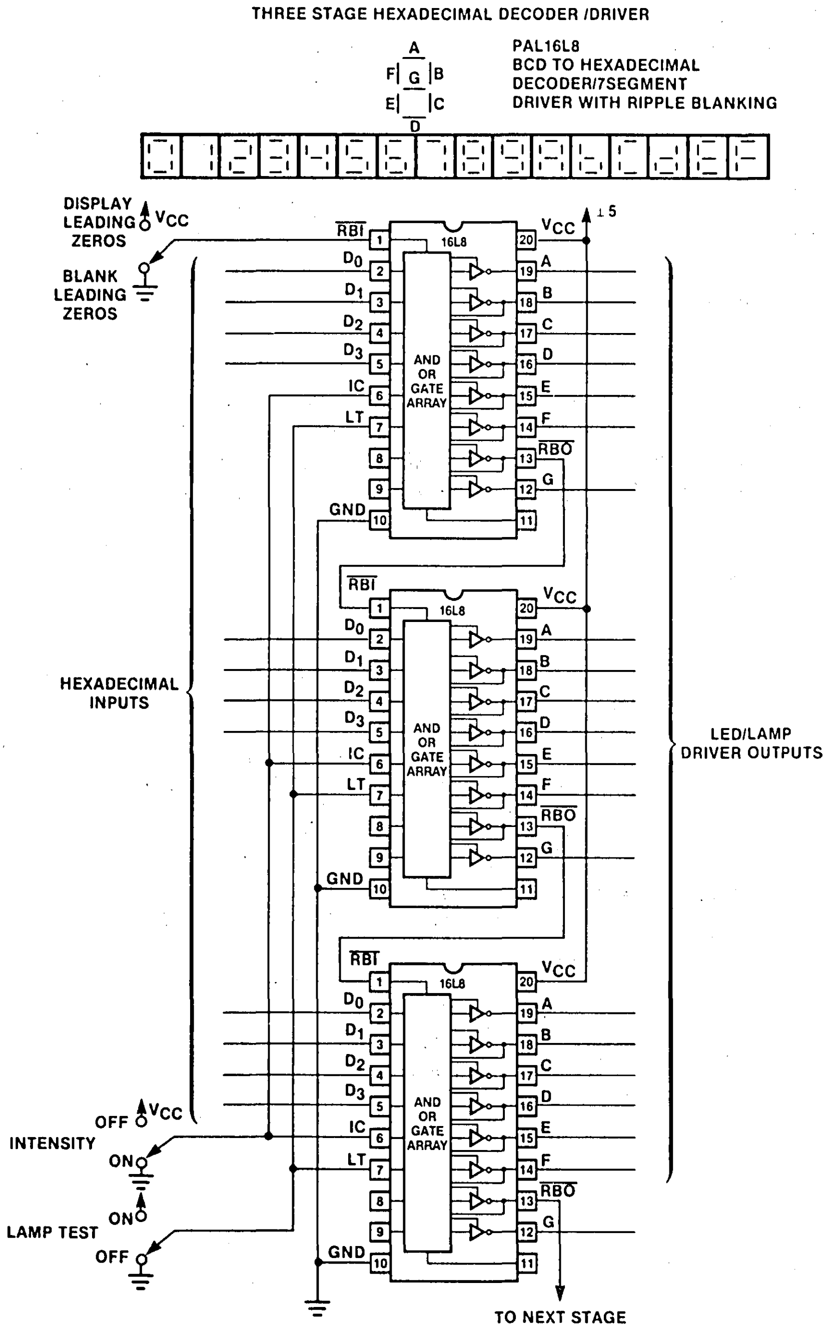


Figure 8.4.1 Hex Display Decoder-Driver, Combinational Logic Diagram

Ripple-blanking input RBI is used to suppress leading zeroes in the display. The signal is propagated from the most significant digit to the least significant digit. If the digit input is zero and RBI is low (indicating that the previous digit is also zero), all segments are left blank and this digit position's ripple-blanking output RBO is set low.

Intensity control signal IC controls the duty cycle of the display driver. When IC is high, all segment drivers are turned off. Pulsing this pin with a duty-cycled signal allows the adjustment of the display's apparent brightness.

Lamp test signal LT lets you check to see if all LED segments are energized.

PAL Device Implementation

The PAL16L8 has both the required I/O pins and the drive current capability to perform as the complete display decoder-driver circuit with seven inputs and eight outputs. The logic equations for this circuit are shown in the listing. One PAL device drives each digit; they may be cascaded without limit. With minor changes, the same logical structure could be used with multiplexer logic to allow a single PAL device to decode and drive multiple digits.

```
PALASM VERSION 1.5

PAL16L8
PAT07
HEX
BLANK
/RBI DO D1 D2 D3 IC LT NC NC GND
NC G /RBO F E D C B A VCC
IF(/IC)/A=/RBO*/DO*/D2+/RBO*/DO*D3+/RBO*D1*D2+
         /RBO*D1*D2*/D3+/RBO*DO*D2*/D3+/RBO*/D1*/D2*D3+LT
IF(IC)/B=/RBO*/D2*/D3+/RBO*/DO*/D2+/RBO*/DO*/D1*/D3+
         /RBO*DO*D1*/D3+/RBO*DO*/D1*/D3+LT
IF(IC)/C=/RBO*DO*/D1+/RBO*DO*/D2+/RBO*/D1*/D2+
         /RBO*D2*/D3+/RBO*/D2*D3+LT
IF(IC)/D=/RBO*/D1*D3+/RBO*/DO*/D2*/D3+
         /RBO*DO*D1*/D2+/RBO*/DO*D1*D2+/RBO*DO*/D1*D2+LT
IF(IC)/E=/RBO*/DO*/D2+/RBO*D2*D3+/RBO*/DO*D1+
         /RBO*D1*D3+LT
IF(IC)/F=/RBO*/DO*/D1+/RBO*/D2*D3+/RBO*D1*D3+
         /RBO*/DO*D2+/RBO*/D1*D2*/D3+LT
IF(VCC)RBO=/DO*/D1*/D2*/D3*/RBI
IF(/IC)/G=/RBO*D1*/D2+/RBO*DO*D3+/RBO*/D2*D3+
         /RBO*/DO*D1+/RBO*/D1*D2*/D3+LT
DESCRIPTION

HEX

             ***************  ***************
             *             * *             *
          ****                              ****
   /RBI   * 1*            P A L             *20*   VCC
          ****                              ****
             *           1 6 L 8            *
          ****                              ****
     DO   * 2*                              *19*   A
          ****                              ****
```

```
              *                             *
           ****                             ****
      D1   * 3*                             *18*   B
           ****                             ****
              *                             *
           ****                             ****
      D2   * 4*                             *17*   C
           ****                             ****
              *                             *
           ****                             ****
      D3   * 5*                             *16*   D
           ****                             ****
              *                             *
           ****                             ****
      IC   * 6*                             *15*   E
           ****                             ****
              *                             *
           ****                             ****
      LT   * 7*                             *14*   F
           ****                             ****
              *                             *
           ****                             ****
      NC   * 8*                             *13*   /RBO
           ****                             ****
              *                             *
           ****                             ****
      NC   * 9*                             *12*   G
           ****                             ****
              *                             *
           ****                             ****
     GND   *10*                             *11*   NC
           ****                             ****
              *                             *
              ********************************
```

```
 HEX

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

BEG*FPLT PAL16L8   8
  0 ---- ---- ---- ---- -X-- ---- ---- ---- /IC
  1 -X-- ---- -X-- ---- ---- ---- --X- ---- /RBO*/DO*/D2
  2 -X-- ---- ---- X--- ---- ---- --X- ---- /RBO*/DO*D3
  3 ---- X--- X--- ---- ---- ---- --X- ---- /RBO*D1*D2
  4 ---- X--- X--- -X-- ---- ---- --X- ---- /RBO*D1*D2*/D3
  5 X--- ---- X--- -X-- ---- ---- --X- ---- /RBO*DO*D2*/D3
  6 ---- -X-- -X-- X--- ---- ---- --X- ---- /RBO*/D1*/D2*D3
  7 ---- ---- ---- ---- ---- X--- ---- ---- LT

  8 ---- ---- ---- ---- X--- ---- ---- ---- IC
  9 ---- ---- -X-- -X-- ---- ---- --X- ---- /RBO*/D2*/D3
 10 -X-- ---- -X-- ---- ---- ---- --X- ---- /RBO*/DO*/D2
 11 -X-- -X-- ---- -X-- ---- ---- --X- ---- /RBO*/DO*/D1*/D3
 12 X--- X--- ---- -X-- ---- ---- --X- ---- /RBO*DO*D1*/D3
 13 X--- -X-- ---- -X-- ---- ---- --X- ---- /RBO*DO*/D1*/D3
 14 ---- ---- ---- ---- ---- X--- ---- ---- LT
 15 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
```

```
16 ---- ---- ---- ---- X--- ---- ---- ---- IC
17 X--- -X-- ---- ---- ---- ---- --X- ---- /RBO*DO*/D1
18 X--- ---- -X-- ---- ---- ---- --X- ---- /RBO*DO*/D2
19 ---- -X-- -X-- ---- ---- ---- --X- ---- /RBO*/D1*/D2
20 ---- ---- X--- -X-- ---- ---- --X- ---- /RBO*D2*/D3
21 ---- ---- -X-- X--- ---- ---- --X- ---- /RBO*/D2*D3
22 ---- ---- ---- ---- ---- X--- ---- ---- LT
23 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

24 ---- ---- ---- ---- X--- ---- ---- ---- IC
25 ---- -X-- ---- X--- ---- ---- --X- ---- /RBO*/D1*D3
26 -X-- ---- -X-- -X-- ---- ---- --X- ---- /RBO*/DO*/D2*/D3
27 X--- X--- -X-- ---- ---- ---- --X- ---- /RBO*DO*D1*/D2
28 -X-- X--- X--- ---- ---- ---- --X- ---- /RBO*/DO*D1*D2
29 X--- -X-- X--- ---- ---- ---- --X- ---- /RBO*DO*/D1*D2
30 ---- ---- ---- ---- ---- X--- ---- ---- LT
31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

32 ---- ---- ---- ---- X--- ---- ---- ---- IC
33 -X-- ---- -X-- ---- ---- ---- --X- ---- /RBO*/DO*/D2
34 ---- ---- X--- X--- ---- ---- --X- ---- /RBO*D2*D3
35 -X-- X--- ---- ---- ---- ---- --X- ---- /RBO*/DO*D1
36 ---- X--- ---- X--- ---- ---- --X- ---- /RBO*D1*D3
37 ---- ---- ---- ---- ---- X--- ---- ---- LT
38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

40 ---- ---- ---- ---- X--- ---- ---- ---- IC
41 -X-- -X-- ---- ---- ---- ---- --X- ---- /RBO*/DO*/D1
42 ---- ---- -X-- X--- ---- ---- --X- ---- /RBO*/D2*D3
43 ---- X--- ---- X--- ---- ---- --X- ---- /RBO*D1*D3
44 -X-- ---- X--- ---- ---- ---- --X- ---- /RBO*/DO*D2
45 ---- -X-- X--- -X-- ---- ---- --X- ---- /RBO*/D1*D2*/D3
46 ---- ---- ---- ---- ---- X--- ---- ---- LT
47 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

48 ---- ---- ---- ---- ---- ---- ---- ----
49 -XX- -X-- -X-- -X-- ---- ---- ---- ---- /DO*/D1*/D2*/D3*/RBI
50 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
51 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

56 ---- ---- ---- ---- -X-- ---- ---- ---- /IC
57 ---- X--- -X-- ---- ---- ---- --X- ---- /RBO*D1*/D2
58 X--- ---- ---- X--- ---- ---- --X- ---- /RBO*DO*D3
59 ---- ---- -X-- X--- ---- ---- --X- ---- /RBO*/D2*D3
60 -X-- X--- ---- ---- ---- ---- --X- ---- /RBO*/DO*D1
61 ---- -X-- X--- -X-- ---- ---- --X- ---- /RBO*/D1*D2*/D3
62 ---- ---- ---- ---- ---- X--- ---- ---- LT
63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

 NUMBER OF FUSES BLOWN = 1496
```

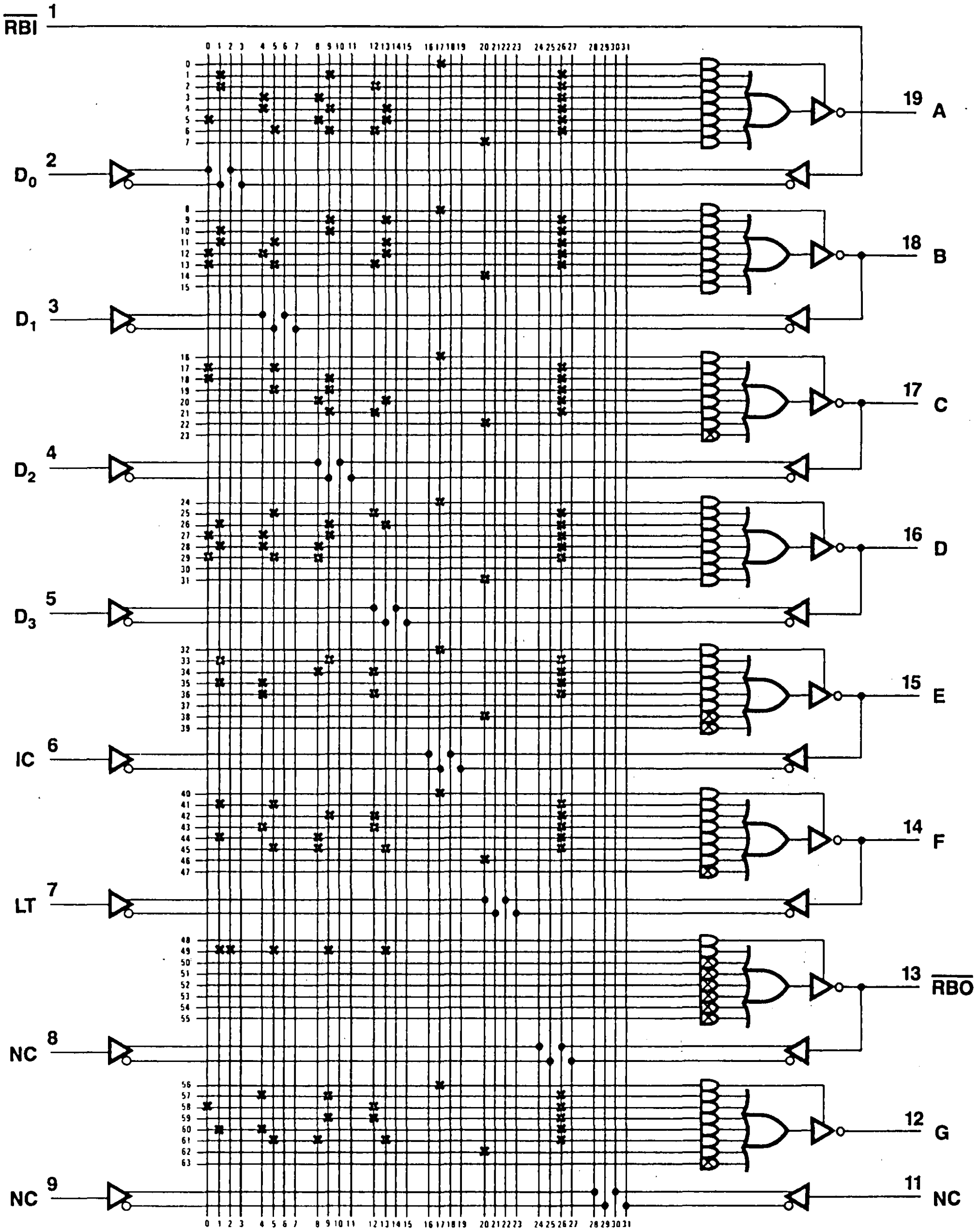
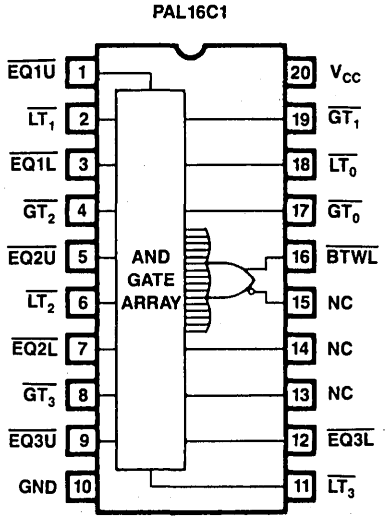


Figure 8.4.2 Logic Diagram PAL16L8

8.5 BETWEEN LIMITS COMPARATOR/LOGIC



LOGIC SYMBOL

Figure 8.5.1 PAL Device 16C1 Limit Checker

```
PALASM VERSION 1.5

PAL16C1
PAT 0021
BETWEEN LIMITS COMPARITOP LOGIC
NSC
/EQ1U /LT1 /EQ1L /GT2 /EQ2U /LT2 /EQ2L /GT3 /EQ3U GND
/LT3 /EQ3L NC NC NC /BTWL /GT0 /LT0 /GT1 VCC
/BTWL = GT3 + GT2*EQ3U + GT1*EQ3U*EQ2U + GT0*EQ3U*EQ2U*EQ1U +
LT3 + LT2*EQ3L + LT1*EQ3L*EQ2L + LT0*EQ3L*EQ2L*EQ1L
DESCRIPTION

BETWEEN LIMITS COMPARITOP LOGIC

                  ***************  ***************
                  *             * *             *
               ****                             ****
       /EQ1U   * 1*            P A L            *20*   VCC
               ****                             ****
                  *           1 6 C 1           *
               ****                             ****
        /LT1   * 2*                             *19*   /GT1
               ****                             ****
                  *                             *
               ****                             ****
       /EQ1L   * 3*                             *18*   /LT0
               ****                             ****
                  *                             *
```

```
                ****                             ****
         /GT2   * 4*                             *17*   /GTO
                ****                             ****
                   *                             *
                ****                             ****
        /EQ2U   * 5*                             *16*   /BTWL
                ****                             ****
                   *                             *
                ****                             ****
         /LT2   * 6*                             *15*   NC
                ****                             ****
                   *                             *
                ****                             ****
        /EQ2L   * 7*                             *14*   NC
                ****                             ****
                   *                             *
                ****                             ****
         /GT3   * 8*                             *13*   NC
                ****                             ****
                   *                             *
                ****                             ****
        /EQ3U   * 9*                             *12*   /EQ3L
                ****                             ****
                   *                             *
                ****                             ****
          GND   *10*                             *11*   /LT3
                ****                             ****
                   *                             *
                   *******************************

 BETWEEN LIMITS COMPARITOP LOGIC

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

BEG*FPLT PAL16C1   8

 24 ---- ---- ---- ---- ---- ---- -X-- ---- GT3
 25 ---- ---- -X-- ---- ---- ---- ---- -X-- GT2*EQ3U
 26 ---- ---X ---- -X-- ---- ---- ---- -X-- GT1*EQ3U*EQ2U
 27 ---X ---- ---- -X-X ---- ---- ---- -X-- GTO*EQ3U*EQ2U*EQ1U
 28 ---- ---- ---- ---- ---- ---- ---- ---X LT3
 29 ---- ---- ---- ---- -X-- ---- ---X ---- LT2*EQ3L
 30 -X-- ---- ---- ---- ---- -X-- ---X ---- LT1*EQ3L*EQ2L
 31 ---- -X-- ---X ---- ---- -X-- ---X ---- LTO*EQ3L*EQ2L*EQ1L

 32 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 33 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 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

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)    - : FUSE BLOWN   (H,P,1)
          O : PHANTOM FUSE   (L,N,0)    O : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOWN =  236
```

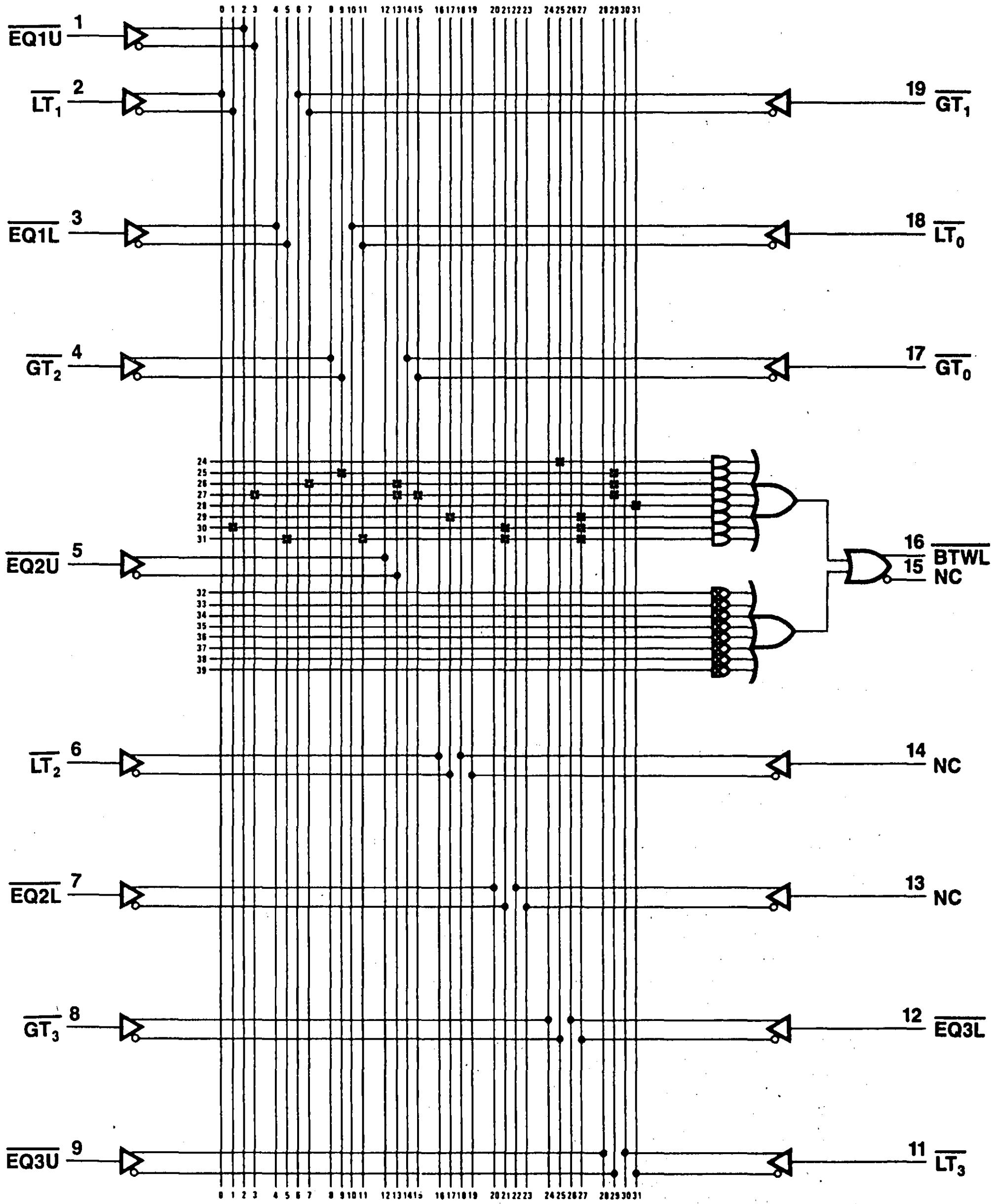
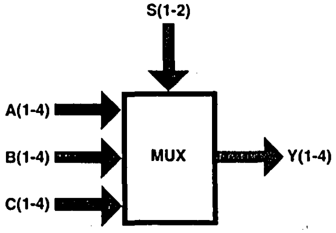


Figure 8.5.2 Logic Diagram PAL16C1

8.6 QUADRUPLE 3-LINE/1-LINE DATA SELECTOR MULTIPLEXER



```
 PALASM VERSION 1.5

 PAL14H4
 PAT0016
 DATA SECLECTOR MULTIPLEXER
 PAL DESIGN
 1A 2A 3A 4A 1B 2B 3B 4B 1C GND
 2C 3C 4C 4Y 3Y 2Y 1Y S1 S0 VCC
 1Y = 1A*/S0*/S1 + 1B*S0*/S1 + 1C*/S0*S1
 2Y = 2A*/S0*/S1 + 2B*S0*/S1 + 2C*/S0*S1
 3Y = 3A*/S0*/S1 + 3B*S0*/S1 + 3C*/S0*S1
 4Y = 4A*/S0*/S1 + 4B*S0*/S1 + 4C*/S0*S1
 DESCRIPTION

 DATA SECLECTOR MULTIPLEXER

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

BEG*FPLT PAL14H4   8


 16 --X- ---X ---X --00 --00 ---- ---- ---- 1A*/S0*/S1
 17 ---- --X- ---X X-00 --00 ---- ---- ---- 1B*S0*/S1
 18 ---- ---X --X- --00 --00 ---- ---- X--- 1C*/S0*S1
 19 XXXX XXXX XXXX XX00 XX00 XXXX XXXX XXXX

 24 X--- ---X ---X --00 --00 ---- ---- ---- 2A*/S0*/S1
 25 ---- --X- ---X --00 X-00 ---- ---- ---- 2B*S0*/S1
 26 ---- ---X --X- --00 --00 ---- ---- --X- 2C*/S0*S1
 27 XXXX XXXX XXXX XX00 XX00 XXXX XXXX XXXX

 32 ---- X--X ---X --00 --00 ---- ---- ---- 3A*/S0*/S1
 33 ---- --X- ---X --00 --00 X--- ---- ---- 3B*S0*/S1
 34 ---- ---X --X- --00 --00 ---- --X- ---- 3C*/S0*S1
 35 XXXX XXXX XXXX XX00 XX00 XXXX XXXX XXXX

 40 ---- ---X X--X --00 --00 ---- ---- ---- 4A*/S0*/S1
 41 ---- --X- ---X --00 --00 ---- X--- ---- 4B*S0*/S1
 42 ---- ---X --X- --00 --00 --X- ---- ---- 4C*/S0*S1
 43 XXXX XXXX XXXX XX00 XX00 XXXX XXXX XXXX


END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)
          0 : PHANTOM FUSE   (L,N,0)   0 : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOWN =  348
```

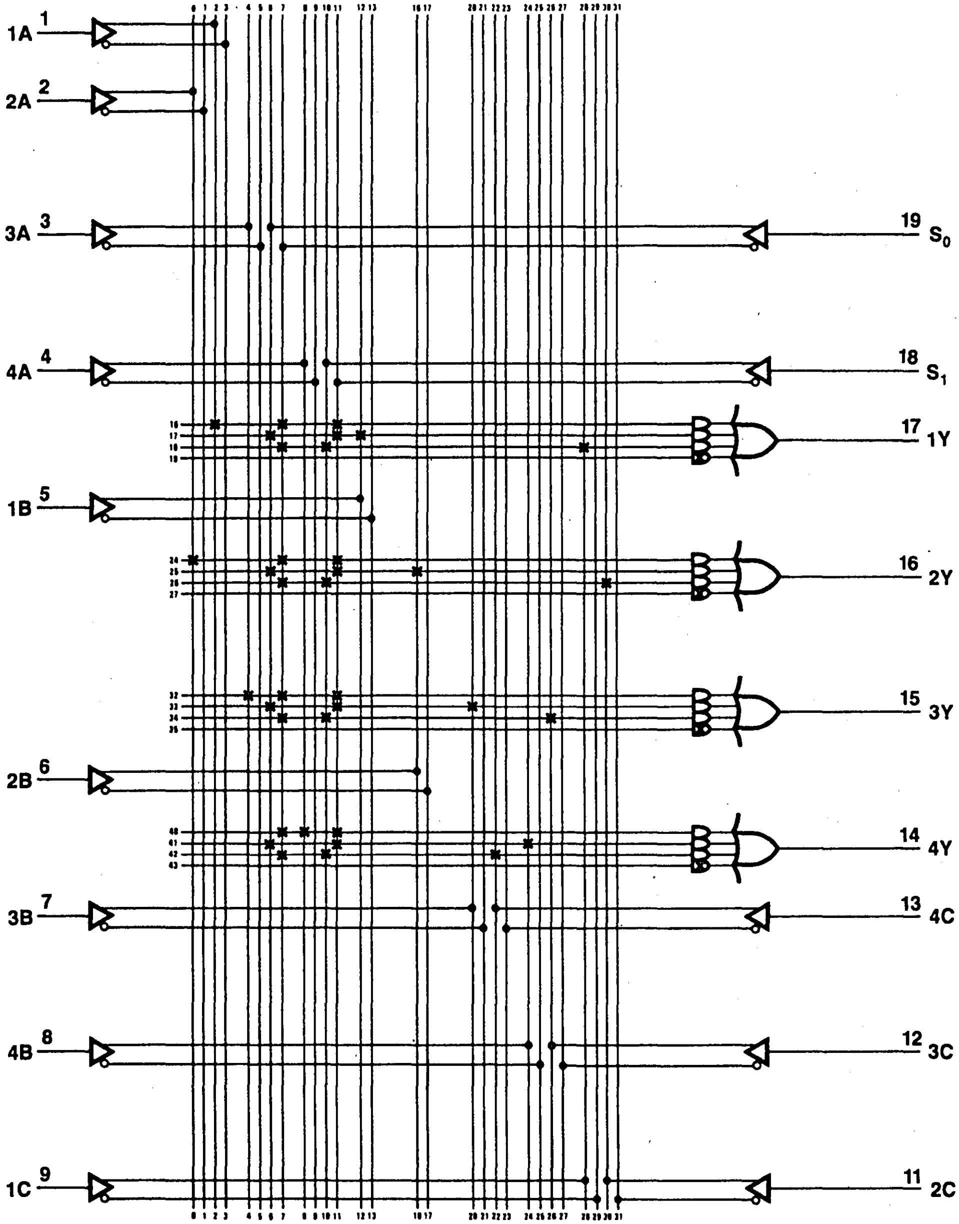


Figure 8.6.1 Logic Diagram PAL14H4

8.7 4-BIT COUNTER WITH 2-INPUT MULTIPLEXER

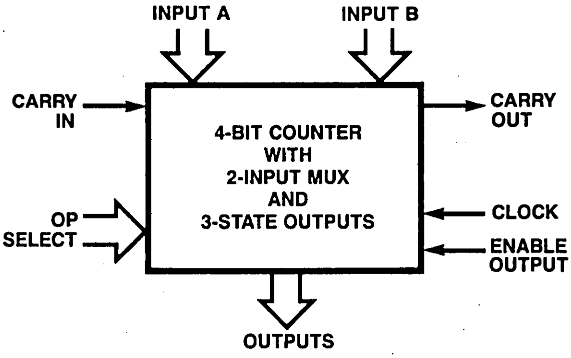


Figure 8.7.1 Four-Bit Counter With Two-Input Multiplexer

```
PALASM VERSION 1.5

PAL16R4
PAT0034
4 BIT COUNTER WITH2 INPUT MUX
NSC
CLOCK A0 A1 A2 A3 B0 B1 B2 B3 GND
/E COUT I1 Q3 Q2 Q1 Q0 I0 CIN VCC
/Q0:=/I1*/I0*/Q0 + /I1*I0*/A0 + I1*/I0*/B0 +
I1*I0*/CIN*/Q0 + I1*I0*CIN*Q0
/Q1:=/I1*/I0*/Q1 + /I1*I0*/A1 + I1*/I0*/B1 +
I1*I0*/CIN*/Q1 + I1*I0*CIN*Q1*Q0 + I1*I0*/Q1*/Q0
/Q2:=/I1*/I0*/Q2 + /I1*I0*/A2 + I1*/I0*B2 + I1*I0*/CIN*/Q2 +
I1*I0*CIN*Q2*Q1*Q0 + I1*I0*/Q2*/Q1 + I1*I0*/Q2*/Q0
/Q3:=/I1*/I0*/Q3 + /I1*I0*/A3 + I1*/I0*/B3 + I1*I0*/CIN*/Q3 +
I1*I0*CIN*Q3*Q2*Q1*Q0 + I1*I0*/Q3*/Q2 + I1*I0*/Q3*/Q1 +
I1*I0*/Q3*/Q0
IF(VCC)/COUT = /CIN + /Q3 + /Q2 + /Q1 + /Q0
DESCRIPTION

4 BIT COUNTER WITH2 INPUT MUX

                  *************** ***************
                  *             * *             *
               ****                             ****
       CLOCK   * 1*            P A L            *20*   VCC
               ****                             ****
                  *           1 6 R 4           *
               ****                             ****
          A0   * 2*                             *19*   CIN
               ****                             ****
                  *                             *
               ****                             ****
          A1   * 3*                             *18*   I0
               ****                             ****
```

```
         *                             *
      ****                             ****
 A2   * 4*                             *17*   Q0
      ****                             ****
         *                             *
      ****                             ****
 A3   * 5*                             *16*   Q1
      ****                             ****
         *                             *
      ****                             ****
 B0   * 6*                             *15*   Q2
      ****                             ****
         *                             *
      ****                             ****
 B1   * 7*                             *14*   Q3
      ****                             ****
         *                             *
      ****                             ****
 B2   * 8*                             *13*   I1
      ****                             ****
         *                             *
      ****                             ****
 B3   * 9*                             *12*   COUT
      ****                             ****
         *                             *
      ****                             ****
GND   *10*                             *11*   /E
      ****                             ****
         *                             *
         *******************************

    4 BIT COUNTER WITH2 INPUT MUX

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

   BEG*FPLT PAL16R4    8
     0 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
     1 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
     2 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
     3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
     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 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
     9 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
    10 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
    11 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
    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 ---X ---- ---- ---- ---X ---- /I1*/I0*/Q0
 17 -X-- --X- ---- ---- ---- ---- ---X ---- /I1*I0*/A0
 18 ---- ---X ---- ---- -X-- ---- --X- ---- I1*/I0*/B0
 19 ---X --X- ---X ---- ---- ---- --X- ---- I1*I0*/CIN*/Q0
 20 --X- --X- --X- ---- ---- ---- --X- ---- I1*I0*CIN*Q0
 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 ---- ---X ---- ---X ---- ---- ---X ---- /I1*/I0*/Q1
 25 ---- -XX- ---- ---- ---- ---- ---X ---- /I1*I0*/A1
 26 ---- ---X ---- ---- ---- -X-- --X- ---- I1*/I0*/B1
 27 ---X --X- ---- ---X ---- ---- --X- ---- I1*I0*/CIN*/Q1
 28 --X- --X- --X- --X- ---- ---- --X- ---- I1*I0*CIN*Q1*Q0
 29 ---- --X- ---X ---X ---- ---- --X- ---- I1*I0*/Q1*/Q0
 30 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 31 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 32 ---- ---X ---- ---- ---X ---- ---X ---- /I1*/I0*/Q2
 33 ---- --X- -X-- ---- ---- ---- ---X ---- /I1*I0*/A2
 34 ---- ---X ---- ---- ---- ---- X-X- ---- I1*/I0*B2
 35 ---X --X- ---- ---- ---X ---- --X- ---- I1*I0*/CIN*/Q2
 36 --X- --X- --X- --X- --X- ---- --X- ---- I1*I0*CIN*Q2*Q1*Q0
 37 ---- --X- ---- ---X ---X ---- --X- ---- I1*I0*/Q2*/Q1
 38 ---- --X- ---X ---- ---X ---- --X- ---- I1*I0*/Q2*/Q0
 39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 40 ---- ---X ---- ---- ---- ---X ---X ---- /I1*/I0*/Q3
 41 ---- --X- ---- -X-- ---- ---- ---X ---- /I1*I0*/A3
 42 ---- ---X ---- ---- ---- ---- --X- -X-- I1*/I0*/B3
 43 ---X --X- ---- ---- ---- ---X --X- ---- I1*I0*/CIN*/Q3
 44 --X- --X- --X- --X- --X- --X- --X- ---- I1*I0*CIN*Q3*Q2*Q1*Q0
 45 ---- --X- ---- ---- ---X ---X --X- ---- I1*I0*/Q3*/Q2
 46 ---- --X- ---- ---X ---- ---X --X- ---- I1*I0*/Q3*/Q1
 47 ---- --X- ---X ---- ---- ---X --X- ---- I1*I0*/Q3*/Q0

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

 56 ---- ---- ---- ---- ---- ---- ---- ----
 57 ---X ---- ---- ---- ---- ---- ---- ---- /CIN
 58 ---- ---- ---- ---- ---- ---X ---- ---- /Q3
 59 ---- ---- ---- ---- ---X ---- ---- ---- /Q2
 60 ---- ---- ---- ---X ---- ---- ---- ---- /Q1
 61 ---- ---- ---X ---- ---- ---- ---- ---- /Q0
 62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

 NUMBER OF FUSES BLOWN =  921
```

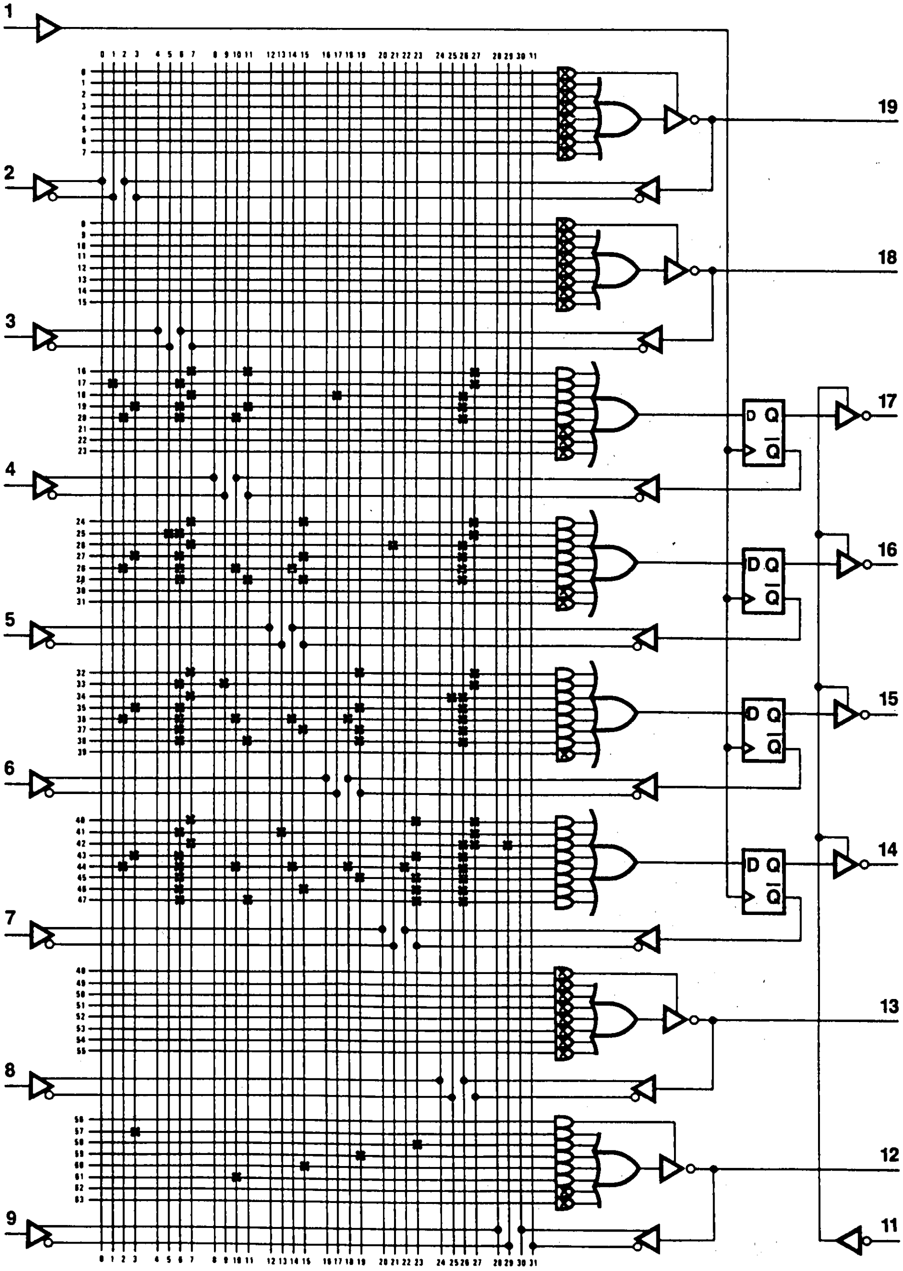


Figure 8.7.2 Logic Diagram PAL16R4

8.8 8-BIT SYNCHRONOUS COUNTER

The 8-bit synchronous counter is used in many systems. The input A0 serves a mode control with LOW for LOAD operation and HIGH for count operation. Input A1 enables the LOAD operation when A0 is set in the LOAD mode and doesn't care when the count mode is chosen. This enables the counter to be cascaded as a multibyte counter with the capability of leading individual byte from a simple byte wide data bus and a common clock. /CIN is the carry input and /COUT is the carry output.

```
PAL20X8
8BIT SYNCHRONOUS COUNTER
LOGIC DESIGN
NSC
CLK A0 X0 X1 X2 X3 X4 X5 X6 X7 A1 GND
/EN /COUT /Y7 /Y6 /Y5 /Y4 /Y3 /Y2 /Y1 /Y0 /CIN VCC
Y0:=/A1*/A0*Y0
+A0*Y0
:+:A1*/A0*X0
+A0*CIN
Y1:=/A1*/A0*Y1+
A0*Y1:+:
A1*/A0*X1 +
A0*CIN*Y0
Y2:=/A1*/A0*Y2+
A0*Y2:+:
A1*/A0*X2 +
A0*CIN*Y0*Y1
Y3:=/A1*/A0*Y3+
A0*Y3:+:A1*/A0*X3 +
A0*CIN*Y0*Y1*Y2
Y4:=/A1*/A0*Y4 +
A0*Y4 :+:
A1*/A0*X4+
A0*CIN*Y0*Y1*Y2*Y3
Y5:=/A1*/A0*Y5 +
A0*Y5:+:
A1*/A0*X5+
A0*CIN*Y0*Y1*Y2*Y3*Y4
Y6:=/A1*/A0*Y6 +
A0*Y6:+:
A1*/A0*X6+
A0*CIN*Y0*Y1*Y2*Y3*Y4*Y5
Y7:=/A1*/A0*Y7 +
A0*Y7:+:
A1*/A0*X7+
A0*CIN*Y0*Y1*Y2*Y3*Y4*Y5*Y6
IF(VCC)COUT = CIN*Y0*Y1*Y2*Y3*Y4*Y5*Y6*Y7
DESCRIPTION
```

```
LOGIC DESIGN

               ***************  **************
               *             * *             *
            ****                             ****
     CLK    * 1*            P A L            *24*   VCC
            ****                             ****
               *           2 0 X 8           *
            ****                             ****
      A0    * 2*                             *23*   /CIN
            ****                             ****
               *                             *
            ****                             ****
      X0    * 3*                             *22*   /Y0
            ****                             ****
               *                             *
            ****                             ****
      X1    * 4*                             *21*   /Y1
            ****                             ****
               *                             *
            ****                             ****
      X2    * 5*                             *20*   /Y2
            ****                             ****
               *                             *
            ****                             ****
      X3    * 6*                             *19*   /Y3
            ****                             ****
               *                             *
            ****                             ****
      X4    * 7*                             *18*   /Y4
            ****                             ****
               *                             *
            ****                             ****
      X5    * 8*                             *17*   /Y5
            ****                             ****
               *                             *
            ****                             ****
      X6    * 9*                             *16*   /Y6
            ****                             ****
               *                             *
            ****                             ****
      X7    *10*                             *15*   /Y7
            ****                             ****
               *                             *
            ****                             ****
      A1    *11*                             *14*   /COUT
            ****                             ****
               *                             *
            ****                             ****
     GND    *12*                             *13*   /EN
            ****                             ****
               *                             *
               *******************************
```

```
 LOGIC DESIGN

                 11 1111 1111 2222 2222 2233 3333 3333
    0123 4567 8901 2345 6789 0123 4567 8901 2345 6789

BEG*FPLT PAL20X8  10
  0 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  1 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  2 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
  3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

  8 -X-- ---X ---- ---- ---- ---- ---- ---- ---- -X-- /A1*/A0*Y0
  9 X--- ---X ---- ---- ---- ---- ---- ---- ---- ---- A0*Y0
 10 -X-- X--- ---- ---- ---- ---- ---- ---- ---- X--- A1*/A0*X0
 11 X--X ---- ---- ---- ---- ---- ---- ---- ---- ---- A0*CIN

 16 -X-- ---- ---X ---- ---- ---- ---- ---- ---- -X-- /A1*/A0*Y1
 17 X--- ---- ---X ---- ---- ---- ---- ---- ---- ---- A0*Y1
 18 -X-- ---- X--- ---- ---- ---- ---- ---- ---- X--- A1*/A0*X1
 19 X--X ---X ---- ---- ---- ---- ---- ---- ---- ---- A0*CIN*Y0

 24 -X-- ---- ---- ---X ---- ---- ---- ---- ---- -X-- /A1*/A0*Y2
 25 X--- ---- ---- ---X ---- ---- ---- ---- ---- ---- A0*Y2
 26 -X-- ---- ---- X--- ---- ---- ---- ---- ---- X--- A1*/A0*X2
 27 X--X ---X ---X ---- ---- ---- ---- ---- ---- ---- A0*CIN*Y0*Y1

 32 -X-- ---- ---- ---- ---X ---- ---- ---- ---- -X-- /A1*/A0*Y3
 33 X--- ---- ---- ---- ---X ---- ---- ---- ---- ---- A0*Y3
 34 -X-- ---- ---- ---- X--- ---- ---- ---- ---- X--- A1*/A0*X3
 35 X--X ---X ---X ---X ---- ---- ---- ---- ---- ---- A0*CIN*Y0*Y1*Y2

 40 -X-- ---- ---- ---- ---- ---X ---- ---- ---- -X-- /A1*/A0*Y4
 41 X--- ---- ---- ---- ---- ---X ---- ---- ---- ---- A0*Y4
 42 -X-- ---- ---- ---- ---- X--- ---- ---- ---- X--- A1*/A0*X4
 43 X--X ---X ---X ---X ---X ---- ---- ---- ---- ---- A0*CIN*Y0*Y1*Y2*Y3

 48 -X-- ---- ---- ---- ---- ---- ---X ---- ---- -X-- /A1*/A0*Y5
 49 X--- ---- ---- ---- ---- ---- ---X ---- ---- ---- A0*Y5
 50 -X-- ---- ---- ---- ---- ---- X--- ---- ---- X--- A1*/A0*X5
 51 X--X ---X ---X ---X ---X ---X ---- ---- ---- ---- A0*CIN*Y0*Y1*Y2*Y3*Y4

 56 -X-- ---- ---- ---- ---- ---- ---- ---X ---- -X-- /A1*/A0*Y6
 57 X--- ---- ---- ---- ---- ---- ---- ---X ---- ---- A0*Y6
 58 -X-- ---- ---- ---- ---- ---- ---- X--- ---- X--- A1*/A0*X6
 59 X--X ---X ---X ---X ---X ---X ---X ---- ---- ---- A0*CIN*Y0*Y1*Y2*Y3*Y4*Y5

 64 -X-- ---- ---- ---- ---- ---- ---- ---- ---X -X-- /A1*/A0*Y7
 65 X--- ---- ---- ---- ---- ---- ---- ---- ---X ---- A0*Y7
 66 -X-- ---- ---- ---- ---- ---- ---- ---- X--- X--- A1*/A0*X7
 67 X--X ---X ---X ---X ---X ---X ---X ---X ---- ---- A0*CIN*Y0*Y1*Y2*Y3*Y4*Y-

 72 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
 73 ---X ---X ---X ---X ---X ---X ---X ---X ---X ---- CIN*Y0*Y1*Y2*Y3*Y4*Y5*Y-
 74 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 75 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)
          0 : PHANTOM FUSE   (L,N,0)   0 : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOW = 1243
```

8.9 6-BIT SHIFT REGISTER WITH THREE-STATE OUTPUTS

```
PALASM VERSION 1.5

PAL16R6
PAT05
6BIT
SVALE
CK SR D0 D1 D2 D3 D4 D5 SL GND
/E RILO Q5 Q4 Q3 Q2 Q1 Q0 LIRO VCC
IF(SR*/SL)/LIRO=/Q0
/Q0:=/SR*/SL*/Q0+SR*/SL*/Q1+/SR*SL*/LIRO+SR*SL*/D0
/Q1:=/SR*/SL*/Q1+SR*/SL*/Q2+/SR*SL*/Q0+SR*SL*/D1
/Q2:=/SR*/SL*/Q2+SR*/SL*/Q3+/SR*SL*/Q1+SR*SL*/D2
/Q3:=/SR*/SL*/Q3+SR*/SL*/Q4+/SR*SL*/Q2+SR*SL*/D3
/Q4:=/SR*/SL*/Q4+SR*/SL*/Q5+/SR*SL*/Q3+SR*SL*/D4
/Q5:=/SR*/SL*/Q5+SR*/SL*/RILO+/SR*SL*/Q4+SR*SL*/D5
IF(/SR*SL)/RILO=/Q5
DESCRIPTION

6BIT

         *************** ***************
         *             * *             *
      ****                             ****
 CK   * 1*            P A L            *20*   VCC
      ****                             ****
         *           1 6 R 6           *
      ****                             ****
 SR   * 2*                             *19*   LIRO
      ****                             ****
         *                             *
      ****                             ****
 D0   * 3*                             *18*   Q0
      ****                             ****
         *                             *
      ****                             ****
 D1   * 4*                             *17*   Q1
      ****                             ****
         *                             *
      ****                             ****
 D2   * 5*                             *16*   Q2
      ****                             ****
         *                             *
      ****                             ****
 D3   * 6*                             *15*   Q3
      ****                             ****
         *                             *
      ****                             ****
 D4   * 7*                             *14*   Q4
      ****                             ****
         *                             *
      ****                             ****
 D5   * 8*                             *13*   Q5
      ****                             ****
         *                             *
      ****                             ****
 SL   * 9*                             *12*   RILO
      ****                             ****
```

```
             *                                *
           ****                               ****
     GND   *10*                               *11*   /E
           ****                               ****
             *                                *
             ********************************

  6BIT

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

BEG*FPLT PAL16R6   8
   0 X--- ---- ---- ---- ---- ---- ---- -X-- SR*/SL
   1 ---- ---X ---- ---- ---- ---- ---- ---- /Q0
   2 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
   3 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
   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-- ---X ---- ---- ---- ---- ---- -X-- /SR*/SL*/Q0
   9 X--- ---- ---X ---- ---- ---- ---- -X-- SR*/SL*/Q1
  10 -X-X ---- ---- ---- ---- ---- ---- X--- /SR*SL*/LIR0
  11 X--- -X-- ---- ---- ---- ---- ---- X--- SR*SL*/D0
  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-- ---- ---X ---- ---- ---- ---- -X-- /SR*/SL*/Q1
  17 X--- ---- ---- ---X ---- ---- ---- -X-- SR*/SL*/Q2
  18 -X-- ---X ---- ---- ---- ---- ---- X--- /SR*SL*/Q0
  19 X--- ---- -X-- ---- ---- ---- ---- X--- SR*SL*/D1
  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 -X-- ---- ---- ---X ---- ---- ---- -X-- /SR*/SL*/Q2
  25 X--- ---- ---- ---- ---X ---- ---- -X-- SR*/SL*/Q3
  26 -X-- ---- ---X ---- ---- ---- ---- X--- /SR*SL*/Q1
  27 X--- ---- ---- -X-- ---- ---- ---- X--- SR*SL*/D2
  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 ---- ---- -X-- /SR*/SL*/Q3
  33 X--- ---- ---- ---- ---- ---X ---- -X-- SR*/SL*/Q4
  34 -X-- ---- ---- ---X ---- ---- ---- X--- /SR*SL*/Q2
  35 X--- ---- ---- ---- -X-- ---- ---- X--- SR*SL*/D3
  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-- /SR*/SL*/Q4
 41 X--- ---- ---- ---- ---- ---- ---X -X-- SR*/SL*/Q5
 42 -X-- ---- ---- ---- ---X ---- ---- X--- /SR*SL*/Q3
 43 X--- ---- ---- ---- ---- -X-- ---- X--- SR*SL*/D4
 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

 48 -X-- ---- ---- ---- ---- ---- ---X -X-- /SR*/SL*/Q5
 49 X--- ---- ---- ---- ---- ---- ---- -X-X SR*/SL*/RILO
 50 -X-- ---- ---- ---- ---- ---X ---- X--- /SR*SL*/Q4
 51 X--- ---- ---- ---- ---- ---- -X-- X--- SR*SL*/D5
 52 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 53 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 54 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 55 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

 56 -X-- ---- ---- ---- ---- ---- ---- X--- /SR*SL
 57 ---- ---- ---- ---- ---- ---- ---X ---- /Q5
 58 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 59 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 60 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 61 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 62 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
 63 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)

 NUMBER OF FUSES BLOWN =  818
```

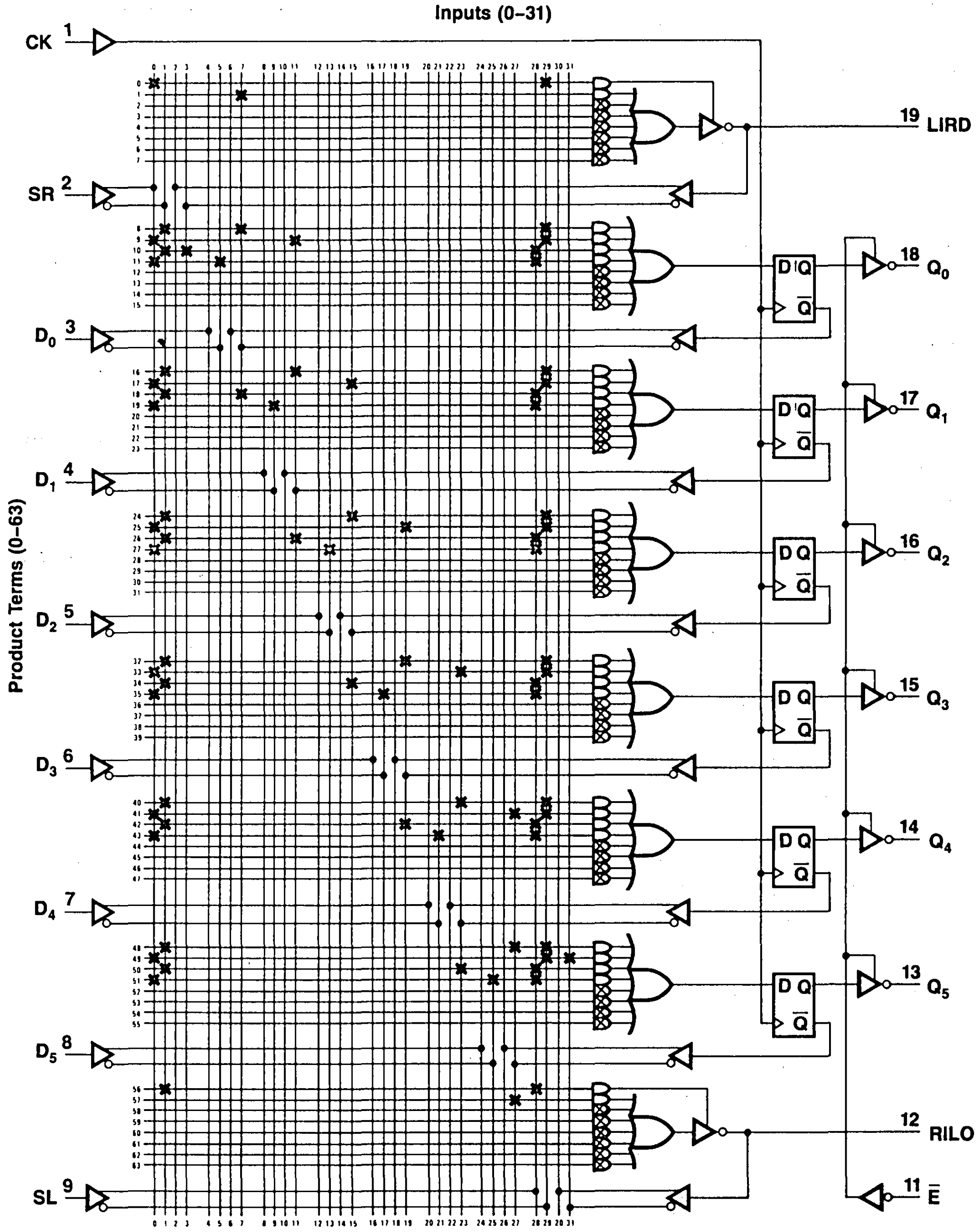


Figure 8.9.1 Logic Diagram PAL16R6

8.10 PORTION OF RANDOM CONTROL LOGIC FOR 8086 CPU BOARD

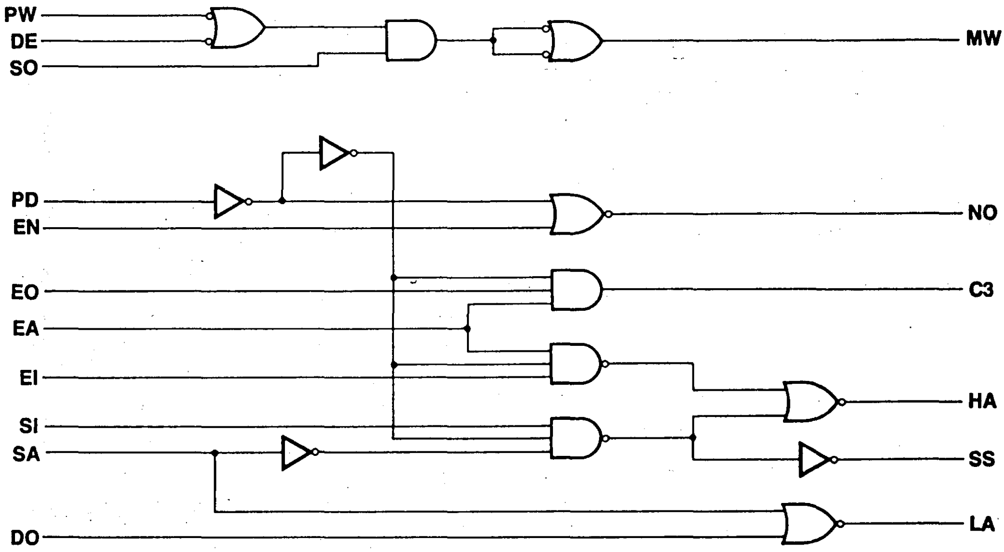


Figure 8.10.1 Control Logic for 8086 CPU Board

```
PALASM VERSION 1.5

PAL12H6
PAT03
8086
CPU
PD EN EO EA S1 SA E1 DO DE GND
SO NC3 NO C3 HA SS LA MW PW VCC
MW=/SO+PW*DE
LA=/SA*/DO
SS=S1*PD*/SA
HA=S1*PD*/SA*EA*E1
C3=PD*EO*EA
NO=PD*/EN
DESCRIPTION

8086

                  ***************  ***************
                  *             * *             *
               ****                             ****
          PD   * 1*            P A L            *20*   VCC
               ****                             ****
                  *           1 2 H 6           *
               ****                             ****
          EN   * 2*                             *19*   PW
               ****                             ****
```

```
              *                             *
           ****                             ****
      EO   * 3*                             *18*   MW
           ****                             ****
              *                             *
           ****                             ****
      EA   * 4*                             *17*   LA
           ****                             ****
              *                             *
           ****                             ****
      S1   * 5*                             *16*   SS
           ****                             ****
              *                             *
           ****                             ****
      SA   * 6*                             *15*   HA
           ****                             ****
              *                             *
           ****                             ****
      E1   * 7*                             *14*   C3
           ****                             ****
              *                             *
           ****                             ****
      DO   * 8*                             *13*   NO
           ****                             ****
              *                             *
           ****                             ****
      DE   * 9*                             *12*   NC3
           ****                             ****
              *                             *
           ****                             ****
     GND   *10*                             *11*   SO
           ****                             ****
              *                             *
              *******************************

 8086

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

BEG*FPLT PAL12H6   8

  8 ---- ---- --00 --00 --00 --00 ---- ---X /SO
  9 ---- --X- --00 --00 --00 --00 ---- X--- PW*DE
 10 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 11 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

 16 ---- ---- --00 --00 -X00 --00 -X-- ---- /SA*/DO
 17 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

 24 --X- ---- --00 X-00 -X00 --00 ---- ---- S1*PD*/SA
 25 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

 32 --X- ---- X-00 X-00 -X00 X-00 ---- ---- S1*PD*/SA*EA*E1
 33 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX

 40 --X- X--- X-00 --00 --00 --00 ---- ---- PD*EO*EA
 41 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 48 -XX- ---- --00 --00 --00 --00 ---- ---- PD*/EN
 49 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 50 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX
 51 XXXX XXXX XX00 XX00 XX00 XX00 XXXX XXXX


END*FPLT

 LEGEND:  X : FUSE NOT BLOWN (L,N,0)   - : FUSE BLOWN   (H,P,1)
          0 : PHANTOM FUSE   (L,N,0)   0 : PHANTOM FUSE (H,P,1)

 NUMBER OF FUSES BLOWN =  206
```

PAL DEVICES FOR EASY INTERFACE BETWEEN DP8408/09* DRAM CONTROLLER AND POPULAR MICROPROCESSORS

High storage density and low cost have made dynamic RAMs the designers choice in most memory applications. However, the major drawbacks of dynamic RAMs are the complex timing involved and periodic refresh needed to keep all memory cells charged. With the introduction of the DP8408/09 Dynamic RAM controller/driver, the above complexities are simplified.

Use of PAL devices adds flexibility in the design as PAL device logic equations can be modified by the user for his/her application and programmed into any of the PAL devices. In addition, PAL devices lower the parts count in memory system design. For most memory operations, the PAL devices (DP8432/322/332) can be directly connected between the control signals from the CPU chip set and the DP8408/09 dynamic RAM controller. The PAL device allows hidden refresh using the DP8408/09. In a standard memory cycle, the access can be slowed by one clock cycle to accommodate slower memories. This extra wait state will not appear during the hidden refresh cycle, so faster devices on the CPU bus will not be affected. Similarly, PAL devices allow for the insertion of wait states for processors operating at high CPU clock frequencies to use slower dynamic RAMs.

The following three applications describe the use of National's PAL16R6, PAL16R4 and PAL16R8 for the ease and flexibility of interfacing DP8408/09 with popular microprocessors such as the 32032, 68000, 8086, and 8088. Today the PAL device family offers the designer flexibility to design desired speed/power PAL device in his memory systems, and achieve the memory operations at very high frequencies with or without wait state conditions.

* *DP8408/09 is part of the interface product line at National Semiconductor Corp.*

8.11 DP84312 DYNAMIC RAM CONTROLLER INTERFACE CIRCUIT FOR THE NS32032 CPU

General Description

The DP84312 dynamic RAM Controller interface is a PAL device for interface between the DP8409 dynamic RAM Controller and the NS32032 microprocessor.

Using timing signals from the NS32032,timing and control unit and the NS32032 the DP84312 supplies all control signals needed to perform memory read, write, byte write, and refresh.

Features

- Low parts count memory system.
- Allows the DP8409 to perform hidden refresh.
- Allows for the insertion of wait states for slow dynamic RAMs.
- Supplies independent $\overline{\text{CASs}}$ for byte writing.
- Possibility of operation at 8MHz with no wait states.
- 20-pin 0.3 inch wide package.
- Standard National Semiconductor PAL device part (PAL16R6).
- PAL device logic equations can be modified by the user for his/her specific application and programmed into any of the National Semiconductor PAL device family, including the new high speed PAL devices.

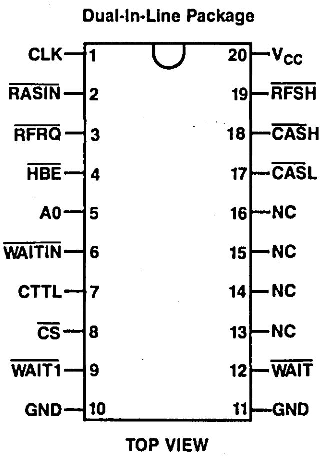


Figure 8.11.1 Connection Diagram

Symbol	Parameter	Min	Typ	Max	Units
V_{CC}	Supply Voltage	4.75	5.00	5.25	V
I_{OH}	High Level Output Current			–3.2	mA
I_{OL}	Low Level Output Current			24 (Note 2)	mA
T_{AA}	Operating Free Air Temperature	0		75	°C

Table 8.11.1 Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2			V
V_{IL}	Low Level Input Voltage				0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min, I_I = – 18 mA			– 1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OH} = Max	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OL} = Max			0.5	V
I_{OZH}	Off-State Output Current High Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 2.4V, V_{IL} = 0.8V			100	μA
I_{OZL}	Off-State Output Current Low Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 0.4V, V_{IL} = 0.8V			– 100	μA
I_I	Input Current at Maximum Input Voltage	V_{CC} = Max, V_I = 5.5V			1.0	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.4V			25	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V			– 250	μA
I_{OS}	Short Circuit Output Current	V_{CC} = Max	– 30		– 130	mA
I_{CC}	Supply Current	V_{CC} = Max		150	225 (Note 1)	mA

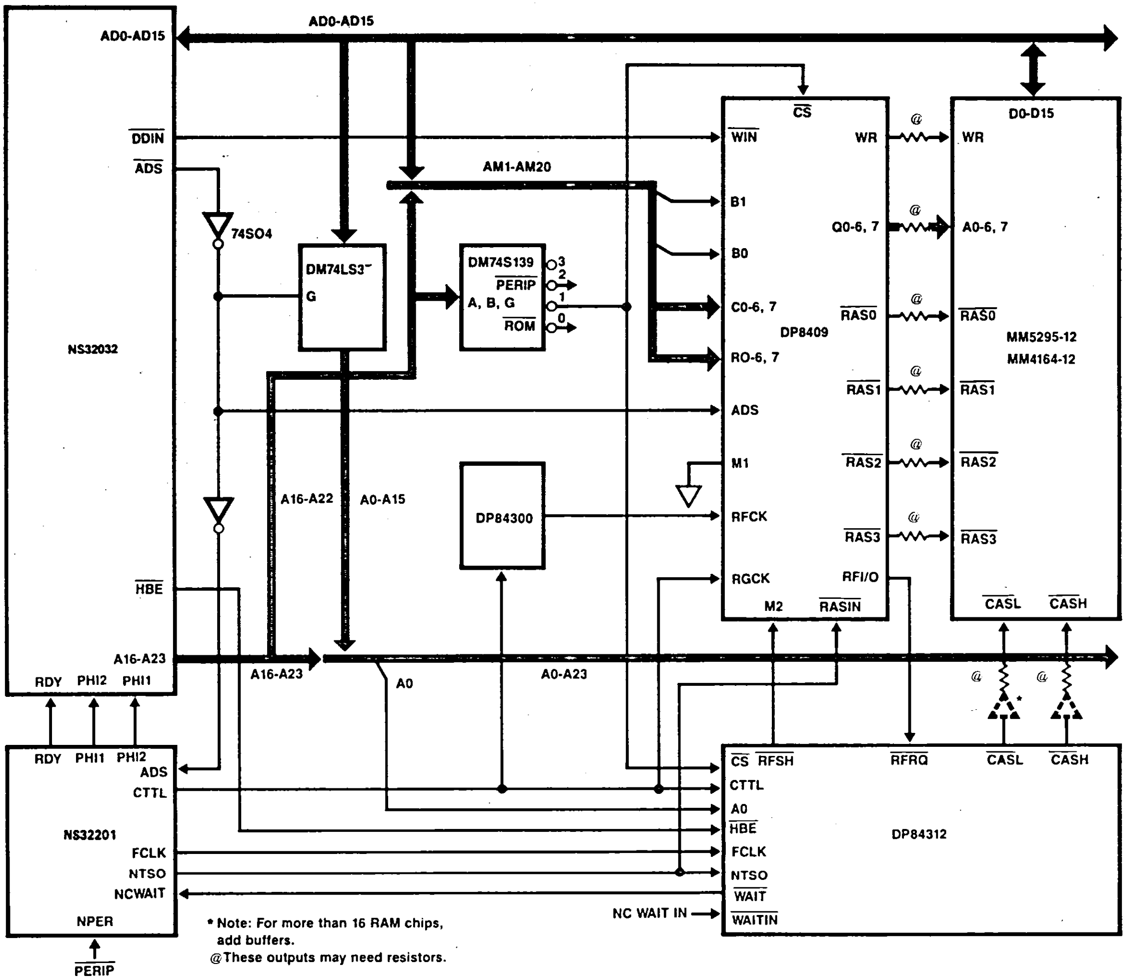
Table 8.11.2 Electrical Characteristics

Symbol	Parameter		Conditions $R_L = 667\Omega$	Commercial T_A = 0°C to + 75°C V_{CC} = 5.0V ± 5%			Units
				Min	Typ	Max	
t_{WD}	$\overline{\text{WAITIN}}$ to $\overline{\text{WAIT}}$ Delay		C_L = 45 pF		25	40	ns
t_{PD}	Clock to Output		C_L = 45 pF		15	25	ns
t_{PZX}	Pin 11 to Output Enable		C_L = 45 pF		15	25	ns
t_{PXZ}	Pin 11 to Output Disable		C_L = 5 pF		15	25	ns
t_w	Width of Clock	High		25			ns
		Low		25			ns
t_{su}	Set-Up Time			40			ns
t_h	Hold Time			0	– 15		ns

Note 1: I_{CC} = max at minimum temperature.

Note 2: One output at a time; otherwise 16 mA.

Table 8.11.3 Switching Chracteristics



* Note: For more than 16 RAM chips, add buffers.
@These outputs may need resistors.

Figure 8.11.2 System Block Diagram

Mnemonic Description

Input Signals

CLK Clock input. This clock comes from the FCLK output of the NS32201 timing and control unit, and supplies timing for the internal logic.

$\overline{\text{RASIN}}$ $\overline{\text{RAS}}$ input. This input is connected to the NTSO pin of the NS32201. This signal marks the start of a memory cycle.

$\overline{\text{RFRQ}}$ Refresh request. The DP8409 requests a forced refresh with this input.

$\overline{\text{HBE}}$, A0 Address select inputs. These inputs select the type of write during a write cycle, and select their respective $\overline{\text{CAS}}$ outputs. These inputs must remain stable throughout the memory cycle.

$\overline{\text{WAITIN}}$ This wait input allows other devices to use the NCWAIT line of the NS16201 clock chip.

CTTL System clock input. This clock is used to synchronize the memory system to the microprocessor clock.

$\overline{\text{CS}}$ Chip select. This input is used to determine if a memory cycle or a hidden refresh cycle is to be performed.

$\overline{\text{WAIT1}}$ Insert one wait state. This input allows the use of slow memories with a microprocessor using a fast clock by inserting a wait state in selected memory cycles.

Output Signals

$\overline{\text{RFSH}}$ Refresh. This output switches the DP8409 to a refresh mode.

$\overline{\text{CASH}}$, $\overline{\text{CASL}}$ $\overline{\text{CAS}}$ outputs. $\overline{\text{CASH}}$ is for controlling the high bank of dynamic RAMs, while $\overline{\text{CASL}}$ controls the $\overline{\text{CAS}}$ line of the lower bank of RAMs. If only eight RAMs are used in each bank, the $\overline{\text{CAS}}$ outputs will directly drive the memories. For large arrays, these outputs should be buffered with a high current driver, such as the DP84244 MOS driver.

$\overline{\text{WAIT}}$ This output controls the insertion of wait states. This output is ORed with $\overline{\text{WAITIN}}$ to allow other devices to insert wait states.

Functional Description

The DP84312 detects the start of a memory cycle when NTSO from the NS32032 timing and control unit (TCU) goes low. The NTSO signal is also used to supply $\overline{\text{RASIN}}$ to the DP8409 dynamic RAM controller. After the DP8409 has latched the row address and supplied the column address to the DRAMs, the DP84312 latches the column address. The DP84312 supplies two $\overline{\text{CAS}}$ outputs: one for the high byte of memory, and the other for the low byte. The ability to control the upper and lower bytes of memory separately is important during a memory write cycle where one byte of memory is to be written (byte write).

By connecting $\overline{\text{WAIT1}}$ of the DP84312 to ground, all selected memory cycles will have one wait state inserted. This allows an NS32032 operating at high CPU clock frequency to use slower dynamic RAMs.

Memory refresh can be achieved in one of two ways: hidden or forced. Hidden refresh is accomplished whenever a refresh is requested (internal to the DP8409) and an unselected memory cycle occurs. With a hidden refresh, the DP84312 does nothing while the DP8409 performs the refresh. If no refresh occurs before the trailing edge of refresh clock, the DP8409 will request a forced refresh. The DP84312 detects this request, and allows the current memory cycle to finish. It then outputs wait states to the CPU, which will hold the CPU if it requests a memory cycle. During this time the DP84312 has switched the dynamic RAM controller to the auto refresh mode, allowing it to perform a refresh. At the end of the refresh cycle, the DP8409 is switched back to the auto access mode, and the wait is removed after a sufficient $\overline{\text{RAS}}$ precharge time. The total forced refresh takes four CPU clock cycles, of which some, none or all may be actual wait states. If the CPU does not request a memory cycle during this refresh cycle, the refresh will not impact the CPU's performance.

The DP84312 can possibly be operated at 8 MHz with no wait states (WAIT1 = "1") given the following conditions:

T2 + T3 = 250 ns
NTSO generation = 15 ns max.
$\overline{\text{RASIN}}$ to $\overline{\text{CAS}}$ delay DP8409-2 = 130 ns max.
External $\overline{\text{CASH,L}}$ generation using 74S02 and 74S240
  7.5 ns (74S02) + 10 ns (74S240) – 7.5 ns (less load on 8409 $\overline{\text{CAS}}$ line) = 10 ns max.
Transceiver delay = 12 ns max.
NS16032 data setup = 20 ns max.
∴ Minimum t_{CAC} = 63 ns
  = 250 – 15 – 130 – 10 – 12 – 20
Minimum t_{RAS} = 250 ns
Minimum t_{RP} = 250 ns
Minimum t_{RAH} = 20 ns

The DP84312 is a standard National PAL device part (PAL 16R6). The user can modify the PAL device equations to support his/her particular application. The DP84312 logic equations, function table (functional test), and logic diagram can be seen at the end of this section.

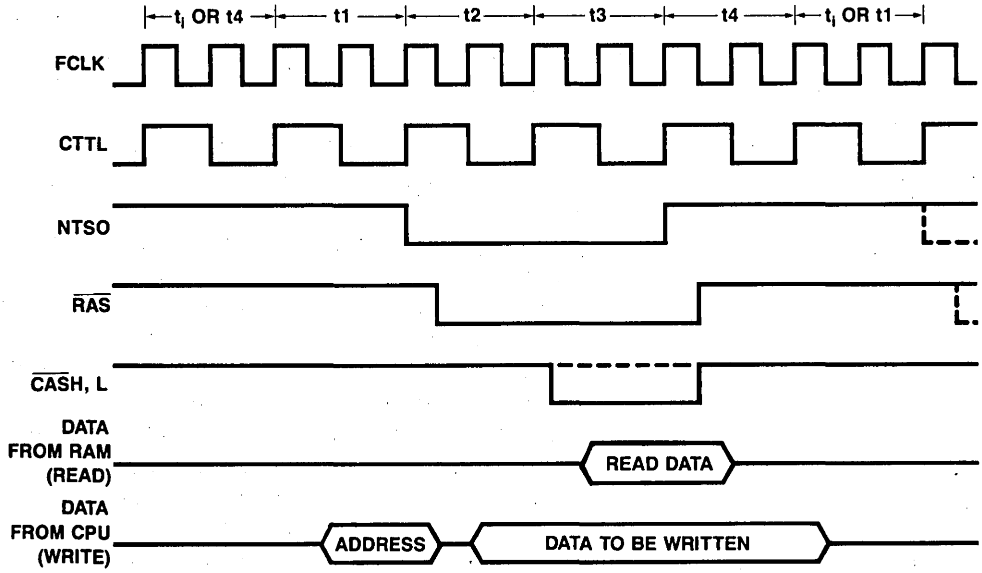


Figure 8.11.3 Timing Diagram; Read, Write or Hidden Refresh Memory Cycle for the NS32032-DP8409 Interface

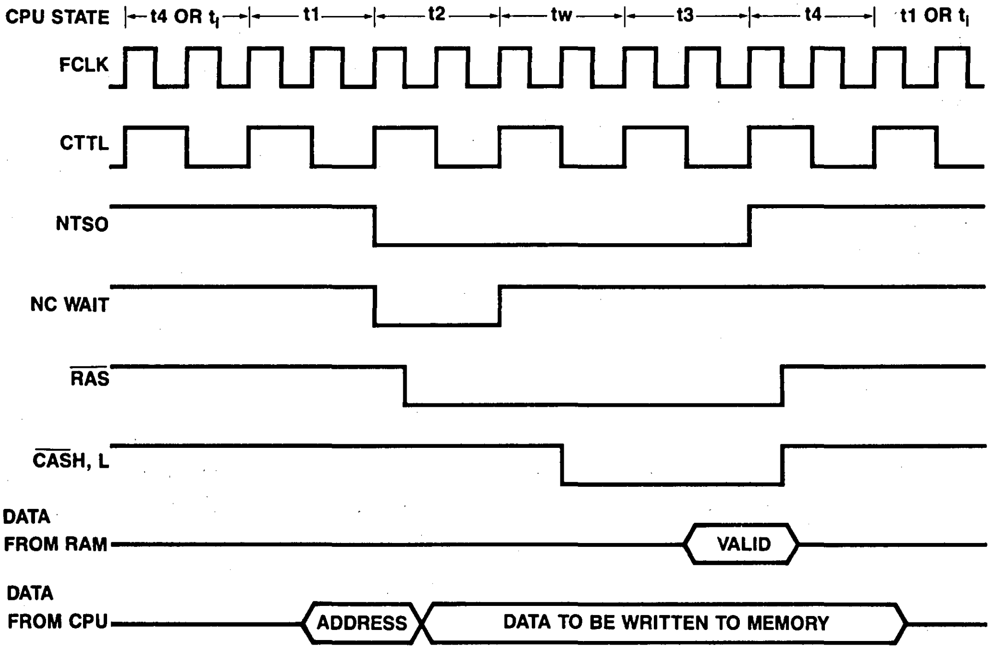


Figure 8.11.4 Timing Diagram; Read, or Write Memory Cycle With One Wait

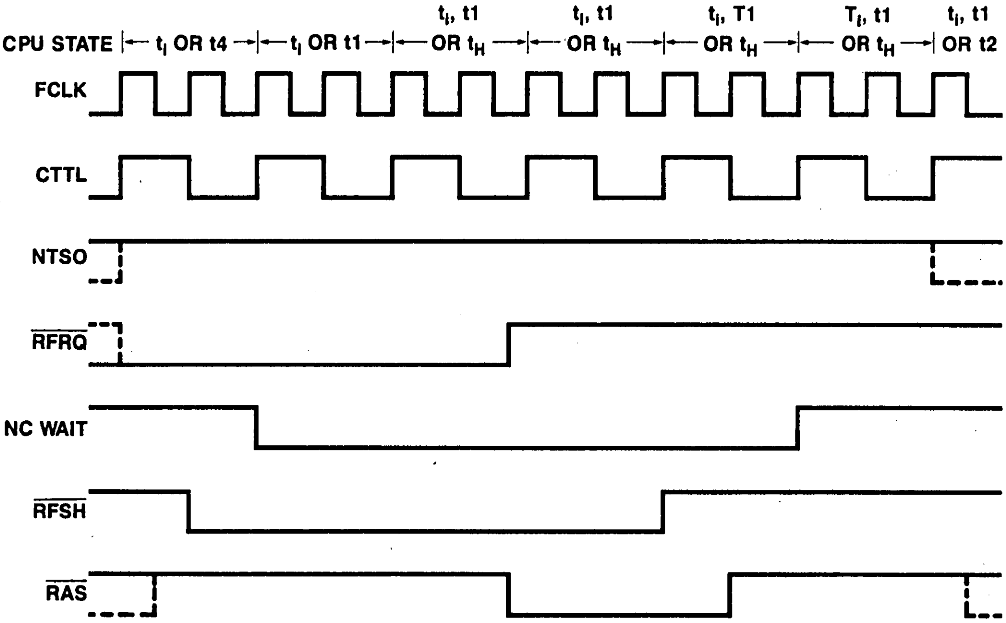


Figure 8.11.5 Timing Diagram; Forced Refresh Cycle

```
PAL16R6
DP84312
Interface Circuit for the NS32032/DP8409
Memory System
CK NTSO  /RFRQ  /HBE A0  /WAITIN CTTL  /CS  /SLOW
GND  /OE  /WAIT  /D  /C  /B  /A  /CASL  /CASH  /RFSH VCC

CASH: = A*/B*C*D* HBE*CS +
        /A*/B*D*HBE*CS

CASL : = A*/B*/C*D*/AO *CS +
         /A*/B*D*/AO*CS
A     : = /A*/B*/C*/D*/NTSO*CS*SLOW +
          B*/C*/D +
          A*/C*/D +
          A*B

B     : = /A*/B*/C*/D*NTSO*RFRQ*CTTL +
          /A*B +
          A*B*/C +
          B*C*D
```

```
C      : = /A*/B*/C*/D*NTSO*RFRQ*CTTL +
           /A*/B*D +
           A*B*D +
           B*C*/D +
           /A*/B*C*/D*/NTSO

D      : = /A*/B*/C*/D*/NTSO*CS*/SLOW +
           /A*/B*/C*/D*/NTSO*/CS +
           A*/C +
           /B*/C*D +
           /A*B*C

IF (VCC) WAIT = /B*/C*/D*/NTSO*CS*SLOW +
          /A*B*D +
          B*/C*/D +
          A*B +
          A*C*/D +
          /CS*WAITIN

IF (VCC) RFSH = /A*B +
          B*/C*/D +
          A*B*/C +
          A*B*C
```

CK	NTSO	$\overline{\text{RFRQ}}$	$\overline{\text{HBE}}$	A0	$\overline{\text{WAITIN}}$	CTTL	$\overline{\text{CS}}$	$\overline{\text{SLOW}}$	$\overline{\text{OE}}$	CASH	CASL	A	B	C	D	WAIT	RFSH
C	H	H	L	L	H	H	H	H	L	X	X	X	X	X	X	X	X
C	H	H	L	L	H	H	H	H	L	L	L	L	L	L	L	L	L
C	L	X	L	L	H	X	L	H	L	L	L	L	L	L	H	L	L
C	L	X	L	L	H	X	L	H	L	H	H	L	L	H	H	L	L
C	X	X	L	L	H	X	L	H	L	H	H	L	L	H	L	L	L
C	H	X	L	L	H	X	L	H	L	L	L	L	L	L	L	L	L
C	L	X	L	H	H	X	L	L	L	L	L	H	L	L	L	H	L
C	X	X	L	H	H	X	L	L	L	L	L	H	L	L	H	L	L
C	X	X	L	H	H	X	L	L	L	H	L	L	L	L	H	L	L
C	X	X	L	H	H	X	L	L	L	H	L	L	L	H	H	L	L
C	X	X	L	H	H	X	L	L	L	H	L	L	L	H	L	L	L
C	H	X	L	L	H	X	H	H	L	L	L	L	L	L	L	L	L
C	L	X	L	L	H	X	H	X	L	L	L	L	L	L	H	L	L
C	X	X	L	L	L	X	H	X	L	L	L	L	L	H	H	H	L
C	H	X	L	L	H	X	H	X	L	L	L	L	L	H	L	L	L
C	H	X	L	L	H	X	H	X	L	L	L	L	L	L	L	L	L
C	H	L	X	X	H	H	X	X	L	L	L	L	H	H	L	L	H
C	H	X	X	X	H	L	X	X	L	L	L	L	H	H	H	H	H
C	H	H	X	X	H	H	X	X	L	L	L	L	H	L	H	H	H
C	H	H	X	X	H	L	X	X	L	L	L	L	H	L	L	H	H
C	H	H	X	X	H	H	X	X	L	L	L	H	H	L	L	H	H
C	H	H	X	X	H	L	X	X	L	L	L	H	H	L	H	H	H
C	H	H	X	X	H	H	X	X	L	L	L	H	H	H	H	H	H
C	H	H	X	X	H	L	X	X	L	L	L	H	H	H	L	H	L
C	H	H	X	X	H	H	X	X	L	L	L	H	L	H	L	H	L
C	H	H	X	X	H	X	H	H	L	L	L	L	L	L	L	L	L
C	L	H	X	X	L	X	H	X	L	L	L	L	L	L	H	H	L
C	L	H	X	X	L	X	H	X	L	L	L	L	L	H	H	H	L
C	L	H	X	X	L	X	H	X	L	L	L	L	L	H	L	H	L
C	L	X	X	X	H	X	H	X	L	L	L	L	L	H	L	L	L
C	H	X	X	X	H	X	H	X	L	L	L	L	L	L	L	L	L
C	H	H	H	H	H	H	H	H	H	Z	Z	Z	Z	Z	Z	Z	Z

Table 8.11.4 Function Table

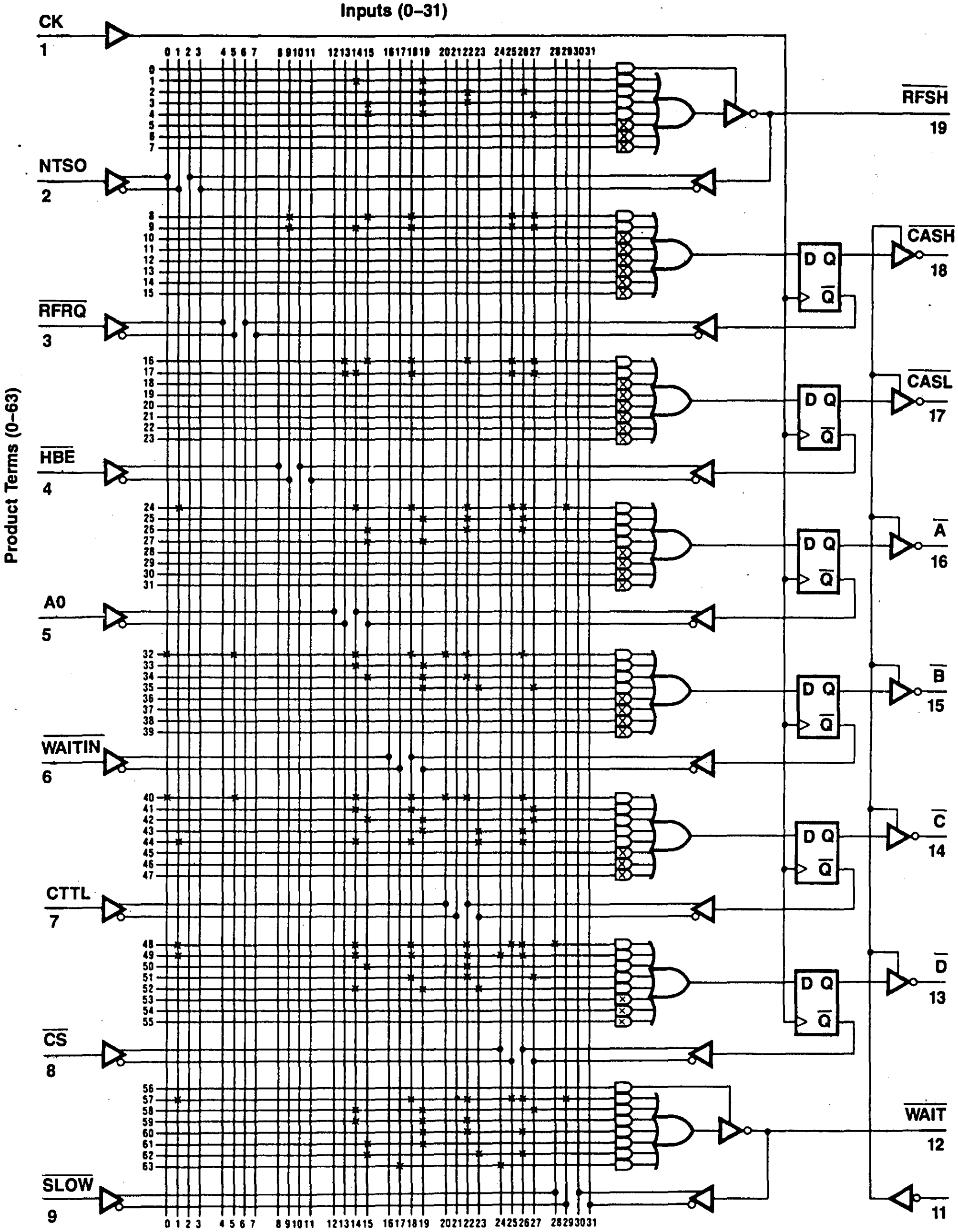


Figure 8.11.6 DP84312 Logic Diagram PAL16R6

8.12 DP84322 DYNAMIC RAM CONTROLLER INTERFACE CIRCUIT FOR THE 68000 CPU

General Description

The DP84322 dynamic RAM controller interface is a PAL device for interface between the DP8409 dynamic RAM Controller and the 68000 microprocessor.

The DP84322 supplies all the control signals needed to perform memory read, write and refresh. Logic is included for inserting a wait state when using fast CPUs.

Features

- Provides 3-chip solution for the 68000 CPU and dynamic RAM interface.
- Works with all 68000 speed versions.
- Possibility of operation at 8 MHz with no wait states.
- Performs hidden refresh.

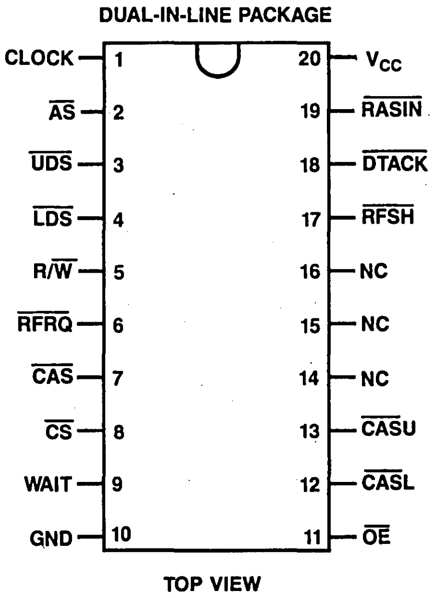


Figure 8.12.1 Connection Diagram

- $\overline{\text{DTACK}}$ is automatically inserted for both memory access and memory refresh.
- Performs forced refresh using typically 4 CPU clocks.
- Standard National Semiconductor PAL device part (PAL16R4).
- PAL device logic equations can be modified by the user for his specific application and programmed into any of National's PAL device family, including the new high speed PAL devices.

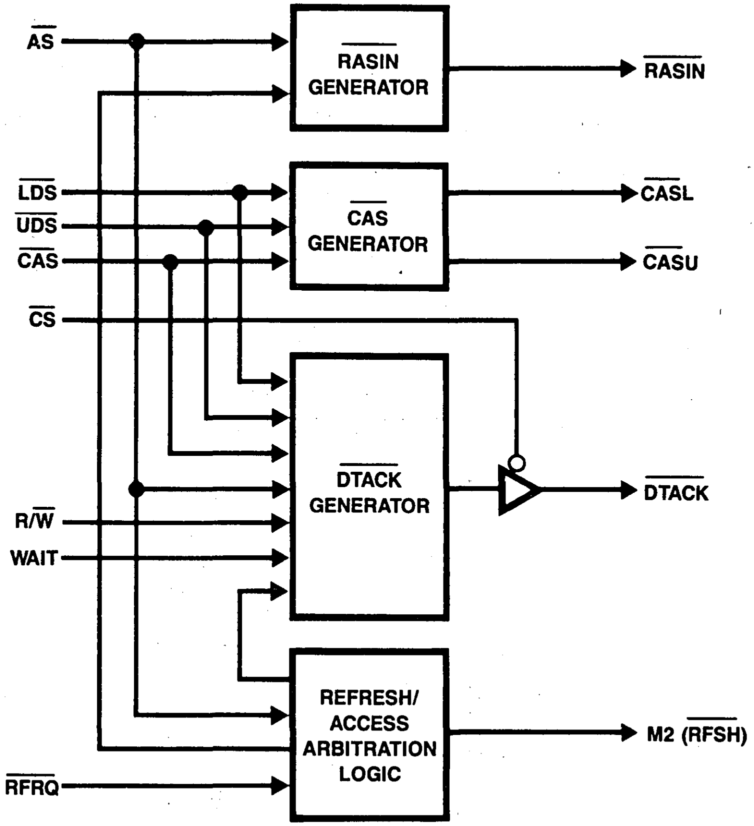

$\overline{\text{AS}}$
$\overline{\text{RASIN}}$ GENERATOR
$\overline{\text{RASIN}}$
$\overline{\text{LDS}}$
$\overline{\text{UDS}}$
$\overline{\text{CAS}}$
$\overline{\text{CAS}}$ GENERATOR
$\overline{\text{CASL}}$
$\overline{\text{CASU}}$
$\overline{\text{CS}}$
$\overline{\text{DTACK}}$ GENERATOR
$\overline{\text{DTACK}}$
R/$\overline{\text{W}}$
WAIT
REFRESH/ ACCESS ARBITRATION LOGIC
M2 ($\overline{\text{RFSH}}$)
$\overline{\text{RFRQ}}$

Figure 8.12.2 Block Diagram

Symbol	Parameter	Min	Typ	Max	Units
V_{CC}	Supply Voltage	4.75	5.00	5.25	V
I_{OH}	High Level Output Current			−3.2	mA
I_{OL}	Low Level Output Current			24 (Note 2)	mA
T_A	Operating Free Air Temperature	0		75	°C

Table 8.12.1 Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2			V
V_{IL}	Low Level Input Voltage				0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min, I_I = − 18 mA			− 1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OH} = Max	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OL} = Max			0.5	V
I_{OZH}	Off-State Output Current High Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 2.4V, V_{IL} = 0.8V			100	μA
I_{OZL}	Off-State Output Current Low Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 0.4V, V_{IL} = 0.8V			− 100	μA
I_I	Input Current at Maximum Input Voltage	V_{CC} = Max, V_I = 5.5V			1.0	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.4V			25	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V			− 250	μA
I_{OS}	Short Circuit Output Current	V_{CC} = Max	− 30		− 130	mA
I_{CC}	Supply Current	V_{CC} = Max		150	225 (Note 1)	mA

Table 8.12.2 Electrical Characteristics

Symbol	Parameter		Test Conditions R_L = 667Ω	Commercial T_A = 0°C to + 75°C V_{CC} = 5.0V ± 5%			Units
				Min	Typ	Max	
t_{PD}	Input to Output		C_L = 45 pF		25	40	ns
t_{PD}	Clock to Output				15	25	ns
t_{PZX}	Pin 11 to Output Enable				15	25	ns
t_{PXZ}	Pin 11 to Output Disable		C_L = 5 pF		15	25	ns
t_{PZX}	Input to Output Enable		C_L = 45 pF		25	40	ns
t_{PXZ}	Input to Output Disable		C_L = 5 pF		25	40	ns
t_w	Width of Clock	High		25			ns
		Low		25			ns
t_{su}	Set-Up Time			40			ns
t_h	Hold Time			0	− 15		ns

Note 1: I_{CC} = max at minimum temperature.

Note 2: One output at a time; otherwise 16 mA.

Table 8.12.3 Switching Characteristics

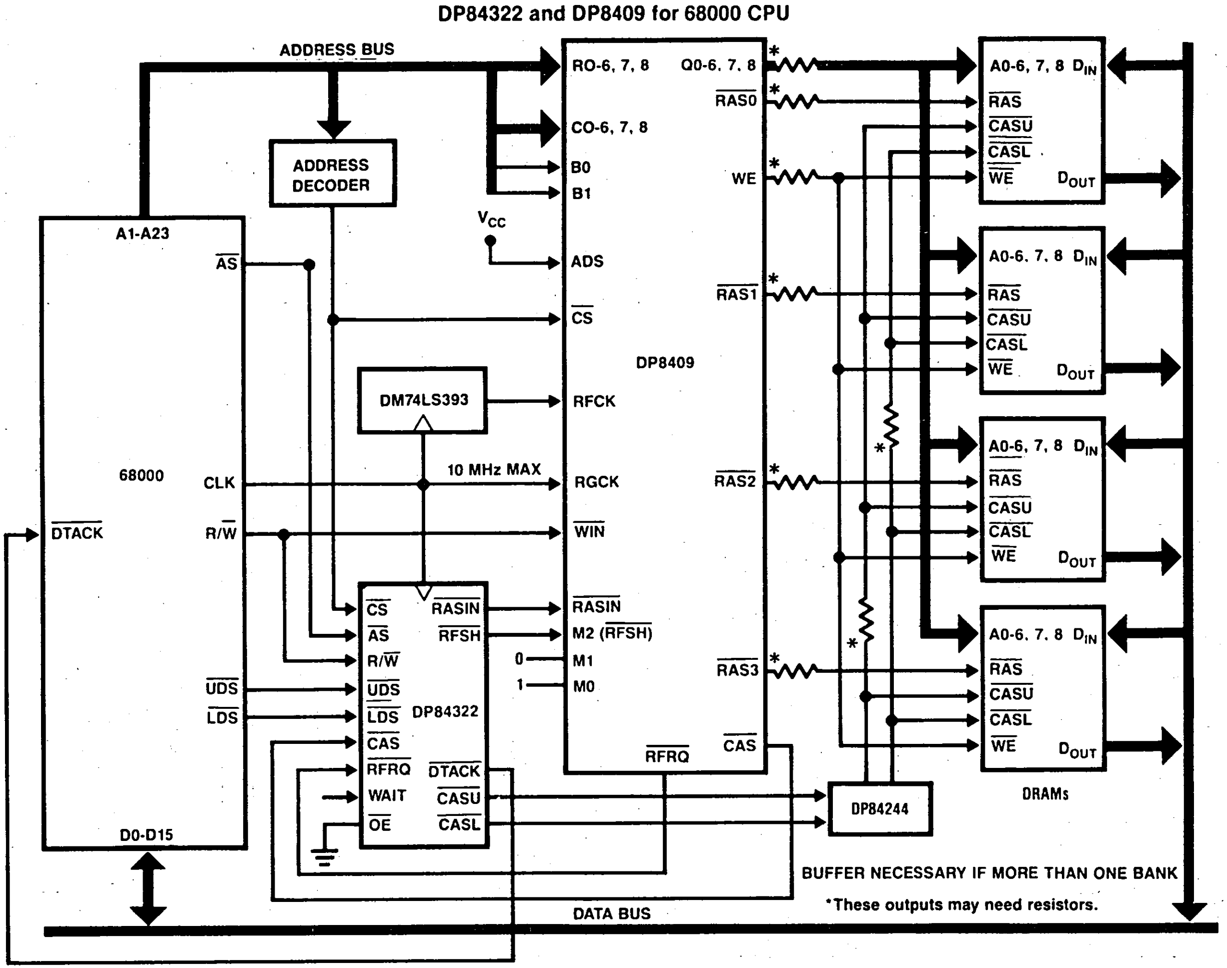


Figure 8.12.3 System Block Diagram

Mnemonic Description

Input Signals

CLOCK The clock signal determines the timing of the outputs and should be connected directly to the 68000 clock input.

$\overline{\text{AS}}$ Address Strobe from the 68000 CPU. This input is used to generate $\overline{\text{RASIN}}$ to the DP8409.

$\overline{\text{UDS}}$, $\overline{\text{LDS}}$ Upper and lower data strobe from the 68000 CPU. These inputs, together with $\overline{\text{AS}}$, R/$\overline{\text{W}}$, provide $\overline{\text{DTACK}}$ to the 68000.

R/$\overline{\text{W}}$ Read/write from the 68000 CPU, when WAIT = 0. Selects processor speed when WAIT = 1 ("1" = 4, to 6 MHz, "0" = 8 MHz).

$\overline{\text{CAS}}$ Column Address Strobe from the DP8409. This input, together with $\overline{\text{LDS}}$ and $\overline{\text{UDS}}$, provides two separate $\overline{\text{CAS}}$ outputs for accessing upper and lower memory data bytes.

$\overline{\text{CS}}$ Chip Select. This input enables $\overline{\text{DTACK}}$ output. CS = 0, $\overline{\text{DTACK}}$ output is enabled; $\overline{\text{CS}}$ = 1, $\overline{\text{DTACK}}$ output is TRI-STATE®.

$\overline{\text{RFRQ}}$ Refresh Request. This input requests the DP84322 for a forced refresh.

WAIT This input allows the necessary wait state to be inserted for memory access cycles.

Output Signals

$\overline{\text{RASIN}}$ This output provides a memory cycle start signal to the DP8409 and provides $\overline{\text{RAS}}$ timing during hidden refresh.

$\overline{\text{CASU}}$, $\overline{\text{CASL}}$ These signals are the separate $\overline{\text{CAS}}$ outputs needed for byte writing.

$\overline{\text{DTACK}}$ This output is used to insert wait states into the 68000 memory cycles when selected and during a forced refresh cycle where the CPU attempts to access the memory. This output is enabled when $\overline{\text{CS}}$ input is low and TRI-STATE when $\overline{\text{CS}}$ is high.

$\overline{\text{RFSH}}$ This output controls the mode of the DP8409. It always goes low for 4 CPU clock periods when $\overline{\text{AS}}$ is inactive and a forced refresh is requested through $\overline{\text{RFRQ}}$ input. This allows the DP8409 to perform an automatic forced refresh.

Functional Description

As a 68000 bus cycle begins, a valid address is output on the address bus A1-A23. This address is decoded to provide Chip Select ($\overline{\text{CS}}$) to the DP8409. After the address becomes valid, $\overline{\text{AS}}$ goes low and it is used to set $\overline{\text{RASIN}}$ low from the DP84322 interface circuit. Note that $\overline{\mathrm{CS}}$ must go low for a minimum of 10 ns before the assertion of $\overline{\mathrm{RASIN}}$ for a proper memory access. As an example, with an 8 MHz 68000, the address is valid for at least 30 ns before $\overline{\mathrm{AS}}$ goes active. $\overline{\mathrm{AS}}$ then has to ripple through the DP84322 to produce $\overline{\mathrm{RASIN}}$. This means the address is valid for a minimum of 40 ns before $\overline{\mathrm{RASIN}}$ goes low, and the decoding of $\overline{\mathrm{CS}}$ should take less than 30 ns. At this speed the DM74LS138 or DM74LS139 decoders can be selected to guarantee the 10 ns minimum required by $\overline{\mathrm{CS}}$ set-up time going low before the access $\overline{\mathrm{RASIN}}$ goes low(t_{CSRL} of the DP8409). This is important because a false hidden refresh may take place when the minimum t_{CSRL} is not met.

Typically $\overline{\mathrm{RASIN}}$ occurs at the end of S2. Subsequently, selected $\overline{\mathrm{RAS}}$ output, row to column select and then $\overline{\mathrm{CAS}}$ will automatically follow $\overline{\mathrm{RASIN}}$ as determined by mode 5 of the DP8409. Mode 5 guarantees a 30 ns minimum for row address hold time (t_{RAH}) and a minimum of 8 ns column address set-up time (t_{ASC}). If the system requires instructions that use byte writing, then $\overline{\mathrm{CASU}}$ and $\overline{\mathrm{CASL}}$ are needed for accessing upper and lower memory data bytes, and they are provided by the DP84322. In the DP84322, $\overline{\mathrm{LDS}}$ and $\overline{\mathrm{UDS}}$ are gated with $\overline{\mathrm{CAS}}$ from the DP8409 to provide $\overline{\mathrm{CASL}}$ and $\overline{\mathrm{CASU}}$. Therefore, designers need not be concerned about delaying $\overline{\mathrm{CAS}}$ during write cycles to assure valid data being written into memory. The 8 MHz 68000 specifies during a write cycle that data output is valid for a minimum of 30 ns before $\overline{\mathrm{DS}}$ goes active. Thus, $\overline{\mathrm{CASL}}$ and $\overline{\mathrm{CASU}}$ will not go low for at least 40 ns after the output data becomes stable, guaranteeing the 68000 valid data is written to memory.

Furthermore, the gating of $\overline{\mathrm{UDS}}$, $\overline{\mathrm{LDS}}$ and $\overline{\mathrm{CAS}}$ allows the DP84322 interface controller to support the test and set instruction (TAS). The 68000 utilizes the read-modify-write cycle to execute this instruction. The TAS instruction provides a method of communication between processors in a multiple processor system. Because of the nature of this instruction, in the 68000 this cycle is indivisible and the Address Strobe $\overline{\mathrm{AS}}$ is asserted through the entire cycle. However, $\overline{\mathrm{DS}}$ is asserted twice for two accesses: a read then a write. The dynamic RAM controller and the DP84322 respond to this read-modify-write instruction as follows (refer to the TAS instruction timing diagram for clarification). First, the selected $\overline{\mathrm{RAS}}$ goes low as a result of $\overline{\mathrm{AS}}$ going low, and this $\overline{\mathrm{RAS}}$ output will remain low throughout the entire cycle. Then the DP84322's selected $\overline{\mathrm{CAS}}$ output ($\overline{\mathrm{CASL}}$ or $\overline{\mathrm{CASU}}$) goes low to read the specified data byte. After this read, $\overline{\mathrm{DS}}$ goes high causing the selected $\overline{\mathrm{CAS}}$ to go high. A few clocks later R/$\overline{\mathrm{W}}$ goes low and then $\overline{\mathrm{DS}}$ is reasserted. As $\overline{\mathrm{DS}}$ goes low, the selected $\overline{\mathrm{CAS}}$ goes low strobing the CPU's modified data into memory, after which the cycle is ended when $\overline{\mathrm{AS}}$ goes high.

The two $\overline{\mathrm{CAS}}$ outputs from the DP84322 however, can only drive one memory bank. For additional driving capability, a memory driver such as the DP84244 should be added to drive loads of up to 500 pF.

Since this DP84322 interface circuit is designed to operate with all of the 68000 speed versions, a status input called WAIT is used to distinguish the 8 MHz from the others. The WAIT input should be set low for a 6 MHz or less allowing full speed of operation with no wait states. Data Transfer Acknowledge input ($\overline{\mathrm{DTACK}}$) of the 68000 at these speeds is automatically inserted during S2 for every memory transaction cycle and is then negated at the end of that cycle when $\overline{UDS}$ and/or $\overline{LDS}$ go high. For the 8 MHz 68000 however, a wait state is required for every memory transaction cycle. At these speeds, the WAIT input is set high, selecting the DP8409's $\overline{CAS}$ output to generate $\overline{DTACK}$ and again $\overline{DTACK}$ is negated at the end of the cycle when $\overline{UDS}$ or $\overline{LDS}$ goes high. Note that $\overline{DTACK}$ output is enabled only when the DP8409's $\overline{CS}$ is low. Therefore when the 68000 is accessing I/O or ROM (in other words, when the DP8409 is not selected), the DP84322's $\overline{DTACK}$ output goes high impedance logic '1' through the external pull-up resistor and it is now up to the designer to supply $\overline{DTACK}$ for a proper bus cycle.

Table 8.12.4 indicates the maximum memory speed in terms of the DRAM timing parameters: t_{CAC} (access-time from $\overline{CAS}$) and t_{RP} ($\overline{RAS}$ precharge time) required by different 68000 speed versions.

Microprocessor Clock	Maximum t_{CAC}	Minimum t_{RP}	Minimum t_{RAS}
8 MHz	125 ns	140 ns	220 ns
6 MHz	90 ns	170 ns	290 ns
4 MHz	270 ns	280 ns	450 ns

Table 8.12.4 Memory Speed

Pin 5 (R/$\overline{W}$ input to the DP84322) is not used as R/$\overline{W}$ when the WAIT input is high. Therefore, when WAIT is high and pin 5 is low, this is configured for the 8 MHz 68000. The dynamic RAM controller in this configuration operates in mode 5 and mode 1.

When both WAIT and pin 5 are high, this is configured for 4 MHz and 6 MHz 68000, allowing only two microprocessor clocks for memory refresh. Furthermore, the designer can use the DP8408 because the dynamic RAM controller now operates in mode 0 and mode 5 or mode 6. In addition, the programmable refresh timer, DP84300, should be used to determine the refresh rate (RFCK) and to provide the refresh request ($\overline{RFRQ}$) input to the DP84322. The refresh timer can provide over two hundred different divisors. $\overline{RFRQ}$ is given at the beginning of every RFCK cycle and remains active until M2 goes low for memory refresh. The DP84322 samples $\overline{RFRQ}$ when $\overline{AS}$ is high, then sets M2 low for two microprocessor clocks, taking the DP8408 or DP8409 to the external control refresh mode. $\overline{RASIN}$ for this refresh is also issued by the DP84322. If a memory access is pending, $\overline{RASIN}$ for this access will not be given until it is delayed for approximately one microprocessor clock, allowing $\overline{RAS}$ precharge time for the dynamic RAMs.

The following table indicates different memory speeds in terms of the DRAM parameters required by 4 MHz and 6 MHz 68000:

Microprocessor Clock	Maximum t_{CAC}	Minimum t_{RAS}	Minimum t_{RP}	Minimum t_{RAH}
4 MHz	290 ns	200 ns	225 ns	20 ns
6 MHz	110 ns	125 ns	140 ns	20 ns

Table 8.12.5 Memory Speed of 68000

When WAIT = 1, pin 5 = 0 (8 MHz), the PAL device controller supports read and write cycles with one inserted wait state, forced refresh with five wait states inserted if $\overline{\text{CS}}$ is valid, and hidden refresh. This PAL device mode does not support the TAS instruction.

When WAIT = pin 5 = 1 (4-6 MHz), the PAL device controller supports read and write cycles with no wait states inserted, and forced refresh with two wait states inserted if $\overline{\text{CS}}$ is valid. This PAL device mode does not support the TAS instruction and only supports hidden refresh when used in mode 5 with the DP8409 controller.

The DP84322 can possibly be operated at 8 MHz with no wait states (WAIT = "0") given the following conditions:

FAST PAL DEVICE (PAL 16R4A)
S2 + S3 + S4 + S5 = 250 ns
$\overline{\text{RASIN}}$ delay = 60 ns ($\overline{\text{AS}}$ low max.)
+ 25 ns (Fast PAL delay) = 85 ns max.
$\overline{\text{RASIN}}$ to $\overline{\text{CAS}}$ delay DP8409-2 = 130 ns max.
External $\overline{\text{CASH,L}}$ generation using 74S02
and 74S240
7.5 ns (74S02) + 10 ns (74S240) – 7.5 ns (less load
on 8409 $\overline{\text{CAS}}$ line) = 10 ns max.
Transceiver delay (74LS245) = 12 ns max.
68000 data setup into S6 = 40 ns min.
$\therefore$ Minimum t_{CAC} = 53 ns
= 250 – 85 – 130 – 10 – 12 + 40
Minimum t_{RAS} = 240 ns
Minimum t_{RP} = 150 ns
Minimum t_{RAH} = 20 ns

Refresh Cycle

Since the access sequence timing is automatically derived from $\overline{\text{RASIN}}$ in mode 5, R/C and $\overline{\text{CASIN}}$ are not used and now become Refresh Clock (RFCK) and $\overline{\text{RAS}}$-generator clock (RGCK) respectively. The Refresh Clock RFCK may be divided down from RGCK, which is the micropocessor clock, using the DM74LS393 or DM74LS390. RFCK provides the refresh time interval and RGCK the fast clock for all-$\overline{\text{RAS}}$ refresh if forced refreshing is necessary. The DP8409 offers both hidden refresh in mode 5 and forced refresh in mode 1 with priority placed on hidden refreshing. Assume 128 rows are being refreshed, then a 16μs maximum clock period is needed for RFCK to distribute refreshing of all the rows over the 2 ms period.

The DP8409 provides hidden refreshing in mode 5 when the refresh clock (RFCK) is high and the microprocessor is accessing RAM. In other words, when the DP8409's chip select is inactive because the microprocessor is not accessing elsewhere, all four $\overline{\text{RAS}}$ outputs follow $\overline{\text{RASIN}}$, strobing the contents of the on-chip refresh counter to every memory bank. $\overline{\text{RASIN}}$ going high terminates the hidden refresh and also increments the refresh counter, preparing it for the next refresh cycle. Once a hidden refresh has taken place, a forced refresh will not be requested by the DP8409 for the current RFCK cycle.

However, if the microprocessor continuously accessed the DP8409 and memory while RFCK was high, a hidden refresh could not have taken place and now the system must force a refresh. Immediately after RFCK goes low, the Refresh Request signal ($\overline{\text{RFRQ}}$) from the DP8409 goes low, indicating a forced refresh is necessary. First, when $\overline{\text{RFRQ}}$ goes low any time during S2 to S7, the controller interface circuit waits until the end of the current memory access cycle and then sets M2 ($\overline{\text{RFSH}}$) low. This refresh takes four microprocessors clocks to complete. If the current cycle is another memory cycle, the 68000 will automatically be put in four wait states.

Alternately, when $\overline{\text{RFRQ}}$ goes low while $\overline{\text{AS}}$ is high during S0 to S1, M2 is now set low at S2. Therefore, it requires an additional microprocessor clock for this refresh. Once the DP8409 is in mode 1 forced refresh, all the $\overline{\text{RAS}}$ outputs remain high until two RGCK trailing edges after M2 goes low, when all $\overline{\text{RAS}}$ outputs go low. This allows a minimum of one and a half clock periods of RGCK for $\overline{\text{RAS}}$ precharge time. As specified in the DP8409 data sheet, the $\overline{\text{RAS}}$ outputs remain low for two clock periods of RGCK. The refresh counter is incremented as the $\overline{\text{RAS}}$ outputs go high. Once the forced refresh has ended, M2 is brought high, the DP8409 back to mode 5 auto access. Note that $\overline{\text{RASIN}}$ for the pending access is not given until it has been delayed for a full microprocessor clock, allowing $\overline{\text{RAS}}$ precharge time for the coming access.

If the 68000 bus is inactive (i.e., the 68000's instruction queue is full, or the 68000 is executing internal operations such as a multiply instruction, or the 68000 is in half state . . .) and a refresh has been requested, a refresh will also take place because $\overline{\text{RFRQ}}$ is continuously sampled while $\overline{\text{AS}}$ is high. Therefore, refreshing under these conditions will be transparent to the microprocessor. Consequently, the system throughput is increased because the DP84322 allows refreshing while the 68000 bus is inactive.

The 84322 is a standard National PAL device part (PAL16R4). The user can modify the PAL equations to support his particular application. The 84322 logic equations, function table, and logic diagram can be seen at the end of this section.

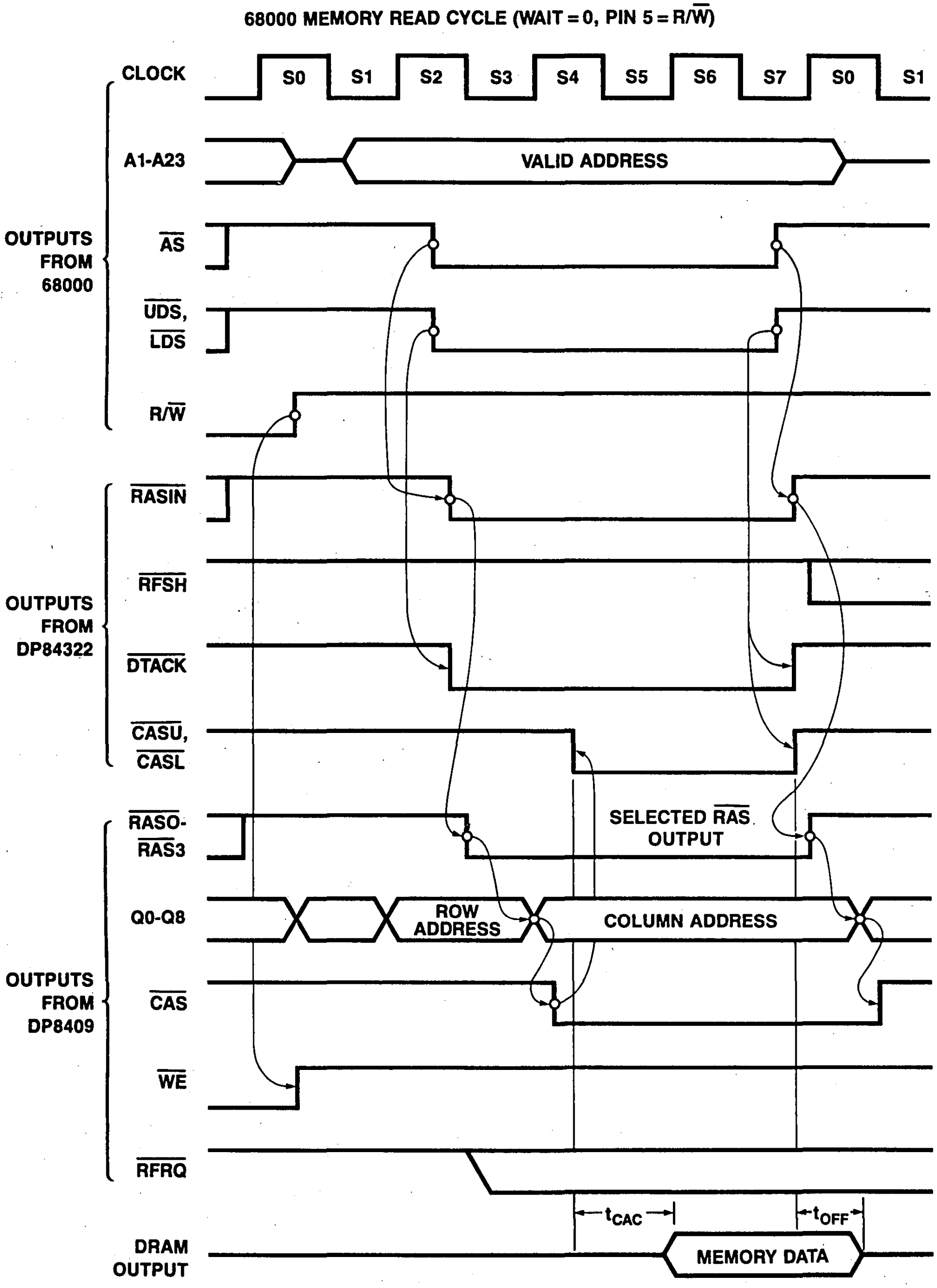


Figure 8.12.4 Timing Diagram; 68000 Memory Read Cycle

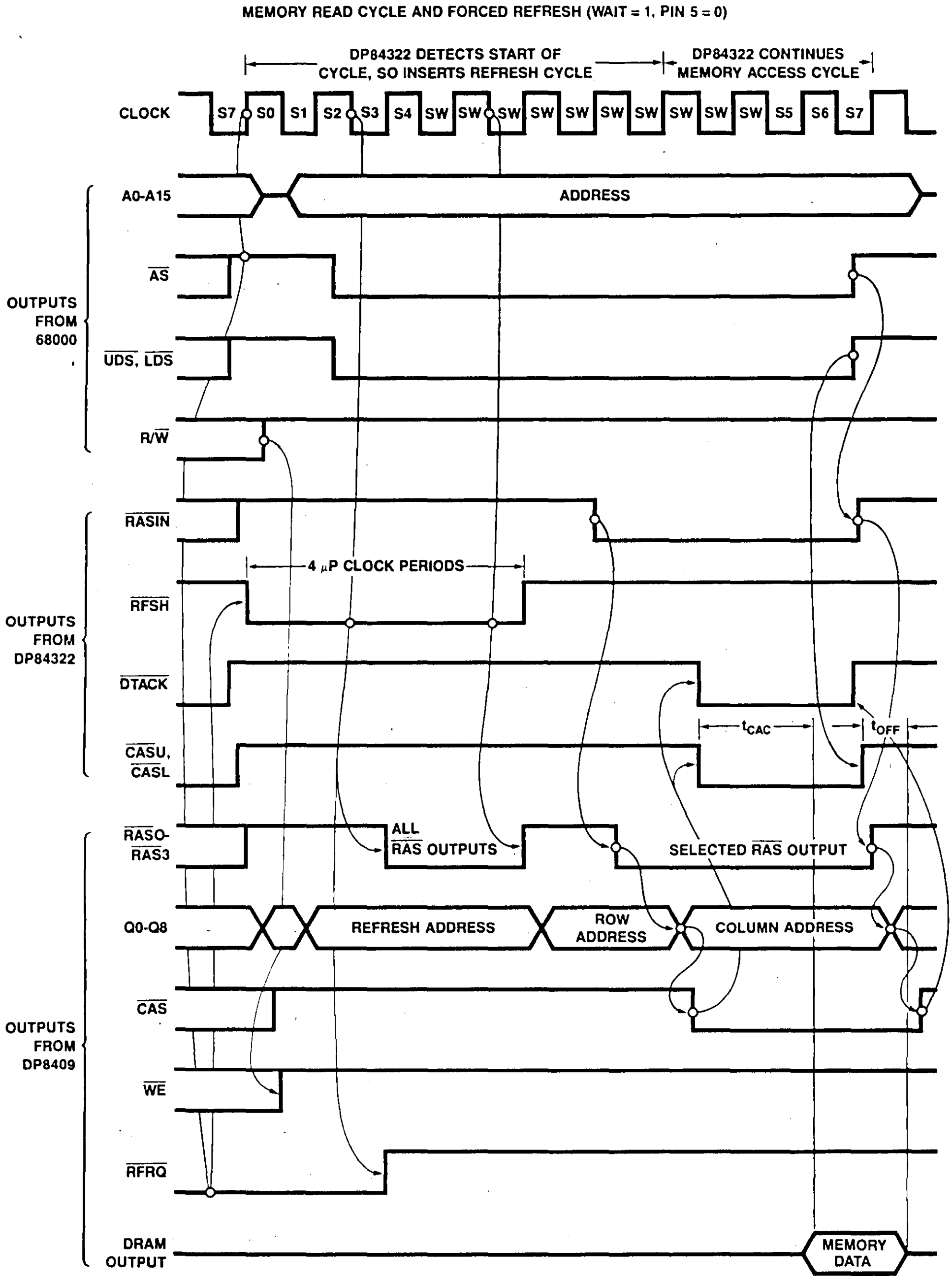


Figure 8.12.5 Timing Diagram; 68000 Memory Read Cycle and Forced Refresh

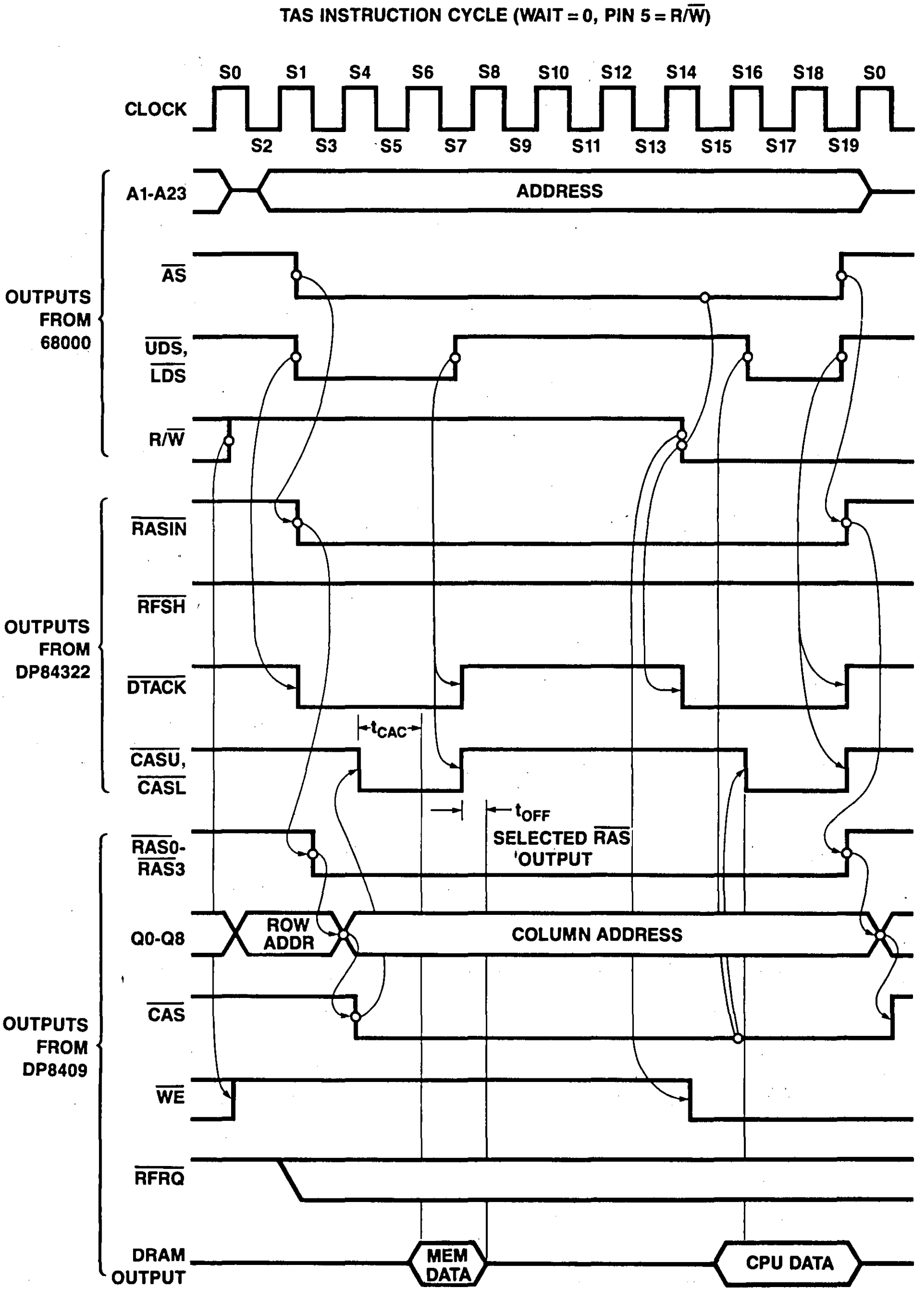


Figure 8.12.6 Timing Diagram; TAS Instruction Cycle

MEMORY READ CYCLE (WAIT = 1, PIN 5 = 0)

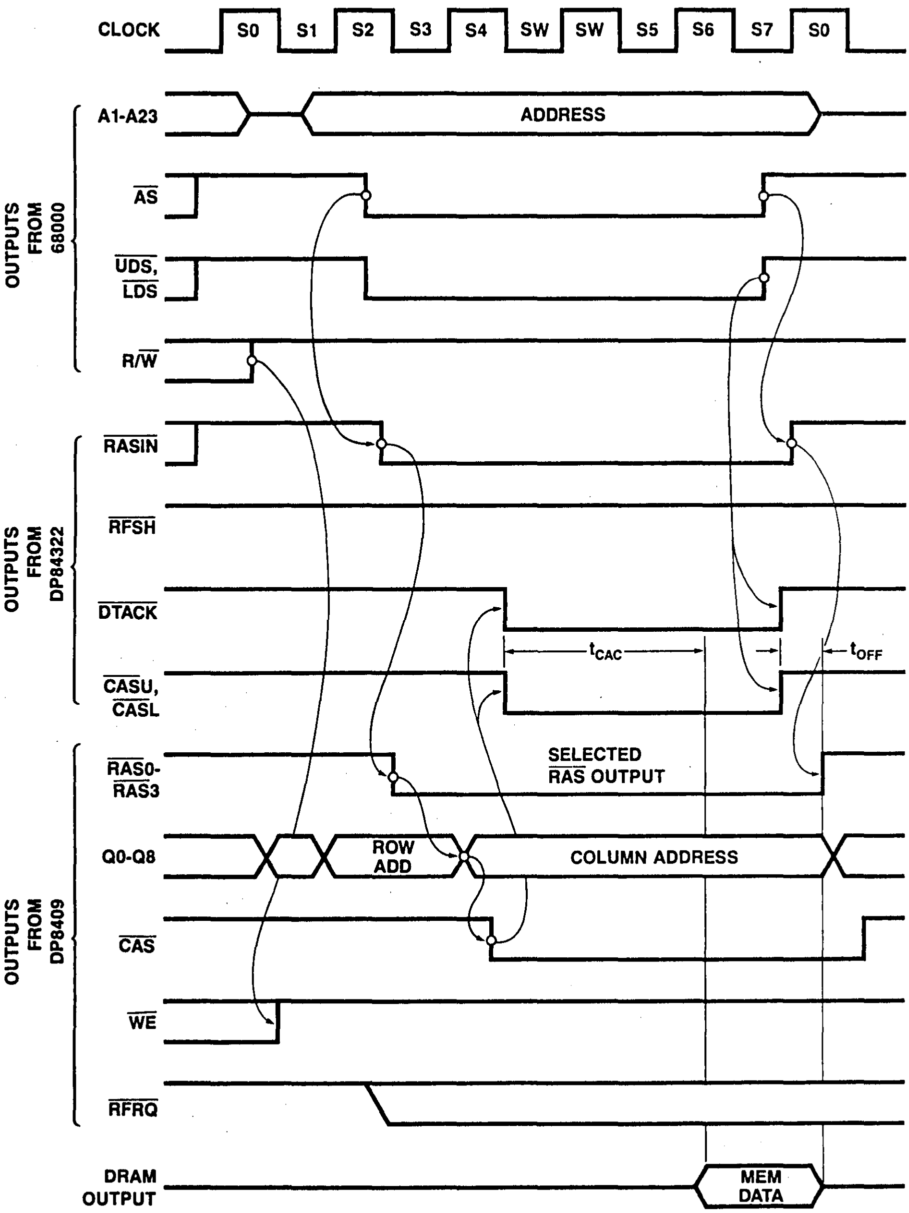


Figure 8.12.7 Timing Diagram; Memory Read Cycle

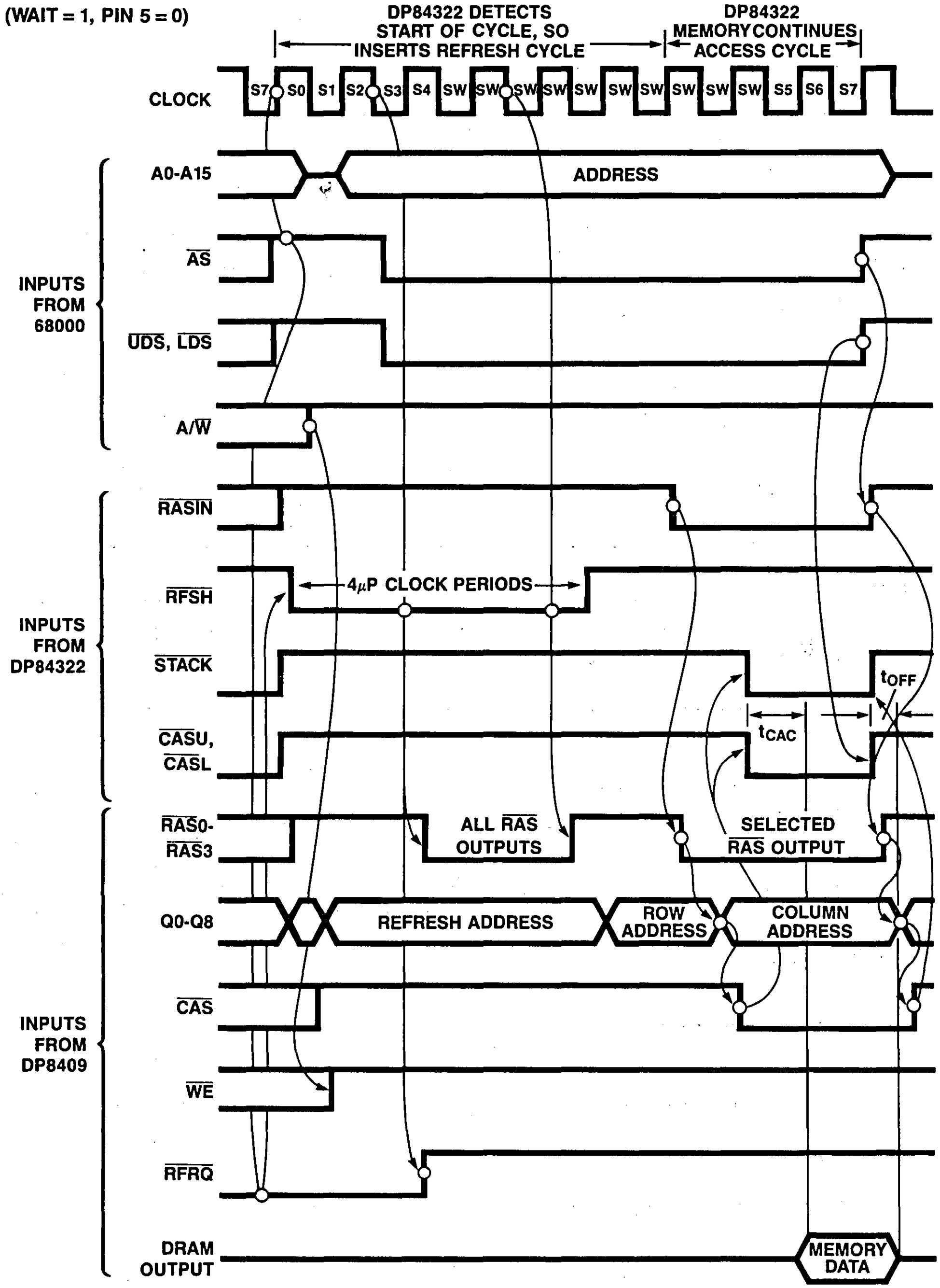
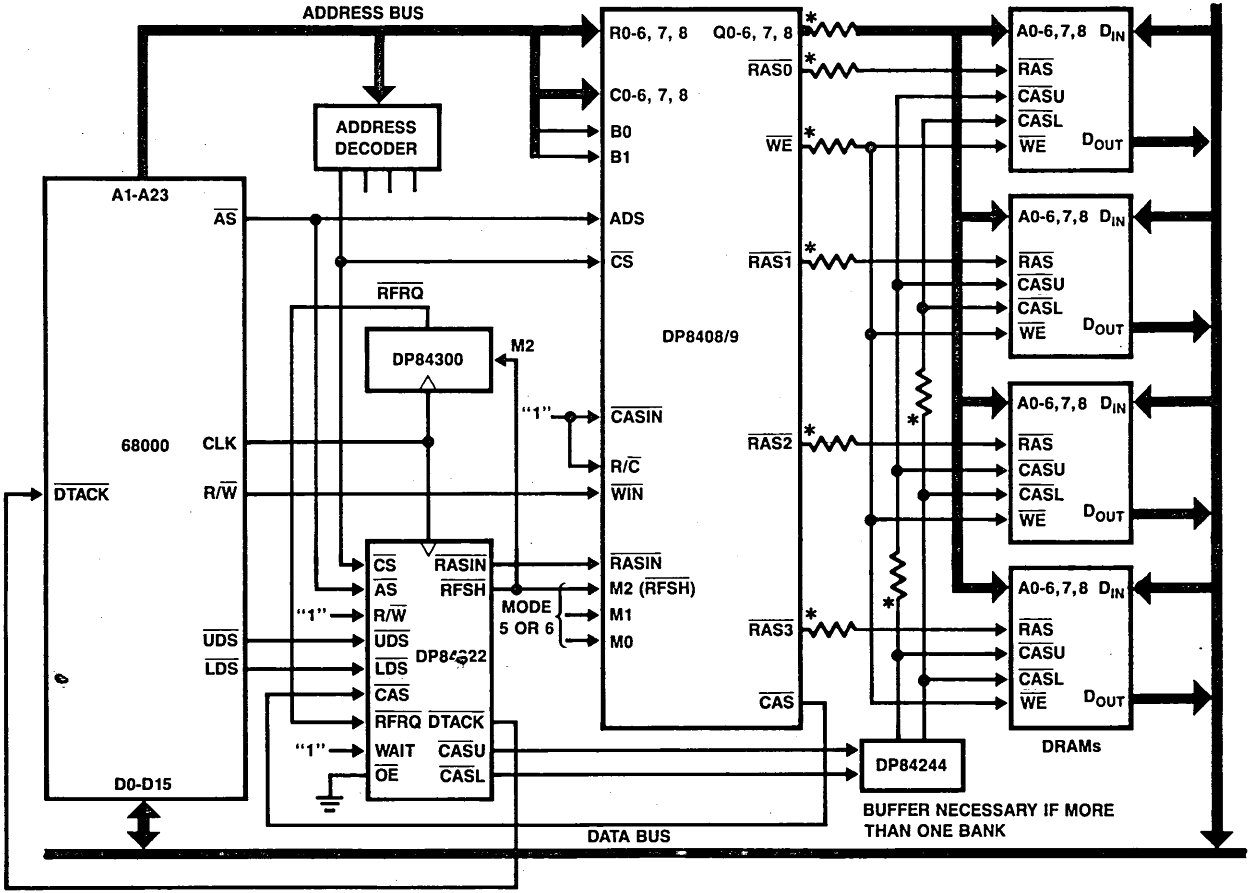


Figure 8.12.8 Timing Diagram; Memory Read Cycle and Forced Refresh

DP8408, DP8409 AND 68000 INTERFACE



*These outputs may need resistors.

Figure 8.12.9 Modified System Block Diagram

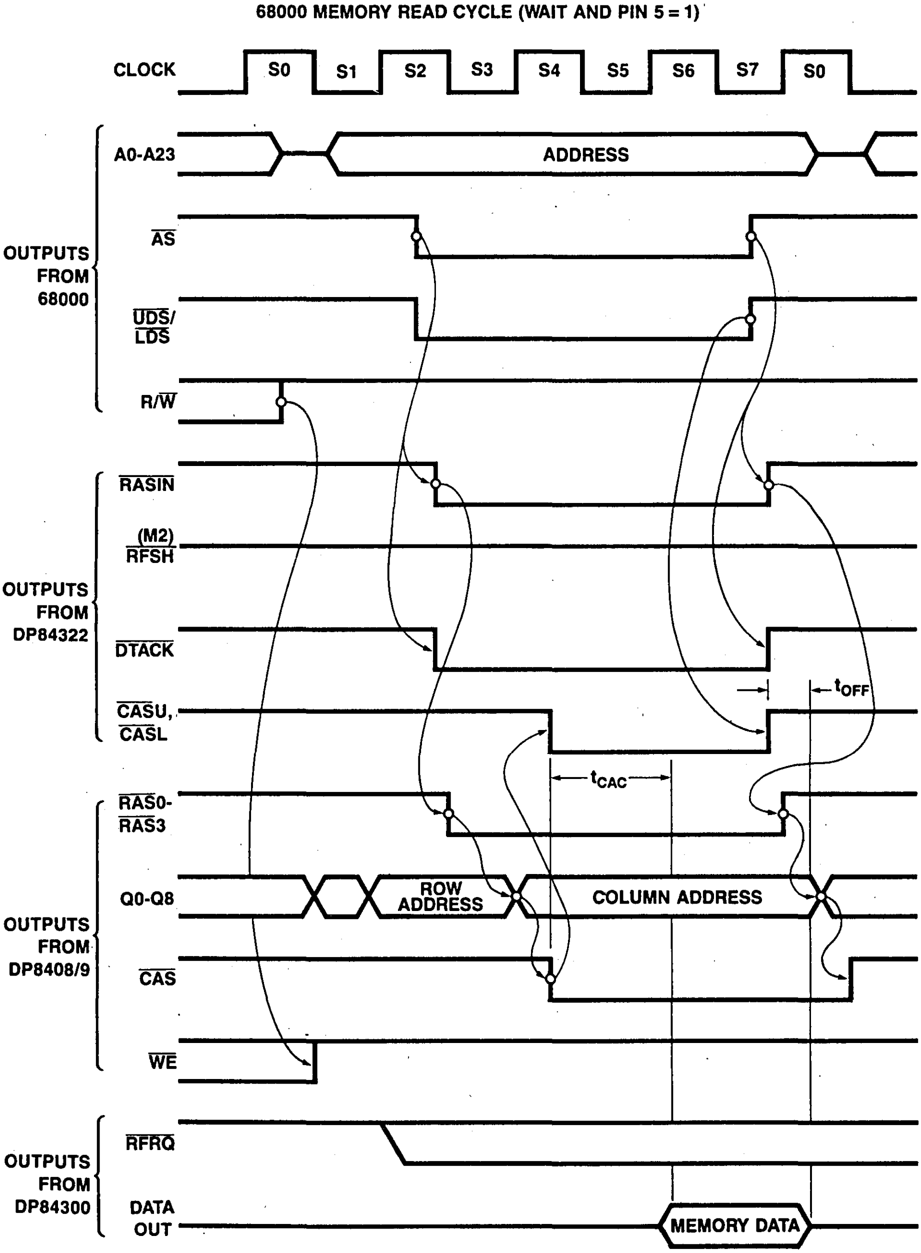


Figure 8.12.10 Timing Diagram; 68000 Memory Read Cycle

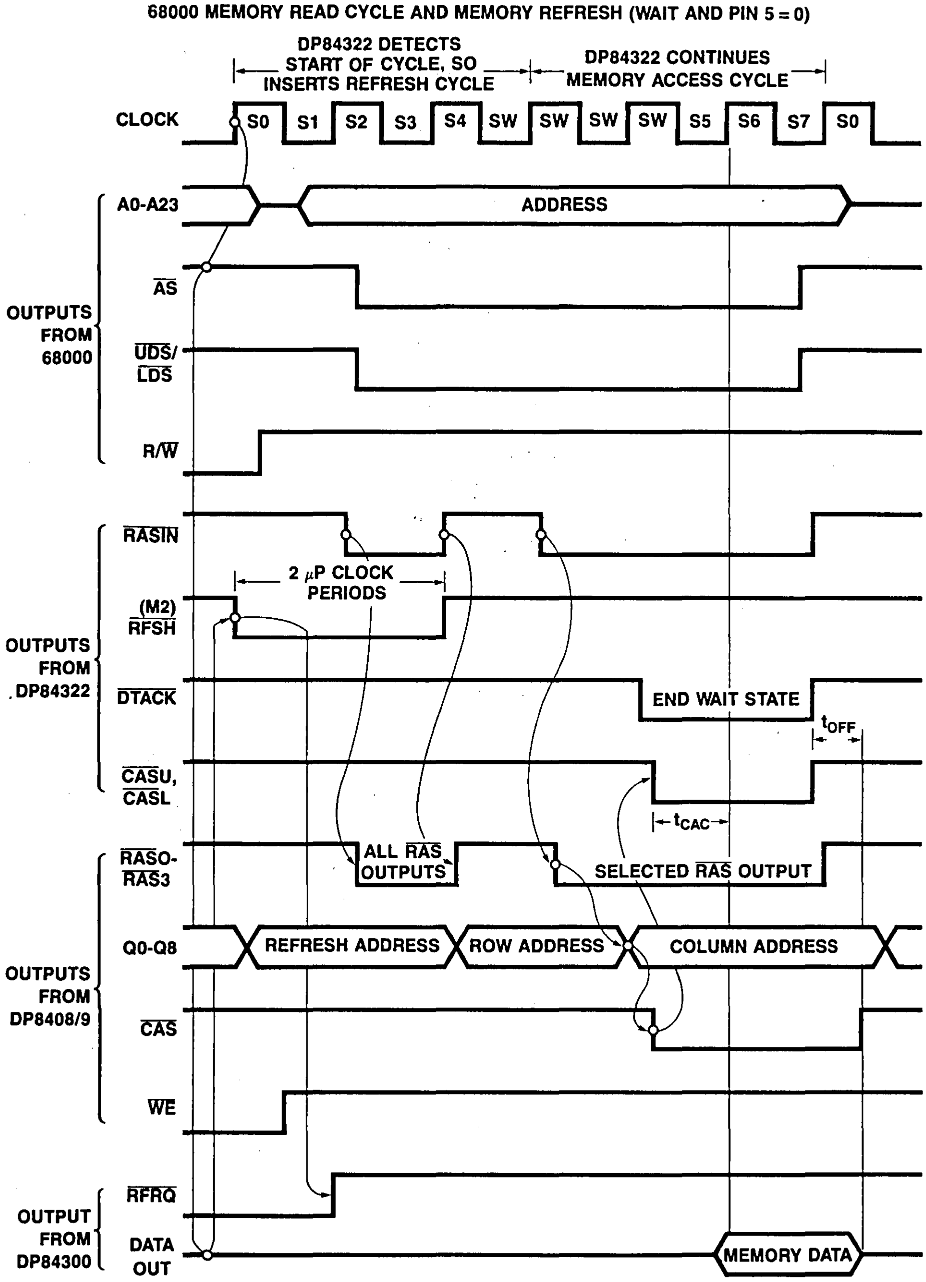


Figure 8.12.11 Timing Diagram; 68000 Memory Read Cycle and Memory Refresh

```
PAL 16R4
DP84322
Dynamic RAM Controller Interface for the
MC68000-DP8409 Memory System
CK /AS /UDS /LDS R /RFRQ /CAS /CS WAIT GND
/OE /CL /CU /C /B /A /RFSH /DTACK /RASIN VCC

 IF (VCC) RASIN = AS*RFSH*/A +
                  RFSH*R*A*WAIT
  IF (CS) DTACK = /R*CAS*WAIT +
                  UDS*/A*/B*/WAIT +
                  LDS*/A*/B*/WAIT +
                  AS*/R*/A*/B*WAIT +
                  AS*/RFSH*R*/A*/B*WAIT

          RFSH: = /AS*RFRQ +
                  RFSH*/R*/C*WAIT +
                  RFSH*R*/A*WAIT +
                  RFSH*/C*/WAIT

             A: = RFSH
             B: = A
             C: = B
    IF (VCC) CU = UDS*CDS
    IF (VCC) CL = LDS*CAS
```

CK	$\overline{\text{AS}}$	$\overline{\text{UDS}}$	$\overline{\text{LDS}}$	R	$\overline{\text{RFRQ}}$	$\overline{\text{CAS}}$	$\overline{\text{CS}}$	WAIT	$\overline{\text{OE}}$	$\overline{\text{CL}}$	$\overline{\text{CU}}$	$\overline{\text{C}}$	$\overline{\text{B}}$	$\overline{\text{A}}$	$\overline{\text{RFSH}}$	$\overline{\text{DTACK}}$	$\overline{\text{RASIN}}$
C	H	L	L	H	H	H	H	L	L	H	H	X	X	X	X	X	H
C	H	L	L	H	H	L	H	L	L	L	L	X	X	X	X	X	H
C	H	L	H	H	H	L	H	L	L	H	L	X	X	X	X	X	H
C	H	H	L	H	H	L	H	L	L	L	H	X	X	X	X	X	H
C	H	H	H	H	H	H	H	L	L	H	H	H	H	H	H	Z	H
C	L	L	H	H	H	H	L	L	L	H	H	H	H	H	H	L	L
C	L	L	H	H	H	L	L	L	L	H	L	H	H	H	H	L	L
C	L	H	H	H	H	L	L	L	L	H	H	H	H	H	H	H	L
C	L	H	H	L	H	L	L	L	L	H	H	H	H	H	H	L	L
C	L	L	H	L	H	L	L	L	L	H	L	H	H	H	H	L	L
C	H	H	H	L	H	H	L	L	L	H	H	H	H	H	H	H	H
C	H	H	H	L	L	H	L	L	L	H	H	H	H	H	L	H	H
C	H	H	H	L	L	H	L	L	L	H	H	H	H	L	L	H	H
C	L	H	L	L	H	H	L	L	L	H	H	H	L	L	L	H	H
C	L	H	L	L	H	H	L	L	L	H	H	L	L	L	L	H	H
C	L	H	L	L	H	H	L	L	L	H	H	L	L	L	H	H	H
C	L	H	L	L	H	H	L	L	L	H	H	L	L	H	H	H	L
C	L	H	L	L	H	L	L	L	L	L	H	L	H	H	H	L	L
C	L	H	L	L	H	L	L	L	L	L	H	H	H	H	H	L	L
C	H	H	H	L	H	L	L	L	L	H	H	H	H	H	H	H	H
C	H	H	H	L	L	H	L	H	L	H	H	H	H	H	L	H	H
C	H	H	H	L	L	H	L	H	L	H	H	H	H	L	L	H	H
C	L	L	L	L	H	H	L	H	L	H	H	H	L	L	L	H	H
C	L	L	L	L	H	H	L	H	L	H	H	L	L	L	L	H	H
C	L	L	L	L	H	H	L	H	L	H	H	L	L	L	H	H	H
C	L	L	L	L	H	H	L	H	L	H	H	L	L	H	H	H	L
C	L	L	L	L	H	L	L	H	L	L	L	L	H	H	H	L	L
C	H	H	H	L	H	L	L	H	L	H	H	H	H	H	H	L	H
C	H	H	H	L	H	H	H	H	L	H	H	H	H	H	H	Z	H
C	H	H	H	H	L	H	L	H	L	H	H	H	H	H	L	H	H
C	H	H	H	H	L	H	L	H	L	H	H	H	H	L	L	H	L
C	L	L	H	H	H	H	L	H	L	H	H	H	L	L	H	H	H
C	L	L	H	H	H	H	L	H	L	H	H	L	L	H	H	H	L
C	L	L	H	H	H	L	L	H	L	H	L	L	H	H	H	L	L
C	H	H	H	H	H	L	L	H	L	H	H	H	H	H	H	H	H
C	H	H	H	H	H	H	L	H	H	H	H	Z	Z	Z	Z	H	H

Table 8.12.6 Function Table

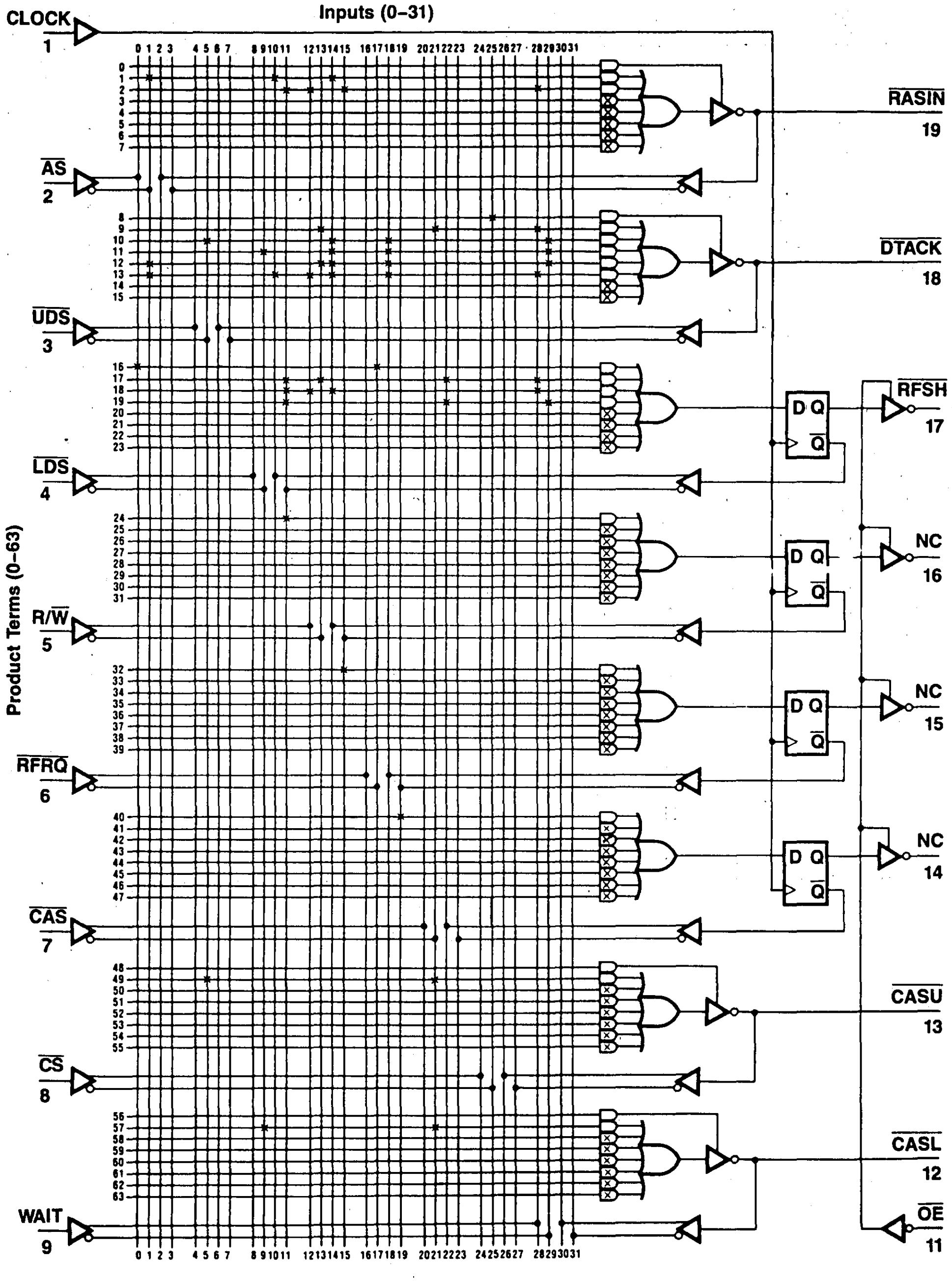


Figure 8.12.12 DP84322 Logic Diagram PAL Device 16R4

8.13 DP84332 DYNAMIC RAM CONTROLLER INTERFACE CIRCUIT FOR THE 8086 AND 8088 CPUS

General Description

The DP84332 dynamic RAM controller interface is a PAL device for interface between the DP8408 dynamic RAM controller and the 8086 and 8088 microprocessors. No wait states are required for memory access. Memory refreshing may be hidden (no wait states) or forced (up to three wait states).

The DP84332 supplies all the control signals needed to perform memory read, write and refresh. Logic is also included to insert a wait state when using slow memory.

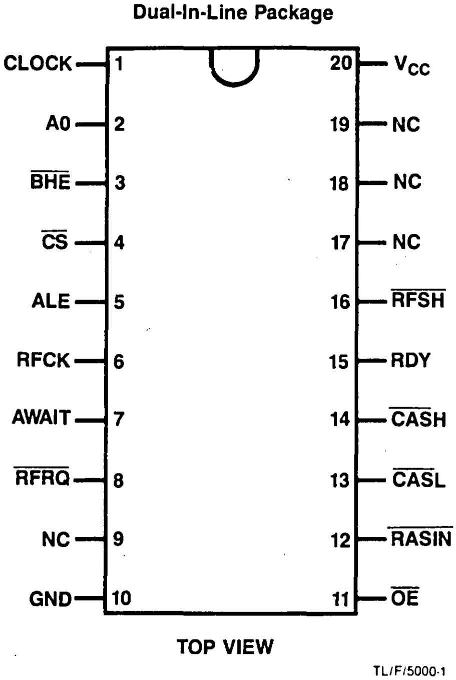


Figure 8.13.1 Connection Diagram

Features

- Low parts count controller for the DP8408/DP8409.
- Works with 8086 systems configured in min or max mode.
- Performs hidden refresh using the DP8408 dynamic RAM controller.
- Compatible with both the 8086 and 8088 microprocessors.
- Capable of working at all CPU clock frequencies up to 8 MHz.
- Standard National Semiconductor PAL device part (PAL16R8).
- PAL device logic equations can be modified by the user for his specific application and programmed into any of the PAL devices in the National Semiconductor family, including the new high speed PAL devices.

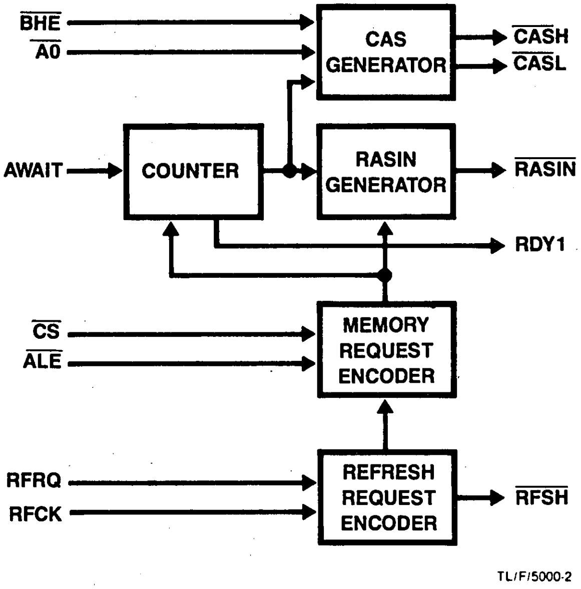


Figure 8.13.2 Block Diagram

Symbol	Parameter	Min	Typ	Max	Units
V_{CC}	Supply Voltage	4.75	5.00	5.25	V
I_{OH}	High Level Output Current			–3.2	mA
I_{OL}	Low Level Output Current			24 (Note 2)	mA
T_A	Operating Free Air Temperature	0		75	°C

Table 8.13.1 Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2			V
V_{IL}	Low Level Input Voltage				0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min, I_I = – 18 mA			– 1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OH} = Max	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, V_{IH} = 2V, V_{IL} = 0.8V, I_{OL} = Max			0.5	V
I_{OZH}	Off-State Output Current High Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 2.4V, V_{IL} = 0.8V			100	μA
I_{OZL}	Off-State Output Current Low Level Voltage Applied	V_{CC} = Max, V_{IH} = 2V, V_O = 0.4V, V_{IL} = 0.8V			– 100	μA
I_I	Input Current at Maximum Input Voltage	V_{CC} = Max, V_I = 5.5V			1.0	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.4V			25	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V			– 250	μA
I_{OS}	Short Circuit Output Current	V_{CC} = Max	– 30		– 130	mA
I_{CC}	Supply Current	V_{CC} = Max		150	225 (Note 1)	mA

Table 8.13.2 Electrical Characteristics

Symbol	Parameter		Conditions $R_L = 667\Omega$	Commercial T_A = 0°C to + 75°C V_{CC} = 5.0V ± 5%			Units
				Min	Typ	Max	
t_{PD}	Clock to Output		C_L = 45 pF		15	25	ns
t_{PZX}	Pin 11 to Output Enable		C_L = 45 pF		15	25	ns
t_{PXZ}	Pin 11 to Output Disable		C_L = 5 pF		15	25	ns
t_W	Width of Clock	High		25			ns
		Low		25			ns
t_{su}	Set-Up Time			40			ns
t_H	Hold Time			0	– 15		ns

Note 1: I_{CC} = max at minimum temperature.
Note 2: One output at a time; otherwise 16 mA.

Table 8.13.3 Switching Characteristics

Mnemonic Description

Input Signals

CLOCK — The CLOCK signal determines the timing of the outputs and should be connected directly to the 8086 clock.

A0, $\overline{\text{BHE}}$ — These inputs come from the 8086 CPU. They must remain stable during the memory cycle for proper operation of the $\overline{\text{CAS}}$ outputs.

$\overline{\text{CE}}$ — Chip enable. This input is used to select the memory and enable the hidden refresh logic.

ALE — Address latch enable. This input is used to indicate the beginning of a memory cycle.

RFCK — Refresh clock. The period of this input determines the refresh interval. The duty cycle of this clock will determine the length of time that the circuit will attempt a hidden refresh.

AWAIT — When connected to VCC, the DP84332 will insert an extra wait state in selected memory cycles.

$\overline{\text{RFRQ}}$ — Refresh request. This input requests the DP84332 to perform a refresh. The state of the RFCK input will determine what type of refresh will be performed.

Output Signals

$\overline{\text{RASIN}}$ — This output provides a memory cycle start signal to the DP8408, and provides $\overline{\text{RAS}}$ timing during refresh.

$\overline{\text{CASH}}$, $\overline{\text{CASL}}$ — These signals are the separate $\overline{\text{CAS}}$s needed for byte writing. Their presence is controlled by $\overline{\text{BHE}}$ and A0 respectively.

RDY — This output is used to insert a wait state into the 8086 memory cycles when selected and during a forced refresh cycle where the 8086 attempts to access the memory. The 8284A clock circuit should be configured so that ASYNC is enabled.

RFSH — This output controls the mode of the DP8408 dynamic RAM controller. When low, it switches the DP8408 into an all $\overline{\text{RAS}}$ refresh mode. This signal is also used to reset the refresh request logic.

Functional Description

A memory cycle starts when chip select ($\overline{\text{CS}}$) and the address latch enable (ALE) are true. RASIN is supplied from the DP84332 to the DP8408 dynamic RAM controller which then supplies a RAS signal to the selected dynamic RAM bank. After the necessary row address hold time, the DP8408 switches the address outputs to the column address. The DP84332 then supplies the required $\overline{\text{CAS}}$ signals ($\overline{\text{CASH}}$, $\overline{\text{CASL}}$) to the RAM. For byte operations, only one $\overline{\text{CAS}}$ will be activated. To differentiate between a read and a write, the DT/$\overline{\text{R}}$ signal from the CPU is inverted and supplied by the DP8408 to the memory array.

A refresh cycle is started by one of two conditions. One is when a refresh is requested (RFRQ is true), refresh clock (RFCK) is high, and a non-selected memory cycle is started (CE is not true, ALE is high). This is called hidden refresh because it is transparent to the CPU. In this case, the address supplied to the memories comes from the refresh counter in the DP8408, and no $\overline{\text{CAS}}$ signals are generated from the DP84332 . The second form of refresh occurs when a refresh is requested, refresh clock is low, and there is no memory cycle in progress. This is called forced refresh, because the CPU will be forced to wait during the next memory cycle to allow for the refresh to be performed. In this case, a refresh is performed as before, but any attempt to access memory is delayed by wait states until after the refresh is finished. In either case, the refresh request is cleared by the refresh line ($\overline{\text{RFSH}}$), which also goes to the DP8408.

In a standard memory cycle, the access can be slowed down by one clock cycle to accommodate slower memories. This extra wait state will not appear during the hidden refresh cycle, so faster devices on the CPU bus will not be affected.

With higher speed systems, memory speed requirements will affect the performance of the system. Table 1 shows memory speed requirements at three different CPU clock speeds.

CPU Clock Frequency	t_{CAC}		t_{RAH}
	No Wait States	1 Wait State	
8 MHz	≤105 ns	≤223 ns	≤30 ns
5 MHz	≤170 ns	≤370 ns	≤30 ns

Table 8.13.4 Memory Speed Requirements

System Description

For memory operation, the DP84332 can be directly connected between the control signals from the CPU chip set and the DP8408 dynamic RAM controller. Each CAS output of the DP84332 is capable of driving eight memory devices. If additional drive is required, a DP84244 buffer can be used to increase the fanout to the full capabilities of the DP8408 (eight memories per output of the DP84244).

The 84332 is a standard National Semiconductor PAL part (PAL 16R8). The user can modify the PAL equations to support his particular application. The 84332 logic equations, function table, and logic diagram can be seen at the end of this section.

INTERFACING THE DP8408 TO AN 8086 SYSTEM

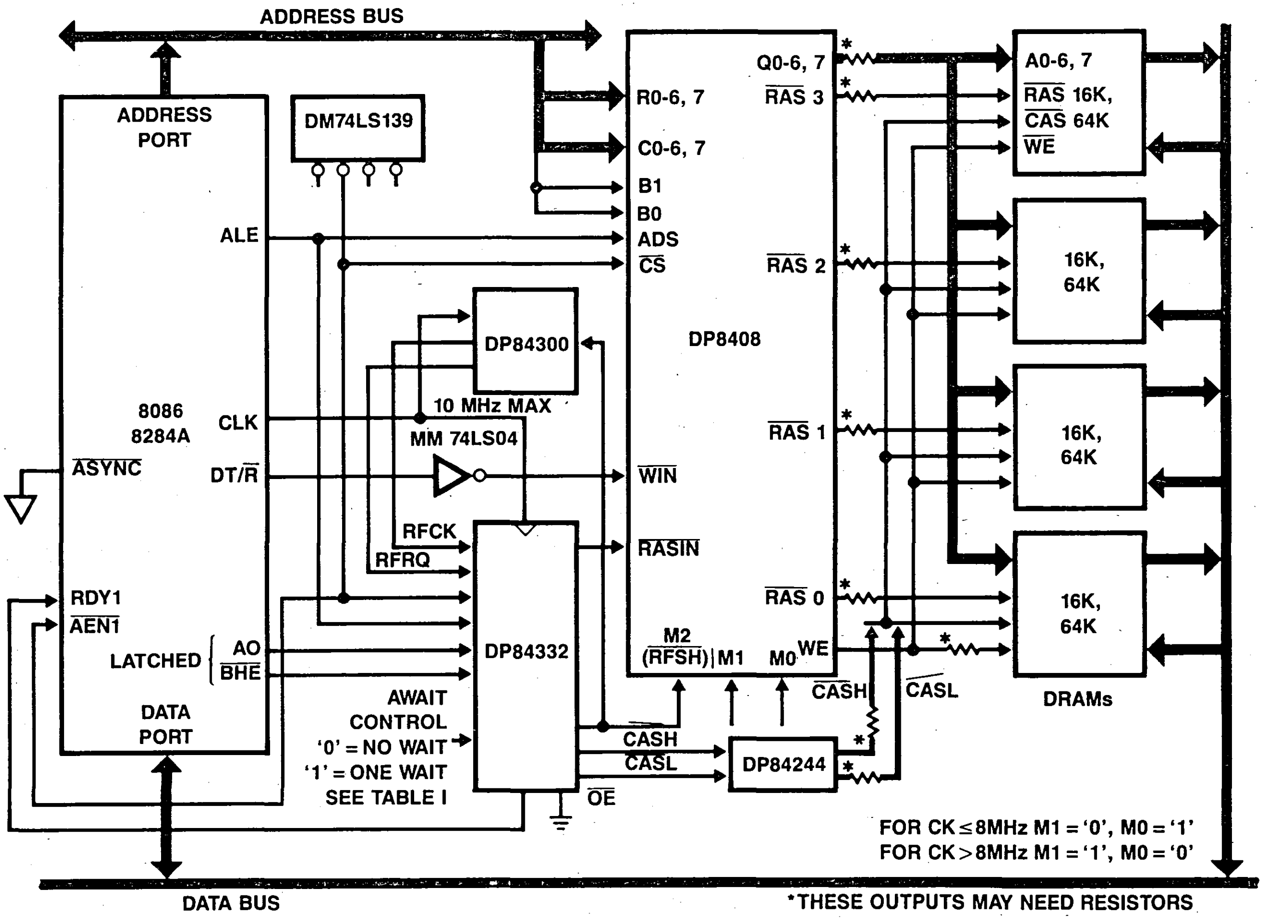


Figure 8.13.3 System Block Diagram

Refresh Request Logic

To generate the refresh request for the DP84332, external circuitry is required. Figure 1 shows how this can be implemented, using standard SSI and MSI logic. A DM74LS393 counter is used to time the period between refresh cycles, while the DM74LS74 flip-flop is used to record the need of a new refresh. A better solution is to use the 24-pin DP84300 programmable refresh timer, as shown in Figure 2. This part allows a maximum amount of time for a hidden refresh to occur before lowering the refresh clock output, and implements the refresh request logic.

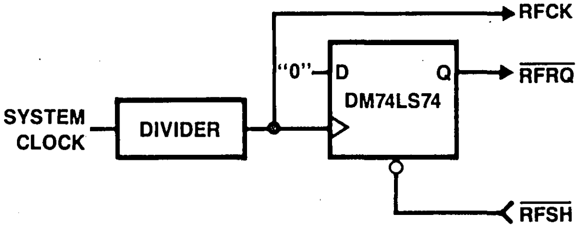


Figure 8.13.4 Using a Flip-Flop and a Counter for Refresh Request Logic

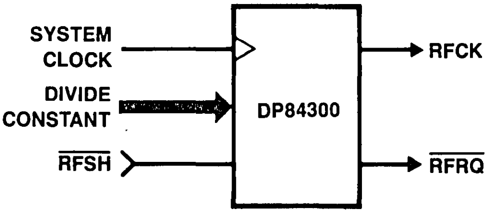


Figure 8.13.5 Using the DP84300 Refresh Counter for Refresh Logic

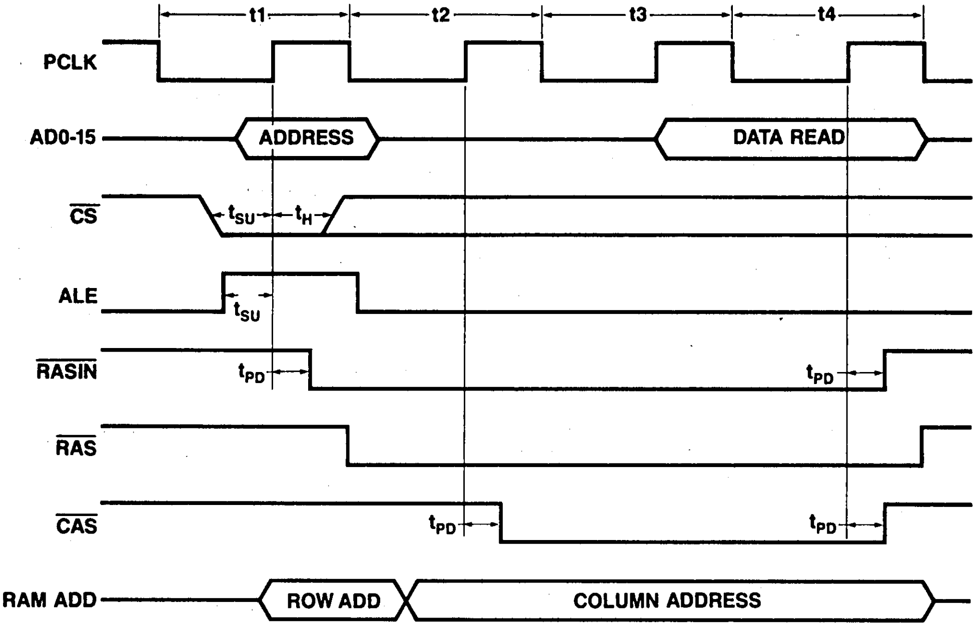


Figure 8.13.6 Timing Diagram; Read Timing

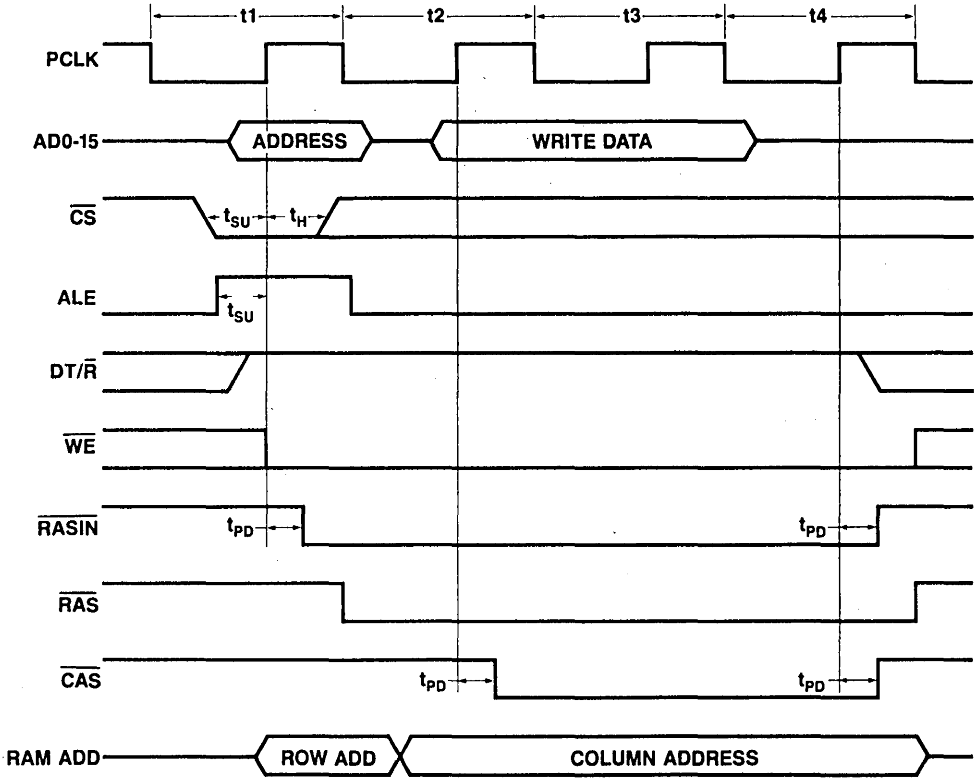


Figure 8.13.7 Timing Diagram; Write Timing

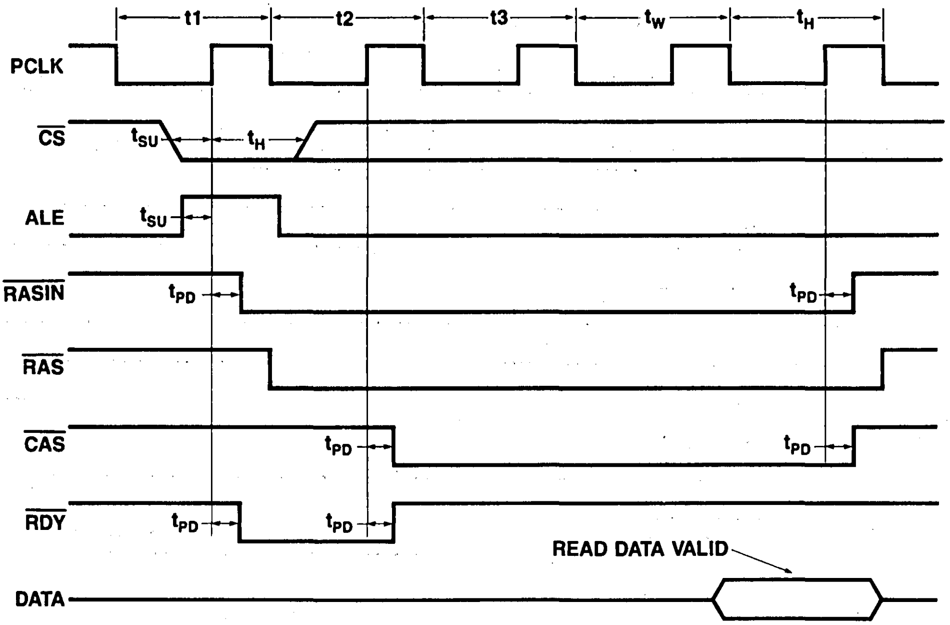


Figure 8.13.8 Timing Diagram; Memory Cycle With 1 Wait State

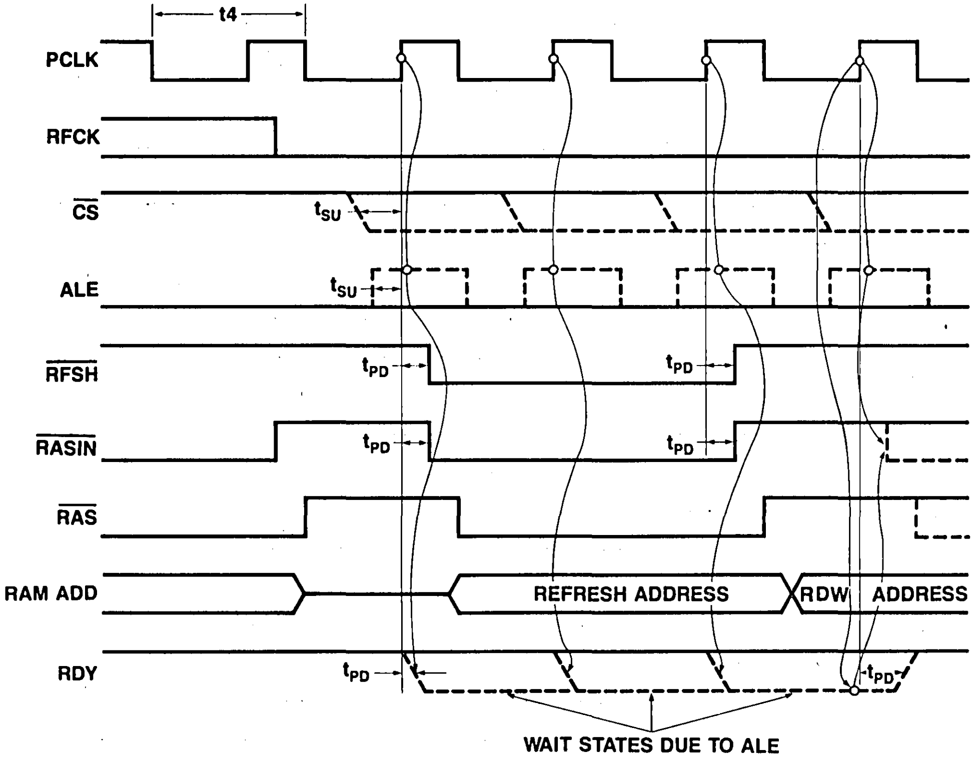


Figure 8.13.9 Timing Diagram; Forced Refresh

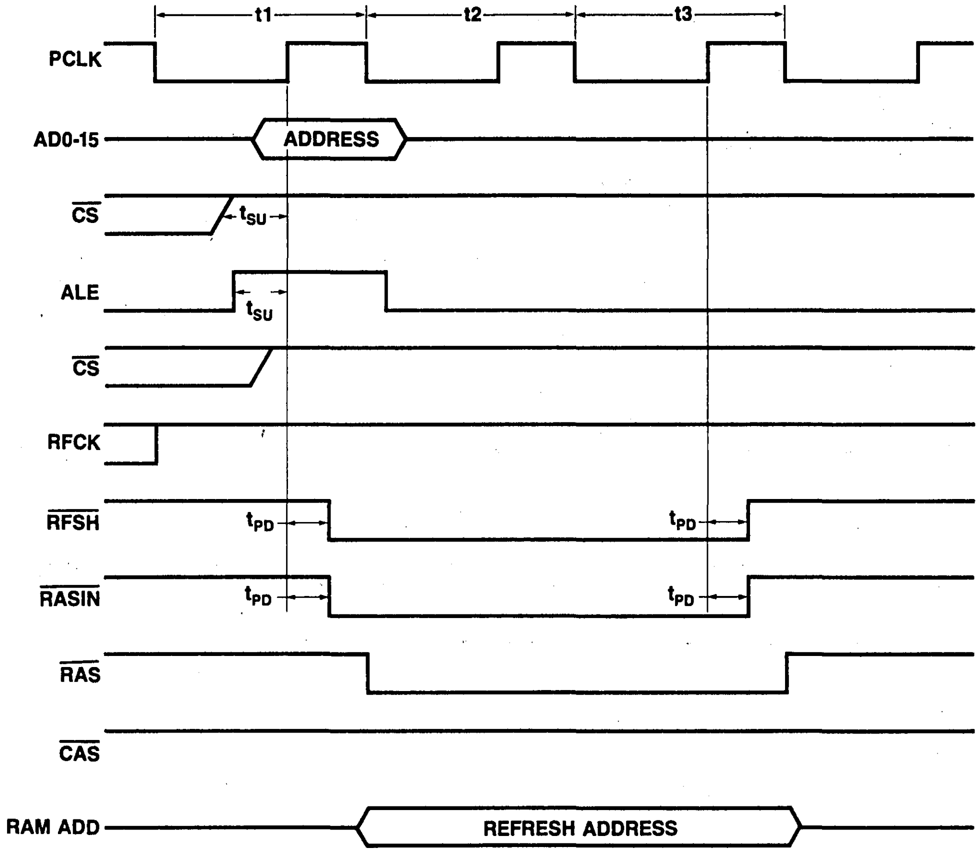


Figure 8.13.10 Timing Diagram, Transparent Refresh

PAL16R8
Dynamic RAM Controller Interface for the 8086-8408 System
CK A0 /BHE /CS ALE RFCK WAIT /RFRQ NC GND /OE /RASIN /CA /CB
RDY /RFSH /A /B /MRQ VCC

```
MRQ: = /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*CS*ALE*/RFCK +
       MRQ*RASIN +
       RAISIN*/CA*/CB*RDY*RFSH*/A*MRQ*CS*ALE

  B: = RASIN*/CA*/CB*RFSH*/A*/B +
       RASIN*/CA*/CB*/RDY*/RFSH*/A*/B*WAIT +
       RASIN*RDY*/RFSH*A*/B
```

```
    A: =  RASIN*ICA*ICB*RDY*/RFSH*/A*/B*/WAIT +
          RASIN*RDY*/RFSH*/A*B +
          RASIN*RDY*/RFSH*A*/B

 RFSH: =  /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*/CS*ALE*RFCK* +
          /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*/RFCK +
          RASIN*/CA*/CB*RFSH*/A*/B

 /RDY: =  /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*MRQ*RFRQ*CS*ALE*/RCFK +
          RASIN*/CA*/CB*RDY*RFSH*/A*/MRQ*CS*ALE +
          /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*/RFRQ*CS*ALE*WAIT +
          /RASIN*/CA*/CB*/RDY*/RFSH*/A*/B*MRQ*/RFRQ*WAIT +
          RASIN*/CA*/CB*/RDY*RFSH*/A +
          /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*CS*ALE*RFCK*WAIT

   CB: =  RASIN*/CA*/CB*/RFSH*/A*/B*BHE +
          RASIN*CB*RDY*/RFSH*/A*B*WAIT +
          RASIN*CB*RDT*/RFSH*A/*B

   CA: =  RASIN*/CA*/CB*/RFSH*/A*/B*BHE +
          RASIN*CA*RDY*/RFSH*/A*B*WAIT +
          RASIN*CA*RDY*RFSH*A*/B

RASIN: =  /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*/RFRQ*CS*ALE +
          /RASIN*/CA*/CB*/RDY*/RFSH*/A*/B*MRQ*/RFRQ +
          RASIN*/CA*/CB*/RFSH*/A*/B +
          RASIN*RDY*/RFSH*/A*B*WAIT +
          /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*ALE*RFCK +
          /RASIN*/CA*/CB*RDY*/RFSH*/A*/B*/MRQ*RFRQ*/RFCK +
          RASIN*/CA*/CB*RFSH*/A*/B +
          RASIN*RDY*/RFSH*A*/B
```

CK	A0	$\overline{BHE}$	$\overline{CS}$	ALE	RFCK	WAIT	$\overline{RFRQ}$	$\overline{OE}$	$\overline{RASIN}$	$\overline{CA}$	$\overline{CB}$	RDY	$\overline{RFSH}$	$\overline{A}$	$\overline{B}$	$\overline{MRQ}$
C	L	L	H	L	H	L	H	L	X	X	X	X	X	X	X	X
C	L	L	H	L	H	L	H	L	H	H	H	H	H	H	H	H
C	X	X	L	H	X	L	H	L	L	H	H	H	H	H	H	H
C	L	H	L	L	X	L	H	L	L	L	H	H	H	L	H	H
C	L	H	L	L	X	L	H	L	L	L	H	H	H	L	L	H
C	X	X	H	L	H	L	H	L	H	H	H	H	H	H	H	H
C	X	X	L	H	X	H	H	L	L	H	H	L	H	H	H	H
C	H	H	L	L	X	H	H	L	L	H	H	H	H	H	L	H
C	H	H	L	L	X	H	H	L	L	H	H	H	H	L	H	H
C	H	H	L	L	X	H	H	L	L	H	H	H	H	L	L	H
C	X	X	H	L	H	H	H	L	H	H	H	H	H	H	H	H
C	X	X	H	H	H	X	L	L	L	H	H	H	L	H	H	H
C	X	X	H	L	X	X	X	L	L	H	H	H	L	H	L	H
C	X	X	X	L	X	X	H	L	H	H	H	H	H	H	H	H
C	X	X	X	L	L	X	L	L	L	H	H	H	L	H	H	H
C	X	X	X	L	X	X	X	L	L	H	H	H	L	H	L	H
C	X	X	X	L	X	X	H	L	H	H	H	H	H	H	H	H
C	X	X	L	H	L	X	L	L	L	H	H	L	L	H	H	L
C	H	L	L	L	X	X	X	L	L	H	H	L	L	H	L	L
C	H	L	L	L	X	X	H	L	H	H	H	L	H	H	H	L
C	H	L	L	L	X	L	H	L	L	H	H	H	H	H	H	H
C	H	L	L	L	X	L	H	L	L	H	L	H	H	L	H	H
C	H	L	L	L	X	L	H	L	L	H	L	H	H	L	L	H
C	X	X	L	L	X	L	H	L	H	H	H	H	H	H	H	H
C	X	X	X	L	L	X	L	L	L	H	H	H	L	H	H	H
C	X	X	L	H	X	H	X	L	L	H	H	L	L	H	L	L
C	L	L	L	L	X	H	X	L	H	H	H	L	H	H	H	L
C	L	L	L	L	X	H	H	L	L	H	H	L	H	H	H	H
C	L	L	L	L	X	H	H	L	L	L	L	H	H	H	L	H
C	L	L	L	L	X	H	H	L	L	L	L	H	H	L	H	H
C	L	L	L	L	X	H	H	L	L	L	L	H	H	L	L	H
C	X	X	L	L	X	H	H	L	H	H	H	H	H	H	H	H
C	X	X	L	H	H	L	L	L	L	H	H	H	H	H	H	H
C	H	L	L	L	X	L	X	L	L	H	L	H	H	L	H	H
C	H	L	L	L	X	L	X	L	L	H	L	H	H	L	L	H
C	X	X	L	L	X	L	X	L	H	H	H	H	H	H	H	H
C	X	X	L	H	H	H	L	L	L	H	H	L	H	H	H	H
C	X	X	L	L	X	X	X	H	Z	Z	Z	Z	Z	Z	Z	Z

Table 8.13.5 Function Table

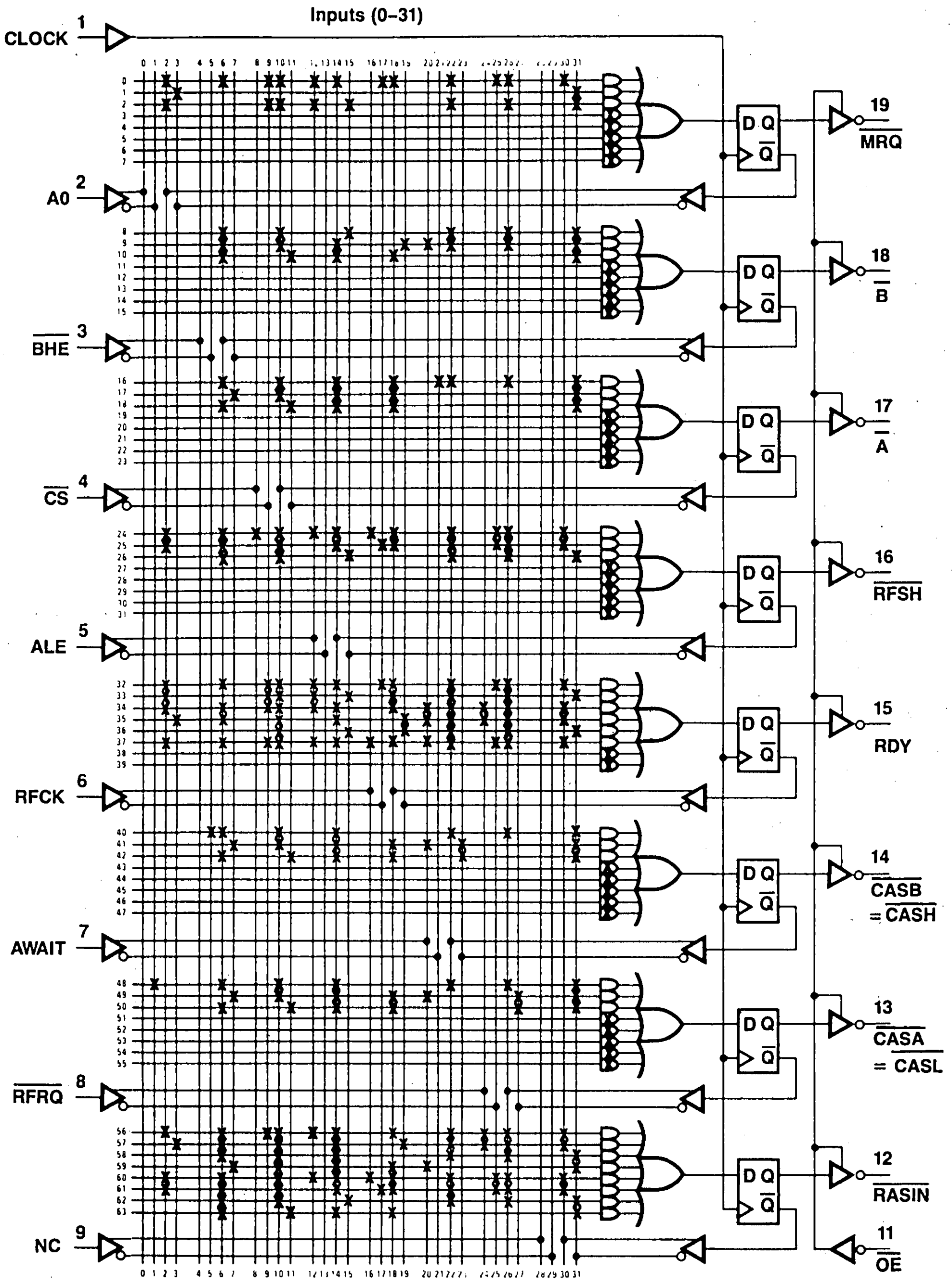


Figure 8.13.11 84332 Logic Diagram PAL16R8

8.14 A PAL DEVICE INTERFACE BETWEEN THE NATIONAL SEMICONDUCTOR NS32032 MICROPROCESSOR, DP8409 DYNAMIC RAM CONTROLLER, AND THE DP8400 EXPANDABLE ERROR CHECKER AND CORRECTOR

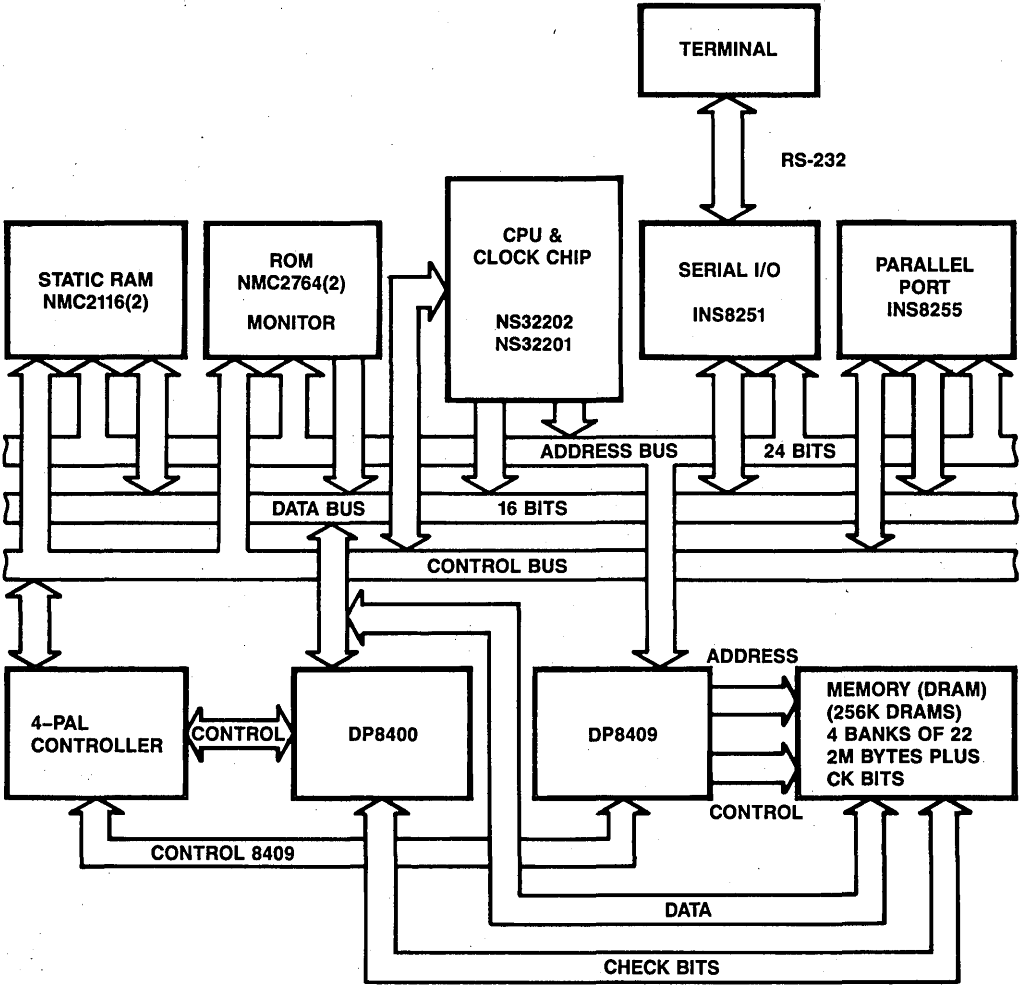


Figure 8.14.1 DP8400, DP8409, NS16032 6 MHz Computer System

• *Application 8.14 is contributed by Webster (Rusty) Meier, Design Engineer of National Semiconductor*

Four PAL devices were used in this application in order to interface between the NS32032, DP8409 and the DP8400. These PAL devices have the following features:

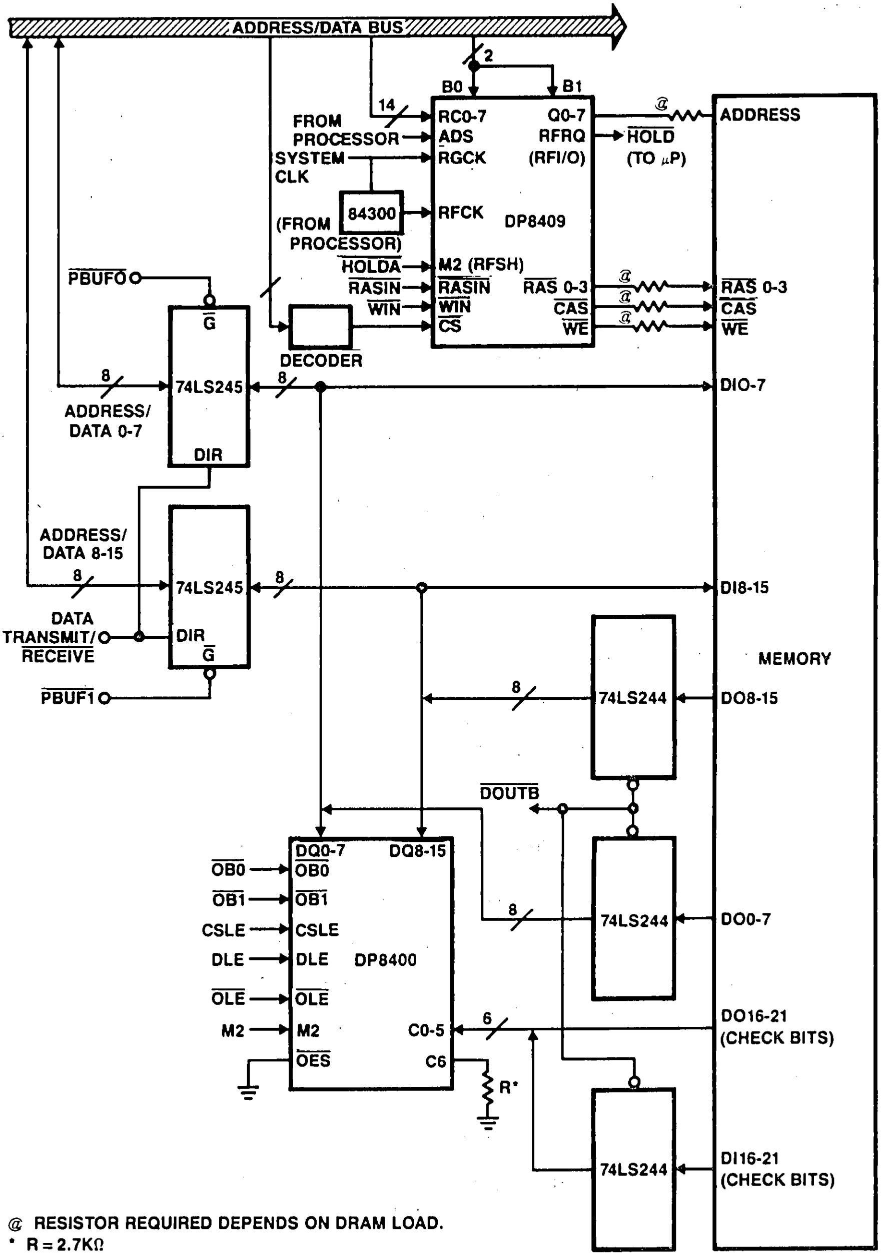
1. The PAL devices control the following types of cycles:
   a) READ cycles with no errors detected, ALWAYS CORRECT MODE
   b) READ cycles with single error detected, the correct data will be written back to memory
   c) WRITE cycles
   d) BYTE WRITE cycles
   e) DRAM REFRESH cycles
   The PAL devices take care of everything, no extra control logic is needed.
2. The outputs of the PAL device control the DP8409, the DP8400 and insert WAIT states at the appropriate times into the NS32032 cycles.
3. The PAL device contains outputs to interrupt the NS32032, or cause a cycle abort if an error greater than a single error is detected (DOUBLERR), or if there is a bus parity error in data transfer from the CPU to memory (PARITYERROR).
4. This PAL device design should work up to 8Mhz with the NS32032. If it is desired to go faster, another WAIT state will have to be inserted into all cycles, and the PAL device equations will have to be adjusted accordingly. Another possibility would be to use the new oxide isolated DP8400 and the new DRAM controller DP8419 (pin compatible with DP8409 in modes 0,1,4,5). These parts would allow considerably more time margin.
5. As can be seen by looking at the PAL device logic diagrams some external logic is needed and some external logic may be added. For example, a system reset input could be added to allow the internal flip-flops to be set to a known state — in this case a refresh state (In PAL device number 1, for example, I used external logic to "NOR" the RFI/O input with a system RESET input). An output enable input was also included to allow all the PAL device outputs to be tri-stated.
6. This PAL device interface performs HIDDEN REFRESHES (CPU not accessing the Dynamic RAM controlled by the DP8409, indicated by /CS being high) assuming a four-T state processor access cycle.
7. Logic diagrams, the PAL device equations, and the timing diagrams follow this introduction section. Basically everything is self-explanatory.
8. I feel that if one is using this interface above 4–6MHz, he should use the fast PAL devices (example "PAL16R8A" instead of "PAL16R8"). The fast PAL devices have an input to output maximum time of 25ns and 15ns if it is a registered output.

The slow PAL devices have an input-to-output maximum time of 35ns and 25ns if it is a registered output. Depending on the specific type of PAL's and logic used, the user can calculate the speed requirements for the DRAM at the specified processor frequency with the timing that I have chosen.

9. The four PAL devices that I have used allow full use of the DP8400 and all its modes of operation. For example, one can perform a complete diagnostic test of the DP8400 without needing to use the external memory. This is possible using an I/O port to control M2 and M1 of the DP8400, along with diagnostic control signals DIAGCS and DIAGD. These signals from the I/O port allow the user complete control over the operating modes of the DP8400 and its data syndrome, and check bit latches.

PAL Device Number 1 Inputs

1.	FCLK	Fast Clock (twice CTTL frequency) from NS32201
2.	CTTL	Output clock from NS32201
3.	/CS	Chip Select for the Dynamic RAM controlled by the DP8409 and DP8400.
4.	/DDIN	Data Direction in, from NS32032, indicates the direction of the data transfer during a bus cycle.
5.	RFI/O	Refresh request output from the DP8409, is also used as a reset input to set PAL to a known state.
6.	INCY	Output from PAL device number 2 indicating that the NS32032 is in an access cycle.
7.	/AOHBE	If address bit 0 AND high byte enable (from NS32032) are both low this input is high. Used to determine when byte operations are in progress.
8.	NTSO	From NS32201, indicating that timing state T2 is starting, it stays low until the beginning of T4.
9.	/ERRLATCH	Output from PAL device number 3 indicating that any error, AE, was valid during a READ access cycle.
10.	/OE	Controlled externally, TRI-STATE PAL outputs.



@ RESISTOR REQUIRED DEPENDS ON DRAM LOAD.
* R = 2.7KΩ

Figure 8.14.2 DP8400/8409 System Interface Block Diagram

PAL Device Number 1 Outputs

1. /RASIN Input to DP8409.
2. /RFSH Input to DP8409, causes the DP8409 to enter mode 1 to do a refresh.
3. /1DLY Delay used by the PAL devices to determine the state of the processor system.
4. /2DLY Delay used by the PAL devices to determine the state of the processor system.
5. /3DLY Delay used by the PAL devices to determine the state of the processor system.
6. /4DLY Delay used by the PAL devices to determine the state of the processor system.
7. /ODCLEN /OLE, DLE, CSLE enable latch signal.
8. /CYCLED Indicates that a processor access cycle is complete.

PAL Device Number 2 Inputs

1. /RFSH Output from PAL device number 1 that indicates whether the DRAMs are being refreshed.
2. /RASIN Output from PAL device number 1.
3. A0 Output from NS32032, address bit 0.
4. /HBE Output from NS32032, high byte enable.
5. /DDIN Data Direction in, from NS32032.
6. /ADS Address strobe from NS32032.
7. NTSO Output from NS32201.
8. /2DLY Output from PAL device number 1.
9. /4DLY Output from PAL device number 1.
10. /ERRLATCH Output from PAL device number 3 indicating that an error has occured during a READ cycle.
11. CSOE Chip select Output Enable, TRI-STATE the outputs of the PAL device when low, and also used for other control purposes.

PAL Device Number 2 Outputs

1. /0B0 Controls DP8400 output buffer for byte "0".
2. 0B1 Controls DP8400 output buffer for byte "1".
3. /PBUF0 Controls the processor buffer transceiver for byte "0".
4. /PBUF1 Controls the processor buffer transceiver for byte "1".
5. /DOUTB Controls memory buffers that interface between the DRAM and the DP8400/memory data bus.
6. /INCY Output indicating that the NS32032 is in an access cycle.
7. /CWAIT Output to NS32201 that causes WAIT states to be inserted into the NS32032 bus cycles.

PAL Device Number 3 Inputs

1. /DDIN Output from NS32032.
2. /RFSH Output from PAL device number 1 indicating a forced refresh of the memory.
3. /AOHBE Output of A0 and /HBE logically NORed together. Therefore, if either input is high this signal will be low. This signal is useful to determine whether words or bytes are being written.
4. /ERRLATCH Output from PAL device number 4 indicating that an error has occurred during a CS READ cycle, it may be a single or multiple bit error.
5. /1DLY Input from PAL device number 1.
6. /2DLY Input from PAL device number 1.
7. /3DLY Input from PAL device number 1.
8. /4DLY Input from PAL device number 1.
9. /RESET Input from external logic that resets the double bit error latch /DOUBLERR or the parity error latch PARITYERR.
10. AE Output from DP8400 indicating an error.
11. E0 Output from DP8400 indicating the type of error.
12. E1 Output from DP8400 indicating the type of error.

13. /PARITYERROR This is an output of this PAL device also. This input indicates that a PARITY error has occurred during a WRITE cycle.

14. CSOE Chip Select Output enable, tristates the registered outputs of the PAL device when low.

PAL Device Number 3 Outputs

1. /WIN Input to the DP8409.

2. /MODECC Input to the DP8400, changes between READ and WRITE modes.

3. /PARITYERR Can be used to interrupt the system when a parity error has been detected during a WRITE cycle.

PAL Device Number 4 Inputs

1. FCLK Fast clock fromNS32201.

2. ODCLEN /OLE, DLE, CSLE latch enable input.

3. DIAGCS Enable input from I/O port for diagnostics to enable CSLE, check bit syndrome latch enable.

4. DIAGD Enable input from I/O port for dagnostics to enable DLE, data latch enable.

5. /RESET Reset input from I/O port to reset PAL error latches.

6. /CYCLED Output from PAL device number 1 indicating that a processor access cycle is complete.

7. AE Output from DP8400 indicating an error.

8. /E01 When this input is low it indicates that either error flag E0 or E1 was high.

9. /3DLY This is an input from PAL device number 1.

10. /0E Output from I/O port that enables the PAL outputs.

11. /DDIN NS32032 input that indicates the direction of the bus transfer during a bus cycle.

12. /RFSH Output from PAL device number 1 indicating a DRAM refresh cycle.

PAL Device Number 4 Outputs

1. DLE — Output that controls the DP8400 Data latch.
2. CSLE — Output that controls the DP8400 Check bit Syndrome latch.
3. /OLE — Output that controls the DP8400 Output latch.
4. /DOUBLERR — Can be used to interrupt the system when a double bit error has been detected during a READ cycle.
5. /ERRLATCH — Used in the PAL device controller to indicate that an error has occurred during a /CS READ cycle, as indicated by AE being valid.

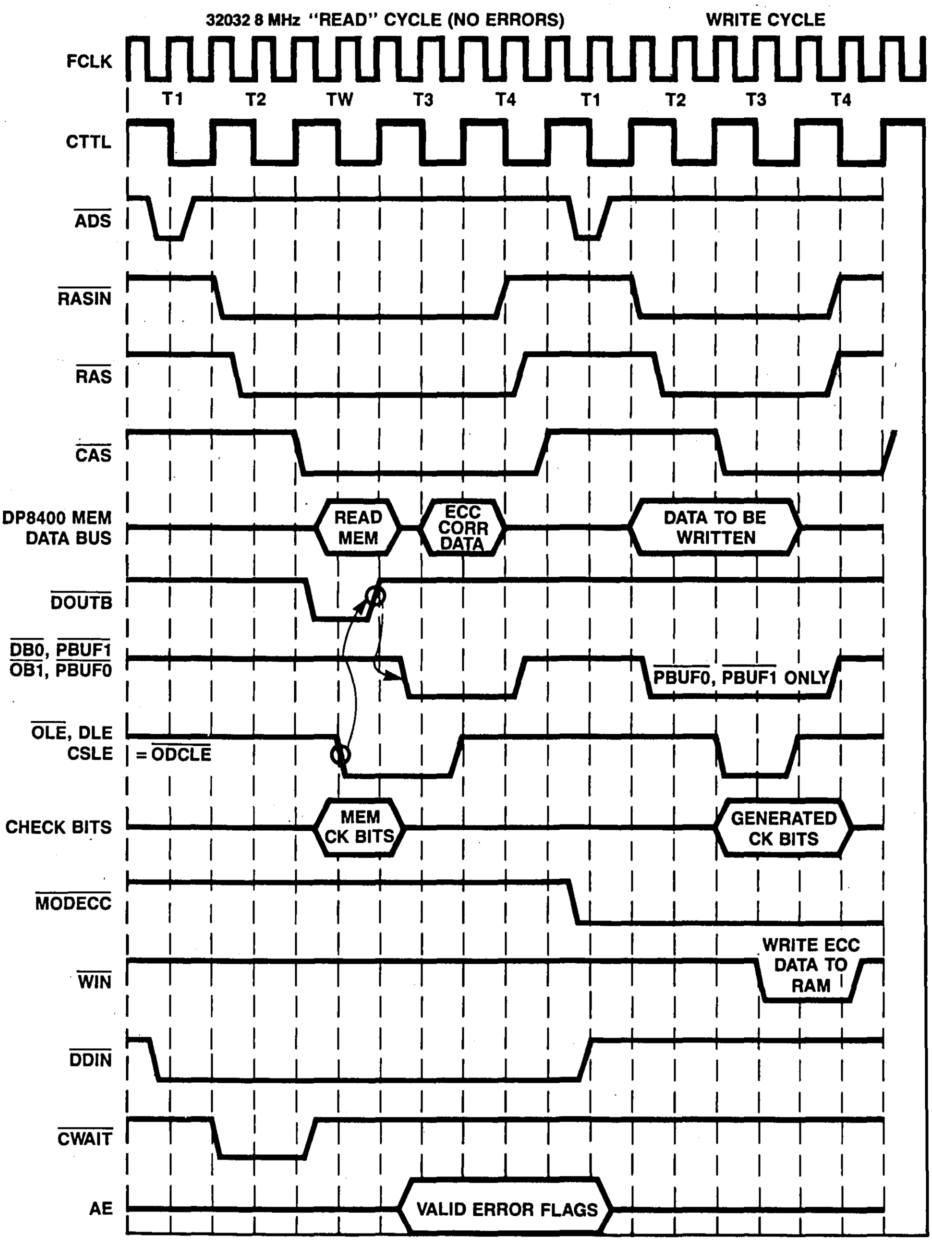


Figure 8.14.3 Timing Diagram; Read Cycle and Write Cycle

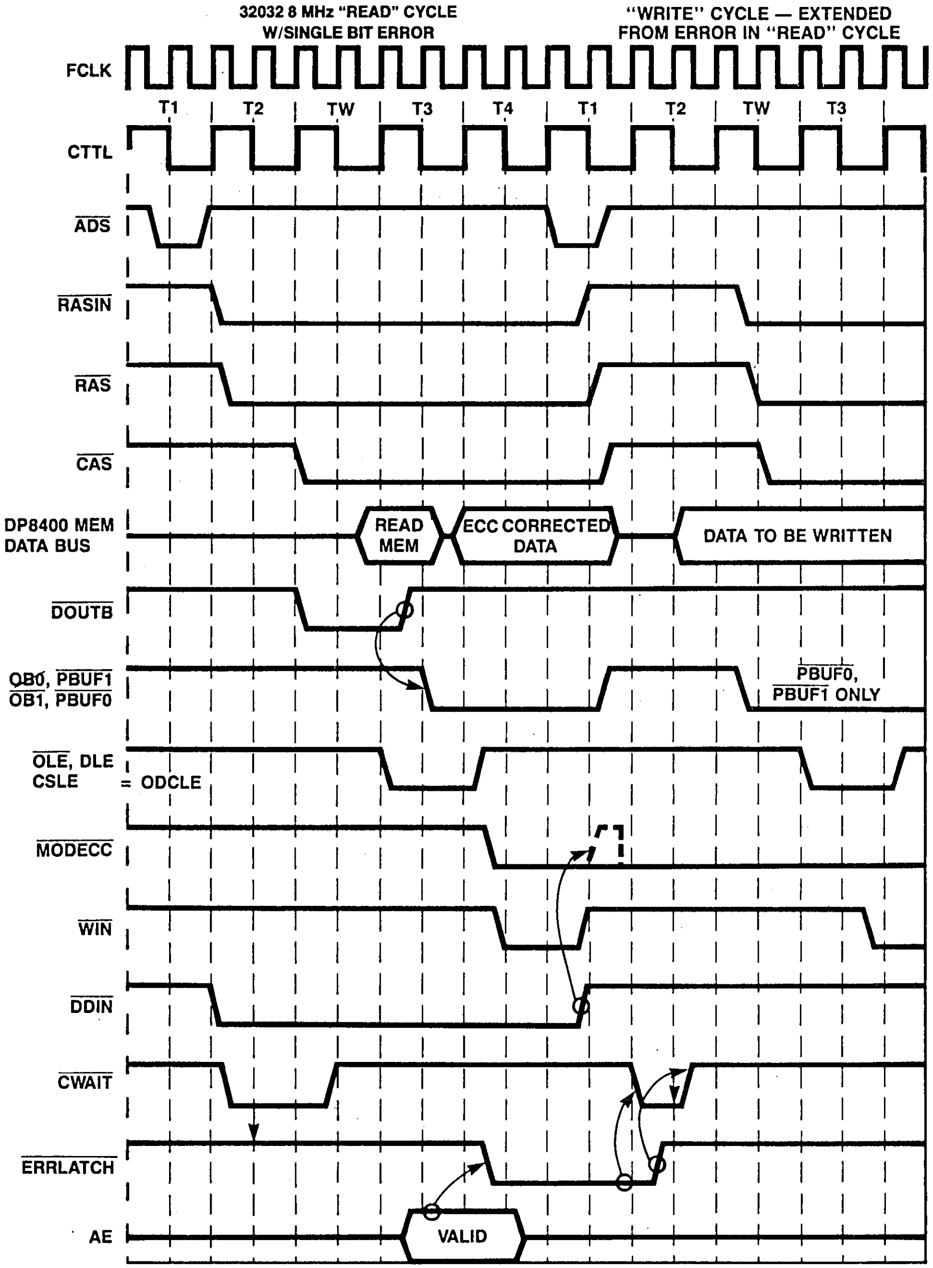


Figure 8.14.4 Timing Diagram; Read Cycle With Simple Bit Error

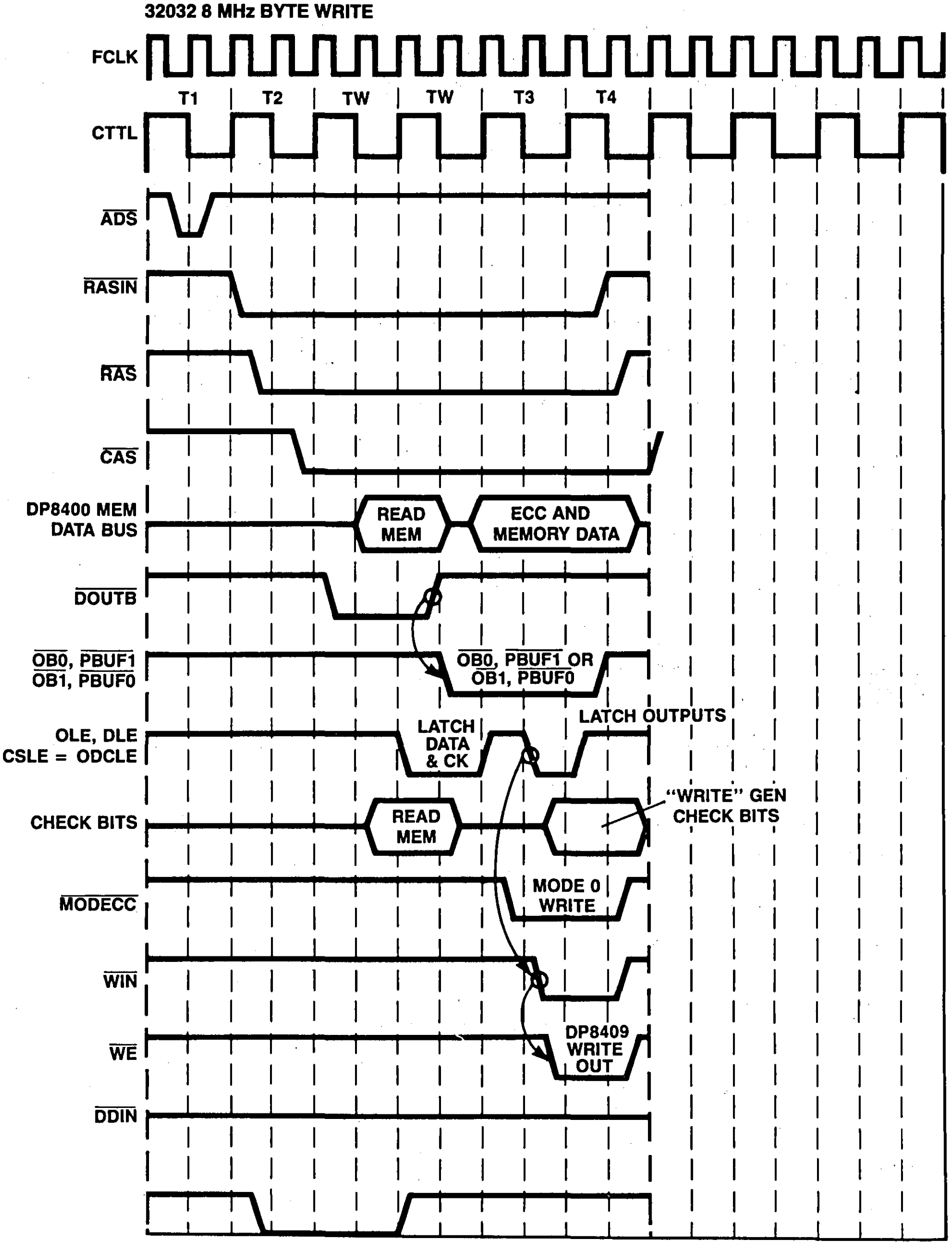


Figure 8.14.5 Timing Diagram; Byte Write

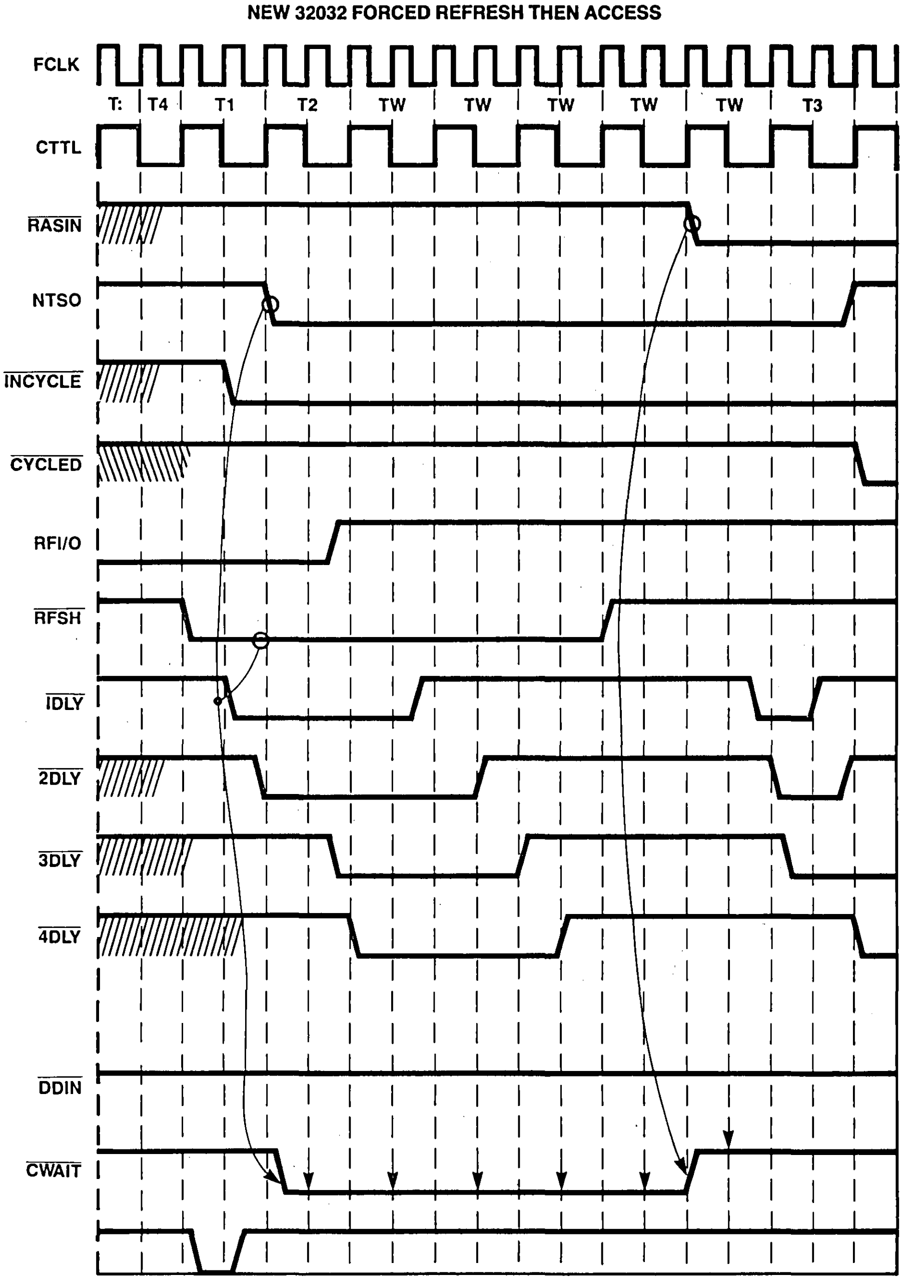


Figure 8.14.6 Timing Diagram; Forced Refresh Then Access

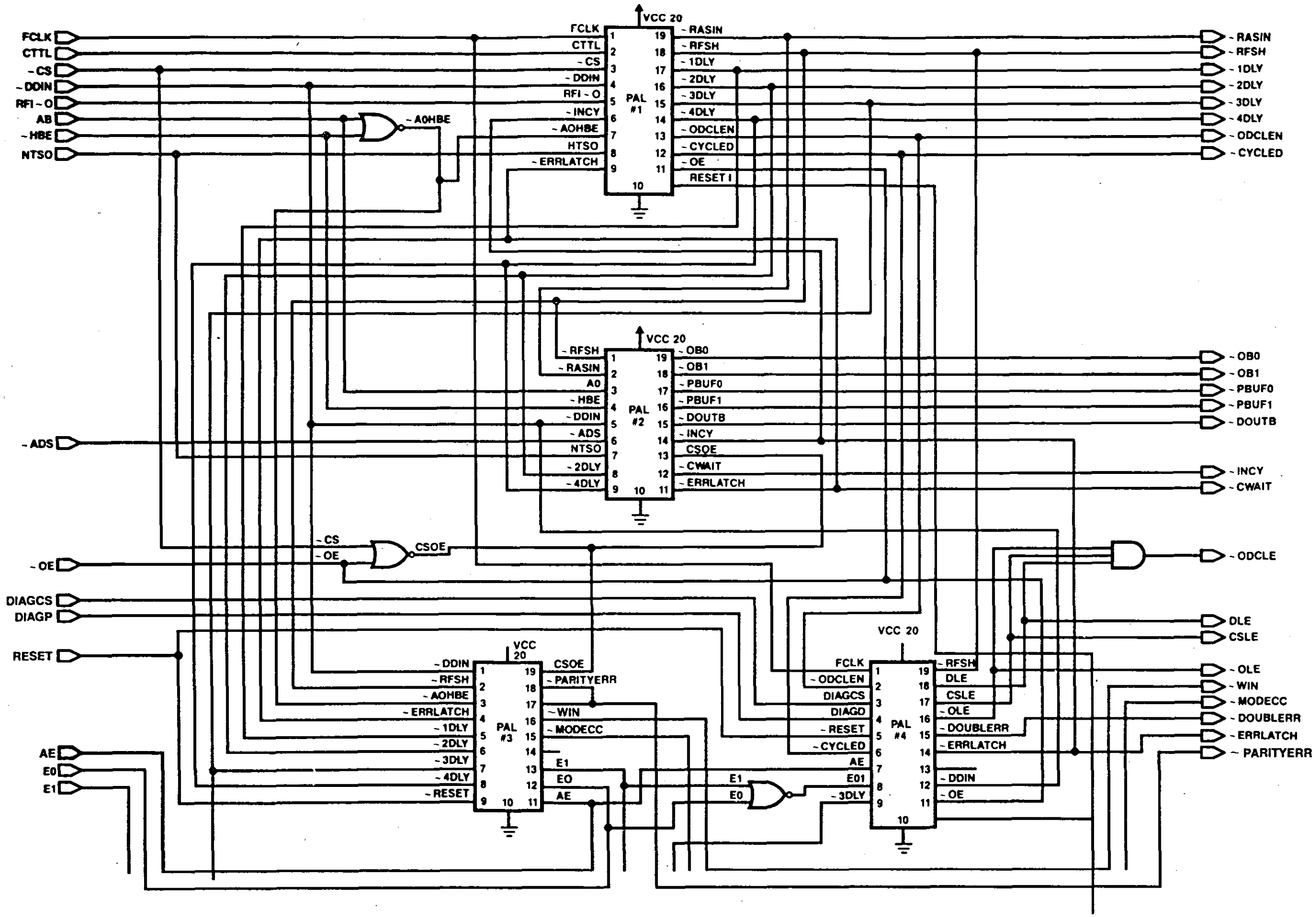


Figure 8.14.7 Simulation Circuit

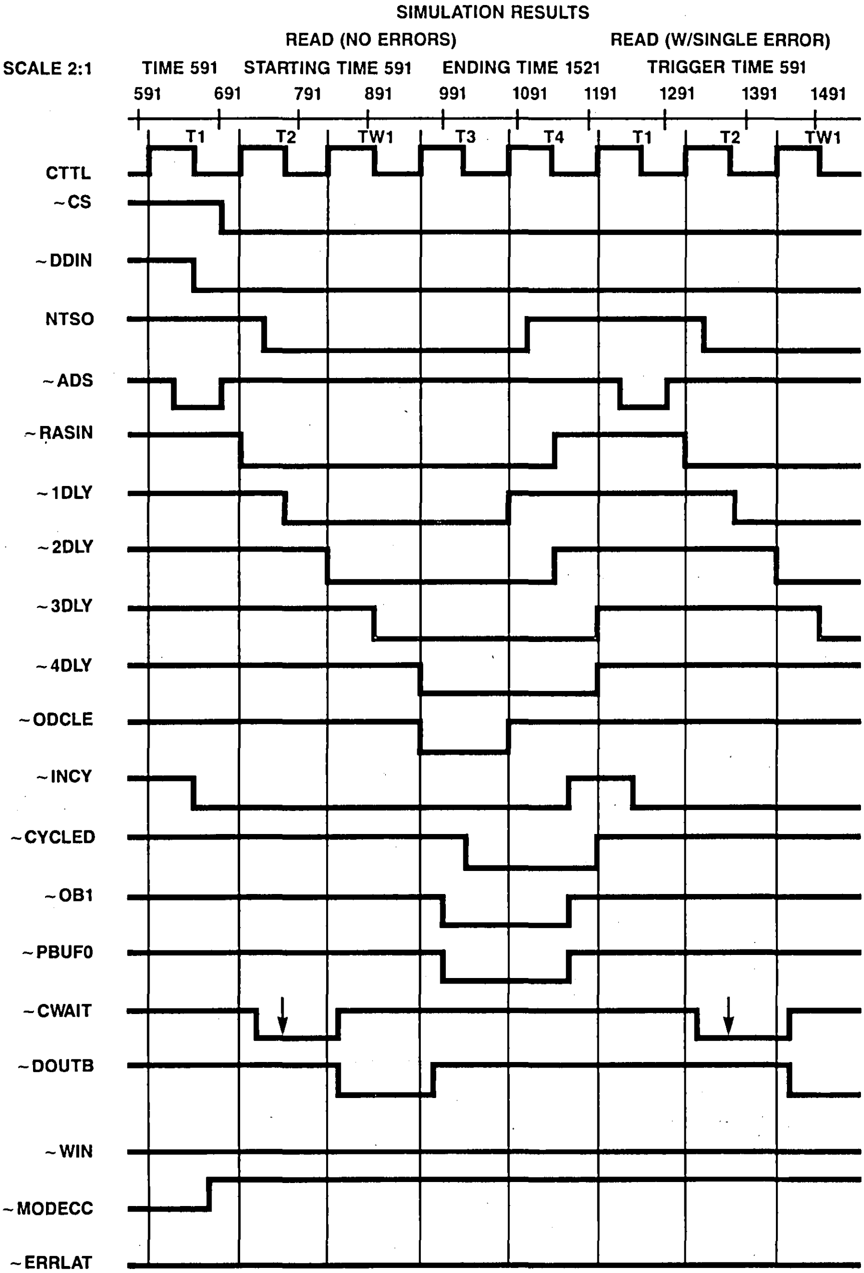


Figure 8.14.8 Simulation Timing Diagram; Read/Write Without Errors

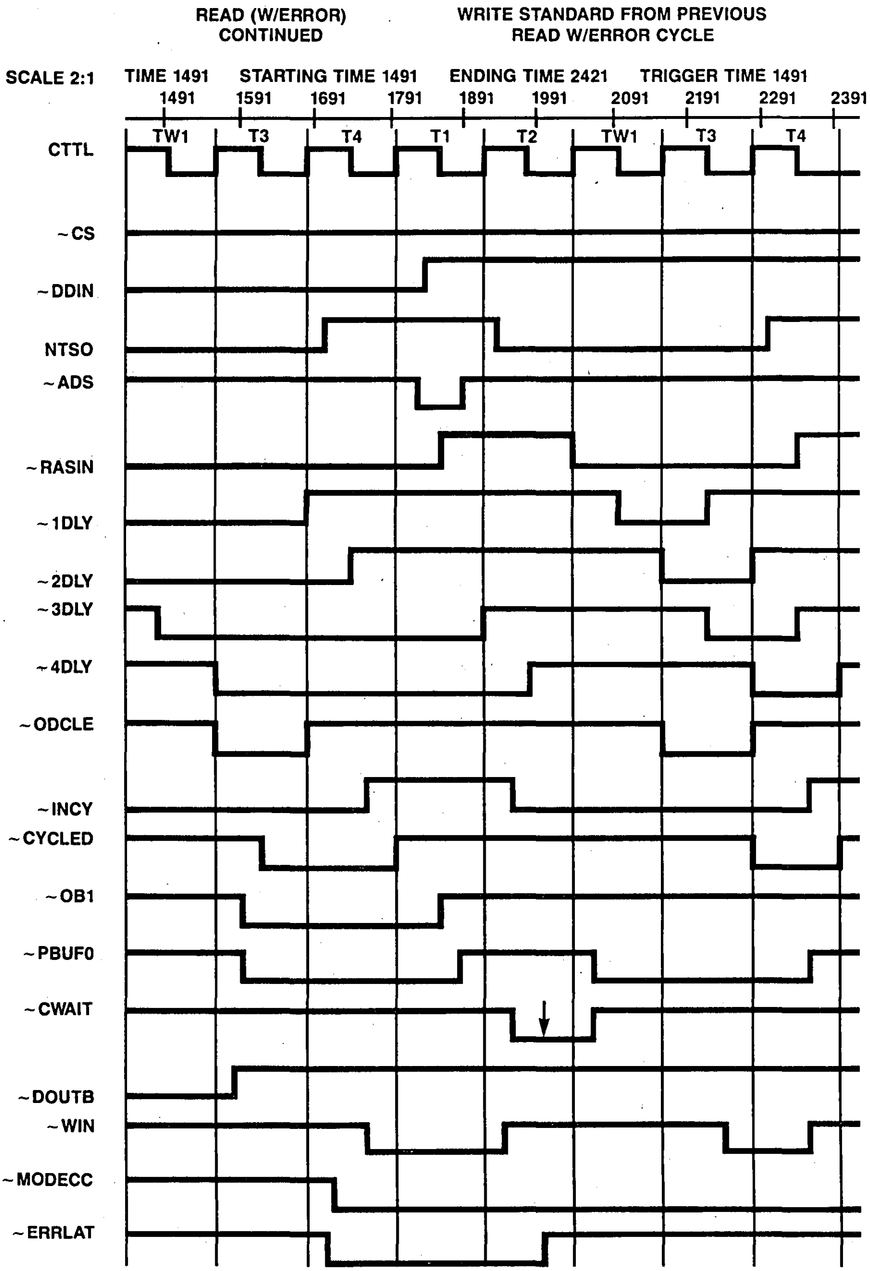


Figure 8.14.9 Simulation Timing Diagram; Read With Error and Write Cycle

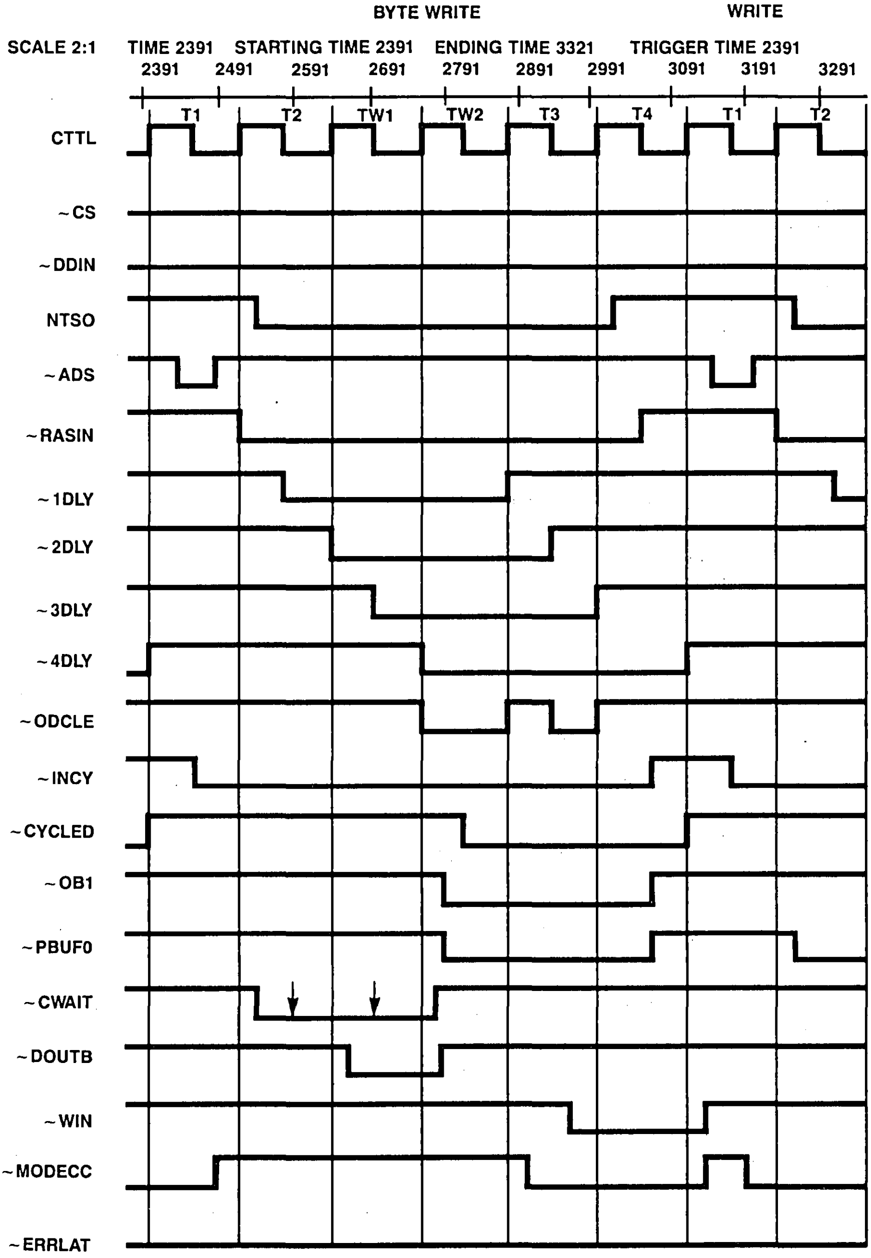


Figure 8.14.10 Simulation Timing Diagram; Byte Write

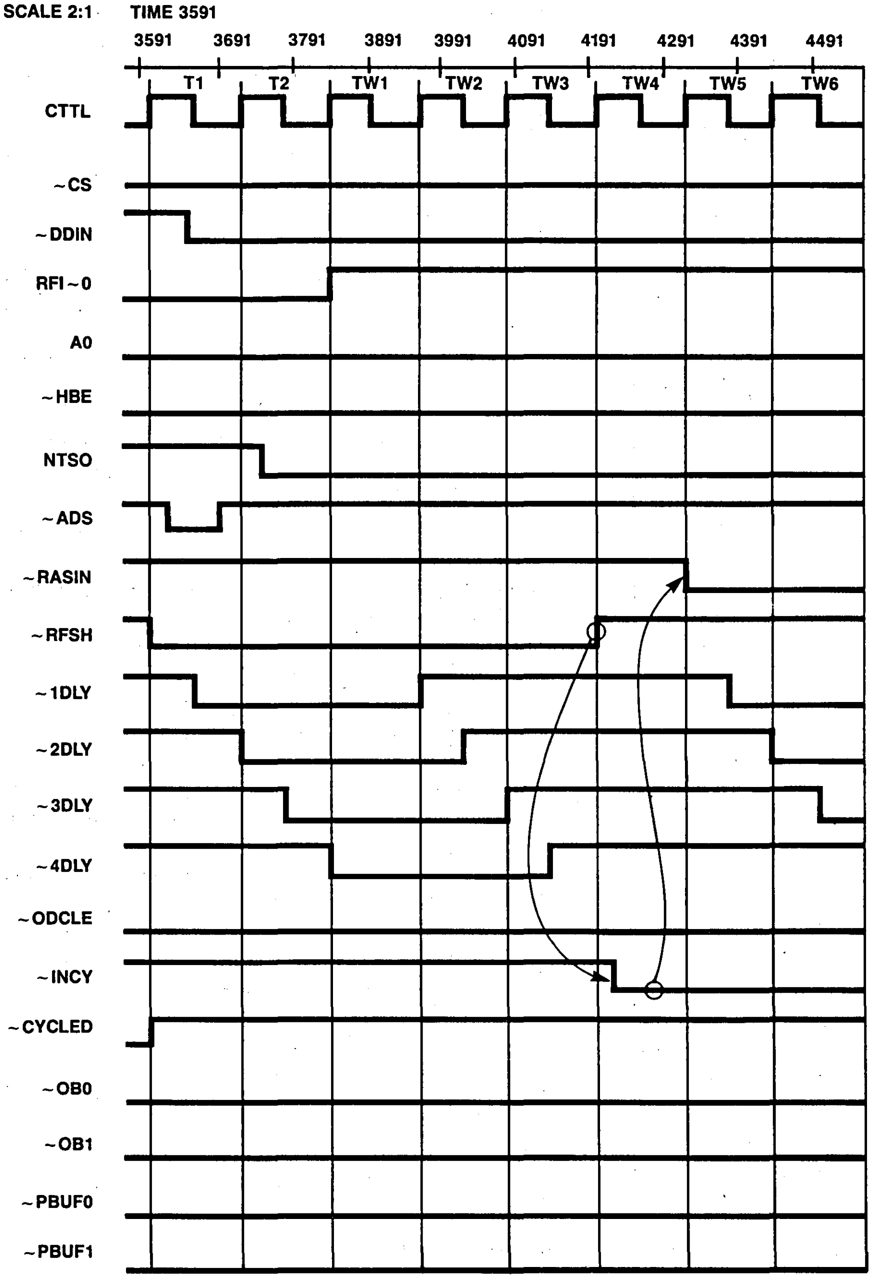


Figure 8.14.11 Simulation Timing Diagram; Forced Refresh Then Access

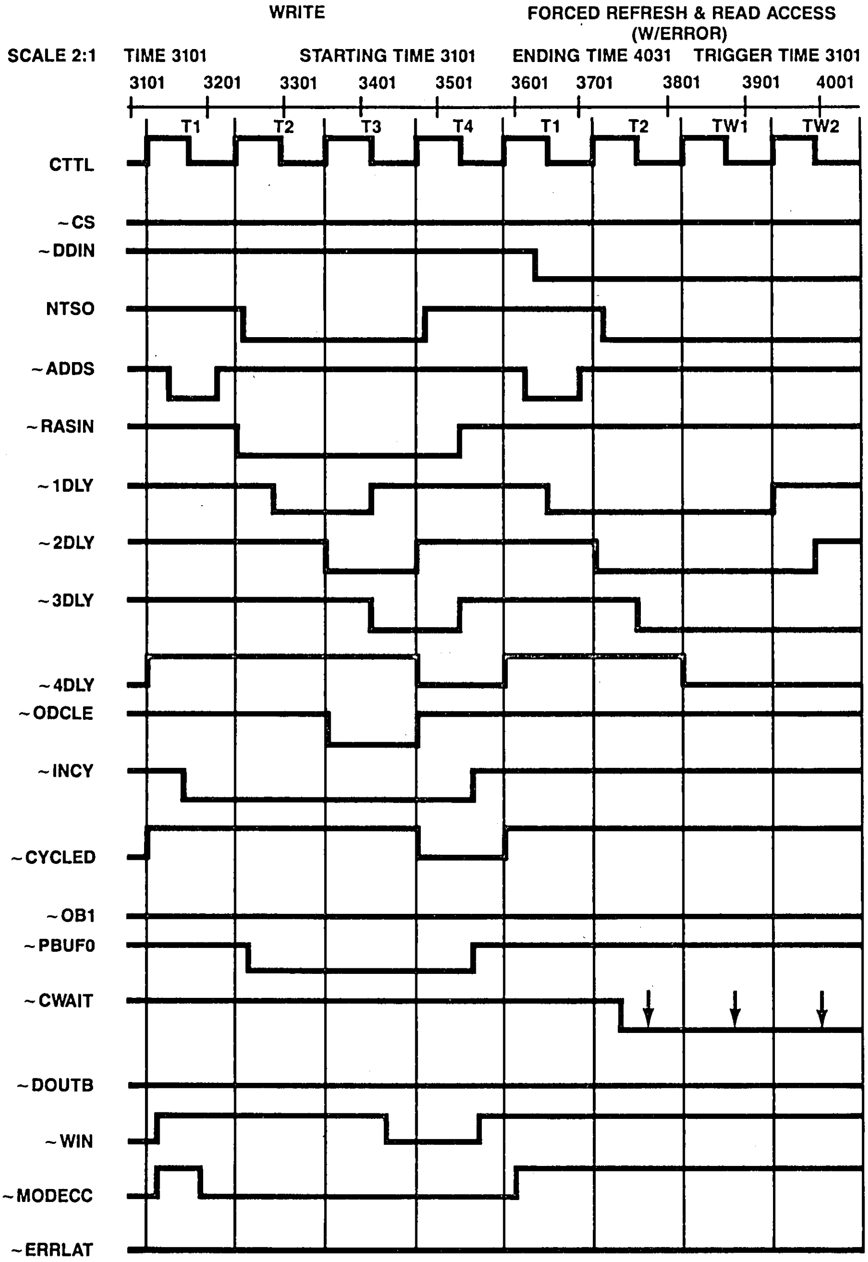


Figure 8.14.12 Simulation Timing Diagram; Write, Forced Refresh and Read Access

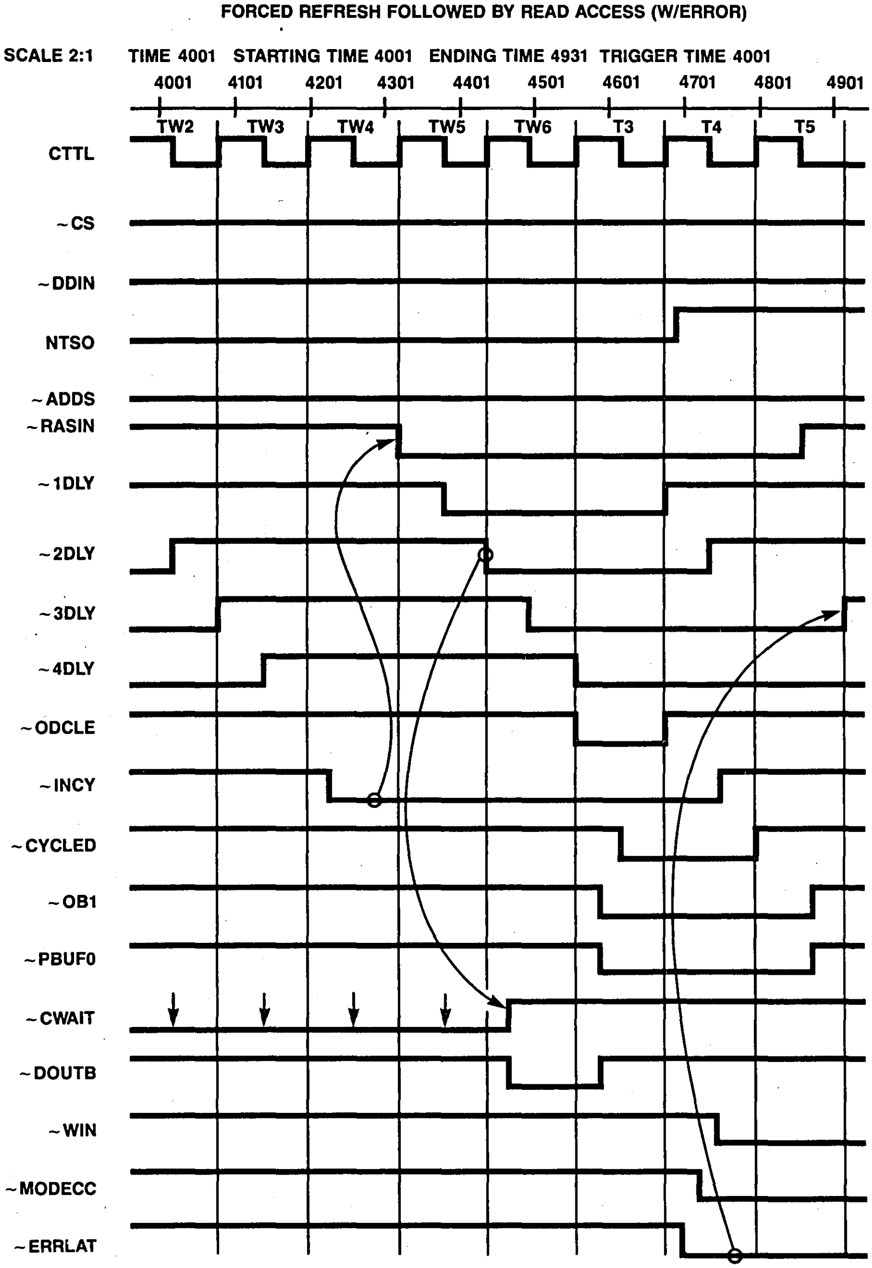


Figure 8.14.13 Simulation Timing Diagram; Forced Refresh Followed by Read Access (With Error)

PAL Device Number 1
This PAL Device is Part of a Four PAL Device Set Needed to Control the 32201, 8409, 8400 Interface

```
PAL16R8A

RFSH : =
/RFI0*/1DLY*/2DLY*/INCY*/CTTL +               ; RFSH in idle states or in long
RFSH*/RFI0 +                                  ; accesses of, other devices or
RFSH*1DLY +                                   ; at the beginning of an access
RFSH*4DLY

1DLY : =
RFSH*/RFI0 +                                  ; Start RFSH 1DLY
RFSH*1DLY*/4DLY +                             ; Hold RFSH 1DLY
RFSH*1DLY*CTTL +                              ; Extend RFSH 1DLY
/RFSH*RASIN*/2DLY*/3DLY*/4DLY +               ; For READs and WRITEs
/RFSH*CS*RASIN*/4DLY*DDIN +                   ; For READs
/RFSH*CS*RASIN*/4DLY*/DDIN*A0HBE +            ; For BYTE WRITEs
/RFSH*CS*1DLY*CTTL*DDIN +                     ; Extend 1DLY during READ
/RFSH*CS*1DLY*CTTL*/DDIN*A0HBE                ; Extend 1DLY during BYTE WRITEs

2DLY : =
1DLY*/4DLY +                                  ; For READs or WRITEs
1DLY*RFSH +                                   ; Extend for RFSH
/RFSH*CS*1DLY*DDIN +                          ; Extend for READ
/RFSH*CS*1DLY*/DDIN*A0HBE                     ; Extend for BYTE WRITE

3DLY : =
2DLY*/4DLY                                    ; For READs or WRITEs
2DLY*RFSH +                                   ; Extend for RFSH
/RFSH*CS*2DLY*DDIN +                          ; Extend for READ
/RFSH*CS*3DLY*ERRLATCH*RASIN +                ; Extend for READ with error
/RFSH*CS*2DLY*/DDIN*A0HBE                     ; Extend for BYTE WRITE

4DLY : =
3DLY*RASIN +                                  ; For READs or WRITEs
3DLY*RFSH +                                   ; Extend for RFSH
/RFSH*CS*3DLY*2DLY*DDIN +                     ; Extend for READ
/RFSH*CS*3DLY*ERRLATCH +                      ; Extend for READ with error
/RFSH*CS*RASIN*4DLY*/DDIN*A0HBE               ; Extend for BYTE WRITE

RASIN : =
/RFSH*INCY*/CYCLED*/4DLY*/CTTL +              ; Start /RASIN
/RFSH*CS*RASIN*DDIN*1DLY +                    ; READ cycle without error
/RFSH*CS*RASIN*DDIN*ERRLATCH*CYCLED +         ; READ cycle with error
/RFSH*CS*RASIN*DDIN*/CYCLED +                 ; WRITE cycle
/RFSH*CS*RASIN*/DDIN*3DLY*A0HBE +             ; BYTE WRITE cycle
/RFSH*INCY*/NTSO*/ERRLATCH*/4DLY*RASIN        ; Hidden RFSH, assume on
                                              ; four 'T' States.
```

```
CYCLED : =
/RFSH*1DLY*2DLY*3DLY*4DLY +                    ; BYTE WRITE or READ cycles
/RFSH*/DDIN*2DLY*3DLY*/A0HBE +                 ; WRITE cycle
CYCLED*CTTL +                                  ; End CYCLED
CYCLED*/NTSO +
CYCLED*RASIN*/DDIN*A0HBE                       ; End BYTE WRITE cycle

ODCLEN : =
CS*/RFSH*DDIN*RASIN*2DLY*                      ; READ and READ with error
/4DLY*/ERRLATCH +
CS*/RFSH*/DDIN*RASIN*/2DLY*                    ; WRITE cycle
/3DLY*/4DLY*/A0HBE +
CS*/RFSH*/DDIN*RASIN*2DLY*/4DLY*A0HBE +        ; BYTE WRITE cycle
CS*/RFSH*/DDIN*RASIN*1DLY*CYCLED*A0HBE         ; BYTE WRITE cycle
```

PAL Device Number 2

```
PAL16L8A

IF (CSOE) 0B0 =
/DOUTB*DDIN*4DLY*RASIN*/RFSH +                  ; READ or READ
                                                ; w/error
/DOUTB*AO*HBE*/DDIN*4DLY*RASIN*/RFSH            ; BYTE WRITE
                                                ; high byte

IF (CSOE) OB1 =
/DOUTB*DDIN*4DLY*RASIN*/RFSH +                  ; READ or READ
                                                ; w/error
/DOUTB*/AO*/HBE*/DDIN*4DLY*RASIN*/RFSH          ; BYTE WRITE
                                                ; low byte

IF (CSOE) PBUF0 =
/DOUTB*/AO*DDIN*4DLY*RASIN*/RFSH +              ; READ,
                                                ; READ/error
/DOUTB*/AO*/HBE*/DDIN*4DLY*RASIN*
/RFSH +                                         ; BYTE WRITE
/DOUTB*/AO*HBE*/DDIN*RASIN*/RFSH                ; Word WRITE

IF (CSOE) PBUF1 =
/DOUTB*IIBE*DDIN*4DLY*RASIN*/RFSH +             ; READ,,
                                                ; READ/error
/DOUTB*AO*HBE*/DDIN*4DLY*RASIN*
/RFSH +                                         ; BYTE WRITE
/DOUTB*/AO*HBE*/DDIN*RASIN*/RFSH                ; Word WRITE

IF (CSOE) DOUTB =
DDIN*/RFSH*2DLY*/4DLY +                         ; READ cycle
/AO*/HBE*/DDIN*/RFSH*2DLY*/4DLY +               ; BYTE WRITE
AO*HBE*/DDIN*/RFSH*2DLY*/4DLY                   ; BYTE WRITE
```

```
IF (VCC) INCY =
/RFSH*ADS*/4DLY +                          ; Start INCY
/RFSH*CSOE*/NTSO*/RASIN +                  ; Start INCY for access
                                           ; after forced refresh
                                           ; or READ w/error
INCY*/4DLY +                               ; Continue INCY
INCY*CSOE*/DDIN*RASIN +                    ; WRITE cycles
INCY*/CSOE*RASIN                           ; Non-/CS cycles

IF (CSOE) CWAIT =
RFSH*CSOE*/NTSO +                          ; Access in RFSH
/RFSH*CSOE*/NTSO*/RASIN +                  ; Access after
                                           ; forced refresh
/RFSH*DDIN*RASIN*2DLY*INCY*/4DLY +         ; READ cycle
/RFSH*/DDIN*/AO*/HBE*RASIN*/4DLY +         ; BYTE WRITE
/RFSH*/DDIN*AO*HBE*RASIN*/RDLY +           ; BYTE WRITE
/RFSH*INCY*ERRLATCH*/2DLY*/NTSO            ; Insert WAITS
                                           ; into the next
                                           ; cycle
```

PAL Device Number 3

```
PAL14L4A

WIN =
/RFSH*ERRLATCH*/2DLY*3DLY*4DLY*CSOE +      ; READ w/error
/RFSH*DDIN*3DLY*/AOHBE*CSOE +              ; Word WRITE
/RFSH*/DDIN*AOHBE*/2DLY*4DLY*CSOE          ; BYTE WRITE

MODECC =
/RFSH*ERRLATCH*/1DLY*4DLY*CSOE +           ; READ w/error
/RFSH*/DDIN*/AOHBE*CSOE +                  ; Word WRITE
/RFSH*/DDIN*AOHBE*/1DLY*4DLY*CSOE          ; BYTE WRITE

PARITYERR =
/RFSH*/DDIN*/RESET*4DLY*                   ; Parity error byte
/AE*EO*/E1*AOHBE*CSOE +                    ; "1" during WRITE
/RFSH*/DDIN*/RESET*4DLY*                   ; Parity error byte
/AE*/EO*E1*AOHBE*CSOE +                    ; "0" during WRITE
/RFSH*DDIN*/RESET*4DLY*
/AE*/EO*/E1*/AOHBE*CSOE +                  ; Parity error
PARITYERR*/RESET*CSOE                      ; both bytes
```

PAL Device Number 4

```
PAL16R6A

/DLE: =
ODCLEN +
DLE*DIAGD                                   ; Hold /DLE for
                                              diagnostics

/CSLE: =
ODCLEN +
CSLE*DIAGCS                                 ; Hold /CLSE for
                                              diagnostics

OLE: = ODCLEN

DOUBLERR: =
/RFSH*/DIAGCS*/DIAGD*/RESET*                ; Double bit error
OLE*CYCLED*AE*/E01 +                        ; during READs
DOUBLERR*/RESET                             ; or BYTE WRITEs

ERRLATCH: =
DDIN*OLE*CYCLED*/DIAGCS*/DIAGD*AE +         ; Error during READ
ERRLATCH*3DLY
```

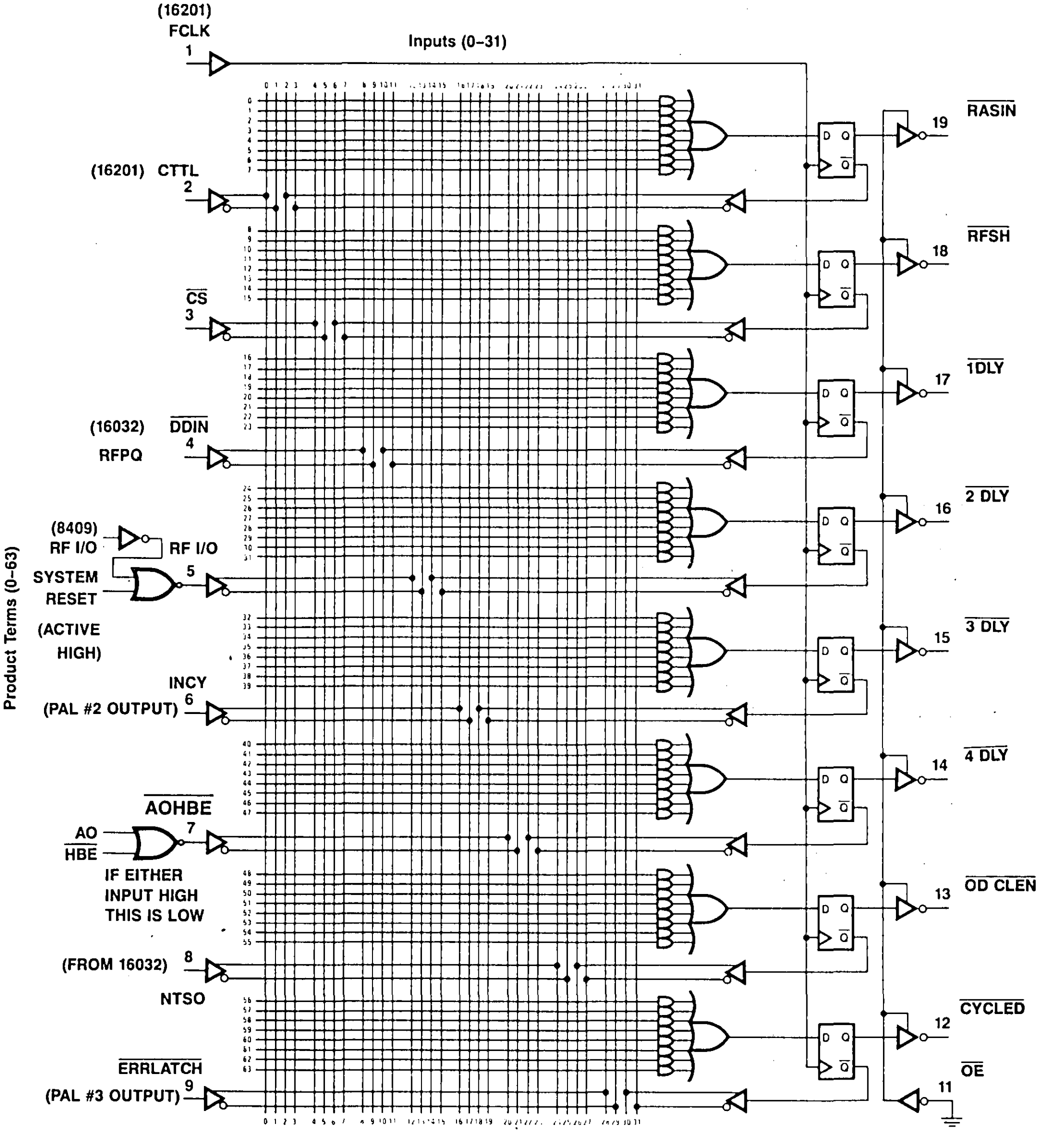


Figure 8.14.14 Logic Diagram of PAL Device #1

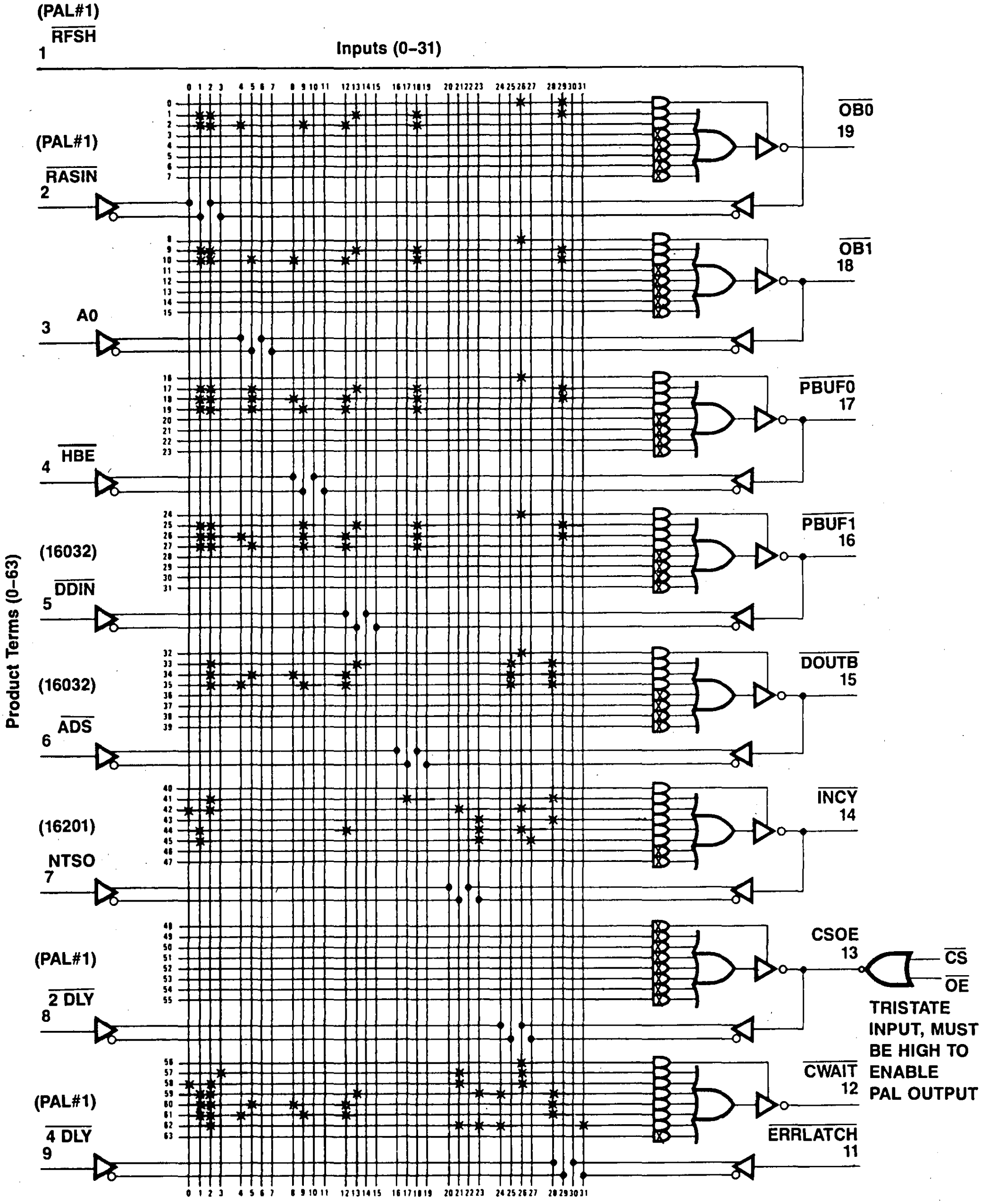


Figure 8.14.15 Logic Diagram of PAL Device #2

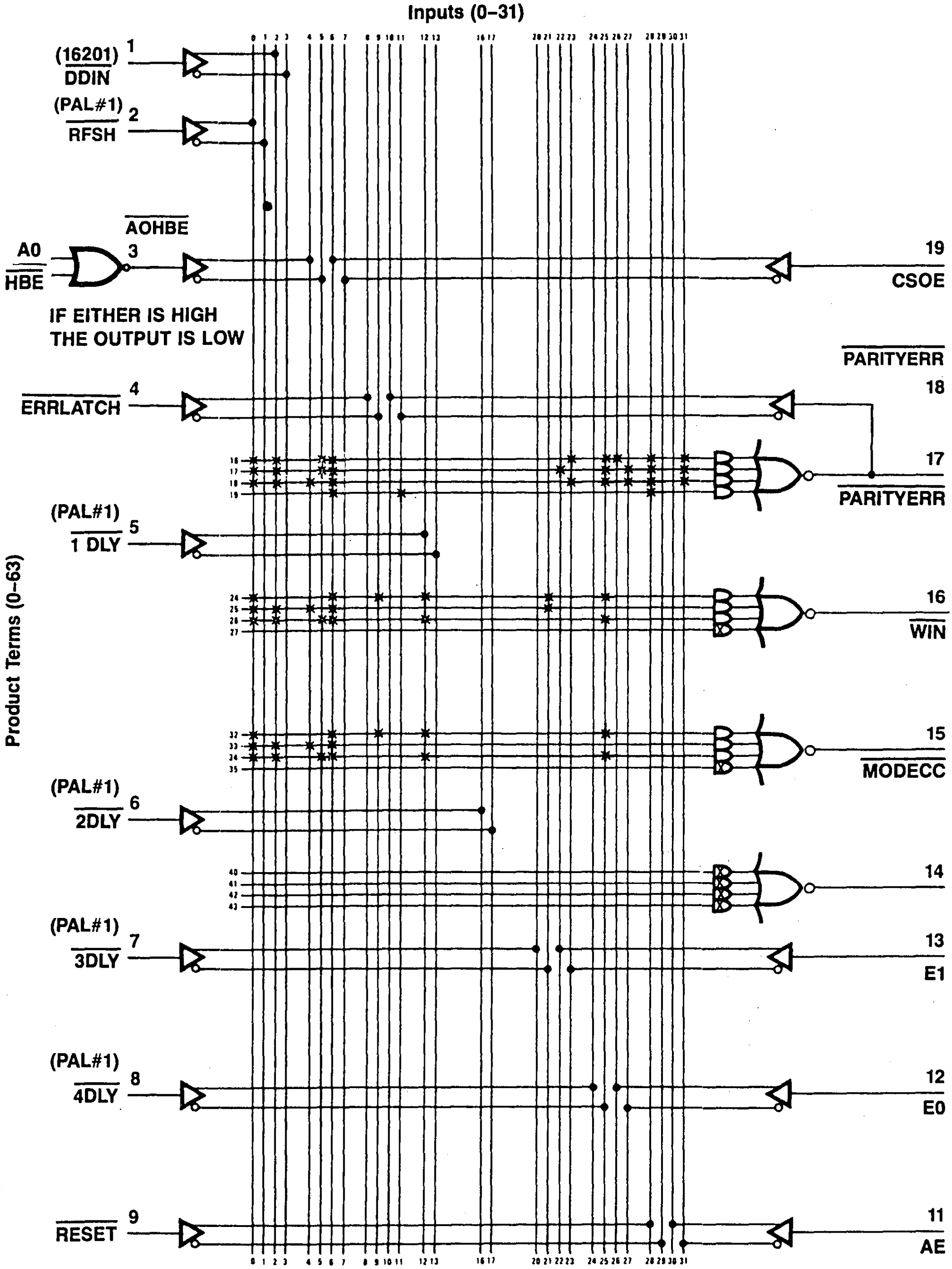


Figure 8.14.16 Logic Diagram of PAL Device #3

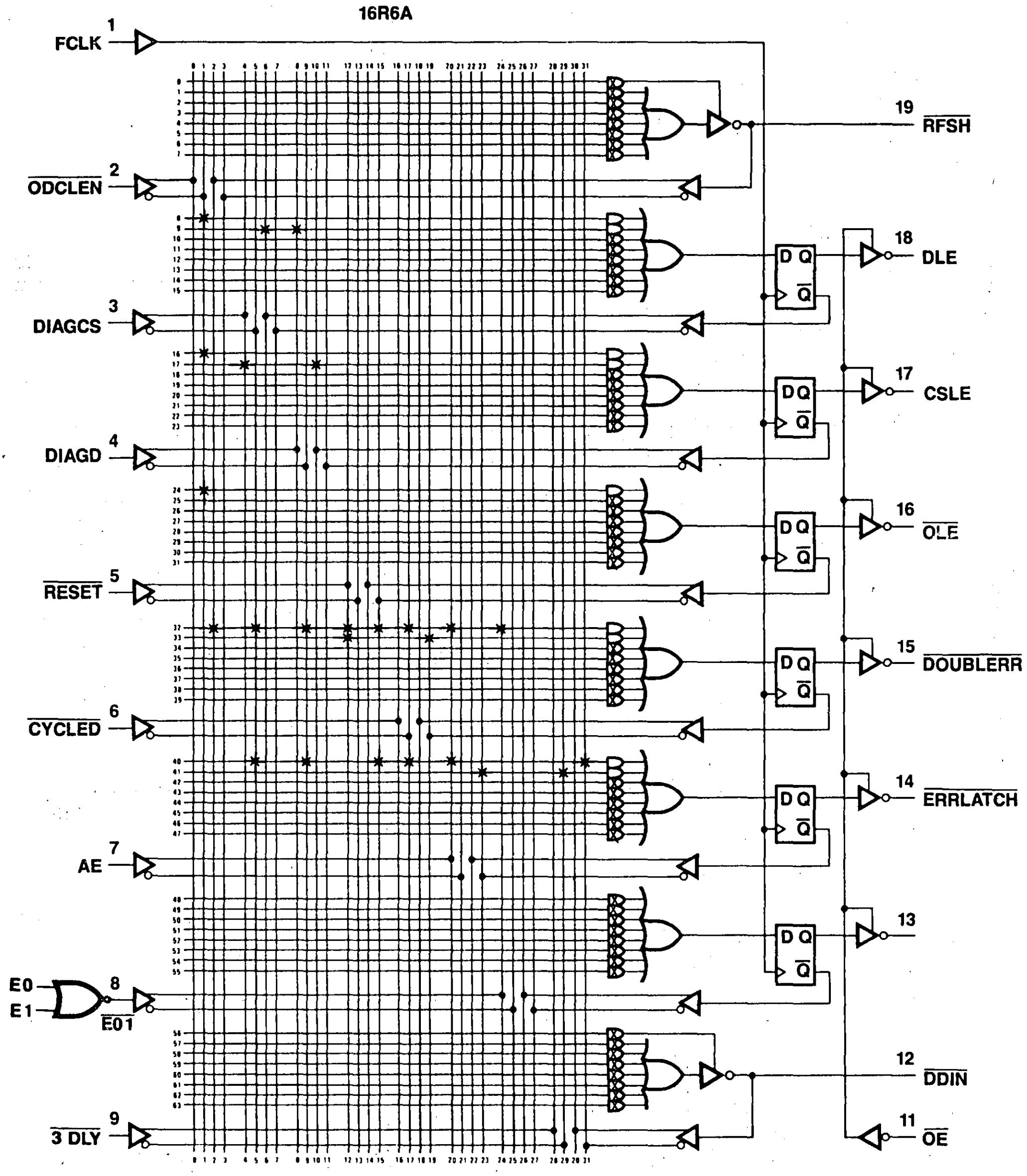


Figure 8.14.17 Logic Diagram of PAL Device #4

9

National Masked Logic (NML)

National Masked Logic (NML) was introduced to provide cost benefits of volume production to programmable logic users who have large volume applications for a given logic pattern. NML devices are mask-programmed and functionally tested in-house by National, thus relieving the customer of programming and testing the devices. Therefore, for these volume applications, the customer can simplify his production line and gain cost savings through the use of NML.

The NML option is available for all of National's programmable logic products. The NML products have the same data sheet specifications as the field-programmable products. The following are the procedures and guidelines involved in using NML.

9.1 NML PROCEDURE

The procedure for using NML is shown in Figure 9.1.1. When a customer has decided on the NML approach, the equations should be supplied to National for generation of programmed parts. These programmed devices are then sent to the customer for verification of the logic pattern in the application. After the logic has been verified by the customer in his circuit, National is notified. At that point orders for the masks are placed in-house at National. At the same time, the Test Engineering and Product Engineering departments prepare to test and qualify the product upon generation of first silicon. After successful testing and qualification, the product is released for routine production.

When the order is placed the customer will also be required to provide test vectors to functionally test the logic. When considering the use of NML, the customer should keep in mind the need for functional testing of the part. He should generate a sequence of test vectors that will test the logic functionality to meet his needs.

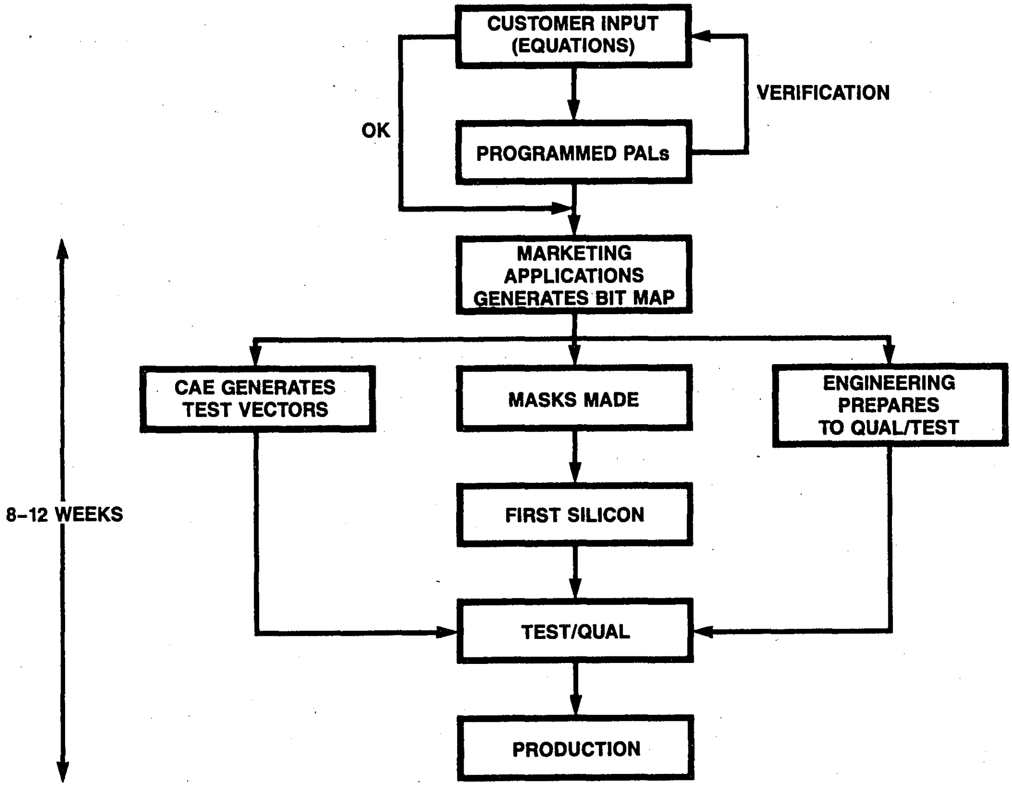

CUSTOMER INPUT (EQUATIONS)
VERIFICATION
OK
PROGRAMMED PALs
MARKETING APPLICATIONS GENERATES BIT MAP
CAE GENERATES TEST VECTORS
MASKS MADE
ENGINEERING PREPARES TO QUAL/TEST
8–12 WEEKS
FIRST SILICON
TEST/QUAL
PRODUCTION

Figure 9.1.1 NML-Procedure

9.2 NML GUIDELINES

In evaluating whether NML is an economic option for a certain application, it is important to keep in mind the following guidelines. The most important and somewhat obvious point is that the logic pattern must be verified and frozen. A minimum quantity for economic justification of NML is at least 10,000 units. At these volumes there is usually a nominal charge for mask generation. The lead time from the point at which the equations are verified to the point at which finished goods are shipped is 8–12 weeks.

NML users typically realize cost savings of between 10–40% over the cost of unprogrammed devices, depending on the volume and the device being used. Keep in mind that NML users do not have to incur programming and testing costs associated with unprogrammed devices.

10

Advantages of National's Programmable Logic Family

National Semiconductor has taken leadership of the programmable logic market through commitments in technology, quality, customer service and support, and by offering a broad product line. In addition, National is also committed to continuing developments in software leading to automated design with programmable logic products.

10.1 TECHNOLOGY

Through innovations in circuit design and process technology, National was the first to introduce the fastest PAL devices, thus clearly establishing itself as the leading technology house for programmable logic devices. The technology used is the proprietary oxide isolated OXISS process that offers higher integration than other bipolar processes and also offers improved performance. The advantages of this superior technology are being harnessed to produce ECL programmable logic devices that will offer speeds at 6 ns. Furthermore, National is also pursuing a major development program to introduce CMOS programmable logic devices.

10.2 BROAD PRODUCT LINE

National's leading technology position has resulted in the broad TTL product line that is currently available. This product line offers a variety of speed, power, and density options as evidenced by the product line description in Chapter 4. For the future, National will offer a broader spectrum of speed and power options through CMOS and ECL devices. More options in the TTL family of programmable products are also forthcoming. Some of the forthcoming features are FPLA-type structures, higher densities, improved testability through register preloads, and scan registers.

To complete the product line, National is also committed to software development and support. PLAN is the first step toward meeting that commitment.

10.3 CUSTOMER SERVICE AND SUPPORT

Within the field offices, National has fully equipped and trained Field Application Engineers (FAEs) who can support customers in designing with programmable logic. The FAEs also have the software and the development systems at their disposal to fully support the customer. In addition, the factory applications and engineering staff are also available to support the customer in programmable logic-based designs.

Customer training seminars are also given, as part of National's service, to inform and train customers on programmable logic products and their applications.

11

Data Sheets

11.1 PAL DEVICE DATA SHEETS

The PAL device data sheets are broken down into two main sections: 20-pin PALs and 24-pin PALs, and within each section the various speed/power groups are shown separately.

Description

The PAL device family utilizes National's Schottky TTL process and bipolar PROM fusible-link technology to provide user-programmable logic to replace conventional SSI/MSI gates and flip-flops. Typical chip count reduction gained by using PAL devices is greater than 4:1.

The family lets the systems engineer customize his chip by opening fusible links to configure AND and OR gates to perform desired logic functions. Complex interconnections that previously required time-consuming layouts are thus transferred from PC board to silicon where they can be easily modified during prototype checkout or production.

The PAL device transfer function is the familiar Sum-of-Products with 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 device family offers these options:

- Variable input/output ratio.
- Programmable TRI-STATE® outputs.
- Registers and feedback.

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 that are loaded on the low-to-high transition of the clock. PAL device 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 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 make verification difficult. This feature gives the user a proprietary circuit that is very difficult to copy.

Features

- Programmable replacement for SSI and MSI TTL Logic.
- Simplifies prototyping and board layout.
- Skinny DIP packages.
- Reliable titanium-tungsten fuses.
- Available in standard, low power and high speed versions.

Part No.	No. of Inputs	No. of Outputs	No. of I/Os	No. of Registers	Output Polarity	Functions
10H8	10	8			AND-OR	AND-OR Array
12H6	12	6			AND-OR	AND-OR Array
14H4	14	4			AND-OR	AND-OR Array
16H2	16	2			AND-OR	AND-OR Array
10L8	10	8			AND-NOR	AND-OR-Invert Array
12L6	12	6			AND-NOR	AND-OR-Invert Array
14L4	14	4			AND-NOR	AND-OR-Invert Array
16L2	16	2			AND-NOR	AND-OR-Invert Array
16C1	16	1			AND-OR/NOR	AND-OR/AND-OR-Invert Array
16L8	10	8	6		AND-NOR	AND-OR-Invert Array
16R8	8	8		8	AND-OR	AND-OR-Invert Register
16R6	8	8	2	6	AND-OR	AND-OR-Invert Register
16R4	8	8	4	4	AND-OR	AND-OR-Invert Register

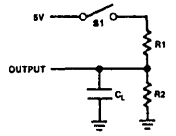
Table 11.1.1 20-Pin PAL Devices

Part No.	No. of Inputs	No. of Outputs	No. of I/Os	No. of Registers	Output Polarity	Functions
12L10	12	10			AND-NOR	AND-OR Invert Gate Array
14L8	14	8			AND-NOR	AND-OR Invert Gate Array
16L6	16	6			AND-NOR	AND-OR Invert Gate Array
18L4	18	4			AND-NOR	AND-OR Invert Gate Array
20L2	20	2			AND-NOR	AND-OR Invert Gate Array
20L8	14	2	6		AND-NOR	AND-OR Invert Gate Array
20L10	12	2	8		AND-NOR	AND-OR Invert Gate Array
20R8	12	8		8	AND-NOR	AND-OR Invert w/Registers
20R6	12	6	2	6	AND-NOR	AND-OR Invert w/Registers
20R4	12	4	4	4	AND-NOR	AND-OR Invert w/Registers
20X10	10	10		10	AND-NOR	AND-OR-XOR Invert w/Registers
20X8	10	8	2	8	AND-NOR	AND-OR-XOR Invert w/Registers
20X4	10	4	6	4	AND-NOR	AND-OR-XOR Invert w/Registers

Table 11.1.2 24-Pin PAL Devices

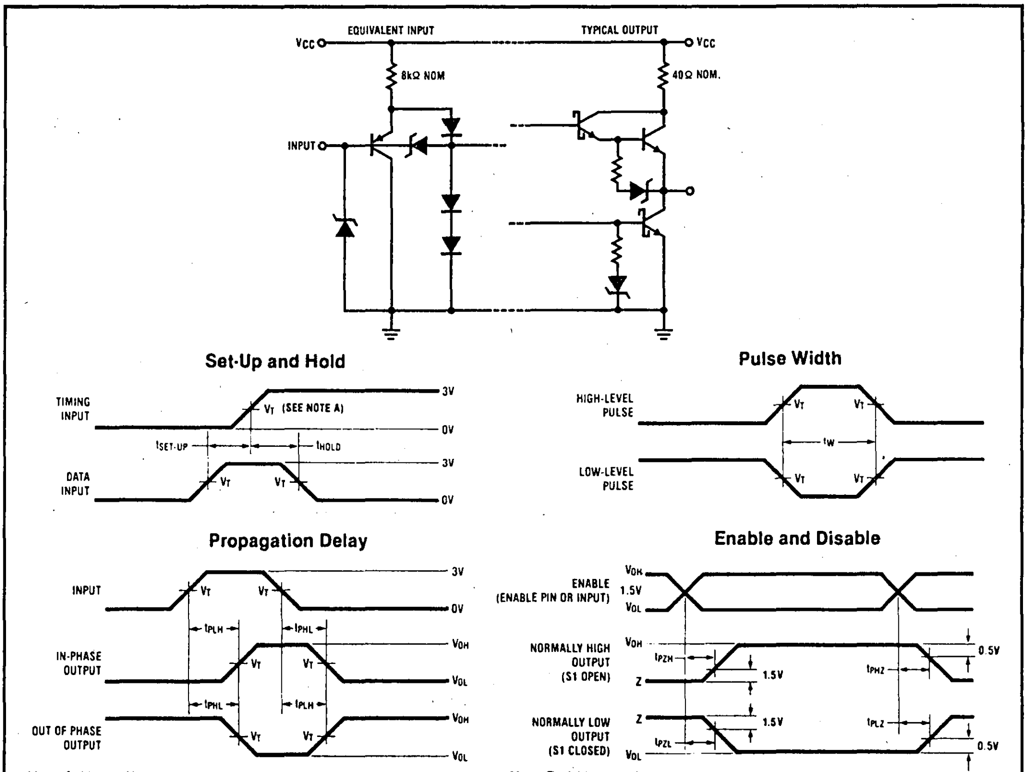
	Operating	Programming
Supply Voltage, V_{CC}	7V	12V
Input Voltage	5.5V	12V
Off-State Output Voltage	5.5V	12V
Storage Temperature Range	−65°C to +150°C	

Table 11.1.3 Absolute Maximum Ratings



MEDPAL		SMPAL
MIL	COM'L	
R1 = 390	R1 = 200	R1 = 560
R2 = 750	R2 = 390	R2 = 1.1k

Table 11.1.4 Standard Test Load



Note A: $V_T = 1.5V$
Note B: C_L includes probe and jig capacitance.
Note C: In the examples above, the phase relationships between inputs and outputs have been chosen arbitrarily.

Note D: All input pulses are supplied by generations having the following characteristics:
$P_{RR} = 1 MHz$, $Z_{OUT} = 50 Ω$.

Figure 11.1.1 Test Waveforms and Schematics of Inputs and Outputs

10H8, 12H6, 14H4, 16H2, 16C1, 10L8, 12L6, 14L4, 16L2 Recommended Operating Conditions

Symbol	Parameter	Military			Commercial			Unit
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply voltage	4.5	5.0	5.5	4.75	5.00	5.25	V
I_{OH}	High-level output current			− 2.0			− 3.2	mA
I_{OL}	Low-level output current			8			8	mA
T_A	Operating free air temperature	− 55		125	0		75	°C

Electrical Characteristics

Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2			V
V_{IL}	Low-level input voltage				0.8	V
V_{IC}	Input clamp voltage	V_{CC} = MIN I_I = − 18 mA			− 1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN V_{IH} = 2V V_{IL} = 0.8V I_{OH} = MAX	2.4			V
V_{OL}	Low-level output voltage	V_{CC} = MIN V_{IH} = 2V V_{IL} = 0.8V I_{OL} = MAX			0.5	V
I_I	Input current at maximum input voltage	V_{CC} = MAX V_I = 5.5 V			1.0	mA
I_{IH}	High-level input current	V_{CC} = MAX V_I = 2.4V			25	μA
I_{IL}	Low-level input current	V_{CC} = MAX V_I = 0.4V			− 250	μA
I_{OS}	Short-circuit output current	V_{CC} = MAX V_O = 0V	− 30		− 130	mA
I_{CC}	Supply current	V_{CC} = MAX		55	90	mA

Switching Characteristics

Over Recommended Ranges of Temperature and V_{CC}

Symbol	Parameter	Test Conditions†† R1 = 560Ω R2 = 1.1 kΩ	Military T_A = − 55° to + 125°C V_{CC} = 5.0V ± 10%			Commercial T_A = 0° to 75°C V_{CC} = 5.0V ± 5%			Unit
			Min	Typ	Max	Min	Typ	Max	
t_{PD}	From any input to any output	C_L = 15pF		25	45		25	35	ns

Table 11.1.5 AC and DC Specifications for 20-Pin Standard Small PAL Devices

16L8, 16R8, 16R6, 16R4
Recommended Operating Conditions

Symbol	Parameter	Military			Commercial			Unit
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply voltage	4.5	5.0	5.5	4.75	5.00	5.25	V
I_{OH}	High-level output current			− 2.0			− 3.2	mA
I_{OL}	Low-level output current			12			24	mA
T_A	Operating free air temperature	− 55		125*	0		75	°C

*Operating Case Temperature only, T_C = 125°C

Electrical Characteristics
Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2			V
V_{IL}	Low-level input voltage				0.8	V
V_{IC}	Input clamp voltage	V_{CC} = MIN I_I = -18 mA			− 1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN V_{IH} = 2V V_{IL} = 0.8V I_{OH} = MAX	2.4			V
V_{OL}	Low-level output voltage	V_{CC} = MIN V_{IH} = 2V V_{IL} = 0.8V I_{OL} = MAX			0.5	V
I_{OZH}	Off-state output current high-level voltage applied	V_{CC} = MAX, V_{IH} = 2V, V_O = 2.4V, VIL = 0.8V			100	μA
I_{OZL}	Off-state output current low-level voltage applied	V_{CC} = MAX, V_{IH} = 2V V_O = 0.4V, VIL = 0.8V			− 100	μA
I_I	Input current at maximum input voltage	V_{CC} = MAX V_I = 5.5 V			1.0	mA
I_{IH}	High-level input current	V_{CC} MAX V_I = 2.4V			25	μA
I_{IL}	Low-level input current	V_{CC} MAX V_I = 0.4V			− 250	μA
I_{OS}	Short-circuit output current	V_{CC} = MAX V_O = 0V	− 30		− 130	mA
I_{CC}	Supply Current 16L8	V_{CC} = MAX		140	180	mA
	16R4,16R6,16R8			150	180	

Table 11.1.6 AC and DC Specifications for 20-Pin Standard, Medium PAL Devices

Switching Characteristics

Over Recommended Ranges of Temperature and V_{CC}

Symbol	Parameter		Test Conditions†† R1, R2	Military			Commercial			Unit
				$T_A = -55°$ to $+125°C$ $V_{CC} = 5.0V \pm 10\%$			$T_A = 0°$ to $75°C$ V_{CC} $5.0V \pm 5\%$			
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input to output		$C_L = 50pF$		25	45		25	35	ns
t_{PD}	Clock to output				15	25		15	25	ns
t_{PZX}	Pin 11 to output enable				15	25		15	25	ns
t_{PXZ}	Pin 11 to output disable		$C_L = 5pF$		15	25		15	25	ns
t_{PZX}	Input to output enable		$C_L = 50pF$		25	45		25	35	ns
t_{PXZ}	Input to output disable		$C_L = 5pF$		25	45		25	35	ns
t_w	Width of clock	High		25			25			ns
		Low		25			25			
t_{su}	Setup time	16R8, 16R6, 16R4		45			35			ns
		16X4, 16A4								
t_h	Hold time			0	− 15		0	− 15		ns

††See Standard Test Load and Definition of Waveforms

Table 11.1.6 AC and DC Specifications for 20-Pin Standard, Medium PAL Devices (Cont.)

10H8A, 12H6A, 14H4A, 16H2A, 16C1A, 10L8A, 12L6A, 14L4A, 16L2A
Recommended Operating Conditions

Symbol	Parameter	Military			Commercial			Units
		Min	Type	Max	Min	Typ	Max	
V_{CC}	Supply Voltage	4.5	5	5.5	4.75	5	5.25	V
I_{OH}	High Level Output Current			−2			−3.2	mA
I_{OL}	Low Level Output Current			12			24	mA
T_A	Operating Free-Air Temperature				0		75	°C
T_C	Operating Case Temperature			125		°C		

Electrical Characteristics
Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{IH}	High Level Input Voltage		2			V
V_{IL}	Low Level Input Voltage				0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = −18mA			−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min., V_{IH} = 2V V_{IL} = 0.8V, I_{OH} = Max.	2.4			V
V_{OL}	Low Level Output Voltage	V_{CC} = Min., V_{IH} = 2V V_{IL} = 0.8V, I_{OL} = Max.			0.5	V
I_I	Input Current at Maximum Input Voltage	V_{CC} = Max., V_I = 5.5V			1	mA
I_{IH}	High Level Input Current	V_{CC} = Max., V_I = 2.4V			25	μA
I_{IL}	Low Level Input Current	V_{CC} = Max., V_I = 0.4V			−0.25	μA
I_{OS}	Short Circuit Output Current	V_{CC} = 5V	−30		−130	mA
I_{CC}	Supply Current	V_{CC} = Max.	55		90	mA

Switching Characteristics
Over Recommended Ranges of Temperature and V_{CC}
Military: T_A = −55°C to +125°C*, V_{CC} = 5V ± 10%
Commercial: T_A = 0 to 75°C, V_{CC} = 5V ± 5%

Symbol	Parameter	Test Conditions	Military			Commercial			Unit
			Min.	Typ.	Max.	Min.	Typ.	Max.	
t_{PD}	From any Input to any Output	C_L = 15pF		15	30		15	25	ns
	16C1A	C_L = 15pF			35			30	ns

Table 11.1.7 AC and DC Specifications for 20-Pin Fast, Small PAL Devices

16L8A, 16R8A, 16R6A, 16R4A
Recommended 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_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	16R8A, 16R6A, 16R4A	30	16		25	16		ns
t_h	Hold Time		0	−10		0	−10		ns
T_A	Operating Free-Air Temperature		−55			0	25	75	°C
T_C	Operating Case Temperature				125				°C

Electrical Characteristics

Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
V_{IH}	High Level Input Voltage			2			V
V_{IL}	Low Level Input Voltage					0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = −18mA			−0.8	−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} = −2mA MIL	2.4	2.8		V
			I_{OH} = −3.2mA COM				
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = 12mA MIL		0.3	0.5	V
			I_{OL} = 24mA*** COM				
I_{OZH}	Off-state Output Current	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 2.4V			100	μA
I_{OZL}			V_O = 0.4V			−100	μA
I_I	Maximum Input Current	V_{CC} = Max., V_I = 5.5V				1	mA
I_{IH}	High Level Input Current	V_{CC} = Max., V_I = 2.4V				25	μA
I_{IL}	Low Level Input Current	V_{CC} = Max., V_I = 0.4V			−0.02	−0.25	mA
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

† I_{CC} = Max. at minimum temperature.

Table 11.1.8 AC and DC Specifications for 20-Pin Fast Medium PAL Devices

Switching Characteristics

Over Recommended Ranges of Temperature and V_{CC}
Military: $T_A = -55°C$ to $+125°C^*$, $V_{CC} = 5V \pm 10\%$
Commercial: $T_A = 0$ to $75°C$, $V_{CC} = 5V \pm 5\%$

Symbol	Parameter	Test Conditions†† R1, R2	Military			Commercial			Unit
			Min.	Typ.	Max.	Min.	Typ.	Max.	
t_{PD}	Input or Feedback to Output	CL = 50pF		15	30		15	25	ns
t_{CLK}	Clock to Output or Feedback			10	20		10	15	ns
t_{PZX}	Pin 11 to Output Enable			10	25		10	20	ns
t_{PXZ}	Pin 11 to Output Disable	$C_L = 5pF$		11	25		11	20	ns
t_{PZX}	Input to Output Enable	$C_L = 50pF$		10	30		10	25	ns
t_{PXZ}	Input to Output Disable	$C_L = 5pF$		13	30		13	25	ns
f_{MAX}	Maximum Frequency		20	30		25	30		ns

††See Waveforms, Test Load on pg. 24-21.

Table 11.1.8 AC and DC Specifications for 20-Pin Fast Medium PAL Devices (Cont.)

16L8B, 16R8B, 16R6B, 16R4B

Recommended Operating Conditions

Symbol	Parameter		Military			Commercial			Units
			Min	Typ	Max	Min	Typ	Max	
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}	Setup Time from Input or Feedback to Clock		50	25		35	25		ns
t_h	Hold Time		0	−5		0	−5		ns
T_A	Operating Free-Air Temperature		−55		125	0	25	75	°C
T_C	Operating Case Temperature				125				°C

Electrical Characteristics

Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions		Min	Typ	Max	Units
V_{IH}*	High Level Input Voltage			2			V
V_{IL}*	Low Level Input Voltage					0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = −18mA			−0.8	−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} = −2mA MIL	2.4	3.4		V
			I_{OH} = −3.2mA COM				
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = 12mA MIL	3.0	0.5		V
			I_{OL} = 24mA COM				
I_{OZH}	Off-State Output Current †	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 2.4V			100	μA
I_{OZL}			V_O = 0.4V			−100	μA
I_I	Maximum Input Current	V_{CC} = Max., V_I = 5.5V				1	mA
I_{IH}	High Level Input Current †	V_{CC} = Max., V_I = 2.4V				25	μA
I_{IL}	Low Level Input Current †	V_{CC} = Max., V_I = 0.4V	Except pins 1 & 11		−0.04	−0.25	mA
			Pins 1 & 11			−0.4	
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

† 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 pin 10 on the device and include 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.

*** Pins 1 and 11 may be raised to 20V max.

Table 11.1.9 AC and DC Specifications for 20-Pin Ultra High-Speed, Medium PAL Devices

Switching Characteristics

Over Recommended Ranges of Temperature and V_{CC}
Military: $T_A = -55°C$ to $+125°C$*, $V_{CC} = 5V \pm 10\%$
Commercial: $T_A = 0$ to $75°C$, $V_{CC} = 5V \pm 5\%$

Sym	Parameter		Test Conditions	Military			Commercial			Units
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feed-back to Output	16R6B 16R4B 16L8B	$R_1 = 200\Omega$ $R_2 = 390\Omega$		11	20		11	15	ns
t_{CLK}	Clock to Output or Feedback				8	15		8	12	ns
t_{PZX}	Pin 11 to Output Enable				10	20		10	15	ns
t_{PXZ}	Pin 11 to Output Disable				10	20		10	15	ns
t_{PZX}	Input to Output Enable	16R6B 16R4B 16L8B			11	25		11	20	ns
t_{PXZ}	Input to Output Disable	16R6B 16R4B 16L8B			11	20		11	15	ns
f_{MAX}	Maximum Frequency	16R8B 16R6B 16R4B		30	50		40	50		MHz

Table 11.1.9 AC and DC Specifications for 20-Pin Ultra High-Speed, Medium PAL Devices (Cont.)

10H8A2, 12H6A2, 14H4A2, 16H2A2, 16C1A2, 10L8A2, 12L6A2, 14L4A2, 16L2A2 Recommended Operating Conditions

Symbol	Parameter		Military			Commercial			Units
			Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage		4.5	5	5.5	4.75	5	5.25	V
I_{OH}	High-Level Output Current				−2.0			−3.2	mA
I_{OL}	Low-Level Output Current	Small PAL †††			8			8	mA
		Medium PAL			12			24	
T_A	Operating Free-Air Temperature		−55		125	0	25	75	°C

Electrical Characteristics Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions		Min	Typ	Max	Units
V_{IH}*	High Level Input Voltage			2			V
V_{IL}*	Low Level Input Voltage					0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = −18 mA			−0.8	−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} = −2 mA MIL	2.4	2.8		V
			I_{OH} = −3.2 mA COM				
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = Max.		0.3	0.5	V
I_{OZH}	Off-State Output Current†	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 2.4V			100	μA
I_{OZL}			V_O = 0.4V			−100	μA
I_I	Maximum Input Current	V_{CC} = Max., V_I = 5.5V				1	mA
I_{IH}	High Level Input Current†	V_{CC} = Max., V_I = 2.4V				25	μA
I_{IL}	Low Level Input Current†	V_{CC} = Max., V_I = 0.4V			−0.02	−0.25	mA
I_{OS}	Output Short-Circuit Current**	V_{CC} = Max., V_O = 0V		−30	−70	−130	mA
I_{CC}	Supply Current	V_{CC} = Max.	Small PAL†††		28	45	mA
			Medium PAL		70	90††	

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

†† Maximum I_{CC} specification applies to unprogrammed devices only. I_{CC} could increase up to 10% for programmed units.

††† Small PAL consists of 10H8A2, 12H6A2, 14H4A2, 16H2A2, 16C1A2, 10L8A2, 12L6A2, 14L4A2 and 16L2A2. Medium PAL consists of 16L8A2, 16R8A2, 16R6A2 and 16R4A2.

* 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.

*** Pins 1 and 11 may be raised to 20V max.

Table 11.1.10 AC and DC Specifications for 20-Pin Fast, Half-Power, Small PAL Devices

Switching Characteristics Over Recommended Ranges of Temperature and V_{CC}
Military: T_A = −55°C to +125°C*, V_{CC} = 5V ± 10%
Commercial: T_A = 0 to 75°C, V_{CC} = 5V ± 5%

Symbol	Parameter		Test Conditions	Military			Commercial			Units
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feedback to Output		C_L = 50 pF		25	45		25	35	ns
t_{CLK}	Clock to Output or Feedback				15	25		15	25	ns
t_{PZX}	Pin 11 to Output Enable				15	25		15	25	ns
t_{PXZ}	Pin 11 to Output Disable		C_L = 5 pF		15	25		15	25	ns
t_{PZX}	Input to Output Enable		C_L = 50 pF		25	45		25	35	ns
t_{PXZ}	Input to Output Disable		C_L = 5 pF		25	45		25	35	ns
f_{MAX}	Maximum Frequency			14	25		16	25		MHz
t_w	Width of Clock	Low		25	10		25	10		ns
		High		25	10		25	10		ns
t_{su}	Setup Time from Input or Feedback to Clock			50	25		35	25		ns
t_h	Hold Time			0	−15		0	−15		ns

Note: The max t_{PD} of 16C1A2 in commercial range is 40 ns.

Table 11.1.10 AC and DC Specifications for 20-Pin Fast, Half-Power, Small PAL Devices (Cont.)

16L8A2, 16R8A2, 16R6A2, 16R4A2 Recommended Operating Conditions

Symbol	Parameter		Military			Commercial			Units
			Min	Typ	Max	Min	Typ	Max	
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}	Setup Time from Input or Feedback to Clock		50	25		35	25		ns
t_h	Hold Time		0	− 15		0	− 15		ns
T_A	Operating Free-Air Temperature		− 55		125	0	25	75	°C
T_C	Operating Case Temperature				125				°C

Electrical Characteristics

Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions		Min	Typ	Max	Units
V_{IH}*	High Level Input Voltage			2			V
V_{IL}*	Low Level Input Voltage					0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = − 18mA			− 0.8	− 1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} = − 2mA MIL I_{OH} = − 3.2mA COM	2.4	3.4		V
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = 12mA MIL I_{OL} = 24mA COM		0.3	0.5	V
I_{OZH}	Off-State Output Current †	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 2.4V			100	μA
I_{OZL}			V_O = 0.4V			− 100	μA
I_I	Maximum Input Current	V_{CC} = Max., V_I = 5.5V				1	mA
I_{IH}	High Level Input Current †	V_{CC} = Max., V_I = 2.4V				25	μA
I_{IL}	Low Level Input Current †	V_{CC} = Max., V_I = 0.4V			− 0.02	− 0.25	mA
I_{OS}	Output Short-Circuit Current**	V_{CC} = 5V, V_O = 0V		− 30	− 70	− 130	mA
I_{CC}	Supply Current††	V_{CC} = Max.			70	90††	mA

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

†† Maximum I_{CC} specification applies to unprogrammed devices only. I_{CC} could increase up to 10% for programmed units.

* 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.

** Pins 1 and 11 may be raised to 20V max.

Table 11.1.11 AC and DC Specifications for 20-Pin Fast, Half Power Medium PAL Devices

Programmable Array Logic PAL Low Power PAL Series 20A2

Symbol	Parameter	Test Conditions††	Military			Commercial			Unit
			Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feedback to Output	$C_L = 50pF$		25	50		25	35	ns
	Clock to Output or Feedback			15	25		15	25	ns
t_{PZX}	Pin 11 to Output Enable			15	25		15	25	ns
t_{PXZ}	Pin 11 to Output Disable	$C_L = 5pF$		15	25		15	25	ns
t_{PZX}	Input to Output Enable	$C_L = 50pF$		25	45		25	35	ns
t_{PXZ}	Input to Output Disable	$C_L = 5pF$		25	45		25	35	ns
f_{MAX}	Maximum Frequency		14	25		16	25		MHz

Table 11.1.11 AC and DC Specifications for 20-Pin Fast, Half Power Medium PAL Devices (Cont.)

16L8B2, 16R8B2, 16R6B2, 16R4B2 Recommended Operating Conditions

Symbol	Parameter		Military			Commercial			Units
			Min	Typ	Max	Min	Typ	Max	
V_{CC}	Supply Voltage		4.5	5	5.5	4.75	5	5.25	V
t_W	Width of Clock	Low	20	10		15	8		ns
		High	20	10		15	8		
t_{SU}	Setup Time from Input or Feedback to Clock	16R8B, 16R6B, 16R4B	25	10		20	10		ns
t_h	Hold Time		0	−5		0	−5		ns
T_A	Operating Free-Air Temperature		−55			0	25	75	°C
T_C	Operating Case Temperature				125				°C

Electrical Characteristics

Over Recommended Operating Temperature Range

Symbol	Parameter	Test Conditions			Min	Typ	Max	Units
V_{IH}*	High Level Input Voltage				2			V
V_{IL}*	Low Level Input Voltage						0.8	V
V_{IC}	Input Clamp Voltage	V_{CC} = Min., I_I = −18mA				−0.8	−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} = −2mA	MIL	2.4	3.4		V
			I_{OH} = −3.2mA	COM				
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = 12mA	MIL		0.3	0.5	V
			I_{OL} = 24mA	COM				
I_{OZH}	Off-State Output Current †	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 2.4V				100	μA
I_{OZL}			V_O = 0.4V				−100	μA
I_I	Maximum Input Current	V_{CC} = Max., V_I = 5.5V					1	mA
I_{IH}	High Level Input Current †	V_{CC} = Max., V_I = 2.4V					25	μA
I_{IL}	Low Level Input Current †	V_{CC} = Max., V_I = 0.4V				−0.01	−0.25	mA
I_{OS}	Output Short-Circuit Current**	V_{CC} = 5V, V_O = 0V			−30	−70	−130	mA
I_{CC}	Supply Current††	V_{CC} = Max.				70	90	mA

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

†† Maximum I_{CC} specification applies to unprogrammed devices only. I_{CC} could increase up to 10% for programmed units.

* 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.

** Pins 1 and 11 may be raised to 20V max.

Table 11.1.12 AC and DC Specifications for 20-Pin Ultra High-Speed, Half Power, Medium PAL Devices

Switching Characteristics

Military: $T_A = -55°C$ to $+125°C$*, $V_{CC} = 5V \pm 10\%$
Commercial: $T_A = 0$ to $75°C$, $V_{CC} = 5V \pm 5\%$

Symbol	Parameter		Test Conditions	Military			Commercial			Unit
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feedback to Output	16R6B 16R4B 16L8B	$R_1 = 200\Omega$ $R_2 = 390\Omega$	15		30		15 11	25	ns
t_{CLK}	Clock to Output or Feedback				8	20		8	15	ns
t_{PZX}	Pin 11 to Output Enable				10	25		10	20	ns
t_{PXZ}	Pin 11 to Output Disable				10	25		10	20	ns
t_{PZX}	Input to Output Enable	16R6B 16R4B 16L8B			11	30		11	25	ns
t_{PXZ}	Input to Output Disable	16R6B 16R4B 16L8B			11	30		11	25	ns
f_{MAX}	Maximum Frequency	16R8B 16R6B 16L4B		20	30 50		25 40	30		MHz

* These are absolute voltage with respect to pin 10 on the device and include all overshoots due to system and/or tester noise. Do not attempt to test these values without suitable equipment.

** Only one output shorted a time.

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

Table 11.1.12 AC and DC Specifications for 20-Pin Ultra High-Speed, Half Power, Medium PAL Devices (Cont.)

12L10, 14L8, 16L6, 18L4, 20L2, 20C1
20L10, 20X10, 20X8, 20X4

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				0		75	°C
T_C	Operating Case Temperature	–55		125				°C

Electrical Characteristics Over Operating Conditions

Symbol	Parameter	Test Conditions		Min	Typ	Max	Unit
V_{IL}	Low Level Input Voltage					0.8	
V_{IH}	High Level Input Voltage			2			
V_{IC}	Input Clamp Voltage	V_{CC} = Min.	I_I = –18 mA			–1.5	
I_{IL}	Low Level Input Current †	V_{CC} = Max.	V_I = 0.4V			–0.25	
I_{IH}	High Level Input Current †	V_{CC} = Max.	V_I = 2.4V			25	
I_I	Maximum Input Current	V_{CC} = Max.	V_I = 5.5V			1	
V_{OL}	Low Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V I_{IH} = 2V	I_{OL} = 12 mA MIL			0.5	V
			I_{OL} = 24 mA COM				
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	I_{OH} =–2 mA MIL	2.4			V
			I_{OH} = –3.2 mA COM				
I_{OZL}	Off-state Output Current †	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	V_O = 0.4V			–100	µA
I_{OZH}			V_O 2.4V			100	µA
I_{OS}	Output Short-Circuit Current**	V_{CC} = Max.	V_O = 0V	–30		–130	mA
I_{CC}	Supply Current	V_{CC} = Max.	20X4, 20X8, 20X10		120	180	mA
			20L10		90	165	
			12L10, 14L8, 16L6 18L4, 20L2, 20C1		60	100	

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

* Pins 1 and 13 may be raised to 22V max.

** Only one output shorted at a time.

Table 11.1.13 AC and DC Specifications for 24-Pin, Standard PAL Devices

Switching Characteristics Over Operating Conditions

Symbol	Parameter		Test Conditions R1, R2	Military			Commercial			Unit
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feedback to Output		20L10, 20X10 20X8, 20X4 $C_L = 50$ pF		35	60		35	50	ns
t_{PD}	Input or Feedback to Output		12L10, 14L8, 16L6 18L4, 20L2, 20C1 $C_L = 50$ pF		25	45		25	40	ns
t_{CLK}	Clock to Output or Feedback		$C_L = 50$ pF		20	40		20	30	ns
t_{PZX}	Pin 13 to Output Enable		$C_L = 50$ pF		20	45		20	35	ns
t_{PXZ}	Pin 13 to Output Disable		$C_L = 5$ pF		20	45		20	35	ns
t_{PZX}	Input to Output Enable		$C_L = 50$ pF		35	55		35	45	ns
t_{PXZ}	Input to Output Disable		$C_L = 5$ pF		35	55		35	45	ns
t_W	Width of Clock	Low		40	20		35	20		ns
		High		30	10		25	10		ns
t_{SU}	Set-Up Time from Input or Feedback			60	38		50	38		ns
t_h	Hold Time			0	–15		0	–15		ns
f_{MAX}	Maximum Frequency			10.0			12.5			MHz

Table 11.1.13 AC and DC Specifications for 24-Pin, Standard PAL Devices (Cont.)

20L8A, 20R8A, 20R6A, 20R4A Operating Conditions

Symbol	Parameter	Military			Commercial			Units
		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				0		75	°C
T_C	Operating Case Temperature	−55		125				°C

Electrical Characteristics Over Operating Conditions

Symbol	Parameter	Test Conditions		Min	Typ	Max	Units
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				−1.5	V
I_{IL}	Low Level Input Current †	V_{CC} = Max., V_I = 0.4V				−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. V_{IL} = 0.8V V_{IH} = 2V	I_{OL} = 12mA MIL			0.5	V
			I_{OL} = 24mA*** COM				
V_{OH}	High Level Output Voltage	V_{CC} = Min. V_{IL} = 0.8V V_{IH} = 2V	$I_{OH} = -2$mA MIL	2.4			V
			$I_{OH} = -3.2$mA COM				
I_{OZL}	Off-State Output Current†	V_{CC} = Max. V_{IL} = 0.8V V_{IH} = 2V	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		−130	mA
I_{CC}	Supply Current	V_{CC} = Max.			160	210	mA

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

* Pins 1 and 13 may be raised to 20V max.

** Only one output shorted at a time.

*** These are absolute voltages with :espect 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.

Table 11.1.14 AC and DC Specifications for 24-Pin, Fast PAL Devices

Switching Characteristics Over Operating Conditions

Symbol	Parameter		Test Conditions	Military			Commercial			Units
				Min	Typ	Max	Min	Typ	Max	
t_{PD}	Input or Feedback to Output		20L8A, 20R6A 20R4A $C_L = 50pF$		18	30		18	25	ns
t_{CLK}	Clock to Output or Feedback		$C_L = 50pF$		12	20		12	15	ns
t_{PZX}	Pin 13 to Output Enable		$C_L = 50pF$		10	25		10	20	ns
t_{PXZ}	Pin 13 to Output Disable		$C_L = 5pF$		11	25		11	20	ns
t_{PZX}	Input to Output Enable		$C_L = 50pF$		10	30		10	25	ns
t_{PXZ}	Input to Output Disable		$C_L = 5pF$		13	30		13	25	ns
t_w	Width of Clock	Low		20	7		15	7		ns
		High		20	7		15	7		ns
t_{su}	Setup Time from Input or Feedback		20R8A, 20R6A, 20R4A	30	18		25	18		ns
t_h	Hold Time			0	−10		0	−10		ns
f_{MAX}	Maximum Frequency			20	40		28.5	40		MHz

Table 11.1.14 AC and DC Specifications for 24-Pin, Fast PAL Devices (Cont.)

11.2 PROGRAMMING/VERIFYING PROCEDURE — 20 PIN PAL DEVICES*

As long as Pin 1 is at HH, Pin 11 is at ground, and Pin 12 is either at HH or Z (as defined in Table 11.2.1) — Pins 16, 17, 18, and 19 are outputs. The other pin functions are: I0 (Pin 2) through I7 (Pin 9) plus Pin 12 address the proper row; A0 (Pin 15), A1 (Pin 14), and A2 (Pin 13) address the proper product lines.

When Pin 11 is at HH, Pin 1 is at ground, and Pin 19 is either at HH or Z — Pins 12, 13, 14, and 15 are outputs. The other pin functions are: I0 (Pin 2) through I7 (Pin 9) plus Pin 19 address the proper row; A0 (now Pin 18), A1 (now Pin 17), and A2 (now Pin 16) address the proper product lines.

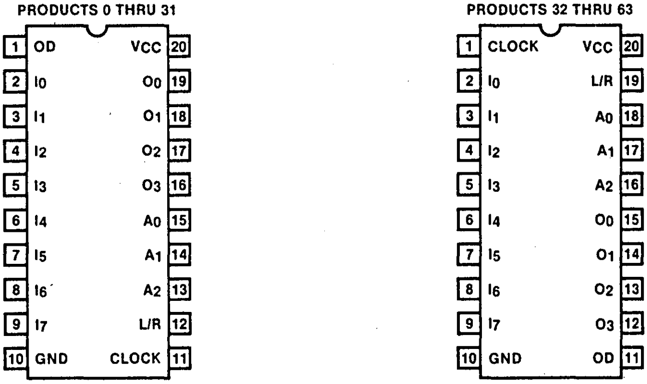


Figure 11.2.1 Pin Assignment for Programming

Pre-Verification

Step 1.1 Raise V_{CC} to 5V.

Step 1.2 Raise Output Disable pin, OD, to V_{IHH}.

Step 1.3 Select an input line by specifying Inputs and L/R as shown in Table 11.2.2.

Step 1.4 Select a product line by specifying A0, A1 and A2 one-of-eight select as shown in Table 11.2.2

Step 1.5 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, O, is in the state corresponding to an unblown fuse.
- — For verified unblown condition, continue procedure from Step 1.3 through Step 1.5.
- — For verified blown condition, stop procedure and reject part.

Note: For programming purposes many PAL pins have double functions.

Input Line Number	Pin Identification								
	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	L/R
0	HH	HH	HH	HH	HH	HH	HH	L	Z
1	HH	HH	HH	HH	HH	HH	HH	H	Z
2	HH	HH	HH	HH	HH	HH	HH	L	HH
3	HH	HH	HH	HH	HH	HH	HH	H	HH
4	HH	HH	HH	HH	HH	HH	L	HH	Z
5	HH	HH	HH	HH	HH	HH	H	HH	Z
6	HH	HH	HH	HH	HH	HH	L	HH	HH
7	HH	HH	HH	HH	HH	HH	H	HH	HH
8	HH	HH	HH	HH	HH	L	HH	HH	Z
9	HH	HH	HH	HH	HH	H	HH	HH	Z
10	HH	HH	HH	HH	HH	L	HH	HH	HH
11	HH	HH	HH	HH	HH	H	HH	HH	HH
12	HH	HH	HH	HH	L	HH	HH	HH	Z
13	HH	HH	HH	HH	H	HH	HH	HH	Z
14	HH	HH	HH	HH	L	HH	HH	HH	HH
15	HH	HH	HH	HH	H	HH	HH	HH	HH
16	HH	HH	HH	L	HH	HH	HH	HH	Z
17	HH	HH	HH	H	HH	HH	HH	HH	Z
18	HH	HH	HH	L	HH	HH	HH	HH	HH
19	HH	HH	HH	H	HH	HH	HH	HH	HH
20	HH	HH	L	HH	HH	HH	HH	HH	Z
21	HH	HH	H	HH	HH	HH	HH	HH	Z
22	HH	HH	L	HH	HH	HH	HH	HH	HH
23	HH	HH	H	HH	HH	HH	HH	HH	HH
24	HH	L	HH	HH	HH	HH	HH	HH	Z
25	HH	H	HH	HH	HH	HH	HH	HH	Z
26	HH	L	HH	HH	HH	HH	HH	HH	HH
27	HH	H	HH	HH	HH	HH	HH	HH	HH
28	L	HH	HH	HH	HH	HH	HH	HH	Z
29	H	HH	HH	HH	HH	HH	HH	HH	Z
30	L	HH	HH	HH	HH	HH	HH	HH	HH
31	H	HH	HH	HH	HH	HH	HH	HH	HH

Table 11.2.1 Input Line Select

Product Line Number	Pin Identification						
	O_3	O_2	O_1	O_0	A_2	A_1	A_0
0,32	Z	Z	Z	HH	Z	Z	Z
1,33	Z	Z	Z	HH	Z	Z	HH
2,34	Z	Z	Z	HH	Z	HH	Z
3,35	Z	Z	Z	HH	Z	HH	HH
4,36	Z	Z	Z	HH	HH	Z	Z
5,37	Z	Z	Z	HH	HH	Z	HH
6,38	Z	Z	Z	HH	HH	HH	Z
7,39	Z	Z	Z	HH	HH	HH	HH
8,40	Z	Z	HH	Z	Z	Z	Z
9,41	Z	Z	HH	Z	Z	Z	HH
10,42	Z	Z	HH	Z	Z	HH	Z
11,43	Z	Z	HH	Z	Z	HH	HH
12,44	Z	Z	HH	Z	HH	Z	Z
13,45	Z	Z	HH	Z	HH	Z	HH
14,46	Z	Z	HH	Z	HH	HH	Z
15,47	Z	Z	HH	Z	HH	HH	HH
16,48	Z	HH	Z	Z	Z	Z	Z
17,49	Z	HH	Z	Z	Z	Z	HH
18,50	Z	HH	Z	Z	Z	HH	Z
19,51	Z	HH	Z	Z	Z	HH	HH
20,52	Z	HH	Z	Z	HH	Z	Z
21,53	Z	HH	Z	Z	HH	Z	HH
22,54	Z	HH	Z	Z	HH	HH	Z
23,55	Z	HH	Z	Z	HH	HH	HH
24,56	HH	Z	Z	Z	Z	Z	Z
25,57	HH	Z	Z	Z	Z	Z	HH
26,58	HH	Z	Z	Z	Z	HH	Z
27,59	HH	Z	Z	Z	Z	HH	HH
28,60	HH	Z	Z	Z	HH	Z	Z
29,61	HH	Z	Z	Z	HH	Z	HH
30,62	HH	Z	Z	Z	HH	HH	Z
31,63	HH	Z	Z	Z	HH	HH	HH

Table 11.2.2 Input Line Select

Programming Algorithm

Step 2.1 Raise Output Disable pin, OD to V_{IHH}.

Step 2.2 Programming pass. For all fuses to be blown:

Step 2.2.1 Lower CLOCK pin to ground.

Step 2.2.2 Select an input line by specifying Inputs and L/R as shown in Table 11.2.2.

Step 2.2.3 Select a product line by specifying A0, A1 and A2 one-of-eight select as shown in Table 11.2.2.

Step 2.2.4 Raise V_{CC} to $_{IHH}$.

Step 2.2.5 Program the fuse by pulsing the output pins of the selected product group one at a time to V_{IHH} (as shown in the Programming Waveforms).

Step 2.2.6 Lower V_{CC} to 5V.

Step 2.2.7 Repeat this procedure from Step 2.2.2 until pattern is complete.

Step 2.3 First verification pass. For all fuse locations:

Step 2.3.1 Select an input line by specifying Inputs and L/R as shown in Tables 11.2.1 and 11.2.2.

Step 2.3.2 Select a product line by specifying A0, `A1, and A2 one-of-eight select as shown in Table 11.2.2.

Step 2.3.3 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, O, is in the correct state.

— For verified output state, continue procedure.

— For overblow condition, stop procedure and reject part.

— For underblow condition, reexecute Steps 2.2.4 through 2.2.6 and 2.2.3. If successful, continue procedure. After three attempts to blow fuse without success, reject part but continue procedure.

Step 2.3.4 Repeat this procedure from Step 2.3.1 until the entire array is exercised.

Step 2.4 High Voltage Verify. For all fuse locations:

Step 2.4.1 Raise V_{CC} to 5.5V.

Step 2.4.2 Select an input line by specifying Inputs and L/R as shown in Tables 11.2.1 or 11.2.2.

Step 2.4.3 Select a product line by specifying A0, A1, and A2 one-of-eight select as shown in Table 11.2.2.

Step 2.4.4 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, O, is in the correct state.

— For verified output state, continue procedure.

— For invalid output state, stop procedure and reject part.

Step 2.4.5 Repeat this procedure from Step 2.4.1 until the entire array is exercised.

Step 2.5 Low Voltage Verify. For all fuse locations:

Step 2.5.1 Lower V_{CC} to 4.5V.

Step 2.5.2 Select an input line by specifying inputs and L/R as shown in Tables 11.2.1 or 11.2.2.

Step 2.5.3 Select a product line by specifying A0, A1, and A2 one-of-eight select as shown in Table 11.2.2.

*NSC programming spec. Rev. 1. The old programming spec. is still valid.

Step 2.5.4 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, O, is in the correct state.
— For verified output state, continue procedure.
— For invalid output state, continue procedure and reject part.

Programming the Security Fuses

Step 3.1 Verify per Step 2.4 and Step 2.5.

Step 3.2 Raise V_{CC} to 6V.

Step 3.3 Program the first fuse by pulsing Pin 1 to V_P. (From 1 to 5 pulses is acceptable.)

Step 3.4 Program the second fuse by pulsing Pin 11 to V_P. (1 to 5 pulses is acceptable.)

Step 3.5 Verify per Step 2.4 and Step 2.5:
— A device is "secure" if either half fails to verify.

Voltage Legend

L = Low level input voltage, V_{IL}

H = High level input voltage, V_{IH}

HH = High level program voltage, V_{IHH}

Z = 10 kΩ to 5V

Note: For programming purposes many PAL device pins have double functions.

11.3 PROGRAMMING/VERIFYING PROCEDURE — 24 PIN PAL DEVICES*

As long as Pin 1 is at HH, Pin 13 is at ground, and Pin 14 is either at HH or Z (as defined in Table 11.3.1) — Pins 19, 20, 21, and 22 are outputs. The other pin functions are: 10 (Pin 2) through 19 (Pin 11) plus Pin 14 address the proper row; A0 (Pin 15), A1 (Pin 16), and A2 (Pin 17) address the proper product lines.

As long as Pin 13 is at HH, Pin 1 is at ground, and Pin 23 is either at HH or Z (as defined in Table 11.3.1) — Pins 15, 16, 17 and 18 are outputs. The other pin functions are: 10 (Pin 2) through I9 (Pin 11) plus Pin 23 address the proper row; AD (Pin 22), A1 (Pin 21), and A2 (Pin 20) address the proper product lines.

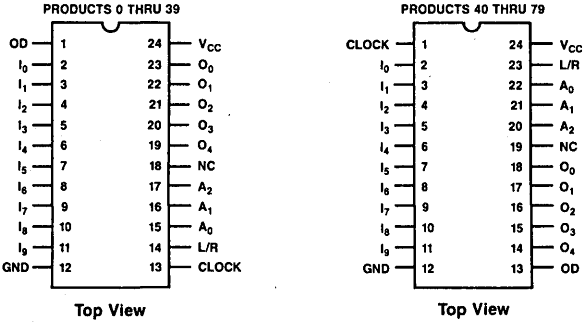


Figure 11.3.1 Pin Assignment for Programming

Pre-Verification

Step 1.1 Raise V_{CC} to 5V.

Step 1.2 Raise Output Disable pin, OD, to V_{IHH}.

Step 1.3 Select an input line by specifying Inputs and L/R as shown in Table 11.3.1.

Step 1.4 Select a product line by specifying A0, A1 and A2 one-of-eight select as shown in Table 11.3.2.

Step 1.5 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, 0H, is in the state corresponding to an unblown fuse.

— For verified unblown condition, continue procedure from Step 1.3 through Step 1.5.

— For verified blown condition, stop procedure and reject part.

Programming Algorithm

Step 2.1 Raise Output Disable pin, OD, to V_{IHH}.

Step 2.2 Programming pass. For all fuses to be blown:

Step 2.2.1 Lower CLOCK pin to ground.

Step 2.2.2 Select an input line by specifying inputs and L/R as shown in Table 11.3.1.

Input Line Number	Pin Identification										
	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	L/R
0	HH	HH	HH	HH	HH	HH	HH	HH	HH	L	Z
1	HH	HH	HH	HH	HH	HH	HH	HH	HH	H	Z
2	HH	HH	HH	HH	HH	HH	HH	HH	HH	L	HH
3	HH	HH	HH	HH	HH	HH	HH	HH	HH	H	HH
4	HH	HH	HH	HH	HH	HH	HH	HH	L	HH	Z
5	HH	HH	HH	HH	HH	HH	HH	HH	H	HH	Z
6	HH	HH	HH	HH	HH	HH	HH	HH	L	HH	HH
7	HH	HH	HH	HH	HH	HH	HH	HH	H	HH	HH
8	HH	HH	HH	HH	HH	HH	HH	L	HH	HH	Z
9	HH	HH	HH	HH	HH	HH	HH	H	HH	HH	Z
10	HH	HH	HH	HH	HH	HH	HH	L	HH	HH	HH
11	HH	HH	HH	HH	HH	HH	HH	H	HH	HH	HH
12	HH	HH	HH	HH	HH	HH	L	HH	HH	HH	Z
13	HH	HH	HH	HH	HH	HH	H	HH	HH	HH	Z
14	HH	HH	HH	HH	HH	HH	L	HH	HH	HH	HH
15	HH	HH	HH	HH	HH	HH	H	HH	HH	HH	HH
16	HH	HH	HH	HH	HH	L	HH	HH	HH	HH	Z
17	HH	HH	HH	HH	HH	H	HH	HH	HH	HH	Z
18	HH	HH	HH	HH	HH	L	HH	HH	HH	HH	HH
19	HH	HH	HH	HH	HH	H	HH	HH	HH	HH	HH
20	HH	HH	HH	HH	L	HH	HH	HH	HH	HH	Z
21	HH	HH	HH	HH	H	HH	HH	HH	HH	HH	Z
22	HH	HH	HH	HH	L	HH	HH	HH	HH	HH	HH
23	HH	HH	HH	HH	H	HH	HH	HH	HH	HH	HH
24	HH	HH	HH	L	HH	HH	HH	HH	HH	HH	Z
25	HH	HH	HH	H	HH	HH	HH	HH	HH	HH	Z
26	HH	HH	HH	L	HH	HH	HH	HH	HH	HH	HH
27	HH	HH	HH	H	HH	HH	HH	HH	HH	HH	HH
28	HH	HH	L	HH	HH	HH	HH	HH	HH	HH	Z
29	HH	HH	H	HH	HH	HH	HH	HH	HH	HH	Z
30	HH	HH	L	HH	HH	HH	HH	HH	HH	HH	HH
31	HH	HH	H	HH	HH	HH	HH	HH	HH	HH	HH
32	HH	L	HH	HH	HH	HH	HH	HH	HH	HH	Z
33	HH	H	HH	HH	HH	HH	HH	HH	HH	HH	Z
34	HH	L	HH	HH	HH	HH	HH	HH	HH	HH	HH
35	HH	H	HH	HH	HH	HH	HH	HH	HH	HH	HH
36	L	HH	HH	HH	HH	HH	HH	HH	HH	HH	Z
37	H	HH	HH	HH	HH	HH	HH	HH	HH	HH	Z
38	L	HH	HH	HH	HH	HH	HH	HH	HH	HH	HH
39	H	HH	HH	HH	HH	HH	HH	HH	HH	HH	HH

Table 11.3.1 Input Line Select

Input Line Number	Pin Identification							
	O_4	O_3	O_2	O_1	O_0	A_2	A_1	A_0
0, 40	Z	Z	Z	Z	HH	Z	Z	Z
1, 41	Z	Z	Z	Z	HH	Z	Z	HH
2, 42	Z	Z	Z	Z	HH	Z	HH	Z
3, 43	Z	Z	Z	Z	HH	Z	HH	HH
4, 44	Z	Z	Z	Z	HH	HH	Z	Z
5, 45	Z	Z	Z	Z	HH	HH	Z	HH
6, 46	Z	Z	Z	Z	HH	HH	HH	Z
7, 47	Z	Z	Z	Z	HH	HH	HH	HH
8, 48	Z	Z	Z	HH	Z	Z	Z	Z
9, 49	Z	Z	Z	HH	Z	Z	Z	HH
10, 50	Z	Z	Z	HH	Z	Z	HH	Z
11, 51	Z	Z	Z	HH	Z	Z	HH	HH
12, 52	Z	Z	Z	HH	Z	HH	Z	Z
13, 53	Z	Z	Z	HH	Z	HH	Z	HH
14, 54	Z	Z	Z	HH	Z	HH	HH	Z
15, 55	Z	Z	Z	HH	Z	HH	HH	HH
16, 56	Z	Z	HH	Z	Z	Z	Z	Z
17, 57	Z	Z	HH	Z	Z	Z	Z	HH
18, 58	Z	Z	HH	Z	Z	Z	HH	Z
19, 59	Z	Z	HH	Z	Z	Z	HH	HH
20, 60	Z	Z	HH	Z	Z	HH	Z	Z
21, 61	Z	Z	HH	Z	Z	HH	Z	HH
22, 62	Z	Z	HH	Z	Z	HH	HH	Z
23, 63	Z	Z	HH	Z	Z	HH	HH	HH
24, 64	Z	HH	Z	Z	Z	Z	Z	Z
25, 65	Z	HH	Z	Z	Z	Z	Z	HH
26, 66	Z	HH	Z	Z	Z	Z	HH	Z
27, 67	Z	HH	Z	Z	Z	Z	HH	HH
28, 68	Z	HH	Z	Z	Z	HH	Z	Z
29, 69	Z	HH	Z	Z	Z	HH	Z	HH
30, 70	Z	HH	Z	Z	Z	HH	HH	Z
31, 71	Z	HH	Z	Z	Z	HH	HH	HH
32, 72	HH	Z	Z	Z	Z	Z	Z	Z
33, 73	HH	Z	Z	Z	Z	Z	Z	HHZ
34, 74	HH	Z	Z	Z	Z	Z	HH	Z
35, 75	HH	Z	Z	Z	Z	Z	HH	HH
36, 76	HH	Z	Z	Z	Z	HH	HH	Z
37, 77	HH	Z	Z	Z	Z	HH	Z	HH
38, 78	HH	Z	Z	Z	Z	HH	HH	Z
39, 79	HH	Z	Z	Z	Z	HH	HH	HH

Table 11.3.2 Product Line Select

Step 2.2.3 Select an input line by specifying inputs and L/R as shown in Table 11.2.2.

Step 2.2.4 Select a product line by specifying A0, A1, and A2 one-of-eight select as shown in Table 11.2.2

Step 2.2.5 Raise V_{CC} to V_{IHH}.

Step 2.2.6 Program the fuse by pulsing the output pins of the selected product group one at a time to V_{IHH} (as shown in the Programming Waveforms).

Step 2.2.7 Lower V_{CC} to 5V.

Step 2.2.8 Repeat this procedure from Step 2.2.2 until pattern is complete.

Step 2.3 First verification pass. For all fuse locations:

Step 2.3.1 Select an input line by specifying Inputs and L/R as shown in Tables 11.2.1 and 11.2.2.

Step 2.3.2 Select a product line by specifying A0, A1, and A2 one-of-eight select as shown in Table 11.2.2.

Step 2.3.3 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, O, is in the correct state.

— For verified output state, continue procedure.

— For overblow condition, stop procedure and reject part.

— For underblow condition, reexecute Steps 2.2.4 through 2.2.6 and 2.2.3. If successful, continue procedure, after three attempts to blow fuse without success, reject part but continue procedure.

Step 2.3.4 Repeat this procedure from Step 2.3.1 until the entire array is exercised.

Step 2.4 High Voltage Verify. For all fuse locations:

Step 2.4.1 Raise V_{CC} to 5.5V.

Step 2.4.2 Select an input line by specifying Inputs and L/R as shown in Table 11.3.1.

Step 2.4.3 Select a product line by specifying A0, A1, and A2 one-of-eight select as shown in Table 11.3.2.

Step 2.4.4 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, 0, is in the correct state.

— For verified output state, continue the procedure.

— For invalid output state, stop procedure and reject part.

Step 2.4.5 Repeat this procedure from step 2.4.1 until the entire array is exercised.

Step 2.5 Low Voltage Verify. For all fuse locations:

Step 2.5.1 Lower V_{CC} to 4.5V.

Step 2.5.2 Select an input line by specifying inputs and L/R as shown in Table 11.3.1.

Step 2.5.3 Select a product line by specifying A0, A1, and A2 one-of-eight as shown in Table 11.3.2.

Step 2.5.4 Pulse the CLOCK pin and verify (with CLOCK at V_{IL}) that the output pin, 0, is in the correct state.
— For verified output state, continue procedure.
— For invalid output state, continue procedure and reject part.

Programming the Security Fuses

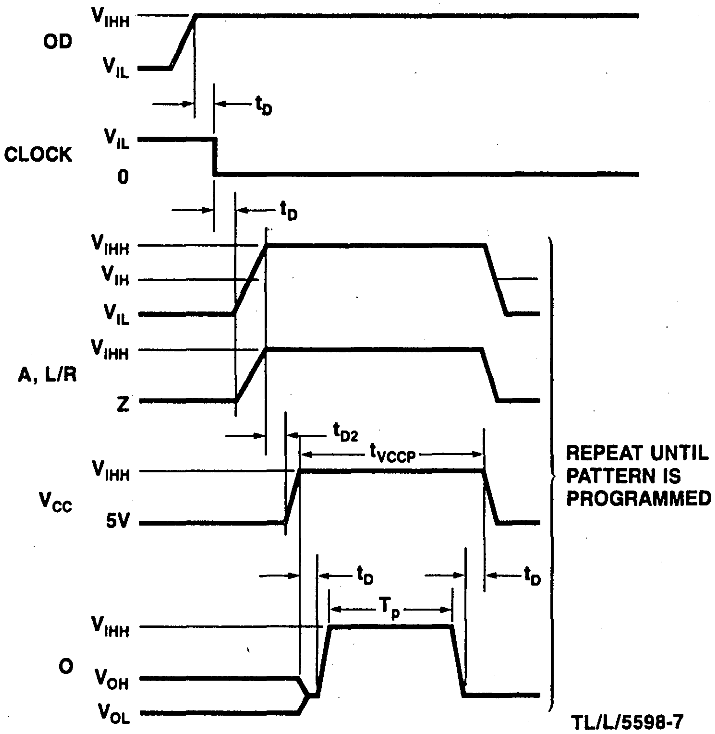
Step 3.1 Verify per Step 2.4 and Step 2.5
Step 3.2 Raise V_{CC} to 6V.
Step 3.3 For PAL 24 and PAL 24A:
— Program the first fuse by pulsing Pin 1 to V_P (From 1 to 5 pulses is acceptable.)
— Program the second fuse by pulsing Pin 13 to V_P (1 to 5 pulses is acceptable.)
Step 3.4 Verify per Step 2.4 and Step 2.5:
— A device is "secure" if either half fails to verify.

Symbol	Parameter		Min	Typ	Max	Units
V_{IHH}	Program Level Input Voltage		11.5	11.75	12	V
I_{IHH}	Program Level Input Current	Output Program Pulse			50	mA
		OD, L/R			50	
		All Other Inputs			10	
I_{CCH}	Program Supply Current				900	mA
t_{VCCP}	Pulse Width of V_{CC} @V_{IHH}				60	μs
t_P	Program Pulse Width		10	20	50	μs
t_D	Delay Time		100			ns
t_{D2}	Delay Time after L/R Pin		10			μs
	V_{CCP} Duty Cycle				20	%
V_P	Security Fuse Programming Voltage		18	18.5	19	V
I_P	Security Fuse Programming Supply Current				400	mA
t_{PP}	Security Fuse Programming Pulse Width		10	40	70	μs
	Security Fuse Programming Duty Cycle				50	%
t_{RP}	Rise Time of Output Programming and Address Pulses		1	1.5	10	V/μs
	Rise Time of Security Fuse Programming Pulses		1	1.5	10	
V_{CCPP}	V_{CC} Value During Security Fuse Programming		5.75	6	6.25	V
	V_{CC} Value for First Verify		4.75	5	5.25	
	V_{CC} Value for High V_{CC} Verify		5.4	5.5	5.6	
	V_{CC} Value for Low V_{CC} Verify		4.4	4.5	4.6	

Table 11.3.3 Programming Parameters

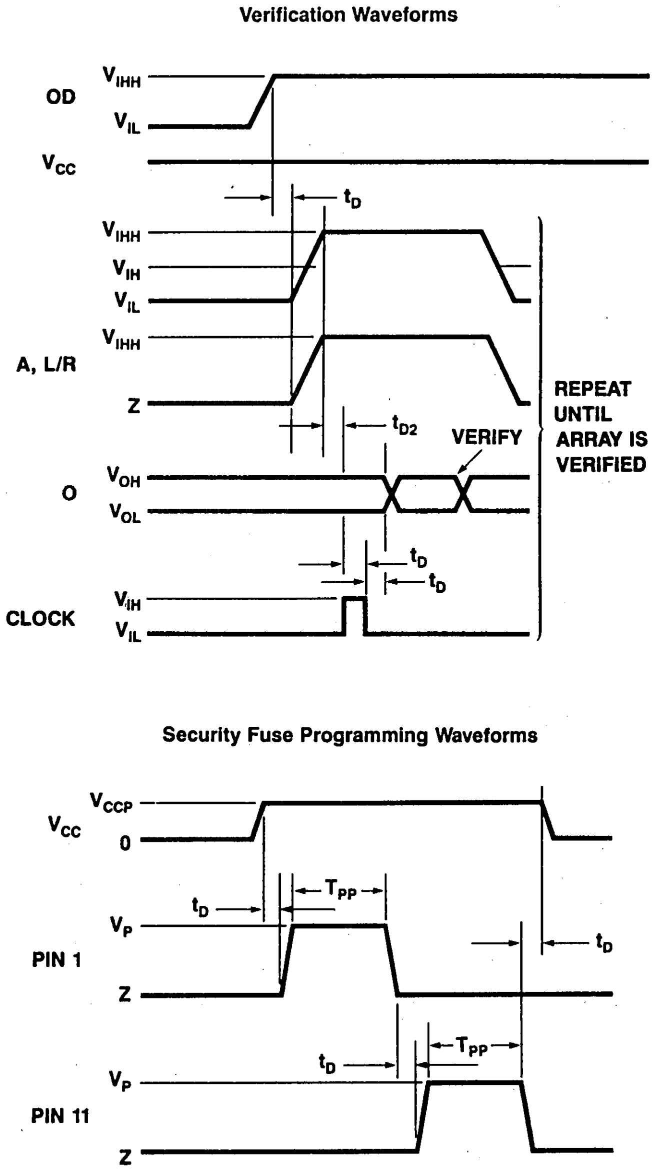
*NSC programming spec. Rev. 1. The old programming spec. is still valid.

Array Programming Waveforms



Note:
V_{CC} (Low Voltage Verify) = 4.5V
V_{CC} (High Voltage Verify) = 5.5V
V_{CC} (First Verify) = 5V
A Delay (t_{D2}) must always precede the Positive
Clock Transition. (e.g. see step 1.2.3.3 for underblow condition)

Figure 11.3.2 Programming Waveforms



Verification Waveforms

Security Fuse Programming Waveforms

Figure 11.3.2 Programming Waveforms (Cont.)

11.4 LOGIC PROM DATA SHEETS

Description

This generic Schottky PROM family by National provides the industry with one of the widest selections in sizes and organizations. Four-bit wide PROMs are provided with 256 to 4096 words in pin compatible 16 and 18-pin dual-in-line packages. The 8-bit wide devices range from 32 to 4096 words in a variety of packages. Being 'generic', all PROMs share a common programming algorithm.

National's new Programmable Read-Only Memories (PROMs) feature titanium-tungsten (Ti:W) fuse links designed to program efficiently with only 10.5 Volts applied. The high peformance and reliability of these PROMs are the result of fabrication by a Schottky bipolar process, of which the titanium-tungsten metallization is an integral part, and an on-chip programming circuit is used.

A major advantage of the titanium-tungsten fuse technology is the low programming voltage of the fuse links. At 10.5 Volts, this voltage level virtually eliminates the need for guard-ring devices and wide spacings required for other fuse technologies. Care is taken, however, to minimize voltage drops across the die and to reduce parasitics. The device is designed to insure that worst-case fuse operating current is low enough for reliable long-term operation. The Darlington programming circuit is liberally designed to insure adequate power density for blowing fuse links. The complete circuit design is optimized to provide high performance over the entire operating ranges of V_{CC} and temperature.

Testability

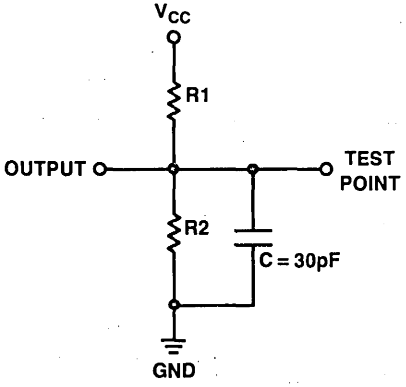
The Schottky PROM die includes extra rows and columns of fusible links for testing the programmability of each chip. These test fuses are placed at the worst-case chip locations to provide the highest possible confidence in the programming tests in the final product. A ROM pattern is also permanently fixed in the additional circuitry and coded to provide a parity check of input address levels. These and other test circuits are used to test for correct operation of the row and column-select circuits and functionality of input and enable gates. All test circuits are available at both wafer and assembled device levels to allow 100% functional and parametric testing at every stage of the test flow.

Reliability

As with all National products, the Ti:W PROMs are subjected to an ongoing reliability evaluation by the Reliability Assurance Department. These evaluations employ accelerated life tests, including dynamic high-temperature operating life, temperature-humidity life, temperature cycling, and thermal shock. To date, nearly 7.4 million Schottky Ti:W PROM device hours have been logged. DIP (N-package) and cerdip (J-package). Device performance in all package configurations is excellent.

Supply Voltage (Note 2)	− 0.5 to + 7.0V
Input Voltage (Note 2)	− 1.2 to + 5.5V
Output Voltage (Note 2)	− 0.5 to + 5.5V
Storage Temperature	− 65 to + 150C
Lead Temperature (10 seconds)	300C

Table 11.4.1 Absolute Maximum Ratings



*Device input waveform charecteristics are;
Repetition rate = 1MHz
Source impedance = 50Ω
Rise and Fall times = 2.5ns max.
(1.0 to 2.0 volt levels)

*TAA is measured with stable enable inputs.

*TEA and TER are measured from the 1.5 volt level on inputs and outputs with all address and enable inputs stable at applicable levels.

*For I_{OL} = 16mA, R1 = 300Ω and R2 = 600Ω for I_{OL} = 12mA, R1 = 400Ω and R2 = 800Ω.

*"C" includes scope and jig capacitance.

Figure 11.4.1 Standard Test Load

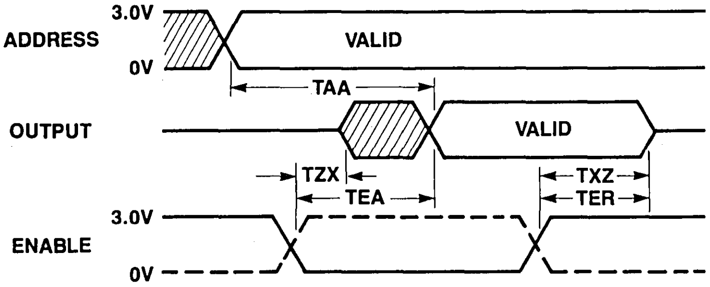


Figure 11.4.2 Switching Time Waveforms Non-Registered PROMs

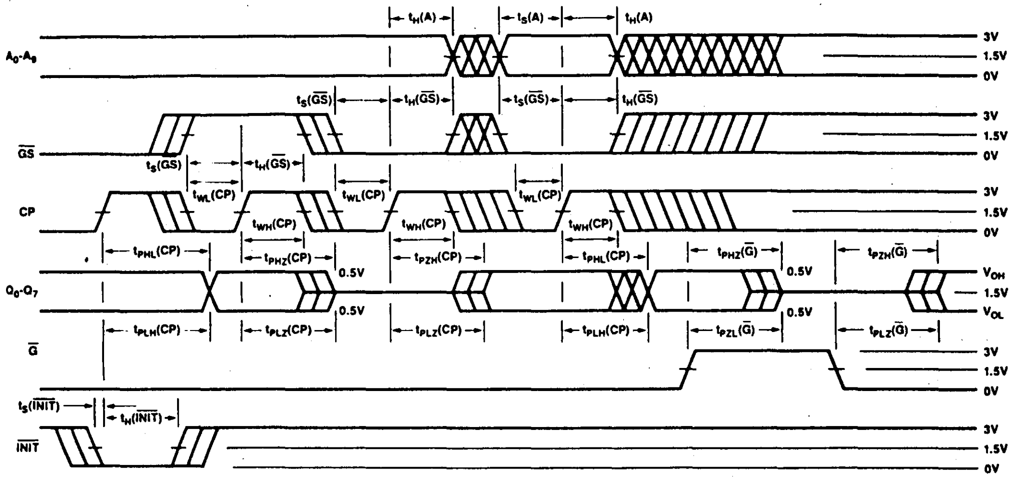


Figure 11.4.3 Switching Waveforms, Registered PROM

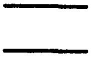
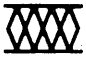
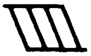
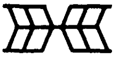
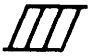
WAVEFORM	INPUTS	OUTPUTS	WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY		DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L		DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H			

Figure 11.4.4 Key to Timing Diagram

11.5 DM54/74S188, DM54/74S288 (32 × 8) 256-BIT TTL PROMs

General Description

These Schottky memories are organized in the popular 32 words by 8 bits configuration. A memory enable input is provided to control the output states. When the device is enabled, the outputs represent the contents of the selected word. When disabled, the 8 outputs go to the OFF or high impedance state. The memories are available in both open-collector and TRI-STATE® versions.

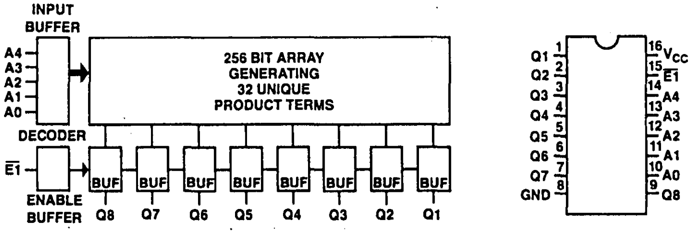
PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions.

Features

- Advanced titanium-tungsten (Ti:W) fuses.
- Schottky-clamped for high speed.
  Address access—22 ns typ.
  Enable access—15 ns typ.
  Enable recovery—15 ns typ.
- PNP inputs for reduced input loading.
- All DC and AC parameters guaranteed over temperature.
- Low voltage TRI-SAFE™ programming.

	Military	Commercial	Open-Collector	TRI-STATE	Package
DM74S188		X	X		N,J
DM74S288		X		X	N,J
DM54S188	X		X		J
DM54S288	X			X	J

Table 11.5.1 (32 × 8) 256-Bit TTL PROM Options



ORDER NUMBER:
DM74S188 J, DM74S288 J,
DM54S188 J, DM54S288 J
SEE NS PACKAGE J16A

ORDER NUMBER:
DM74S188 N OR DM74S288 N
SEE NS PACKAGE N16A

Figure 11.5.1 Block and Connection Diagram

DM54/74S188, DM54/74S288 (32 × 8) 256-BIT TTL PROMs

DC Electrical Characteristics

(Note 3)

Sym	Parameter	Conditions	DM54S188/288			DM74S188/288			Units
			Min	Typ	Max	Min	Typ	Max	
I_{IL}	Input Load Current	V_{CC} = Max, V_{IN} = 0.45V		-80	-250		-80	-250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max, V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max, V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
I_{OZ}	Output Leakage Current	V_{CC} = Max, V_{CEX} = 2.4V			50			50	μA
	(Open-Collector Only)	V_{CC} = Max, V_{CEX} = 5.5V			100			100	μA
V_C	Input Clamp Voltage	V_{CC} = Min, I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25C, 1MHz		4.0			4.0		pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V T_A = 25C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max, Inputs Grounded All Outputs Open		70	110		70	110	mA
TRI-STATE® Parameters									
I_{OS}	Short Circuit Output Current	V_O = 0V, V_{CC} = Max (Note 4)	-20		-70	-20		-70	mA
I_{OZ}	Output Leakage	V_{CC} = Max, V_O = 0.45 to 2.4V			+50			+50	μA
	(TRI-STATE)	Chip Disabled			-50			-50	μA
V_{OH}	Output Voltage High	I_{OH} = -2.0mA	2.4	3.2					V
		I_{OH} = −6.5mA				2.4	3.2		V

AC Electrical Characteristics

(With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol	DM54S188/288			DM74S188/288			Units
			Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV		22	45		22	35	ns
TEA	Enable Access Time	TEVQV		15	30		15	20	ns
TER	Enable Recovery Time	TEXQX		15	35		15	25	ns
TZX	Output Enable Time	TEVQX		15	30		15	20	ns
TXZ	Output Disable Time	TEXQZ		15	35		15	25	ns

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.5.2 AC and DC Specifications for (32 × 8) 256-Bit TTL PROMs

11.6 PL77X288/PL87X288 (32 × 8) 256-BIT TTL LOGIC PROMs

General Description

These Schottky programmable logic devices are organized in the popular 32 words by 8-bit configuration. An enable input is provided to control the output states. When the device is enabled, the outputs represent the contents of the selected word. When disabled, the 8 outputs go to the OFF or high impedance state. The memories are available in the TRI-STATE® version only.

PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions.

Features

- Advanced titanium-tungsten (Ti-W) fuses
- Schottky-clamped for high speed
  - Addressed access—10 ns typ
  - Enable access—8 ns typ
  - Enable recovery—8 ns typ
- PNP inputs for reduced input loading
- All DC and AC parameters guaranteed over temperature
- Low voltage TRI-SAFE™ programming

	Military	Commercial	Open-Collector	TRI-STATE	Package
PL87X288		X		X	N, J
PL77X288	X			X	J

Table 11.6.1 (32 × 8) 256-Bit TTL PROM Options

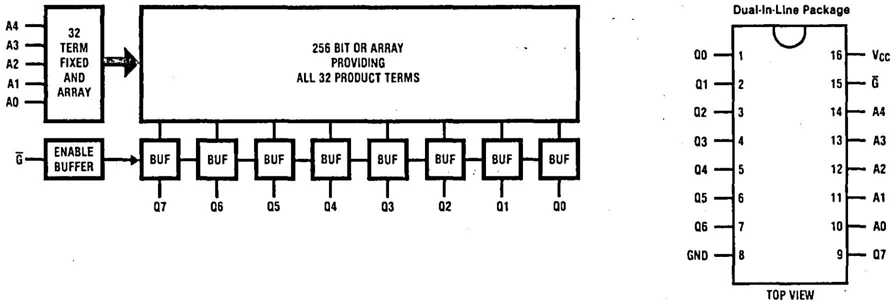


Figure 11.6.1 Block and Connection Diagram

PL77X288/PL87X288 (32 × 8) 256-BIT TTL LOGIC PROMs

DC Electrical Characteristics (Note 3)

Symbol	Parameter	Conditions	PL77X288			PL87X288			Units
			Min	Typ	Max	Min	Typ	Max	
I_{IL}	Input Load Current	V_{CC} = Max, V_{IN} = 0.4V		−80	−250		−80	−250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max, V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max, V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 24 mA (Com) I_{OL} = 12 mA (Mil)		0.35	0.50		0.35	0.50	V
V_{IL}	Low Level Input Voltage	(Note 7)			0.80			0.80	V
V_{IH}	High Level Input Voltage	(Note 7)	2.0			2.0			V
V_C	Input Clamp Voltage	V_{CC} = Min, I_{IN} = −18 mA		−0.8	−1.5		−0.8	−1.5	V
C_I	Input Capacitance	V_{CC} = 5.0V, V_{IN} = 2.0V T_A = 25°C, 1 MHz		4.0			4.0		pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V T_A = 25°C, 1 MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max, Inputs Grounded All Outputs Open		110	140		110	140	mA
TRI-STATE									
I_{OS}	Short Circuit Output Current	V_O = 0V, V_{CC} = Max (Note 4)	−30		−130	−30		−130	mA
I_{OZ}	Output Leakage (TRI-STATE)	V_{CC} = Max, V_O = 0.4V to 2.4V Chip Disabled			100 −100			100 −100	μA μA
V_{OH}	Output Voltage High	I_{OH} = −2.0 mA	2.4	3.2					V
		I_{OH} = −3.2 mA				2.4	3.2		V

AC Electrical Characteristics with standard load and operating conditions

Symbol	Parameter	JEDEC Symbol	PL77X288			PL87X288			Units
			Min	Typ	Max	Min	Typ	Max	
t_{AA}	Address Access Time (Note 5)	TAVQV		10	20		10	15	ns
t_{EA}	Enable Access Time (Note 5)	TEVQV		8	15		8	12	ns
t_{ER}	Enable Recovery Time (Note 6)	TEXQX		8	15		8	12	ns
t_{ZX}	Output Enable Time (Note 5)	TEVQX		8	15		8	12	ns
t_{XZ}	Output Disable Time (Note 6)	TEXQZ		8	15		8	12	ns

Note 1: Absolute maximum ratings are those values beyond which the device may be permanently damaged. They do not mean that the device may be operated at these values.

Note 2: These limits do not apply during programming. For the programming ratings, refer to the programming parameters.

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25°C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Note 5: C_L = 50 pF.

Note 6: C_L = 5 pF.

Note 7: 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.

Table 11.6.2 AC and DC Specifications for (32 × 8) 256-Bit TTL Logic PROMs

11.7 DM54/74LS471 (256 × 8) 2048-BIT TTL PROMs

General Description

These Schottky memories are organized in the popular 256 words by 8 bits configuration. Memory enable inputs are provided to control the output states. When the device is enabled, the outputs represent the contents of the selected word. When disabled, the 8 outputs go to the "OFF" of high impedance state.

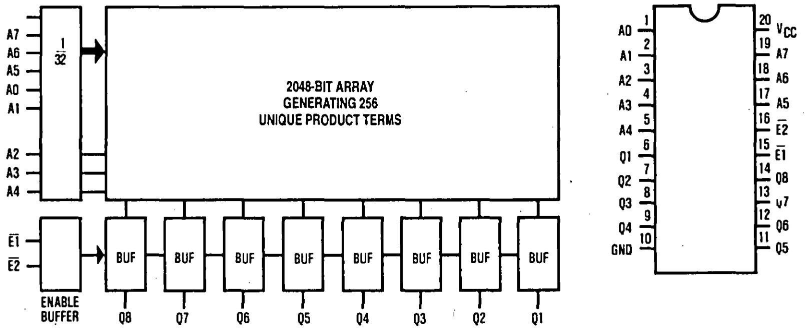
PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions.

Features

- Advanced titanium-tungsten (Ti-W) fuses
- Schottky-clamped for high speed
  - Addressed access—40 ns typ
  - Enable access—15 ns typ
  - Enable recovery—15 ns typ
- PNP inputs for reduced input loading
- All DC and AC parameters guaranteed over temperature
- Low voltage TRI-SAFE™ programming

	Military	Commercial	Open-Collector	TRI-STATE	Package
DM74LS471		X		X	N,J
DM54LS471	X			X	J

Table 11.7.1 (256 × 8) 2048-Bit TTL PROM Options



Order Number:
DM74LS471 J,
DM54LS471 J,
See NS Package J20B

Order Number:
DM74LS471 N
See NS Package N20A

Figure 11.7.1 Block and Connection Diagram

DM54/74LS471 (256 × 8) 2048-BIT TTL PROMs

DC Electrical Characteristics (Note 3)

Sym	Parameter	Conditions	DM54LS471			DM74LS471			Units
			Min	Typ	Max	Min	Typ	Max	
I_{IL}	Input Load Current	V_{CC} = Max, V_{IN} = 0.45V		-80	-250		-80	-250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max, V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max, V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
V_C	Input Clamp Voltage	V_{CC} = Min, I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25C, 1MHz		4.0			4.0		pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V T_A = 25C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max, Inputs Grounded All Outputs Open		75	100		75	100	mA

TRI-STATE® Parameters

I_{OS}	Short Circuit Output Current	V_O = 0V, V_{CC} = Max (Note 4)	-20		-70	-20		-70	mA
I_{OZ}	Output Leakage (TRI-STATE)	V_{CC} = Max, V_O = 0.45 to 2.4V			+50			+50	μA
		Chip Disabled			-50			-50	μA
V_{OH}	Output Voltage High	I_{OH} = -2.0mA	2.4	3.2					V
		I_{OH} = 6.5mA				2.4	3.2		V

AC Electrical Characteristics (With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol	DM54LS471			DM74LS471			Units
			Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV		45	70		40	60	ns
TEA	Enable Access Time	TEVQV		15	35		15	30	ns
TER	Enable Recovery Time	TEXQX		15	35		15	30	ns
TZX	Output Enable Time	TEVQX		15	35		15	30	ns
TXZ	Output Disable Time	TEXQZ		15	35		15	30	ns

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.7.2 AC and DC Specifications for (256 × 8) 2048-Bit TTL PROMs

11.8 DM54/74S473, DM54/74S472; DM54/74S473A, DM54/74S472A; DM54/74S472B (512 × 8) 4K-BIT TTL PROMs

General Description

These Schottky memories are organized in the popular 512 words by 8 bits configuration. A memory enable input is provided to control the output states. When the device is enabled, the outputs represent the contents of the selected word. When disabled, the 8 outputs go to the OFF or high impedance state. The memories are available in both open-collector and TRI-STATE® versions.

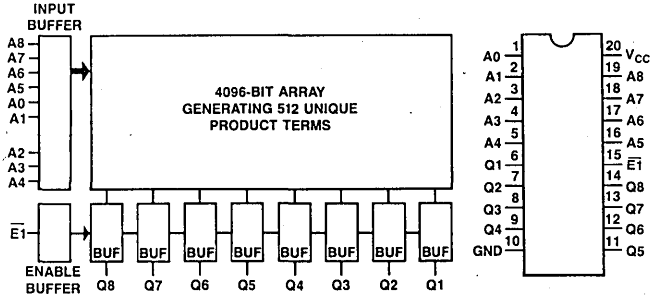
PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions.

Features

- Advanced titanium-tungsten (Ti:W) fuses.
- Schottky-clamped for high speed.
  Address access—25 ns typ.
  Enable access—15 ns typ.
  Enable recovery—15 ns typ.
- PNP inputs for reduced input loading.
- All DC and AC parameters guaranteed over temperature.
- Low voltage TRI-SAFE™ programming.

	Miltary	Commercial	Open-Collector	TRI-STATE	Package
DM74S473		X	X		N,J
DM74S472		X		X	N,J
DM54S473	X		X		J
DM54S472	X			X	J

Table 11.8.1 512 × 8 4096-Bit TTL PROM Optics



Order Number:
DM74S473 J, DM74S472 J,
DM54S473 J, or DM54S472 J
See NS Package J208

Order Number:
DM74S473 N or DM74S472 N
See NS Package N20A

Figure 11.8.1 Block and Connection Diagram

DM54/74S473, DM54/74S472, DM54/74S473A, DM54/74S472A, DM54/74S472B DC Electrical Characteristics

(Note 3)

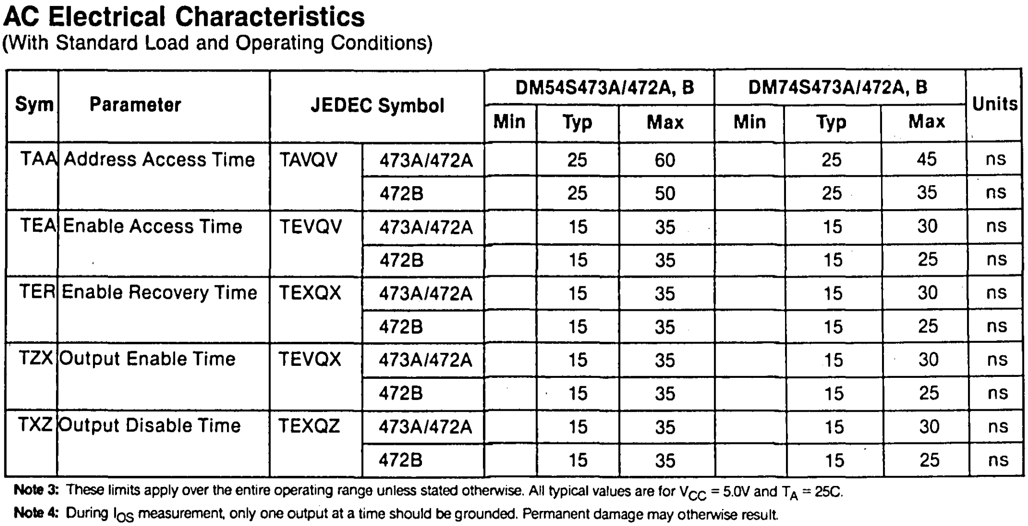
Sym	Parameter	Conditions	DM54S473/472			DM74S473/472			Units
			Min	Typ	Max	Min	Typ	Max	
I_{IL}	Input Load Current	V_{CC} = Max, V_{IN} = 0.45V		-80	-250		-80	-250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max, V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max, V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
I_{OZ}	Output Leakage Current	V_{CC} = Max, V_{CEX} = 2.4V			50			50	μA
	(Open-Collector Only)	V_{CC} = Max, V_{CEX} = 5.5V			100			100	μA
V_C	Input Clamp Voltage	V_{CC} = Min, I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25C, 1MHz		4.0			4.0		pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V T_A = 25C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max, Inputs Grounded All Outputs Open		110	155		110	155	mA
TRI-STATE® Parameters									
I_{OS}	Short Circuit Output Current	V_O = 0V, V_{CC} = Max (Note 4)	-20		-70	-20		-70	mA
I_{OZ}	Output Leakage	V_{CC} = Max, V_O = 0.45 to 2.4V			+50			+50	μA
	(TRI-STATE)	Chip Disabled			-50			-50	μA
V_{OH}	Output Voltage High	I_{OH} = -2.0mA	2.4	3.2					V
		I_{OH} = 6.5mA				2.4	3.2		V

AC Electrical Characteristics

(With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol	DM54S473/472			DM74S473/472			Units
			Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV		40	75		40	60	ns
TEA	Enable Access Time	TEVQV		15	35		15	30	ns
TER	Enable Recovery Time	TEXQX		15	35		15	30	ns
TZX	Output Enable Time	TEVQX		15	35		15	30	ns
TXZ	Output Disable Time	TEXQZ		15	35		15	30	ns

Table 11.8.2 AC and DC Specifications for (512 × 8) 4096-Bit TTL PROM



AC Electrical Characteristics

(With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol		DM54S473A/472A, B			DM74S473A/472A, B			Units
				Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV	473A/472A		25	60		25	45	ns
			472B		25	50		25	35	ns
TEA	Enable Access Time	TEVQV	473A/472A		15	35		15	30	ns
			472B		15	35		15	25	ns
TER	Enable Recovery Time	TEXQX	473A/472A		15	35		15	30	ns
			472B		15	35		15	25	ns
TZX	Output Enable Time	TEVQX	473A/472A		15	35		15	30	ns
			472B		15	35		15	25	ns
TXZ	Output Disable Time	TEXQZ	473A/472A		15	35		15	30	ns
			472B		15	35		15	25	ns

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.8.2 AC and DC Specifications for (512 × 8) 4096-Bit TTL PROM (Cont.)

11.9 DM54/74S475, DM54/74S474; DM54/74S475A, DM54/74S474A; DM54/74S474B, (512 × 8) K-BIT TTL PROMs

General Description

These Schottky memories are organized in the popular 512 words by 8 bits configuration. Memory enable inputs are provided to control the output states. When the device is enabled, the outputs represent the contents of the selected word. When disabled, the 8 outputs go to the OFF or high impedance state. The memories are available in both open-collector and TRI-STATE® versions.

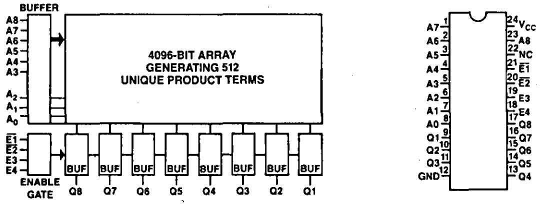
PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions.

Features

- Advanced titanium-tungsten (Ti:W) fuses.
- Schottky-clamped for high speed.
  Address access—25 ns typ.
  Enable access—15 ns typ.
  Enable recovery—15 ns typ.
- PNP inputs for reduced input loading.
- All DC and AC parameters guaranteed over temperature.
- Low voltage TRI-SAFE™ programming.

	Military	Commercial	Open-Collector	TRI-STATE	Package
DM74S475		X	X		N,J
DM74S474		X		X	N,J
DM54S475	X		X		J
DM54S474	X			X	J

Table 11.9.1 (512 × 8) 4096-Bit TTL PROM



ORDER NUMBER:
DM74S475 J, DM74S475 J,
DM45S475 J, DM54S474 J
SEE NS PACKAGE J24A

ORDER NUMBER:
DM74S475 N OR DM74S474 N
SEE NS PACKAGE N24A

Figure 11.9.1 Block and Connection Diagram

DM54/74S745, DM54/74S474, DM54/74S475A, DM54/74S474A, DM54/74S474B

DC Electrical Characteristics

(Note 3)

Sym	Parameter	Conditions	DM54S475/474			DM74S475/474			Units
			Min	Typ	Max	Min	Typ	Max	
I_{IL}	Input Load Current	V_{CC} = Max, V_{IN} = 0.45V		-80	-250		-80	-250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max, V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max, V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
I_{OZ}	Output Leakage Current (Open-Collector Only)	V_{CC} = Max, V_{CEX} = 2.4V			50			50	μA
		V_{CC} = Max, V_{CEX} = 5.5V			100			100	μA
V_C	Input Clamp Voltage	V_{CC} = Min, I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25C, 1MHz		4.0			4.0		pF
C_O	Output Capacitance	V_{CC} = 5.0V, V_O = 2.0V T_A = 25C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max, Inputs Grounded All Outputs Open		115	170		115	170	mA
TRI-STATE® Parameters									
I_{OS}	Short Circuit Output Current	V_O = 0V, V_{CC} = Max (Note 4)	-20		-70	-20		-70	mA
I_{OZ}	Output Leakage (TRI-STATE)	V_{CC} = Max, V_O = 0.45 to 2.4V			+50			+50	μA
		Chip Disabled			-50			-50	μA
V_{OH}	Output Voltage High	I_{OH} = -2.0mA	2.4	3.2					V
		I_{OH} = 6.5mA				2.4	3.2		V

AC Electrical Characteristics

(With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol	DM54S475/474			DM74S475/474			Units
			Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV		40	75		40	65	ns
TEA	Enable Access Time	TEVQV		20	40		20	35	ns
TER	Enable Recovery Time	TEXQX		20	40		20	35	ns
TZX	Output Enable Time	TEVQX		20	40		20	35	ns
TXZ	Output Disable Time	TEXQZ		20	40		20	35	ns

Table 11.9.2 AC and DC Specifications for (512 × 8) 4096-Bit TTL High Speed PROM

AC Electrical Characteristics

(With Standard Load and Operating Conditions)

Sym	Parameter	JEDEC Symbol		DM54S475A/474A, B			DM74S475A/474A, B			Units
				Min	Typ	Max	Min	Typ	Max	
TAA	Address Access Time	TAVQV	475A/474A		25	60		25	45	ns
			474B		25	50		25	35	ns
TEA	Enable Access Time	TEVQV			15	35		15	25	ns
TER	Enable Recovery Time	TEXQX			15	35		15	25	ns
TZX	Output Enable Time	TEVQX			15	35		15	25	ns
TXZ	Output Disable Time	TEXQZ			15	35		15	25	ns

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.9.2 AC and DC Specifications for (512 × 8) 4096-Bit TTL High Speed RPOM (Cont.)

11.10 DM77/87SR474, DM77/87SR474B (512 × 8) 4K-BIT REGISTERED TTL PROM

General Description

The DM77/87SR474 is an electrically programmable Schottky TTL read-only memory with D-type, master-slave registers on-chip. This device is organized as 512 words by 8-bits and is available in the TRI-STATE output version. Designed to optimize system performance, this device also substantially reduces the cost and size of pipelined microprogrammed systems and other designs wherein accessed PROM data is temporarily stored in a register. The DM77/87SR474 also offers maximal flexibility for memory expansion and data bus control by providing both synchronous and asynchronous output enables. All outputs will go into the OFF state if the synchronous chip enable ($\overline{\text{GS}}$) is high before the rising edge of the clock, or if the asynchronous chip enable ($\overline{\text{G}}$) is held high. The outputs are enabled when $\overline{\text{GS}}$ is brought low before the rising edge of the clock and $\overline{\text{G}}$ is held low. The $\overline{\text{GS}}$ flip-flop is designed to power-up to the OFF state with the application of V_{CC}.

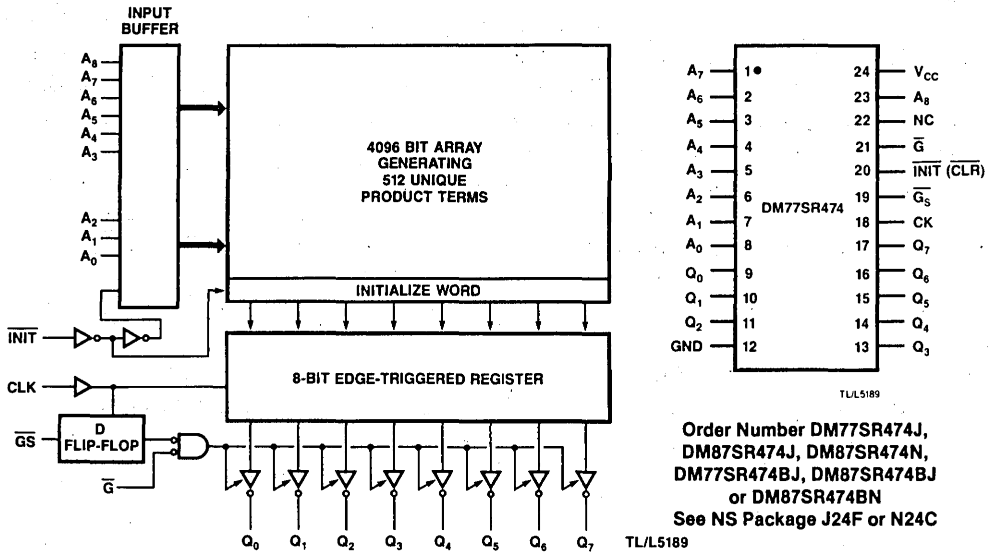
Data is read from the PROM by first applying an address to inputs A_0-A_8. During the setup time the output of the array is loaded into the master flip-flop of the data register. During the rising edge (low-to-high transition) of the clock, the data is then transferred to the slave of the flip-flop and will appear on the output if the output is enabled. Following the rising edge clock transition, the addresses and synchronous chip enable can be removed and the output data will remain stable.

The DM77/87SR474 also features an initialize function, $\overline{\text{INIT}}$. The initialize function provides the user with an extra word of programmable memory which is accessed with single pin control by applying a low on $\overline{\text{INIT}}$. The initialize function is synchronous and is loaded into the output register on the next rising edge of the clock. The unprogrammed state of the $\overline{\text{INIT}}$ is all lows, providing a CLEAR function when not programmed.

PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions. Once programmed, it is impossible to go back to a low.

Features

- On-chip, edge-triggered registers.
- Synchronous and asynchronous enables for word expansion.
- Programmable synchronous register INITIALIZE.
- 24-pin, 300 mil thin-DIP package.
- 35 ns address setup and 20 ns clock to output for maximum system speed.
- Highly reliable, titanium tungsten fuses.
- TRI-STATE® outputs.
- Low voltage TRI-SAFE™ programming.
- All parameters guaranteed over temperature.
- Pinout compatible with DM77SR181 (1K × 8) Registered PROM for future expansion.



Order Number DM77SR474J, DM87SR474J, DM87SR474N, DM77SR474BJ, DM87SR474BJ or DM87SR474BN
See NS Package J24F or N24C

Figure 11.10.1 Block and Connection Diagrams

DM77/87SR474
DC Electrical Characteristics (Note 3)

Symbol	Parameter	Conditions	DM77SR474			DM87SR474			Units
			Min.	Typ.	Max.	Min.	Typ.	Max.	
L_{IL}	Input Load Current	V_{CC} = Max., V_{IN} = 0.45V		−80	−250		−80	−250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max., V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max., V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min., I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
I_{OZ}	Output Leakage Current	V_{CC} = Max., V_{CEX} = 2.4V			50			50	μA
V_C	Input Clamp Voltage	V_{CC} = Min., I_{IN} = −18mA		−0.8	−1.2		−0.8	−1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25°C, 1MHz		4.0			4.0		pF
C_0	Output Capacitance	V_{CC} = 5.0V, V_0 = 2.0V T_A = 25°C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max., Inputs Grounded All Outputs Open		135	185		135	185	mA
	TRI-STATE Parameters								
I_{OS}	Short Circuit Output Current	V_0 = 0V, V_{CC} = Max. (Note 4)	−20		−70	−20		−70	mA
I_{OZ}	Output Leakage (TRI-STATE)	V_{CC} = Max., V_0 = 0.45 to 2.V Chip Disabled	−50		+50	−50		+50	μA
V_{OH}	Output Voltage High	I_{OH} = −2.0mA	2.4	3.2					V
		I_{OH} = −6.5mA				2.4	3.2		V

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25°C.
Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.10.1 AC and DC Specifications for (512 × 8) 4K-Bit Registered TTL PROMs

DM77/87SR474B
Switching Characteristics

Symbol	Parameter		DM77SR474			DM87SR474			Units
			Min.	Typ.	Max.	Min.	Typ.	Max.	
$t_{S(A)}$	Address to CLK (High) Setup Time	SR474	55	20		50	20		
		SR474B	40	20		35	20		
$t_{H(A)}$	Address to CLK (High) Hold Time		0	−5		0	−5		ns
$t_{S(INIT)}$	$\overline{\text{INIT}}$ to CLK (High) Setup Time		30	20		25	20		ns
$t_{H(INIT)}$	$\overline{\text{INIT}}$ to CLK (High) Hold Time		0	−5		0	−5		ns
$t_{PHL(CLK)}$ $t_{PLH(CLK)}$	Delay from CLK (High) to Output (High or Low)	SR474		15	30		15	27	ns
		SR474B		15	25		15	20	
$t_{WH(CLK)}$ $t_{WL(CLK)}$	CLK Width (High or Low)		25	13		20	13		ns
$t_{S(GS)}$	$\overline{\text{GS}}$ to CLK (High) Setup Time		10	0		10	0		ns
$t_{H(GS)}$	$\overline{\text{GS}}$ to CLK (High) Hold Time		5	0		5	0		ns
$t_{PZL(CLK)}$ $t_{PZH(CLK)}$	Delay from CLK (High) to Output Active (High or Low)			20	35		20	30	ns
$t_{PZL(G)}$ $t_{PZH(G)}$	Delay from $\overline{\text{G}}$ (Low) to Output Active (High or Low)			15	30		15	25	ns
$t_{PLZ(CLK)}$ $t_{PHZ(CLK)}$	Delay from CLK (High) to Output Inactive (TRI-STATE)			20	35		20	30	ns
$t_{PLZ(G)}$ $t_{PHZ(G)}$	Delay from $\overline{\text{G}}$ (Low) to Output Inactive (TRI-STATE)			15	30		15	25	ns

Table 11.10.1 AC and DC Specifications for (512 × 8) 4K-Bit Registered TTL PROMs (Cont.)

11.11 DM77/87SR476, DM77/87SR25, DM77/87SR476B, DM77/87SR25B (512 × 8) 4K-BIT REGISTERED TTL PROMs

General Description

The DM77/87SR476 is an electrically programmable schottky TTL read-only memory with D-type, masterslave registers on-chip. This device is organized as 512 words by 8-bits and is available in the TRI-STATE® output version. Designed to optimize system performance, this device also substantially reduces the cost and size of pipelined microprogrammed systems and other designs wherein accessed PROM data is temporarily stored in a register. The DM77/87SR476 also offers maximal flexibility for memory expansion and data bus control by providing both synchronous and asynchronous output enables. All outputs will go into the OFF state if the synchrounous chip enable ($\overline{GS}$) is high before the rising edge of the clock, or if the asynchrounous chip enable ($\overline{G}$) is held high. The outpus are enabled when $\overline{GS}$ is brought low before the rising edge of the clock and $\overline{G}$ is held low. The $\overline{GS}$ flip-flop is designed to power up to the OFF state with the application of V_{CC}.

Data is read from the PROM by first applying an address to inputs A0-A8. During the rising edge (low-to-high transition) of the clock, the data is then transferred to the slave of the flip-flop and will appear on the output if the output is enabled. Following the rising edge clock transition, the addresses and synchronous chip enable can be removed and the output data will remain stable.

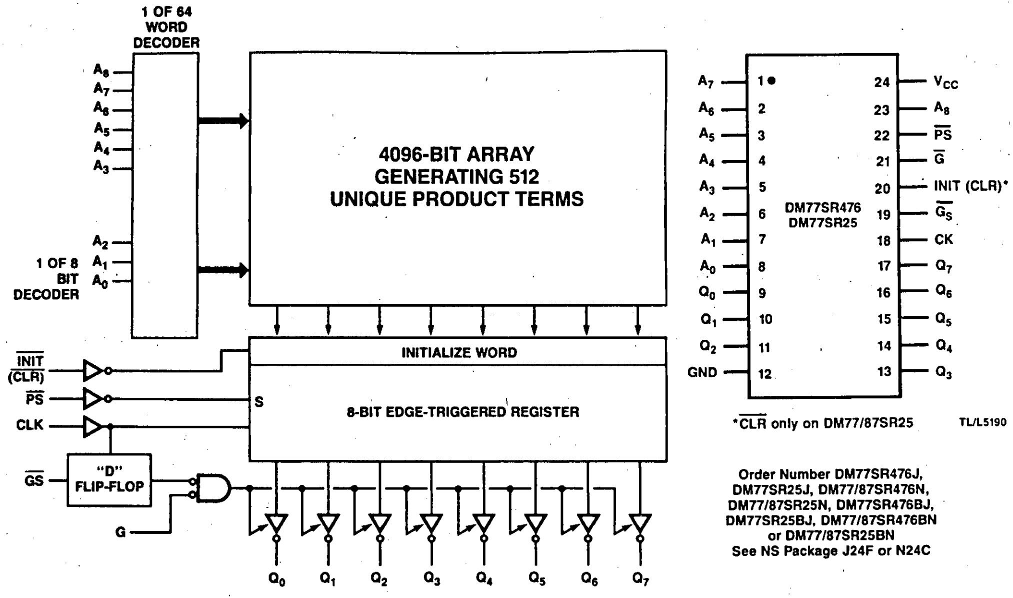
The DM77SR476 also features an initialize function, $\overline{INIT}$. The initialize function provides the user with an extra word of programmable memory which is accessed with single pin control by applying a low on $\overline{INIT}$. The initialize function is asynchronous and is loaded into the output register when $\overline{INIT}$ is brought low. The unprogrammed state of the $\overline{INIT}$ is all lows, which makes it compatible with the CLEAR function on the AM27S25. $\overline{PS}$ loads lows into the output registers when brought low.

PROMs are shipped from the factory with lows in all locations. A high may be programmed into any selected location by following the programming instructions. Once programmed, it is impossible to go back to a low.

Features

- Functionally compatible with AM27S25.
- On-chip, edge-triggered registers.
- Synchronous and asynchronous enables for word expansion.
- Programmable asynchronous INITIALIZE (SR476 only).
- 24-pin, 300 mil thin-DIP package.
- 35 ns address setup and 20 ns clock to output for maximum system speed.

- Highly reliable, titanium tungsten fuses.
- TRI-STATE outputs.
- Low voltage TRI-SAFE™ programming.
- All parameters guaranteed over temperature.
- Preset input.



*CLR only on DM77/87SR25

Order Number DM77SR476J, DM77SR25J, DM77/87SR476N, DM77/87SR25N, DM77SR476BJ, DM77SR25BJ, DM77/87SR476BN or DM77/87SR25BN
See NS Package J24F or N24C

Figure 11.11.1 Block and Connection Diagrams

DM77/87SR476, DM77/87RS25, DM77/87SR476B, DM77/87SR25B

DC Electrical Characteristics (Note 3)

Symbol	Parameter	Conditions	DM77SR474			DM87SR474			Units
			Min.	Typ.	Max.	Min.	Typ.	Max.	
L_{IL}	Input Load Current	V_{CC} = Max., V_{IN} = 0.45V		−80	−250		−80	−250	μA
I_{IH}	Input Leakage Current	V_{CC} = Max., V_{IN} = 2.7V			25			25	μA
		V_{CC} = Max., V_{IN} = 5.5V			1.0			1.0	mA
V_{OL}	Low Level Output Voltage	V_{CC} = Min., I_{OL} = 16mA		0.35	0.50		0.35	0.45	V
V_{IL}	Low Level Input Voltage				0.80			0.80	V
V_{IH}	High Level Input Voltage		2.0			2.0			V
I_{OZ}	Output Leakage Current	V_{CC} = Max., V_{CEX} = 2.4V			50			50	μA
V_C	Input Clamp Voltage	V_{CC} = Min., I_{IN} = −18mA		−0.8	−1.2		−0.8	−1.2	V
C_I	Input Capacitance	V_{CC} = 5.0, V_{IN} = 2.0V T_A = 25°C, 1MHz		4.0			4.0		pF
C_0	Output Capacitance	V_{CC} = 5.0V, V_0 = 2.0V T_A = 25°C, 1MHz, Outputs Off		6.0			6.0		pF
I_{CC}	Power Supply Current	V_{CC} = Max., Inputs Grounded All Outputs Open		135	185		135	185	mA
	TRI-STATE Parameters								
I_{OS}	Short Circuit Output Current	V_0 = 0V, V_{CC} = Max. (Note 4)	−20		−70	−20		−70	mA
I_{OZ}	Output Leakage (TRI-STATE)	V_{CC} = Max., V_0 = 0.45 to 2.V Chip Disabled	−50		+50	−50		+50	μA
V_{OH}	Output Voltage High	I_{OH} = −2.0mA	2.4	3.2					V
		I_{OH} = −6.5mA				2.4	3.2		V

Note 3: These limits apply over the entire operating range unless stated otherwise. All typical values are for V_{CC} = 5.0V and T_A = 25°C.

Note 4: During I_{OS} measurement, only one output at a time should be grounded. Permanent damage may otherwise result.

Table 11.11.1 AC and DC Specifications for (512 × 8) 4K-Bit Registered TTL PROMs

Switching Characteristics

Symbol	Parameter		DM77SR476, 476B DM77SR25, 25B			DM87SR476, 476B DM87SR25, 25B			Units
			Min.	Typ.	Max.	Min.	Typ.	Max.	
$t_{S(A)}$	Address to CLK (High) Setup Time	SR476, SR25	55	20		50	20		ns
		SR476B, SR25B	40	20		35	20		
$t_{H(A)}$	Address to CLK (High) Hold Time		0	−5		0	−5		ns
$t_{PHL(CLK)}$ $t_{PLH(CLK)}$	Delay from CLK (High) to Output (High or Low)	SR476, SR25		15	30		15	27	ns
		SR476B, SR25B		15	25		15	20	
$t_{WH(CLK)}$ $t_{WL(CLK)}$	CLK Width (High or Low)		25	13		25	13		ns
$t_{S(GS)}$	$\overline{GS}$ to CLK (High) Setup Time		10	0		10	0		ns
$t_{H(GS)}$	$\overline{GS}$ to CLK (High) Hold Time		5	0		5	0		ns
$t_{PLH(PS)}$	Delay from $\overline{PS}$ (Low) to Output (High)			20	30		20	25	ns
$t_{PLH(INIT)}$ $t_{PHL(INIT)}$	Delay from $\overline{INIT}$ (Low) to Output (Low or High)			20	30		20	25	ns
$t_{WL(PS)}$	$\overline{PS}$ Pulse Width (Low)		15	10		15	10		ns
$t_{WL(INIT)}$	$\overline{INIT}$ Pulse Width (Low)		15	10		15	10		
$t_{s(PS)}$	$\overline{PS}$ Recovery (High) to CLK (High)		25	10		20	10		ns
$t_{S(INIT)}$	$\overline{INIT}$ Recovery (High) to CLK (High)		25	10		20	10		ns
$t_{PZL(CLK)}$ $t_{PZH(CLK)}$	Delay from CLK (Low) to Active Output (High or Low)			20	35		20	30	ns
$t_{PZL(G)}$ $t_{PZH(G)}$	Delay from $\overline{G}$ (Low) to Active Output (Low or High)			15	30		15	25	ns
$t_{PLZ(CLK)}$ $t_{PHZ(CLK)}$	Delay from CLK (High) to Inactive Output (TRI-STATE)			20	35		20	30	ns
$t_{PLZ(G)}$ $t_{PHZ(G)}$	Delay from $\overline{G}$ (High) to Inactive Output (TRI-STATE)			15	30		15	25	ns

Table 11.11.1 AC and DC Specifications for 20-Pin Ultra High-Speed, Medium PAL Devices (Cont.)

11.12 REGISTERED PROM PROGRAMMING PROCEDURE

National Schottky PROMs are shipped from the factory with all fuses intact. As a result, the outputs will be low (logical "0") for all addresses. To generate high (logical "1") levels at the outputs, the device must be programmed. Information regarding commercially available programming equipment can be obtained from National. If it is desired to build your own programmer, the following conditions must be observed:

1. Programming should be attempted only at ambient temperatures between 15 ° and 30 °C.
2. Address and enable inputs must be driven with TTL logic levels during programming and verification.
3. Programming will occur at the selected address when V_{CC} is at 10.5V, and at the selected bit location when the output pin representing that bit is at 10.5V, and the device is subsequently enabled. To achieve these conditions in the appropriate sequence, the following procedure must be followed:

   a) Select the desired word by applying high or low levels to the appropriate address inputs. Disable the device by applying a high level to the asynchronous Chip Enable input $\overline{G}$. $\overline{GS}$ is held low during the enable programming time.

   b) Increase V_{CC} from nominal to 10.5 volts (± 0.5V) with a slew rate between 1.0 and 10.0V/μs. Since V_{CC} is the source of the current required to program the fuse as well as the I_{CC} for the device at the programming voltage, it must be capable of supplying 750mA at 11.0V.

   c) Select the output where a logical high is desired by raising that output voltage to 10.5V (± 0.5V). Limit the slew rate from 1.0 to 10.0V/μs. This voltage change may occur simultaneously with the increase in V_{CC}, but must not precede it. It is critical that only one output at a time be programmed since the internal circuits can only supply programming current to one bit at a time. Outputs not being programmed must be left open or connected to a high impedance source of 20kΩ minimum. (Remember that the outputs of the device are disabled at this time.)

   d) Enable the device by taking the chip enable ($\overline{G}$) to a low level. This is done with a pulse of 10μs. The 10μs duration refers to the time that the circuit (device) is enabled. Normal input levels are used, and rise and fall times are not critical.

   e) Verify that the bit has been programmed by first removing the programming voltage from the output and then reducing V_{CC} to 4.0V (± 0.2V) for one verification and to 6.0V (± 0.2V) for a second verification. Verification at a V_{CC} level of 4.0V and 6.0V will guarantee proper output states over the V_{CC} and temperature range of the programmed part. Each data verification must be preceded by a positive going (low-to-high) clock edge to load the data from the array into the output register. The device must be enabled to sense the state of the outputs. During verification, the loading of the output must be within specified I_{OL} and I_{OH} limits. Steps b, c, and d must be repeated up to 10 times or until verification that the bit has been programmed.

f) The initialize word is programmed by setting INIT input to a logic low and programming the initialize word output by output in the same manner as any other address. This can be accomplished by inverting the A9 address input from the PROM programmer and applying it to the INIT input. Using this method, the initialize word will program at address 512.

g) Following verification, apply five additional programming pulses to the bit being programmed. The programming procedure is now complete for the selected bit.

h) Repeat steps a through f for each bit to be programmed to a high level. If the procedure is performed on an automatic programmer, the duty cycle of V_{CC} at the programming voltage must be limited to a maximum of 25%. This is necessary to minimize device junction temperatures. After all selected bits are programmed, the entire contents of the memory should be verified.

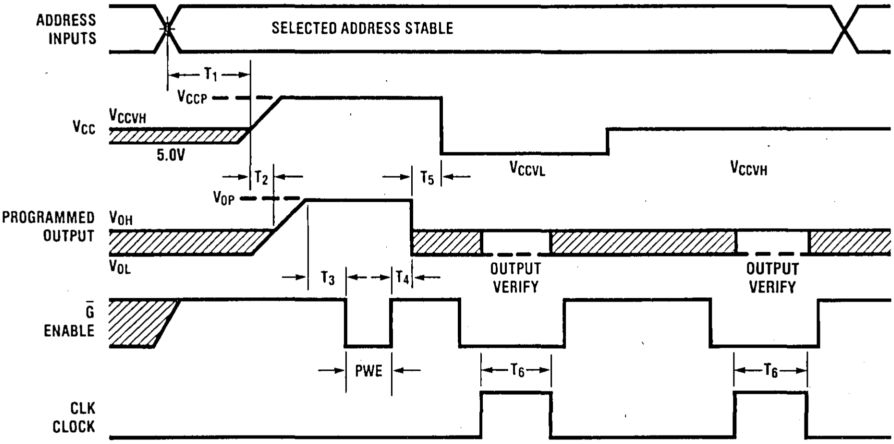
Programming Parameters

Do not test or you may program the device

Symbol	Parameter	Test Conditions	Min.	Recommended Value	Max.	Units
V_{CCP}	Required V_{CC} for Programming		10	10.5	11	V
I_{CCP}	I_{CC} During Programming	$V_{CC} = 11V$			750	mA
V_{OP}	Required Output Voltage for Programming		10	10.5	11	V
I_{OP}	Output Current While Programming	$V_{OUT} = 11V$			20	mA
I_{RR}	Rate of Voltage Change of V_{CC} or Output		1		10	V/μs
P_{WE}	Programming Pulse Width (Enabled)		9	10	11	μs
V_{CCVL}	Required Low V_{CC} for Verification		3.8	4	4.2	V
V_{CCVH}	Required High V_{CC} for Verification		5.8	6	6.2	V
M_{DC}	Maximum Duty Cycle for V_{CC} at V_{CCP}			25	25	%

Table 11.12.1 Programming Parameters. Do Not Test or You May Program the Device.

Programming Waveforms



$T_1 = 100$ ns MIN.
$T_2 = 5$ μs MIN. (T_2 MAY BE > 0 IF V_{CCP} RISES AT THE SAME RATE OR FASTER THAN V_{OP}.)
$T_3 = 100$ ns MIN.
$T_4 = 100$ ns MIN.
$T_5 = 100$ ns MIN.
$T_6 = 50$ ns MIN.

Figure 11.12.1 Programming Waveforms Registered PROM

11.13 NON-REGISTERED PROM PROGRAMMING PROCEDURE

National Schottky PROMs are shipped from the factory with all fuses intact. As a result, the outputs will be low (logical "0") for all addresses. To generate high (logical "1") levels at the outputs, the device must be programmed. Information regarding commercially available programming equipment can be obtained from National. If it is desired to build your own programmer, the following conditions must be observed:

1. Programming should be attempted only at ambient temperatures between 15 and 30 degrees Celsius.

2. Address and enable inputs must be driven with TTL logic levels during programming and verification.

3. Programming will occur at the selected address when V_{CC} is at 10.5 volts, and at the selected bit location when the output pin representing that bit is at 10.5 volts, and the device is subsequently enabled. To achieve these conditions in the appropriate sequence, the following procedure must be followed:

a) Select the desired word by applying high or low levels to the appropriate address inputs. Disable the device by applying a high level to asynchronous Chip Enable input G. GS is held low during the enable programming time.

b) Increase V_{CC} from nominal to 10.5 volts (± 0.5V) with a slew rate between 1.0 and 10.0V/μs. Since V_{CC} is the source of the current required to program the fuse as well as the I_{CC} for the device at the programming voltage, it must be capable of supplying 750 mA at 11.0 V.

c) Select the output where a logical high is desired by raising that output voltage to 10.5 volts (± 0.5V). Limit the slew rate from 1.0 to 10.0V/μs. This voltage change may occur simultaneously with the increase in V_{CC}, but must not precede it. It is critical that only one output at a time be programmed since the internal circuits can only supply programming current to one bit at a time. Outputs not being programmed must be left open or connected to a high impedance source of 20kΩ minimum. (Remember that the outputs of the device are disabled at this time.)

d) Enable the device by taking the chip enable ($\overline{G}$) to a low level. This is done with a pulse of 10μs. The 10μs duration refers to the time that the circuit (device) is enabled. Normal input levels are used and rise and fall times are not critical.

e) Verify that the bit has been programmed by first removing the programming voltage from the output and then reducing V_{CC} to 4.0V (± 0.2V) for one verification and to 6.0V (± 0.2V) for a second verification. Verification at a V_{CC} level of 4.0V and 6.0V will guarantee proper output states over the V_{CC} and temperature range of the programmed part. Each data verification must be preceded by a positive going (low-to-high) clock edge to load the data from the array into the output register. The device must be enabled to sense the state of the outputs. During verification, the loading of the output must be within specified I_{OL} and I_{OH} limits. Steps b, c, and d must be repeated up to 10 times or until verification that the bit has been programmed.

f) Following verification, apply five additional programming pulses to the bit being programmed. The programming procedure is now complete for the selected bit.

g) Repeat steps a through f for each bit to be programmed to a high level. If the procedure is performed on an automatic programmer, the duty cycle of V_{CC} at the programming voltage must be limited to a maximum of 25%. This is necessary to minimize device junction temperatures. After all selected bits are programmed, the entire contents of the memory should be verified.

Note: Since only an enabled device is programmed, it is possible to program these parts at the board level if all of the programming parameter are complied with.

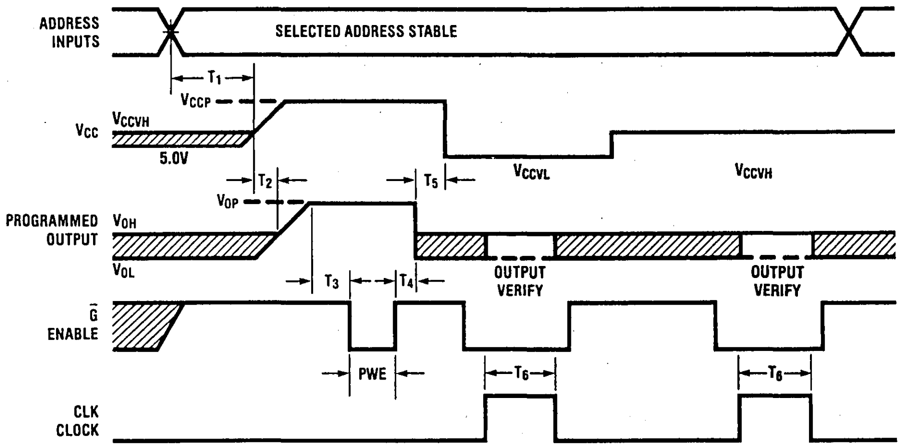
TRI-STATE® is a registered trademark of National Semiconductor Corp.
TRI-SAFE™ is a trademark of National Semiconductor Corp.

Programming Parameters
Do not test or you may program the device

Symbol	Parameter	Test Conditions	Min.	Recommended Value	Max.	Units
V_{CCP}	Required V_{CC} for Programming		10	10.5	11	V
I_{CCP}	I_{CC} During Programming	$V_{CC} = 11V$			750	mA
V_{OP}	Required Output Voltage for Programming		10	10.5	11	V
I_{OP}	Output Current While Programming	$V_{OUT} = 11V$			20	mA
I_{RR}	Rate of Voltage Change of V_{CC} or Output		1		10	V/μs
P_{WE}	Programming Pulse Width (Enabled)		9	10	11	μs
V_{CCVL}	Required Low V_{CC} for Verification		3.8	4	4.2	V
V_{CCVH}	Required High V_{CC} for Verification		5.8	6	6.2	V
M_{DC}	Maximum Duty Cycle for V_{CC} at V_{CCP}			25	25	%

Table 11.13.1 Programming Parameters
Do Not Test or You May Program the Device

Programming Waveforms Non-Registered PROM



$T_1 = 100$ ns MIN.
$T_2 = 5$ μs MIN. (T_2 MAY BE > 0 IF V_{CCP} RISES AT THE SAME RATE OR FASTER THAN V_{OP}.)
$T_3 = 100$ ns MIN.
$T_4 = 100$ ns MIN.
$T_5 = 100$ ns MIN.
$T_6 = 50$ ns MIN.

Figure 11.13.1 Programming Waveforms Non-Registered PROM

MANUFACTURER	SYSTEM #
DATA I/O	5/17/19/29A
PRO-LOG	M910, M980
KONTRON	MPP80S
STAG	PPX
AIM	RP400
DIGELEC	UP803
STARPLEX™	

Table 11.13.2 Approved Programmers for NSC PROMs

11.14 QUALITY ENHANCEMENT PROGRAMS

A + PROGRAM*			B + PROGRAM		
Test	Condition	Guaranteed LOT AQL 5	Test	Condition	Guaranteed LOT AQL 5
D.C. Parametric And Functionality	25°C	0.05	D.C. Parametric And Functionality	25°C	0.05
	Each Temperature Extreme	0.05		Each Temperature Extreme	0.05
A.C. Parametric	25°C	0.4	A.C. Parametric	25°C	0.4
Mechanical	Critical	0.01	Mechanical	Critical	0.01
	Major	0.28		Major	0.28
Seal Tests Hermetic	Fine Leak (5 x 10 $^{-8}$)	0.4	Seal Tests Hermetic	Fine Leak (5 x 10 $^{-8}$)	0.4
	Gross	0.4		Gross	0.4

*Includes 160 hours of burn-in at 125°C.

Table 11.14.1 Quality Enhancement Program for Bipolar Memory

12

Package Outlines

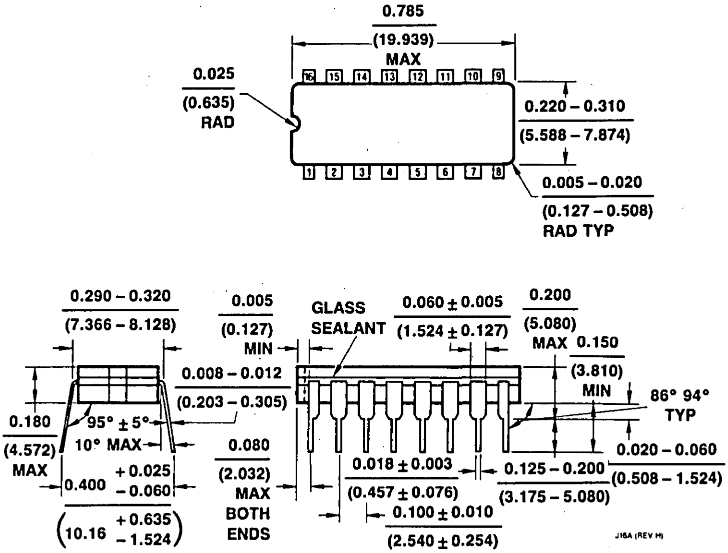


Figure 12.1 NS Package J16A 16-Lead Cavity DIP (J)

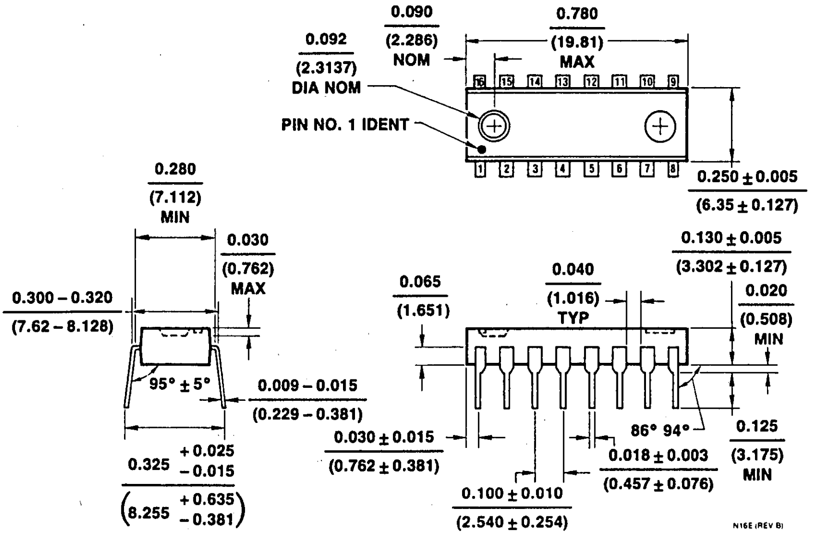


Figure 12.2 NS Package N16E 16-Lead Molded DIP (N) (Substitute for N16A)

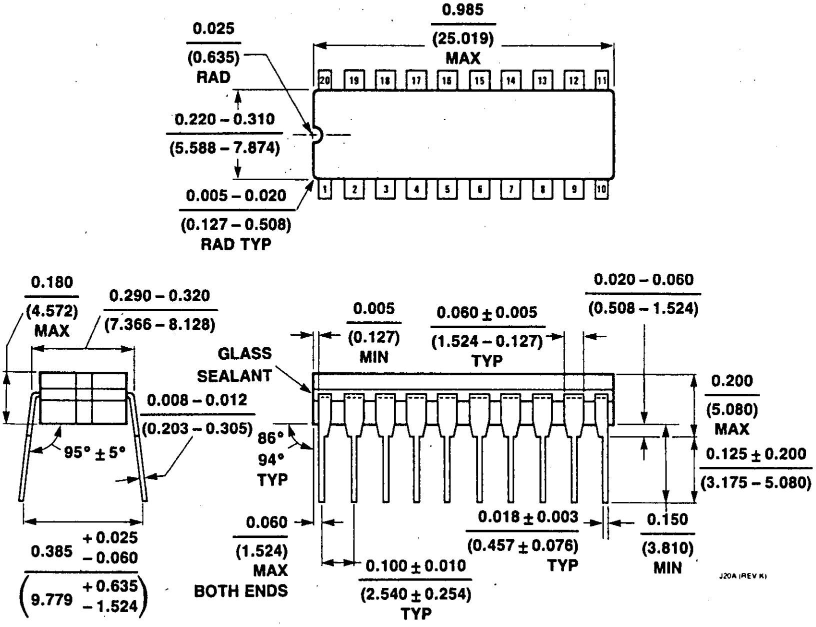


Figure 12.3 NS Package J20A 20-Lead Cavity DIP (J)

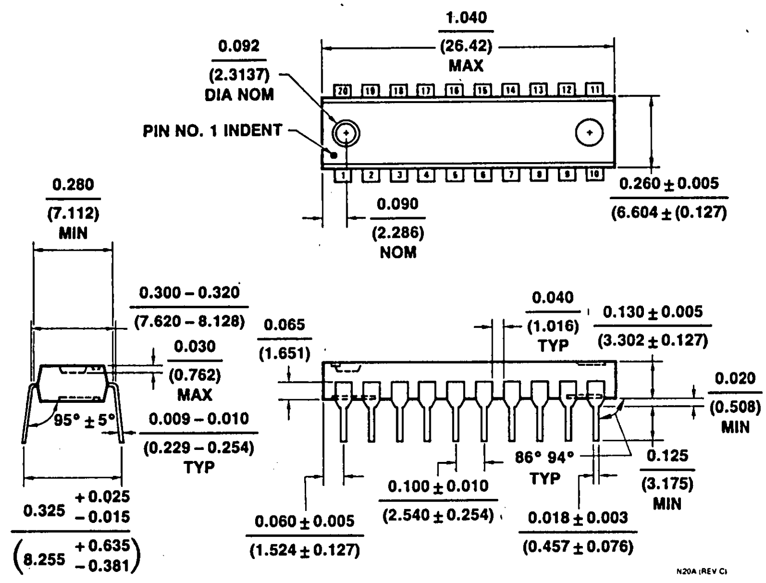


Figure 12.4 NS Package N20A 20-Lead Molded DIP, (N)

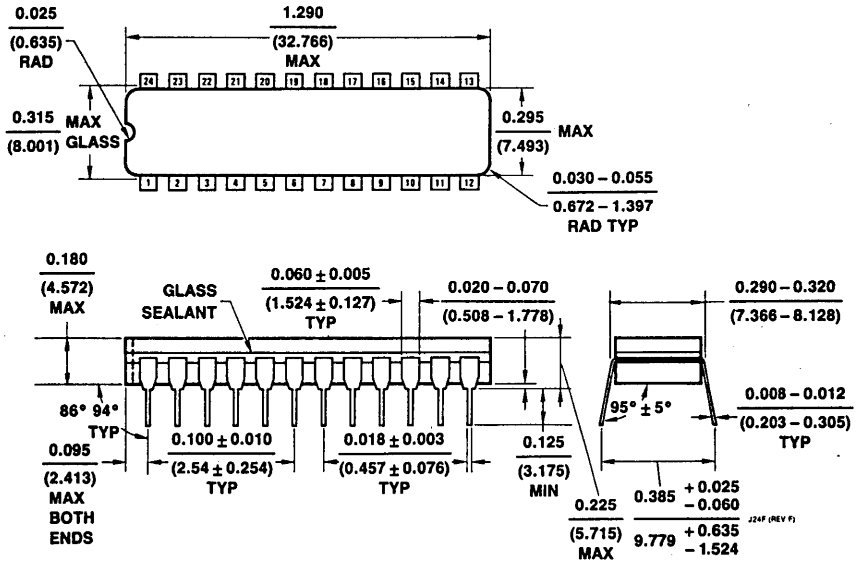


Figure 12.5 NS Package J24F 24-Lead Cavity DIP (J)

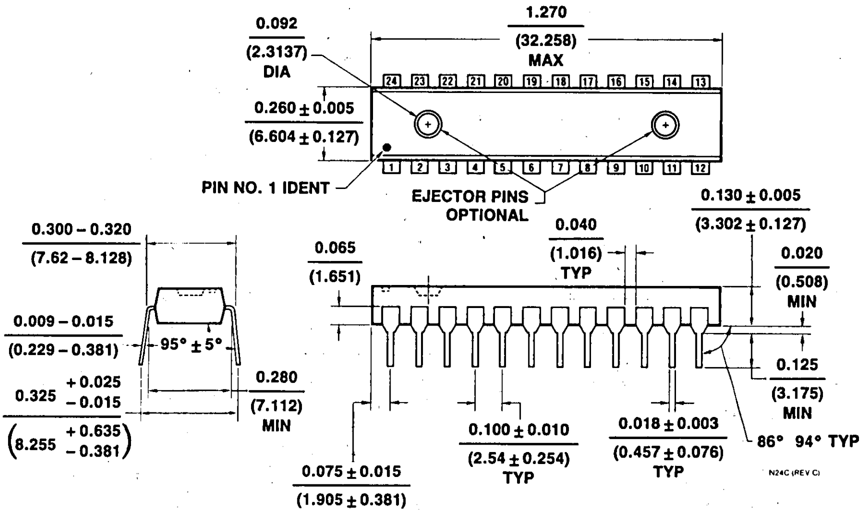


Figure 12.6 NS Package N24C 24-Lead Molded DIP (N)

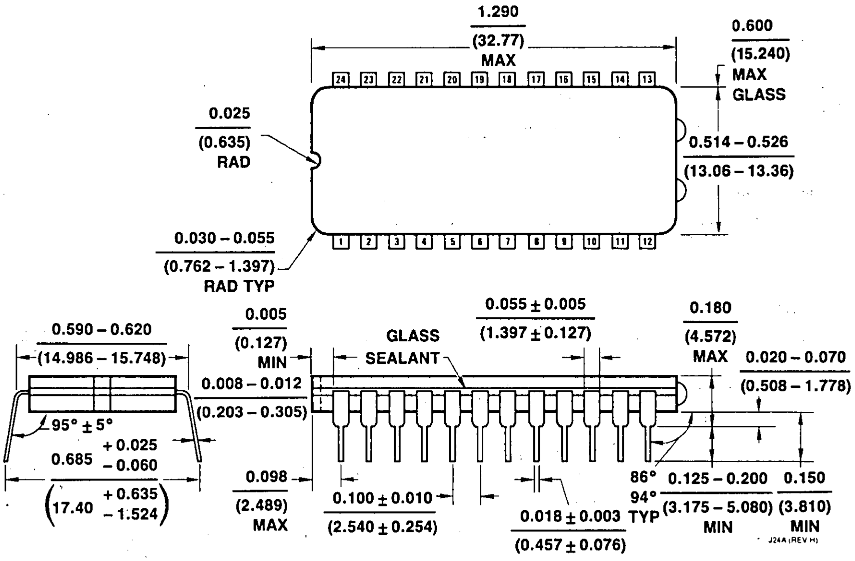


Figure 12.7 NS Package J24A 24-Lead Cavity DIP (J)

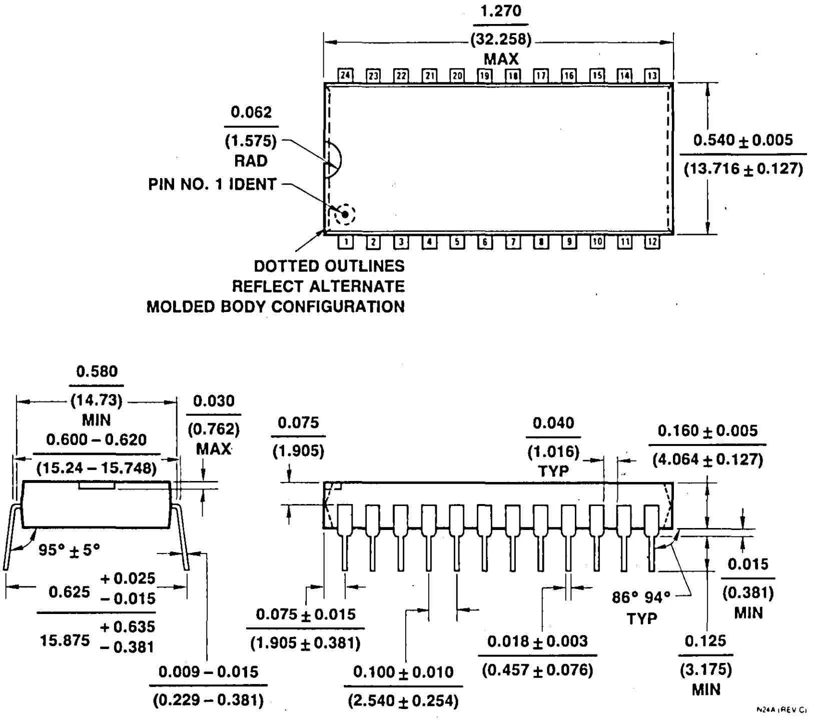


Figure 12.8 NS Package N24A 24-Lead Molded DIP (N)

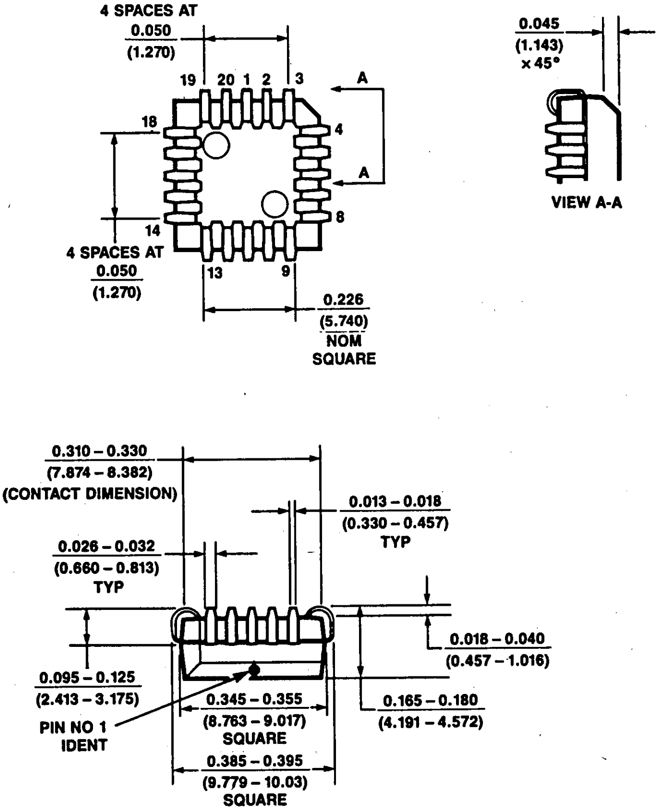


Figure 12.9 NS Package PCC-20 20-Lead Plastic Leaded Chip Carrier (V)

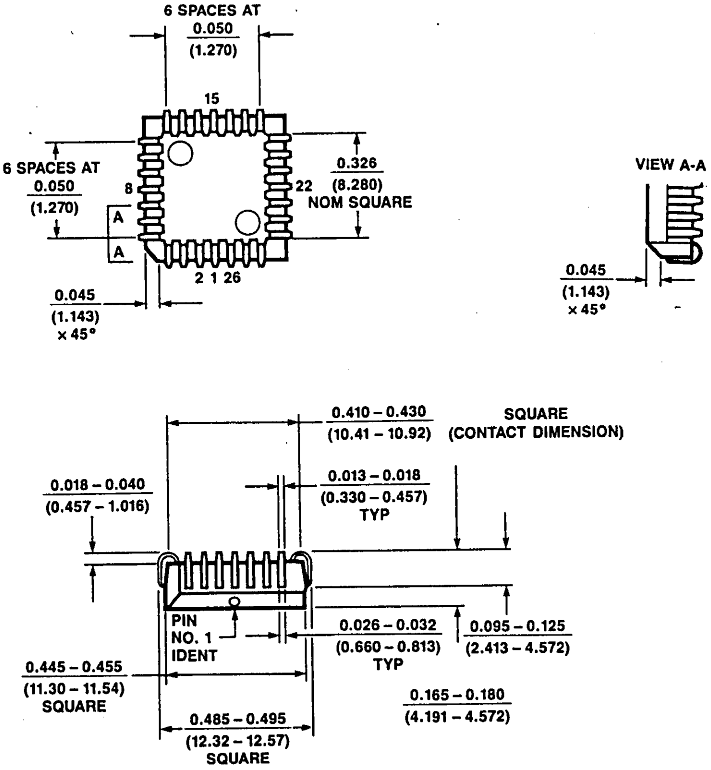


Figure 12.10 NS Package PCC-28 28-Lead Plastic Leaded Chip Carrier (V)

13

Terminology

Term	Explanation
PAL Device	Programmable Array Logic. AND-OR Array with a programmable AND array and a fixed OR array.
PROM	Programmable Read-Only Memory. AND-OR Array with a fixed AND array and a programmable OR array.
FPLA	Field-Programmable Logic Array. AND-OR Array with a programmable AND array and a programmable OR array.
Product Term (Pn)	Logical AND operation on input variables. Example: $P_0 = A_0A_1A_{15}$, $P_{10} = A_2A_5$
Summing Term (Sn)	Logical OR operation on product terms. Example: $S_1 = P_0 + P_{10}$ $= A_0A_1A_{15} + A_2A_5$
Output Polarity	Inversion or Non-inversion of summing term outputs.
Don't Care	Variable can take any logic state without affecting logic operation.
Active High	Output is a logic high when Sum-of-Products expression is true. Within programmable logic context, refers to a non-inverted output.
Active Low	Output is a logic low when Sum-of-Products expression is true. Within programmable logic context, refers to an inverted output.
+	Fixed connection.
⋇	Programmable connection in virgin array.

Term	Explanation
+	Unconnected in programmed part.
✳	Programmed, connected.
Maximum Clock Frequency, f_{MAX}:	The highest rate at which the clock input of a bistable circuit can be driven through its required sequence while maintaining stable transitions of logic level at the output with input conditions established that should cause changes of output logic level in accordance with the specification.
High Level Input Current, I_{IH}:	The current into an input when a high level voltage is applied to that input.*
High Level Output Current, I_{OH}:	The current into an output with input conditions applied that, according to the product specification, will establish a high level at the output.*
Low Level Input Current, I_{IL}:	The current into an input when a low level voltage is applied to that input.*
Low Level Output Current, I_{OL}:	The current into an output with input conditions applied that, according to the product specification, will establish a low level at the output.*
Off-State (High-Impedance State) Output Current of a 3-State Output), I_{OZ}:	The current into an output having 3-state capability with input conditions applied that, according to the product specification, will establish the high-impedance state at the output.*
Short-Circuit Output Current, I_{OS}:	The current into an output when that output is short-circuited to ground (or other specified potential) with input conditions applied to establish the output logic level farthest from ground potential (or other specified potential).*
Supply Current, I_{CC}:	The current into the V_{CC} Supply terminal of an integrated circuit.*

Term	Explanation
Hold Time, t_h:	The interval during which a signal is retained at a specified input terminal after an active transition occurs at another specified input terminal. Notes: 1. The hold time is the actual time between two events and may be insufficient to accomplish the intended result. A minimum value is specified that is the shortest interval for which correct operation of the logic element is guaranteed. 2. The hold time may have a negative value, in which case the minimum limit defines the longest interval (between the release of data and the active transition) for which correct operation of the logic element is guaranteed.
Output Enable Time (of a 3-State Output) to High Level, t_{PZH}(or Low Level, t_{PZL}):	The propagation delay time between the specified reference points on the input and output voltage waveforms with the 3-state output changing from a high-impedance (off) state to the defined high (or low) level.
Output Enable Time (of a 3-State Output) to High or Low Level, t_{PZX}:	The propagation delay time between the specified reference points on the input and output voltage waveforms with the 3-state output changing from a high-impedance (off) state to either of the defined active levels (high or low).
Output Disable Time (of a 3-State Output) from High Level, t_{PHZ}(or Low Level, t_{PLZ}):	The propagation delay time between the specified reference points on the input and output voltage waveforms with the 3-state output changing from the defined high (or low) level to a high-impedance (off) state.
Output Disable Time (of a 3-State Output) from High or Low Level, t_{PXZ}:	The propagation delay time between the specified reference points on the input and output voltage waveforms with the 3-state output changing from either of the defined active levels (high or low) to a high-impedance (off) state.

Term	Explanation
Propagation Delay Time, t_{PD}:	The time between the specified reference points on the input and output voltage waveforms with the output changing from one defined level (high or low) to the other defined level.
Propagation Delay Time, Low-to-High Level Output, t_{PLH}:	The time between the specified reference points on the input and output voltage waveforms with the output changing from the defined low level to the defined high level.
Propagation Delay Time, High-to-Low Level Output, t_{PHL}:	The time between the specified reference points on the input and output voltage waveforms with the output changing from the defined high level to the defined low level.
Pulse Width, t_W:	The time interval between specified reference points on the leading and trailing edges of the pulse waveform.
Setup Time, t_{su}	The time interval between the application of a signal that is maintained at a specified input terminal and a consecutive active transition at another specified input terminal. Notes: 1. The setup time is the actual time between two events and may be insufficient to accomplish the setup. A minimum value is specified that is the shortest interval for which correct operation of the logic element is guaranteed. 2. The setup time may have a negative value in which case the minimum limit defines the longest interval (between the active transition and the application of the other signal) for which correct operation of the logic element is guaranteed.
High Level Input Voltage, V_{IH}:	An input voltage within the more positive (less negative) of the two ranges of values used to represent the binary variables. Note: A minimum is specified that is the least positive value of high level voltage for which

Term	Explanation
	operation of the logic elements within specification limits is guaranteed.
High Level Output Voltage, V_{OH}:	The voltage at an output terminal with input conditions applied that, according to the product specification, will establish a high level at the output.
Input Clamp Voltage, V_{IC}:	An input voltage in a region of relatively low differential resistance that serves to limit the input voltage swing.
Low Level Input Voltage, V_{IL}:	An input voltage level within the less positive (more negative) of the two ranges of values used to represent the binary variables. Note: A maximum is specified that is the most positive value of the low level input voltage for which operation of the logic element within specification limits is guaranteed.
Low Level Output Voltage, V_{OL}:	The voltage at an output terminal with input conditions applied that according to the product specification will establish a low level at the output.
Negative-Going Threshold Voltage, V_{T-}	The voltage level at a transition-operated input that causes operation of the logic element according to specification as the input voltage falls from a level above the positive-going threshold voltage, V_{T+}.
Positive-Going Threshold Voltage, $V_T +$:	The voltage level at a transition-operated input that causes operation of the logic element according to specification as the input voltage rises from a level below the negative-going threshold voltage, V_{T-}.

*Current out of a terminal is given as a negative value.

Appendix – an Overview of LSI Testing Techniques

The growth in the complexity ana performance of digital circuits can only be described as explosive. Large-scale integrated circuits are being used today in a variety of applications, many of which require highly reliable operation. This is causing concern among designers of tests for LSI circuits. The testing of these circuits is difficult for several reasons:

- The number of faults that has to be considered is large, since an LSI circuit contains thousands of gates, memory elements, and interconnecting lines, all individually subject to different kinds of faults.
- The observability and controllability of the internal elements of any LSI circuit are limited by the available number of I/O pins. As more and more elements are packed into one chip, the task of creating an adequate test becomes more difficult. A typical LSI chip may contain 5000 gates but only 40 I/O pins.
- The implementation details of the circuits usually are not disclosed by the manufacturer. For example, the only source on information about commercially available microprocessors is the user's manual, which details the instruction set and describes the architecture of the microprocessor at the register-transfer level, with some information of the system timing. The lack of implementation information eliminates the use of many powerful test generation techniques that depend on the actual implementation of the unit under test.
- As more and more gates and flip-flops are packed into one chip, new failure modes — such as pattern-sensitivity faults — arise.[1] These new types of faults are difficult to detect and require lengthy test patterns.
- The dynamic nature of LSI devices requires high-speed test systems that can test the circuits when they are operating at their maximum speeds.
- The bus structure of most LSI systems makes fault isolation more difficult because many devices — any of which can cause a fault — share the same bus.

M.S. Abadir, H.K. Reghbati, Authors.

- Solving the problems above increases the number of test patterns required for a successful test. This in turn increases both the time required for applying that test and the memory needed to store the test patterns and their results.

LSI testing is a challenging task. Techniques that worked well for SSI and MSI circuits, such as the D-algorithm, do not cope with today's complicated LSI and VLSI circuits. New testing techniques must be developed. In what follows, we describe some basic techniques developed to solve the problems associated with LSI testing.

A.1 TESTING METHODS

There are many test methods for LSI circuits, each with its own way of generating and processing test data. These approaches can be divided into two broad categories — *concurrent* and *explicit.*[2]

In concurrent approaches, normal user-application input patterns serve as diagnostic patterns. Thus testing and normal computation proceed concurrently. In explicit approaches, on the other hand, special input patterns are applied as tests. Hence, normal computation and testing occur at different times.

Concurrent Testing

Systems that are tested concurrently are designed such that all the information transferred among various parts of the system is coded with different types of error detecting codes. In addition, special circuits monitor this coded data continuously and signal detection of any fault.

Different coding techniques are required to suit the different types of information used inside LSI systems. For example, m-out-of-n codes (n-bit patterns with exactly m 1's and $n - m$ 0's) are suitable for coding control signals, while arithmetic codes are best suited for coding ALU operands.[3]

The monitoring circuits — *checkers* — are placed in various locations inside the systems so that they can detect most of the faults. A checker is sometimes designed in a way that enables it to detect a fault in its own circuitry as well as in the monitored data. Such a checker is called a *self-checking checker.*[3]

Hayes and McCluskey surveyed various concurrent testing methods that can be used with microprocessor-based LSI systems.[2] Concurrent testing approaches provide the following advantages:

- Explicit testing expenses (e.g., for test equipment, down time, and test pattern generation) are eliminated during the life of the system, since the data patterns used in normal operation serve as test patterns.

- The faults are detected instantaneously during the use of the LSI chip, hence the first faulty data pattern caused by a certain fault is detected. Thus, the user can rely on the correctness of his output results within the degree of fault coverage provided by the error detection code used. In explicit approaches, on the other hand, nothing can be said about the correctness of the results until the chip is explicitly tested.

- Transient faults, which may occur during normal operation, are detected if they cause any faulty data pattern. These faults cannot be detected by any explicit testing method.

Unfortunately, the concurrent testing approach suffers from several problems that limit its usage in LSI testing:

- The application patterns may not exercise all the storage element or all the internal connection lines. Defects may exist in places that are not exercised, and hence the faults these defects would produce will not be detected. Thus, the assumption that faults are detected as they occur, or at least before any other fault occurs, is no longer valid. Undetected faults will cause fault accumulation. As a result, the fault detection mechanism may fail because most error detection codes have a limited capability for detecting multiple faults.
- Using error detecting codes to code the information signals used in an LSI chip requires additional I/O pins. At least two extra pins are needed as error signal indicators. (A single pin cannot be used, since such a pin stuck at the good value could go undetected.) Because of constraints on pin count, however, such requirements cannot be fulfilled.
- Additional hardware circuitry is required to implement the checkers and to increase the width of the data carriers used for storing and transferring the coded information.
- Designing an LSI circuit for concurrent testing is a much more complicated task than designing a similiar LSI circuit that will be tested explicitly.
- Concurrent approaches provide no control over critical voltage or timing parameters. Hence, devices cannot be tested under marginal timing and electrical conditions.
- The degree of fault coverage usually provided by concurrent methods is less than that provided by explicit methods.

The above-mentioned problems have limited the use of concurrent testing for most commercially available LSI circuits. However, as digital systems grow more complex and difficult to test, it becomes increasingly attractive to build test procedures into the UUT (unit under test) itself. We will not consider the concurrent approach further in this article. For a survey of work in concurrent testing, see Hayes and McCluskey.[2]

Explicit Testing

All explicit testing methods separate the testing process from normal operation. In general, an explicit testing process involves three steps:

- **Generating the test patterns.** The goal of this step is to produce those input patterns which will exercise the UUT under different modes of operation while trying to detect any existing fault.

- **Applying the test patterns to the UUT.** There are two ways to accomplish this step. The first is external testing — the use of special test equipment to apply the test patterns externally. The second is internal testing — the application of test patterns internally by forcing the UUT to execute a self-testing procedure.[2] Obviously, the second method can only be used with systems that can execute programs (for example, with microprocessor-based systems.) External testing gives better control over the test process and enables testing under different timing and electrical conditions. On the other hand, internal testing is easier to use because it does not need special test equipment or engineering skills.
- **Evaluating the responses obtained from the UUT.** This step is designed with one of two goals in mind. The first is the detection of an erroneous response, which indicates the existence of one or more faults (*go/no-go testing*). The other is the isolation of the fault, if one exists, in an easily replaceable module (*fault location testing*). Our interest in this article will be go/no-go testing, since fault location testing of LSI circuits sees only limited use.

Many explicit test methods have evolved in the last decade. They can be distinguished by the techniques used to generate the test patterns and to detect and evaluate the faulty responses (Figure A.1.1). In what follows, we concentrate on explicit testing

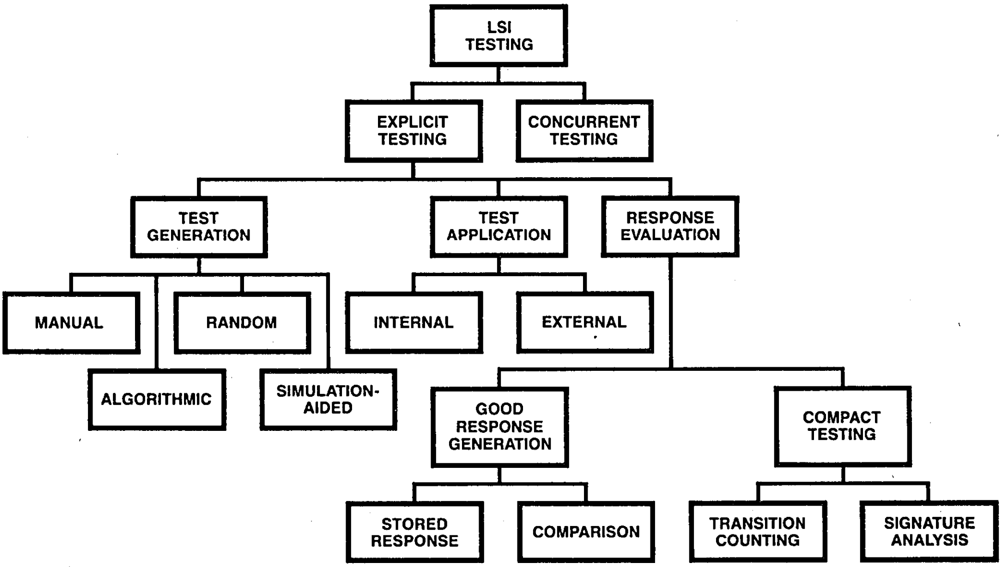


Figure A.1.1 LSI Test Technology

and present in-depth discussions of the methods of test generation and response evaluation employed with explicit testing.

A.2 TEST GENERATION TECHNIQUES

The test generation process represents the most important part of any explicit testing method. Its main goal is to generate those test patterns that, when applied to the UUT, sensitize existing faults and propagate a faulty response to an observable output of the UUT. A test sequence is considered good if it can detect a high percentage of the possible UUT faults; it is considered good, in other words, if its degree of *fault coverage* is high.

Rigorous test generation should consist of three main activities:

- Selecting a good descriptive model, at a suitable level, for the system under consideration. Such a model should reflect the exact behavior of the system in all its possible modes of operation.
- Developing a fault model to define the types of faults that will be considered during test generation. In selecting a fault model, the percentage of possible faults covered by the model should be maximized, and the test costs associated with the use of the model should be minimized. The latter can be accomplished by keeping the complexity of the test generation low and the length of the tests short. Clearly these objectives contradict one another — a good fault model is usually found as a result of a trade-off between them. The nature of the fault model is usually influenced by the model used to describe the system.
- Generating tests to detect all the faults in the fault model. This part of test generation is the soul of the whole test process. Designing a test sequence to detect a certain fault in a digital circuit usually involves two problems. First, the fault must be *excited*; i.e., a certain test sequence must be applied that will force a faulty value to appear at the fault site if the fault exists. Second, the test must be *made sensitive* to the fault; i.e., the effect of the fault must propagate through the network to an observable output.

Rigorous test generation rests heavily on both accurate descriptive (system) models and accurate fault models.

Test generation for digital circuits is usually approached either at the gate-level or at the functional level. The classical approach of modeling digital circuits as a group of connected gates and flip-flops has been used extensively. Using this level of description, test designers introduced many types of fault models, such as the classical stuck-at model. They also assumed that such models could describe physical circuit failures in terms of logic. This assumption has sometimes restricted the number of physical failures that can be modeled, but it has also reduced the complexity of test generation since failures at the elementary level do not have to be considered.

Many algorithms have been developed for generating tests for a given fault in combinational networks.[1,4,5,6,7] However, the complexity of these algorithms depends on the topology of the network; it can become very high for some circuits. Ibarra and

NP-COMPLETE PROBLEMS

The theory of NP-completeness is perhaps the most important theoretical development in algorithm research in the past decade.[29] Its results have meaning for all researchers who are developing computer algorithms.

It is an unexplained phenomenon that for many of the problems we know and study, the best algorithms for their solution have computing times which cluster into two groups. The first group consists of problems whose solution is bounded by a polynomial of small degree. Examples include ordered searching, which is 0(log n), polynomial evaluation, which is 0(n), and sorting, which is 0(n log n).[30]

The second group contains problems whose best-known algorithms are nonpolynomial. For example, the best algorithms described in Horowitz and Sahni's book[2] for the traveling salesman and the knapsack problems have a complexity of 0($n^2 2^n$) and 0($2^{n/2}$), respectively. In the quest to develop efficient algorithms, no one has been able to develop a polynomialtime algorithm for any problem in the second group.

The theory of NP-completeness does not provide a method for obtaining polynomial-time algorithms for these problems. But neither does it say that algorithms of this complexity do not exist. What it does show is that many of the problems for which there is no known polynomial-time algorithm are computationally related. In fact, a problem that is NP-complete has the property that it can be solved in polynomial time if all other NP-complete problems can also be solved in polynomial time.

References

29. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman, San Francisco, 1978.

30. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, Washington, DC, 1978.

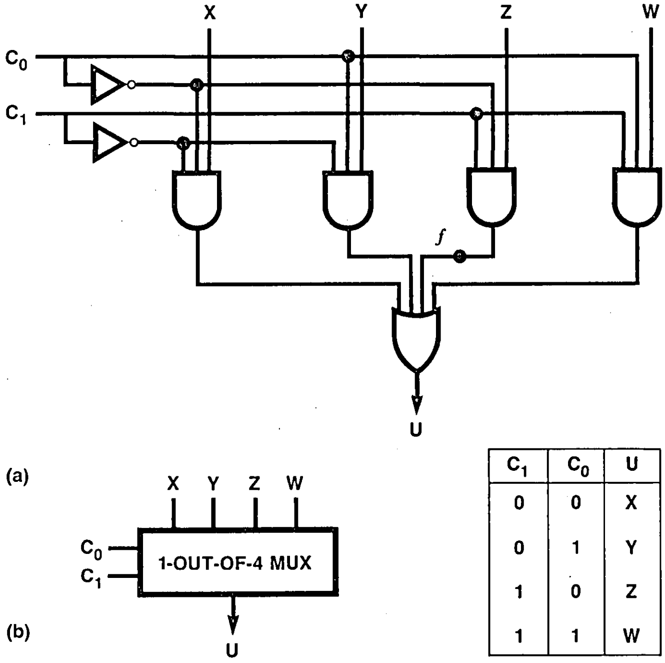
Sahni have shown that the problem of generating tests to detect single stuck-at faults in a combinational circuit modeled at the gate level is an NP-complete problem.[8] Moreover, if the circuit is sequential, the problem can become even more difficult depending on the deepness of the circuit's sequential logic.

Thus, for LSI cicuits having many thousands of gates, the gate level approach to the test generation problem is not very feasible. A new appoach, the functional level, is needed.

Another important reason for considering faults at the functional level is the constraint imposed on LSI testing by a user environment — the test patterns have to be generated without a knowledge of the implementation details of the chip at the gate level.

The only source of information usually available is the typical IC catalog, which details the different modes of operation and describes the general architecture of the circuit. With such information, the test designer finds it easier to define the functional behavior of the circuit and to associate faults with the functions. He can partition the UUT into various modules such as registers, multiplexers, ALUs, ROMs, and RAMs. Each module can be treated as a "black box" performing a specified input/output mapping. These modules can then be tested for *functional failures*; explicit consideration of faults affecting the internal lines is not necessary. The example given below clarifies the idea.

Consider a simple one-out-of-four multiplexers such as the one shown in Figure A.2.1. This multiplexer can be modeled at the gate level as shown in Figure A.2.1(a), or at the functional level as shown in Figure A.2.1(b).



C_1	C_0	U
0	0	X
0	1	Y
1	0	Z
1	1	W

Figure A.2.1 (a) A One-out-of-four Multiplexer-gate-level Description; (b) Functional-level Description.

A possible fault model for the gate-level description is the single stuck-at fault model. With this model, the fault list may contain faults such as the line labeled with "f" is stuck at 0, or the control line "C_0" is stuck at 1.

At the functional level, the multiplexer is considered a black box with a well-defined function. Thus, a fault model for it may specify the following as possible faults: selection of wrong source, selection of no source, or presence of stuck-at faults in the input lines or in the multiplexer output. With this model, the fault list may contain faults such as source "X" is selected instead of source "Y," or line "Z" is stuck at 1.

Ad hoc methods — which determine what faults are the most probable — are sometimes used to generate fault lists. But if no fault model is assumed, then the tests derived must be either exhaustive or a rather ad hoc check of the functionality of the system. Exhaustive tests are impossible for even small systems because of the enormous number of possible states, and superficial tests provide neither good coverage nor even an indication of what faults are covered.

Once the fault list has been defined, the next step is to find the test patterns required to detect the faults in the list. As previously mentioned, each fault first has to be excited so that an error signal will be generated somewhere in the UUT. Then this signal has to be sensitized at one of the observable outputs of the UUT. The three examples below describe how to excite and sensitize different types of faults in the types of modules usually encountered in LSI circuits.

Consider the gate-level description of the three-bit incrementer shown in Figure A.2.2.

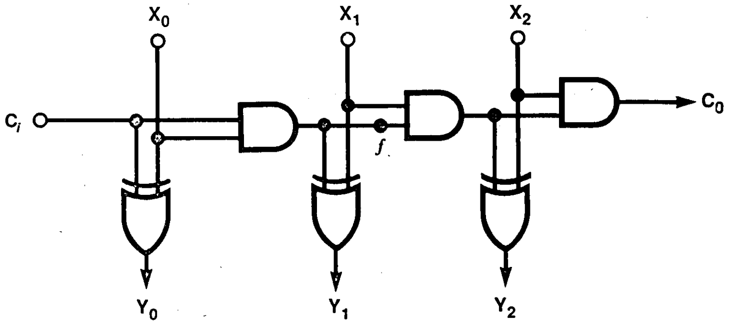


Figure A.2.2 Gate-level Description of a Three-Bit Incrementer

The incrementer output, Y_2 Y_1 Y_0, is the binary sum of C_i and the three-bit binary number $X_2X_1X_0$, while C_0 is the carry-out bit of the sum. Note that $X_0(Y_0)$ is the least significant bit of the incrementer input (output).

Assume we want to detect the fault "line *f* is stuck at 0." To excite that fault we will force a 1 to appear on line *f* so that, if it is stuck at 0, a faulty value will be generated at the fault site. To accomplish this both X_0 and C_i must be set to 1. To sensitize the faulty 0 at *f*, we have to set X_1 to 1; this will propagate the fault to Y_2 independent of the value of X_2. Note that if we set X_1 to 0, the fault will be masked since the AND gate output will be 0, independent of the value at *f*. Note also that X_2 was not specified in the above test. However, by setting X_2 to 1, the fault will propagate to both Y_2 and C_0, which makes the response evaluation task easier.

Consider a microprocessor RAM and assume we want to generate a test sequence to detect the fault "accessing word *i* in the RAM results in accessing the word *j* instead."

To excite such a fault, we will use the following sequence of instructions (assume a microprocessor with single-operand instructions):

Load the word 00 . . . 0 into the accumulator.

Store the accumulator contents into memory address *j*.

Load the word 11 . . . 1 into the accumulator.

Store the accumulator contents into memory address *i*.

If the fault exists, these instructions will force a 11 . . . 1 word to be stored in memory address *j* instead of 00 . . . 0. To sensitize the fault, we need only read what is in memory address *j*, using the appropriate instructions. Note that the RAM and its fault have been considered at the functional level, since we did not specify how the RAM is implemented.

Consider the program counter (PC) of a microprocessor and assume we want to generate a test sequence that will detect any fault in the incrementing mode of this PC, i.e., any fault that makes the PC unable to be incremented from x to $x + 1$ for any address x. One way to excite this fault is to force the PC to step through all the possible addresses. This can be easily done by initializing the PC to zero and then executing the no-operation instruction $x + 1$ times. As a result, the PC will contain an address different than $x + 1$. By executing another no-operation instruction, the wrong address can be observed at the address bus and the fault detected. In practice, such an exhaustive test sequence is very expensive, and more economical tests have to be used. Note that, as in the example immediately above, the problem and its solution have been considered at the functional level.

Four methods are currently used to generate test patterns for LSI circuits: manual test generation, algorithmic test generation, simulation-aided test generation, and random test generation.

Manual Test Generation

In manual test generation, the test designer carefully analyzes the UUT. This analysis can be done at the gate level, at the functional level or at a combination of the two. The analysis of the different parts of the UUT is intended to determine the specific patterns that will excite and sensitize each fault in the fault list. At one time, the manual approach was widely used for medium-and small-scale digital circuits. Then, the formulation of the D-algorithm and similar algorithms eliminated the need for analyzing each circuit manually and provided an efficient means to generate the required test patterns.[1,5] However, the arrival of LSI circuits and microprocessors required a shift back toward manual test generation techniques, because most of the algorithmic techniques used with SSI and MSI circuits were not suitable for LSI circuits.

Manual test generation tends to optimize the length of the test patterns and provides a relatively high degree of fault coverage. However, generating tests manually takes a considerable amount of effort and requires persons with special skills. Realizing that test generation has to be done economically, test designers are now moving in the direction of automatic test generation.

One good example of manual test generation is the work done by Sridhar and Hayes,[9] who generated test patterns for a simple bit-sliced microprocessor at the functional level.

A bit-sliced microprocessor is an array of n identical ICs called slices, each of which is a simple processor for operands of $k$bit length, where k is typically 2 or 4. The interconnections among the n slices are such that the entire array forms a processor for nkbit operands. The simplicity of the individual slices and the regularity of the interconnections make it feasible to use systematic methods for fault analysis and test generation.

Sridhar and Hayes considered a one-bit processor slice as a simplified model for the commercially available bit-sliced processors such as the Am2901.[10] A slice can be modeled as a collection of modules interconnected in a known way. These modules are regarded as black boxes with well-defined input-output relationships. Examples of these functional modules are ALUs, multiplexers, and registers. Combinational modules are described by their truth tables, while sequential modules are defined by their state tables (or state diagrams).

The following fault categories were considered:

- For combinational modules, all possible faults that induce arbitrary changes in the truth table of the module, but that cannot convert it into a sequential circuit.
- For sequential modules, all possible faults that can cause arbitrary changes in the state table of the module without increasing the number of states.

Only one module was assumed to be faulty at any time.

To test for the faults allowed by the above-mentioned fault model, all possible input patterns must be applied to each combinational module (exhaustive testing), and a checking sequence[11] to each sequential module. In addition, the responses of each module must be propagated to observable output lines. The tests required by the individual modules were easily generated manually — a direct consequence of the small operand size ($k = 1$). And because the slices were identical, the tests for one slice were easily extended to the whole array of slices. In fact, Sridhar and Hayes showed that an arbitrary number of simple interconnected slices could be tested with the same number of tests as that required for a single slice, as long as only one slice was faulty at one time. This property is called *C-testability.* Note that the use of carry-lookahead when connecting slices eliminates C-testability. Also note that slices with operand sizes equal to 2 or more usually are not C-testable.

The idea of modeling a digital system as a collection of interconnected functional modules can be used in modeling any LSI circuit. However, using exhaustive tests and checking sequences to test individual modules is feasible only for toy systems. Hence, the fault model proposed by Sridhar and Hayes, though very powerful, is not directly applicable to LSI testing.

PATH SENSITIZATION AND THE D-ALGORITHM

One of the classical fault detection methods at the gate and flip-flop level is the D-algorithm[1,5] employing the path sensitization testing technique.[4] The basic principle involved in path sensitization is relatively simple. For an input X; to detect a fault "line a is stuck at $j, j = 0, 1$," the input X; must cause the signal a in the normal (fault-free) circuit to take the value $\overline{j}$. This condition is necessary but not sufficient to detect the fault. The error signal must be propagated along some path from its site to an observable output.

To generate a test to detect a stuck-at fault in a combinational circuit, the following path sensitization procedure must be followed:

- Excitation—The inputs must be specified so as to generate the appropriate value (0 for stuck-at 1 and 1 for stuck-at 0) at the site of the fault.
- Error propagation—A path from the fault site to an observable output must be selected, and additional signal values to propagate the fault signal along this path must be specified.
- Line justification—Input values must be specified so as to produce the signals values specified in the step above.

There may be several possible choices for error propagation and line justification. Also, in some cases there may be a choice of ways in which to excite the fault. Some of these choices may lead to an inconsistency, and so the procedure must backtrack and consider the next alternative. If all the alternatives lead to an inconsistency, this implies that the fault cannot be detected.

To facilitate the path sensitization process, we introduce the symbol D to represent a signal which has the value 1 in a normal circuit and 0 in a faulty circuit, and $\overline{\mathrm{D}}$ to represent a signal which has the value 0 in a normal circuit and 1 in a faulty circuit. The path sensitization procedure can be formulated in terms of a cubical algebra[1,5] to enable automatic generation of test. This also facilitates test generation for more complex fault models and for fault propagation through complex logic elements.

We shall define three types of cubes (i.e., line values specified in positional notation):

- For a circuit element E which realizes the combinational function f, the "primitive cubes" offer a typical presentation of the prime implicants of f and $\overline{f}$. These cubes concisely represent the logical behavior of E.
- A "primitive D-cube of a fault" in a logic element E specifies the minimal input conditions that must be applied to E in order to produce an error signal (D or $\overline{\mathrm{D}}$) at the output of E.

- The "propagation D-cubes" of a logic element E specify the minimal input conditions to the logic element that are required to propagate an error signal on an input (or inputs) to the output of that element.

To generate a test for a stuck-at fault in a combinational circuit, the D-algorithm must perform the following:

1. Fault excitation—A primitive D-cube of the fault under consideration must be selected. This generates the error signal D or $\overline{D}$ at the site of the fault. (Usually a choice exists in this step. The initial choice is arbitrary, and it may be necessary to backtrack and consider another choice).

2. Implication—In Step 1 some of the gate inputs or outputs may be specified so as to uniquely imply values on other signals in the circuit. The implication procedure is performed both forwards and backwards through the circuit. Implication is performed as follows: Whenever a previously unspecified signal value becomes specified, all the elements associated with this signal are placed on a list B and processed one at a time (and removed). For each element processed, it is determined if new values of 0, 1, D, and $\overline{D}$ are implied, based on the previously specified inputs and outputs. These implied line values are determined by intersecting the test cube (which specifies all the previously determined signal values of the circuit) with the primitive cubes of the element. If any line values are implied, the are specified in the test cube, and the associated gates are placed on the list B. An inconsistency occurs when a value is implied on a line which has been specified previously to a different value. If an inconsistency occurs, the procedure must backtrack to the last point a choice existed, reset all lines to their values at that point, and begin again with the next choice.

3. D-propagation—All the elements in the circuit whose output values are unspecified and whose input has some signal D or $\overline{D}$ are placed on a list called the D-frontier. In this step, an element from the D-frontier is selected and values are assigned to its unspecified inputs so as to propagate the D or $\overline{D}$ on its inputs to one of its outputs. This is accomplished by intersecting the current test cube describing the circuit signal values with a propagation D-cube of the selected element of the D-frontier, resulting in a new test cube. If such intersection is impossible, a new element in the D-frontier is selected. If intersection fails for all the elements in the D-frontier, the procedure backtracks to the last point at which a choice existed.

4. Implication of D-propagation—Implication is performed for the new test cube derived in Step 3.

5. Steps 3 and 4 are repeated until the faulty signal has been propagated to an output of the circuit.

6. Line justification—Execution of Steps 1 to 5 may result in specifying the output value of an element E but leaving some of the inputs to the element unspecified. The unspecified inputs of such an element are assigned values so as to produce the desired output value. This is done by intersecting the test cube with any primitive cube of the element which has no specified signal values that differ from those of the test cube.

7. Implication of line justification—Implication is performed on the new test cube derived in Step 6.

8. Steps 6 and 7 are repeated until all specified element outputs have been justified. Backtracking may again be required.

References

1. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Washington, DC, 1976.

4. D.B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinatorial Nets," *IEEE Trans. Electronic Computers*, Vol. EC-15, No. 2, Feb. 1966, pp. 63-73.

5. J. P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronic Computers*, Vol. EC-16, No. 5, Oct. 1967, pp. 567-580.

Algorithmic test generation

In algorithmic test generation, the test designer devises a set of algorithms to generate the 1's and 0's needed to test the UUT. Algorithmic test techniques are much more economical than manual techniques. They also provide the test designer with a high level of flexibility. Thus, he can improve the fault coverage of the tests by replacing or modifying parts of the algorithms. Of course, this task is much simpler than modifying the 1's and 0's in a manually generated test sequence.

Techniques that use the gate-level description of the UUT, such as path sensitization[4] and the D-algorithm,[5] can no longer be used in testing complicated LSI circuits. Thus, the problem of generating meaningful sets of tests directly from the functional description of the UUT has become increasingly important. Relatively little work has been done on functional-level testing of LSI chips that are not memory elements.[9,12,13,14,15,16,17] Functional testing of memory chips is relatively simple because of the regularity of their design and also because their components can be easily controlled and observed from the outside. Various test generation algorithms have been developed to detect different types of faults in memories.[1,18] In the rest of this section we will concentrate on the general problem of generating tests for irregular LSI chips, i.e., for LSI chips which are not strictly memory chips.

It is highly desirable to find an algorithm that can generate tests for any LSI circuit, or at least most LSI circuits. One good example of work in this area is the technique proposed by Thatte and Abraham for generating tests for microprocessors.[12,13] Another approach, pursued by the authors of this article, is a test generation procedure capable of handling general LSI circuits.[15,16,17]

The Thatte-Abraham Technique

Microprocessors constitute a high percentage of today's LSI circuits. Thatte and Abraham[12,13] approached the microprocessor test generation problem at the functional level.

The test generation procedure they developed was based on:

- A functional description of the microprocessor at the register-transfer level. The model is defined in terms of data flow among storage units during the execution of an instruction. The functional behavior of a microprocessor is thus described by information about its instruction set and the functions performed by each instruction.

- A fault model describing faults in the various functional parts of the UUT (e.g., the data transfer function, the data storage function, the instruction decoding and control function). This fault model describes the faulty behavior of the UUT without knowing its implementation details.

The microprocessor is modeled by a graph. Each register in the microprocessor (including general-purpose registers and accumulator, stack, program counter, address buffer, and processor status word registers) is represented by a node of the graph. Instructions of the microprocessors are classified as being of transfer, data manipulation, or branch type. There exists a directed edge (labeled with an instruction) from one node to another if during an execution of the instruction data flow occurs from the register represented by the first node to that represented by the second. Examples of instruction representation are given in Figure A.2.3.

Having described the function or the structure of the UUT, one needs an appropriate fault model in order to derive useful tests. The approach used by Thatte and Abraham is to partition the various functions of a microprocessor into five classes: the register decoding function, the instruction decoding and control function, the data storage function, the data transfer function, and the data manipulation function. Fault models are derived for each of these functions at a higher level and independently of the details of implementation for the microprocessor. The fault model is quite general. Tests are derived allowing any number of faults, but only in one function at a time; this restriction exists solely to cut down the complexity of test generation.

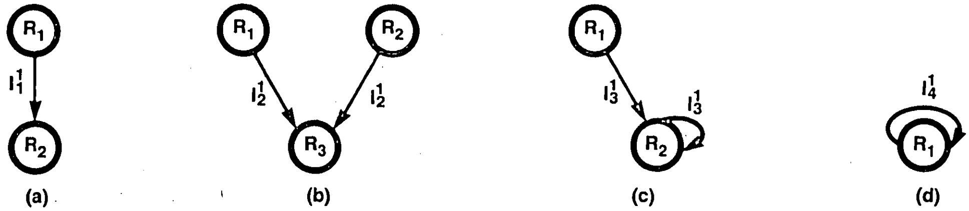


Figure A.2.3 Representations of Microprocessor Instruction — I_1, (a) Transfer Instruction, $R_2 \leftarrow R_1$; (b) Add Instruction, $R_3 \leftarrow R_1 + R_2$; (c) I_3, OR Instruction, $R_2 \leftarrow R_1$ OR R_2; (d) I_4 Rotate Left Instruction.

The fault model for the register decoding function allows any possible set of registers to be accessed instead of a particular register. (If the set is null then no register is accessed.) This fault model is thus very general and independent of the actual realization of the decoding mechanism.

For the instruction decoding and control function, the faulty behavior of the microprocessor is specified as follows. When instruction I_j, is executed any one of the following can happen:

- Instead of instruction I_j, some other instruction I_k is executed. This fault is denoted by $F(I_j/I_k)$.
- In addition to instruction I_j, some other instruction I_k is activated. This fault is denoted by $F(I_j/I_j + I_k)$.
- No instruction is executed. This fault is denoted by $F(I_j/\phi)$.

Under this specification, any number of instructions can be faulty.

In the fault model for the data storage function, any cell in any data storage module is allowed to be stuck at 0 or 1. This can occur in any number of cells.

The fault model for the data transfer function includes the following types of faults:

- A line in a path used in the execution of an instruction is stuck at 0 or 1.
- Two lines of a path used in the instruction are coupled; i.e., they fail to carry different logic values.

Note that the second fault type cannot be modeled by single stuck-at faults. The transfer paths in this fault model are logical paths and thus will account for any failure in the actual physical paths.

Since there is a variety of designs for the ALU and other functional units such as increment or shift logic, no specific fault model is used for the data manipulation function. It is assumed that complete test sets can be derived for the functional units for a given fault model.

By carefully analyzing the logical behavior of the microprocessor according to the fault models presented above, Thatte and Abraham formulated a set of algorithms to generate the necessary test patterns. These algorithms step the microprocessor through a precisely defined set of instructions and addresses. Each algorithm was designed for detecting a particular class of faults, and theorems were proved which showed exactly the kind of faults detected by each algorithm. These algorithms employ the excitation and sensitization concepts previously described.

To gain insight into the problems involved in using the algorithms, Thatte investigated the testing of an eight-bit microprocessor from Hewlett-Packard.[12] He generated the test patterns for the microprocessor by hand, using the algorithms. He found that 96 percent of the single stuck-at faults that could affect the microprocessor were detected by the test sequence he generated. This figure indicates the validity of the technique.

The Abadir-Reghbati technique

Here we will briefly describe a test generation technique we developed for LSI circuits.[15,16] We assumed that the tests would be generated in a user environment in which the gate-and flip-flop-level details of the chip were not known.

We developed a module-level model for LSI circuits. This model bypasses the gate and flip-flop levels and directly describes blocks of logic (modules) according to their functions. Any LSI circuit can be modeled as a network of interconnected modules such as counters, registers, ALUs, ROMs, RAMs, multiplexers, and decoders.

Each module in an LSI circuit was modeled as a black box having a number of functions defined by a set of *binary decision diagrams* (see box, next page).[19] This type of diagram, a functional description tool introduced by Akers in 1978, is a concise means for completely defining the logical operation of one or more digital functions in an implementation-free form. The information usually found in an IC catalog is sufficient to derive the set of binary decision diagrams describing the functions performed by the different modules in a device. These diagrams — like truth tables and state tables — are amenable to extensive logical analysis. However, unlike truth tables and state tables, they do not have the unpleasant property of growing exponentially with the number of variables involved. Moreover, the diagrams can be stored and processed easily in a digital computer. An important feature of these diagrams is that they state exactly how the module will behave in every one of its operation modes. Such information can be extracted from the module's diagrams in the form of a set of *experiments*.[15,20] Each of these experiments describes the behavior of the module in one of its modes of operation. The structure of these experiments makes them suitable for use in automatic test generation.

We also developed a functional-level fault model describing faulty behavior in the different modules of an LSI chip. This model is quite independent of the details of implementation and covers functional faults that alter the behavior of a module during one of its modes of operation. It also covers stuck-at faults affecting any input or output pin or any interconnection line in the chip.

Using the above-mentioned models, we proposed a functional test generation procedure based on path sensitization and the D-algorithm.[15] The procedure takes the module-level model of the LSI chip and the functional description of its modules as parameters and generates tests to detect faults in the fault model. The *fault collapsing technique*[1] was used to reduce the length of the test sequence. As in the D-algorithm, the procedure employs three basic operations, namely implication, D-propagation, and line justification. However, these operations are performed on functional modules.

We also presented algorithmic solutions to the problems of performing these operations on functional modules.[16] For each of the three operations, we gave an algorithm which takes the module's set of experiments and current state (i.e., the values assigned to the module inputs, outputs, and internal memory elements) as parameters and generates all the possible states of the module after performing the required operation.

We have also reported our efforts to develop test sequences based on our test generation procedure for typical LSI circuits.[17] More specifically, we considered a one-bit microprocessor slice C that has all the basic features of the four-bit Am2901 microprocessor slice.[10] The circuit C was modeled as a network of eight functional modules: an ALU, a latch register, an addressable register, and five multiplexers. The functions of the individual modules were described in terms of binary decision diagrams or equivalent sets of experiments. Tests capable of detecting various faults covered by the fault model were then generated for the circuit C. We showed that if the fault collapsing technique is used, a significant reduction in the length of the final test sequence results.

The test generation effort was quite straightforward, indicating that the technique can be automated without much difficulty. Our study also shows that for a simplified version of the circuit C the length of the test sequence generated by our technique is very close to the length of the test sequence manually generated by Sridhar and Hayes[9] for the same circuit. We also described techniques for modeling some of the features of the Am2909 four-bit microprogram sequencer[10] that are not covered by the circuit C.

The results of our case study were quite promising and showed that our technique is a viable and effective one for generating tests for LSI circuits.

Simulation-aided Test Generation

Logic simulation techniques have been used widely in the evaluation and verification of new digital circuits. However, an important application of logic simulation is to interpret the behavior of a circuit under a certain fault or faults. This is known as *fault simulation.* To clarify how this technique can be used to generate tests for LSI systems, we will first describe its use with SSI/MSI-type circuits.

To generate a fault simulator for an SSI/MSI circuit, the following information is needed:[1]

- the gate-level description of the circuit, written in a special language;
- the initial conditions of the memory elements; and
- a list of the faults to be simulated, including classical types of faults such as stuck-at faults and adjacent pin shorts.

The above is fed to a simulation package which generates the fault simulator of the circuit under test. The resulting simulator can simulate the behavior of the circuit under normal conditions as well as when any faults exist.

Now, by applying various input patterns (either generated by hand, by an algorithm, or at random), the simulator checks to see if the output response of the correct circuit differs from one of the responses of the faulty circuits. If it does, then this input pattern detects the fault which created the wrong output response; otherwise the input pattern is useless. If an input pattern is found to detect a certain fault, this fault is deleted from the fault list and the process continues until either the input patterns or the faults are finished. At the end, the faults remaining in the fault list are those which cannot be detected by the input patterns. This directly measures the degree of fault coverage of the input patterns used.

Two examples of this type of logic simulator are LAMP — the Logic Analyzer for Maintenance Planning developed at Bell Laboratories,[21] and the Testaid III fault simulator developed at the Hewlett-Packard Company.[12] Both work primarily at the gate level and simulate stuck-at faults only. One of the main applications of such fault simulators is to determine the degree of fault coverage provided by a test sequence generated by any other test generation technique.

There are two key requirements that affect the success of any fault simulator:

- the existence of a software model for each primitive element of the circuit, and
- the existence of a good fault model for the UUT which can be used to generate a fault list covering most of the actual physical faults.

These two requirements have been met for SSI/MSI circuits, but they pose serious problems for LSI circuits. If it can be done at all, modeling LSI circuits at the gate level requires great effort. One part of the problem is the lack of detailed information about the internal structure of most LSI chips. The other is the time and memory required to simulate an LSI circuit containing thousands of gates. Another severe problem facing almost all LSI test generation techniques is the lack of good fault models at a level higher than the gate level.

The Abadir-Reghbati description model proposed in the previous section permits the test designer to bypass the gate-level description and, using binary decision diagrams, to define blocks of logic according to their functions. Thus, the simulation of complex LSI circuits can take place at a higher level, and this eliminates the large time and memory requirements. Furthermore, the Abadir-Reghbati fault model is quite efficient and is suitable for simulation purposes. In fact, the implication operation[16] employed by the test generation procedure represents the main building block of any fault simulator. It must be noted that fault simulation techniques are very useful in optimizing the length of the test sequence generated by any test generation technique.

BINARY DECISION DIAGRAMS

Binary decision diagrams are a means of defining the logical operation of digital functions.[19] They tell the user how to determine the output value of a digital function by examining the values of the inputs. Each node in these diagrams is associated with a binary variable, and there are two branches coming out from each node. The right branch is the "1" branch, while the left branch is the "0" branch. Depending on the value of the node variable, one of the two branches will be selected when the diagram is processed.

To see how binary decision diagrams can be used, consider the half-adder shown in Figure A.2.4(a). Assume we are interested in defining a procedure to determine the value of C, given the binary values of X and Y. We can do this by looking at the value of X. If X = 0,then C = 0, and we are finished. If X = 1, we look at Y. If Y = 0, then C = 0, else C = 1, and in either case we are finished. Figure A.2.4(b) shows a simple diagram of this procedure. By entering the diagram at the node indicated by the arrow labeled with C and then proceeding through the diagram following the appropriate branches until a 0 or 1 value is reached, we can determine the value C. Figure A.2.4(c) shows the diagram representing the function S of the half-adder.

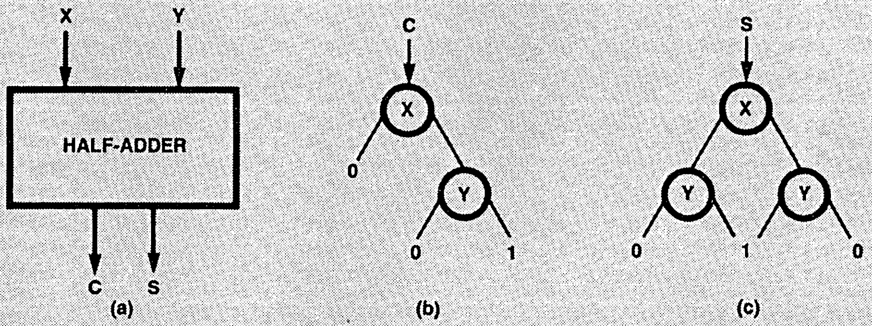


Figure A.2.4 (a) A Half-adder; (b) Binary Decision Diagram for $C = X \bullet Y$; (c) Binary Decision Diagram for $S = X \oplus Y$

To simplify the diagrams, any diagram node which has two branches as exit branches can be replaced by the variable itself or its complement. These variables are called exit variables. Figure A.2.5 shows how this convention is used to simplify the diagrams describing the half-adder.

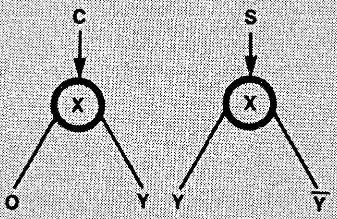


Figure A.2.5 Simplified Binary Decision Diagrams for the Half-adder

In the previous discussion, we have considered only simple diagrams in which the variables within the nodes are primary input variables. However, we can expand the scope of these diagrams by using auxiliary variables as the node variables. These auxiliary variables are defined by their diagrams. Thus, when user encounters such a node variable, say *g*, while tracing a path, he must first process the diagram defining *g* to determine the value of *g*, and then return to the original node and take the appropriate branch. This process is similar to the use of subroutines in high-level programming languages.

For example, consider the full-adder defined by:

$$C_{j+1} = E_jC_j + \overline{E}_jA_j$$

$$S_j = E_j + C_j,$$

where $E_j = A_j + B_j$. Figure A.2.6 shows the diagrams for these three equations. If the user wants to know the value of C_{j+1} when the values of the three primary inputs A_j, B_j, and C are all 1's, he enters the C_{j+1} diagram, where he encounters

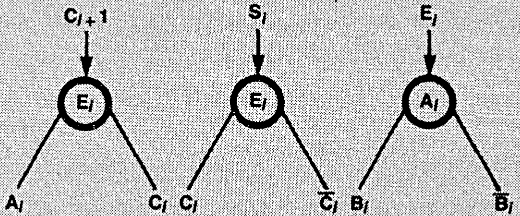


Figure A.2.6 Binary Decision Diagrams for a Full-adder

the node variable E_j. by traversing the E_j diagram, he obtains a value of 0. Returning to the original C_{j+1} diagram with $E_j = 0$ will result in taking the 0 branch and exiting with $C_{j+1} = A_j = 1$.

Since node variables can refer to other auxiliary functions, we can simply describe complex modules by breaking their functions into small subfunctions. Thus, the system diagram will consist of small diagrams connected in a hierarchical structure. Each of these diagrams describes either a module output or an auxilliary variable.

Akers[19] described two procedures to generate the binary decision diagram of a combinational function *f*. The first one uses the truth table description of *f*, while the other uses the boolean expression of *f*. A similar procedure can be derived to generate the binary decision diagram for any sequential function defined by a state table.

Binary decision diagrams can be easily stored and processed by a computer through the use of binary tree structures. Each node can be completely defined by an ordered triple: the node variable and two pointers to the two nodes to which its 0 and 1 branches are directed. Binary decision diagrams can be used in functional testing.[20]

References

19. S.B. Akers, "Binary Decision Diagram," *IEEE Trans Computers,* Vol. C-27, No. 6, June 1978, pp. 509-516.

20. S.B. Akers, "Functional Testing with Binary Decision Diagram," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978, pp. 82-92.

Random Test Generation

This method can be considered the simplest method for testing a device. A random number generator is used to simultaneously apply random input patterns both to the UUT and to a copy of it known to be fault-free. (This copy is called the *golden unit.*) The results obtained from the two units are compared, and if they do not match, a fault in the UUT is detected. This response evaluation technique is known as comparison testing; we will discuss it later. It is important to note that every time the UUT is tested, a new random test sequence is used.

The important question is how effective the random test is, or, in other words, what fault coverage a random test of given length provides. This question can be answered by employing a fault simulator to simulate the effect of random test patterns of various lengths. The results of such experiments on SSI and MSI circuits show that random test generation is most suitable for circuits without deep sequential logic.[1,22,23] However, by combining random patterns with manually generated ones, test designers can obtain very good results.

The increased sequentiality of LSI circuits reduces the applicability of random testing. Again, combining manually generated test patterns with random ones improves the degree of fault coverage. However, two factors restrict the use of the random test generation technique:

- The dependency on the golden unit, which is assumed to be fault-free, weakens the level of confidence in the results.
- There is no accurate measure of how effective the test is, since all the data gathered about random tests are statistical data. Thus, the amount of fault coverage provided by a particular random test process is unpredictable.

A.3 RESPONSE EVALUATION TECHNIQUES

Different methods have been used to evaluate UUT responses to test patterns. We restrict our discussion to the case where the final goal is only to detect faults or, equivalently, to detect any wrong output response. There are two ways of achieving this goal — using a good response generator or using a compact testing technique.

Good Response Generation

This technique implements an ideal strategy: comparing UUT responses with good response patterns to detect any faulty response. Clearly, the key problems are how to obtain a good response and at what stage in the testing process that response will be generated. In current test systems, two approaches to solving these problems are taken — *stored response testing and comparison testing.*

Stored Response Testing

In stored response testing, a one-shot operation generates the good response patterns at the end of the test generation stage. These patterns are stored in an auxiliary memory (usually a ROM). A flow diagram of the stored response testing technique is shown in Figure A.3.1.

Different methods can be used to obtain good responses of a circuit to a particular test sequence. One way is to do it manually by analyzing the UUT and the test patterns. This method is the most suitable if the test patterns were generated manually in the first place.

The method most widely used to obtain good responses from the UUT is to apply the test patterns either to a known good copy of the UUT — the golden unit — or to a software-simulated version of the UUT. Of course, if fault simulation techniques were used to generate the test patterns, the UUT's good responses can be obtained very easily as a partial product from the simulator.

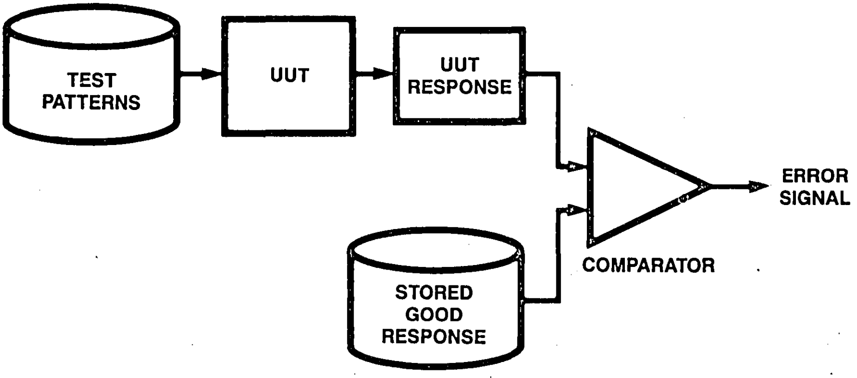


Figure A.3.1 Stored Response Testing

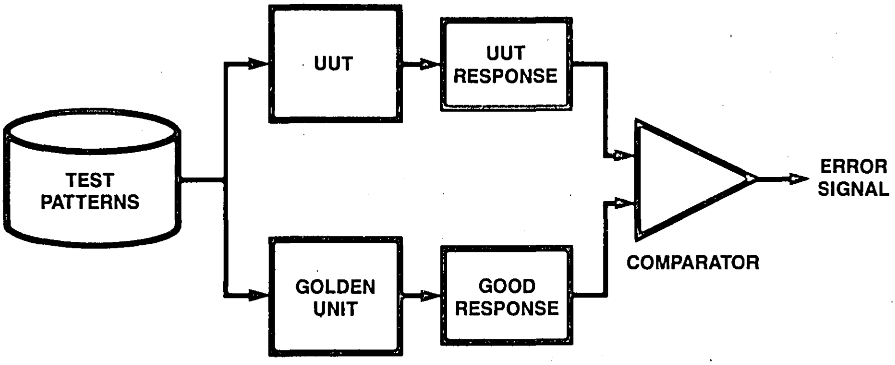


Figure A.3.2 Comparison Testing

The use of a known good device depends on the availability of such a device. Hence, different techniques must be used for the user who wants to test his LSI system and for the designer who wants to test his prototype design. However, golden units are usually available once the device goes into production. Moreover, confidence in the correctness of the responses can be increased by using three or five good devices together to generate the good responses.

The major advantage of the stored response technique is that the good responses are generated only once for each test sequence, thus reducing the cost of the response evaluation step. However, the stored response technique suffers from various disadvantages:

- Any change in the test sequence requires the whole process to be repeated.

- A very large memory is usually needed to store all the good responses to a reasonable test sequence, because both the length and the width of the responses are relatively large. As a result, the cost of testing equipment increases.
- The speed with which the test patterns can be applied to the UUT is limited by the access time of the memory used to store the good responses.

Comparison Testing

Another way to evaluate the responses of the UUT during the testing process is to apply the test patterns simultaneously to both the UUT and a golden unit and to compare their responses to detect any faulty response. The flow diagram of the comparison testing technique is shown in Figure A.3.2. The use of comparison testing makes possible the testing of the UUT at different speeds under different electrical parameters, given that these parameters are within the operating limits of the golden unit, which is assumed to be ideal.

Note that in comparison testing the golden unit is used to generate the good responses every time the UUT is tested. In stored response testing, on the other hand, the golden unit is used to generate the good responses only once.

The disadvantages of depending on a golden unit are more serious here, however, since every explicit testing process requires one golden unit. This means that every tester must contain a golden copy of each LSI circuit tested by that tester.

One of the major advantages of comparison testing is that nothing has to be changed in the response evaluation stage if the test sequence is altered. This makes comparison testing highly desirable if test patterns are generated randomly.

Compact Testing

The major drawback of good response generation techniques in general, and stored response testing in particular, is the huge amount of response data that must be analyzed and stored. Compact testing methods attempt to solve this by compressing the response data R into a more compact form *f*(R) from which most of the fault information in R can be derived. Thus, because only the compact form of the good responses has to be stored, the need for large memory or expensive golden units is eliminated. An important property of the compression function *f* is that it can be implemented with simple circuitry. Thus, compact testing does not require much test equipment and is especially suited for field maintenance work. A general diagram of the compact testing technique is shown in Figure A.3.3.

Several choices for the function *f* exist, such as "the number of 1's in the sequence," "the number of 0 to 1 and 1 to 0 transitions in the sequence" (*transition counting*),[24] or "the signature of the sequence" (*signature analysis*).[25] For each compression function *f*, there is a slight probability that a response R1 different from the fault-free response R0 will be compressed to a form equal to *f*(R0), i.e., *f*(R1) = *f*(R0).

Thus, the fault causing the UUT to produce R1 instead of R0 will not be detected, even though it is covered by the test patterns.

The two compression functions that are the most widely accepted commercially are transition counting and signature analysis.

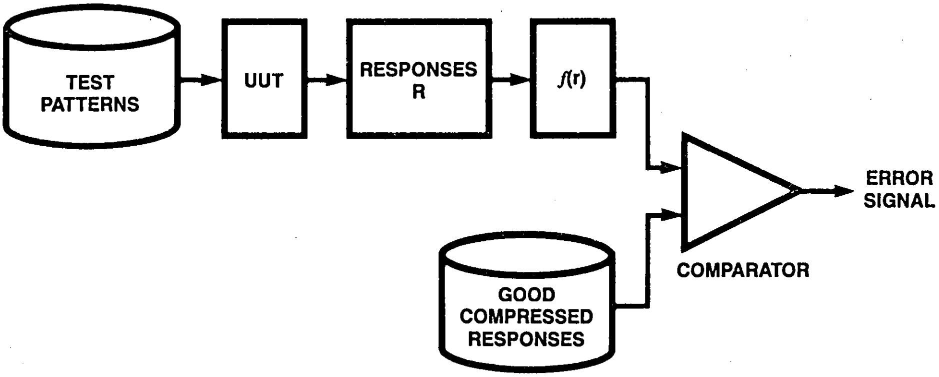


Figure A.3.3 Compact Testing

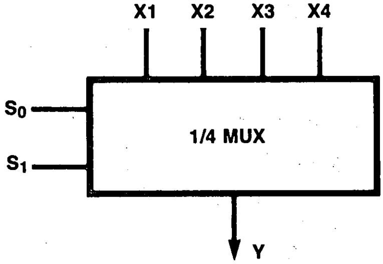
Transition Counting

In transition counting, the number of logical transitions (0 to 1 and vice versa) is computed at each output pin by simply running each output of the UUT into a special counter. Thus, the number of counters needed is equal to the number of output pins observed. For every m-bit output data stream (at one pin), an n-bit counter is required, where $n = [\log_2 m]$. As in stored response testing, the transition counts of the good responses are obtained by applying the test sequence to a golden copy of the UUT and counting the number of transitions at each output pin. This latter information is used as a reference in any explicit testing process.

In the testing of an LSI circuit by means of transition counting, the input patterns can be applied to the UUT at a very high rate, since the response evaluation circuitry is very fast. Also, the size of the memory needed to store the transition counts of the good responses can be very small. For example, a transition counting test using 16 million patterns at a rate of one MHz will take 16 seconds, and the compressed stored response will occupy only K 24-bit words, where K is the number of output pins. This can be contrasted with the 16 million K-bit words of storage space needed if regular stored response testing is used.

The test patterns used in a transition counting test system must be designed such that their output responses maximize the fault coverage of the test.[24] The example below shows how this can be done.

Consider the one-out-of-four multiplexer shown in Figure A.3.4. To check for multiple stuck-at faults in the multiplexer input lines, eight test patterns are required, as shown in Table A.3.1. The sequence of applying these eight patterns to the multiplexer is not important if we want to evaluate the output responses one by one. However, this sequence will greatly affect the degree of fault coverage if transition counting is used. To illustrate this fact, consider the eight single stuck-at faults in the four input lines X1,X2,X3, and X4 (i.e., X1 stuck-at 0, X1 stuck-at 1, X2 stuck-at 0, and so on). Each of these faults will be detected by only one pattern among the eight test patterns. For example, the fault "X1 stuck-at 0" will be detected by applying the first test pattern in Table A.3.1, but the other seven test patterns will not detect this fault. Now, suppose we want to use transition counting to evaluate the output responses of the multiplexer. Applying the eight test patterns in the sequence shown in Table A.3.1 (from top to bottom) will produce the output response 10101010 (from left to right), with a transition count of seven. Any possible combination of the eight faults described above will change the transition count to a number different from seven, and the fault will be detected. (Note that no more than four of the eight faults can occur at any one time.) Thus, the test sequence shown in Table A.3.1 will detect all single and multiple stuck-at faults in the four input lines of the multiplexer.



S_0	S_1	Y
0	0	X1
0	1	X2
1	0	X3
1	1	X4

Figure A.3.4 One-Out-of-Four Multiplexer

Now, if we change the sequence of the test patterns to the one shown in Table A.3.2, the fault coverage of the test will decrease considerably. The output responses of the sequence of Table A.3.2 will be 11001100, with a transition count of three. As a result, six of the eight single stuck-at faults will not be detected, because the transition count of the six faulty responses will remain three. For example, the fault "X1 stuck-at 1" will change the output response to 11101100, which has a transition count of three. Hence, this fault will not be detected. Moreover, most of the multiple combinations of the eight faults will not change the transition count of the output, and hence they will not be detected either.

It is clear from the above example that the order of applying the test patterns to the UUT greatly affects the fault coverage of the test. When testing combinational circuits, the test designer is completely free to choose the order of test patterns. However, he

S_0	S_1	X1	X2	X3	X4	Y
0	0	1	0	0	0	1
0	0	0	1	1	1	0
0	1	0	1	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	0	0	1	1
1	1	1	1	1	0	0

Table A.3.1 The eight test patterns used for testing the multiplexer of Figure A.3.4

S_0	S_1	X1	X2	X3	X4	Y
0	0	1	0	0	0	1
0	1	0	1	0	0	1
0	0	0	1	1	1	0
0	1	1	0	1	1	0
1	0	0	0	1	0	1
1	1	0	0	0	1	1
1	0	1	1	0	1	0
1	1	1	1	1	0	0

Table A.3.2 A different sequence of the eight multiplexer test patterns

cannot do the same with test patterns for sequential circuits. More seriously, because he is dealing with LSI circuits that probably have multiple output lines, he will find that a particular test sequence may give good results at some outputs and bad results at others. One way to solve these contradictions is to use simulation techniques to find the optimal test sequence. However, because of the limitations discussed here, transition counting cannot be recognized as a powerful compact LSI testing method.

Signature Analysis

In 1977 Hewlett-Packard Corporation introduced a new compact testing technique called signature analysis, intended for testing LSI systems.[25-28] In this method, each output response is passed through a 16-bit linear feedback shift register whose contents *f*(R), after all the test patterns have been applied, are called the test *signature*. Figure A.3.5 shows an example of a linear feedback shift register used in signature analysis.

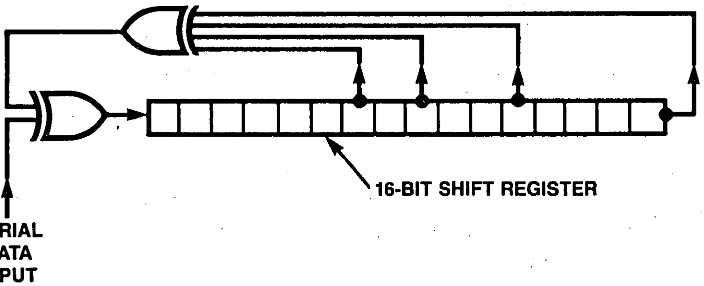


Figure A.3.5 The 16-bit Linear Feedback Shift Register Used in Signature Analysis

The signature provided by linear feedback shift registers can be regarded as a unique fingerprint — hence, test designers have extremely high confidence in these shift registers as tools for catching errors. To better understand this confidence, let us examine the 16-bit linear feedback shift register shown in Figure A.3.5. Let us assume a data stream of length n is fed to the serial data input line (representing the output response to be evaluated). There are 2^n possible combinations of data streams, and each one will be compressed to one of the 2^{16} possible signatures. Linear feedback shift registers have the property of equally distributing the different combinations of data streams over the different signatures.[27] This property is illustrated by the following numerical examples.

- Assume $n = 16$. Then each data stream will be mapped to a distinctive signature (one-to-one mapping).
- Assume $n = 17$. Then exactly two data streams will be mapped to the same signature. Thus, for a particular data stream (the UUT good output response), there is only one other data stream (a faulty output response) that will have the same signature; i.e., only one faulty response out of $2^{17} — 1$ possible faults will not be detected.
- Assume $n = 18$. Then four different data streams will be mapped to the same signature. Hence, only three faults out of $2^{18} — 1$ possible faults will not be detected.

We can generalize the results obtained above. For any response data stream of length $n > 16$, the probability of missing a faulty response when using a 16-bit signature analyzer is[27]

$$\frac{2^{n-16}-1}{2^n-1} \cong 2^{-16}, \text{ for } n >> 16.$$

Hence, the possibility of missing an error in the bit stream is very small (on the order of 0.002 percent). Note also that a great percentage of the faults will affect more than one output pin — hence the probability of not detecting these kind of faults is even lower.

Signature analysis provides a much higher level of confidence for detecting faulty output responses than that provided by transition counting. But, like transition counting, it requires only very simple hardware circuitry and a small amount of memory for storing the good signatures. As a result, the signatures of the output responses can be calculated even when the UUT is tested at its maximum speed. Unlike transition counting, the degree of fault coverage provided by signature analysis is not sensitive to the order of the test patterns. Thus, it is clear that signature analysis is the most attractive solution to the response evaluation problem.

The rapid growth of the complexity and performance of digital circuits presents a testing problem of increasing severity. Although many testing methods have worked well for SSI and MSI circuits, most of them are rapidly becoming obsolete. New techniques are required to cope with the vastly more complicated LSI circuits.

In general, testing techniques fall into the concurrent and explicit categories. In this article, we gave special attention to explicit testing techniques, especially those approaching the problem at the functional level. The explicit testing process can be partitioned into three steps: generating the test, applying the test to the UUT, and evaluating the UUT's responses. The various testing techniques are distinguished by the methods they use to perform these three steps. Each of these techniques has certain strengths and weaknesses.

We have tried to emphasize the range of testing techniques available, and to highlight some of the milestones in the evolution of LSI testing. The details of an individual test method can be found in the sources we have cited.

References

1. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Washington, DC, 1976.

2. J.P. Hayes and E.J. McCluskey, "Testing Considerations in Microprocessor-Based Design," *Computer*, Vol. 13, No. 3, Mar. 1980, pp. 17-26.

3. J. Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications*, American Elsevier, New York, 1978.

4. D.B. Armstrong, "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinatorial Nets," *IEEE Trans. Electronic Computers,* Vol. EC-15, No. 2, Feb. 1966, pp. 63-73.

5. J.P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronica Computers,* Vol. EC-16, No. 5, Oct. 1967, pp. 567-580.

6. S.B. Akers, "Test Generation Techniques," *Computer,* Vol. 13, No. 3, Mar. 1980, pp. 9-15.

7. E.I. Muehldorf and A.D. Savkar, "LSI Logic Testing — An Overview," *IEEE Trans. Computers*, Vol. C-30, No. 1, Jan. 1981, pp. 1-17.

8. O.H. Ibarra and S.K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Trans. Computers*, Vol. C-24, No. 3, Mar. 1975, pp 242-249.

9. T. Sridhar and J.P. Hayes, "Testing Bit-Sliced Microprocessors," *Proc. 9th Int'l Symp. Fault-Tolerant Computing*, 1979, pp. 211-218.

10. *The Am2900 Family Data Book*, Advanced Micro Devices, Inc., 1979.

11. Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1970.

12. S.M. Thatte, "Test Generation for Microprocessors," PhD thesis, University of Illinois, Urbana, 1979.

13. S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors," *IEEE Trans. Computers*, Vol. C-29, No. 6, June 1980, pp. 429-441.

14. M.A. Breuer and A.D. Friedman, "Functional Level Primitives in Test Generation," *IEEE Trans. Computers*, Vol. C-29, No. 3, Mar. 1980, pp. 223-235.

15. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: A New Approach," Tech. Report 81-7, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.

16. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: Basic Operations," Tech. Report 81-8, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.

17. M.S. Abadir and H.K. Reghbati, "Test Generation for LSI: A Case Study," Tech. Report 81-9, Dept. of Computational Science, University of Saskatchewan, Saskatoon, 1981.

18. M.S. Abadir and H.K. Reghbati, "Functional Testing of Semiconductor Random Access Memories," Tech. Report 81-6, Dept. of Computational Science, Univeristy of Saskatchewan, Saskatoon, 1981.

19. S.B. Akers, "Binary Decision Diagram," *IEEE Trans Computers,* Vol. C-27, No. 6, June 1978, pp. 509-516.

20. S.B. Akers, "Functional Testing with Binary Decision Diagram," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978, pp. 82-92.

21. B.A. Zimmer, "Test Techniques for Circuit Boards Containing Large Memories and Microprocessors," *Proc. 1976 Semiconductor Test Symp.*, pp. 16-21.

22. P. Agrawal and V.D. Agrawal, "On Improving the Efficiency of Monte Carlo Test Generation," *Proc. 5th Int'l Symp. Fault-Tolerant Computing*, June 1975, pp. 205-209.

23. D. Bastin, E. Girard, J.C. Rault, and R. Tulloue, "Probabilistic Test Generation Methods," *Proc. 3rd Int'l Symp. Fault-Tolerant Computing*, June 1973, p. 171.

24. J.P. Hayes, "Transition Count Testing of Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. C-25, No. 6, June 1976, pp. 613-620.

25. "Signature Analysis," *Hewlett Packard J.*, Vol.28, No. 9, May 1977.

26. R. David, "Feedback Shift Register Testing," *Proc. 8th Int'l Symp. Fault-Tolerant Computing*, June 1978.

27. H.J. Nadig, "Testing a Microprocessor Product Using Signature Analysis," *Proc. 1978 Semiconductor Test Symp.*, pp. 159-169.

28. J.B. Peatman, *Digital Hardware Design*, McGraw-Hill, New York, 1980.

29. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman, San Francisco, 1978.

30. E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, Washington, DC, 1978.

National Semiconductor Corporation
P.O. Box 58090
2900 Semiconductor Drive
Santa Clara, CA 95052-8090
Tel: (408) 721-5000
TWX: (910) 339-9240

Electronica NSC de Mexico SA
Juventino Rosas No. 118-2
Col Guadalupe Inn
Mexico, 01020 D.F. Mexico
Tel: (905) 524-9402

National Semicondutores
Do Brasil Ltda.
Av. Brig. Faria Lima, 830
8 Andar
01452 Sao Paulo, SP. Brasil
Tel: (55/11) 212-5066
Telex: 391-1131931 NSBR BR

National Semiconductor GmbH
Westendstrasse 193-195
D-8000 Munchen 21
West Germany
Tel: (089) 5 70 95 01
Telex: 522772

National Semiconductor (UK) Ltd.
301 Harpur Centre
Horne Lane
Bedford MK40 1TR
United Kingdom
Tel: 0234-47147
Telex: 826 209

National Semiconductor Benelux
Ave Charles Quint 545
B-1080 Bruxelles
Belgium
Tel: (02) 4661807
Telex: 61007

National Semiconductor (UK) Ltd.
1, Bianco Lunos Alle
DK-1868 Copenhagen V
Denmark
Tel: (01) 213211
Telex: 15179

National Semiconductor
Expansion 10000
28, Rue de la Redoute
F-92 260 Fontenay-aux-Roses
France
Tel: (01) 660-8140
Telex: 250956

National Semiconductor S.p.A.
Via Solferino 19
20121 Milano
Italy
Tel: (02) 345-2046/7/8/9
Telex: 332835

National Semiconductor AB
Box 2016
Stensatravagen 4/11 TR
S-12702 Skarholmen
Sweden
Tel: (08) 970190
Telex: 10731

National Semiconductor
Calle Nunez Morgado 9
(Esc. Dcha. 1-A)
E-Madrid 16
Spain
Tel: (01) 733-2954/733-2958
Telex: 46133

National Semiconductor Switzerland
Alte Winterthurerstrasse 53
Postfach 567
CH-8304 Wallisellen-Zurich
Tel: (01) 830-2727
Telex: 59000

National Semiconductor
Pasilanraitio 6C
SF-00240 Helsinki 24
Finland
Tel: (90) 14 03 44
Telex: 124854

NS Japan Ltd.
4-403 Ikebukuro, Toshima-ku
Tokyo 171, Japan
Tel: (03) 988-2131
Fax: 011-81-3-988-1700

National Semiconductor
Hong Kong Ltd.
Southeast Asia Marketing
Austin Tower, 4th Floor
22-26 Austin Avenue
Tsimshatsui, Kowloon, H.K.
Tel: 3-7231290, 3-7243645
Cable: NSSEAMKTG
Telex: 52996 NSSEA HX

National Semiconductor (Australia)
PTY, Ltd.
21/3 High Street
Bayswater, Victoria 3153
Tel: (03) 729-6333
Telex: AA32096

National Semiconductor (PTE), Ltd.
10th Floor
Pub Building, Devonshire Wing
Somerset Road
Singapore 0923
Tel: 652700047
Telex: NAT SEMI RS 21402

National Semiconductor (Far East)
Ltd.
Taiwan Branch
P.O. Box 68-332 Taipei
7th Floor, Nan Shan Life Bldg.,
302 Min Chuan East Road,
Taipei, Taiwan R.O.C.
Tel: (02) 501-7227
Telex: 22837 NSTW
Cable: NSTW TAIPEI

National Semiconductor (Far East)
Ltd.
Korea Office
Third Floor, Hankyung Bldg.
4-25 Hannam-Dong
Yongsam-Ku, Seoul 140, Korea
Tel: 797-8001/3
Telex: K24942 NSRK

RRD20M075/Printed in U.S.A.