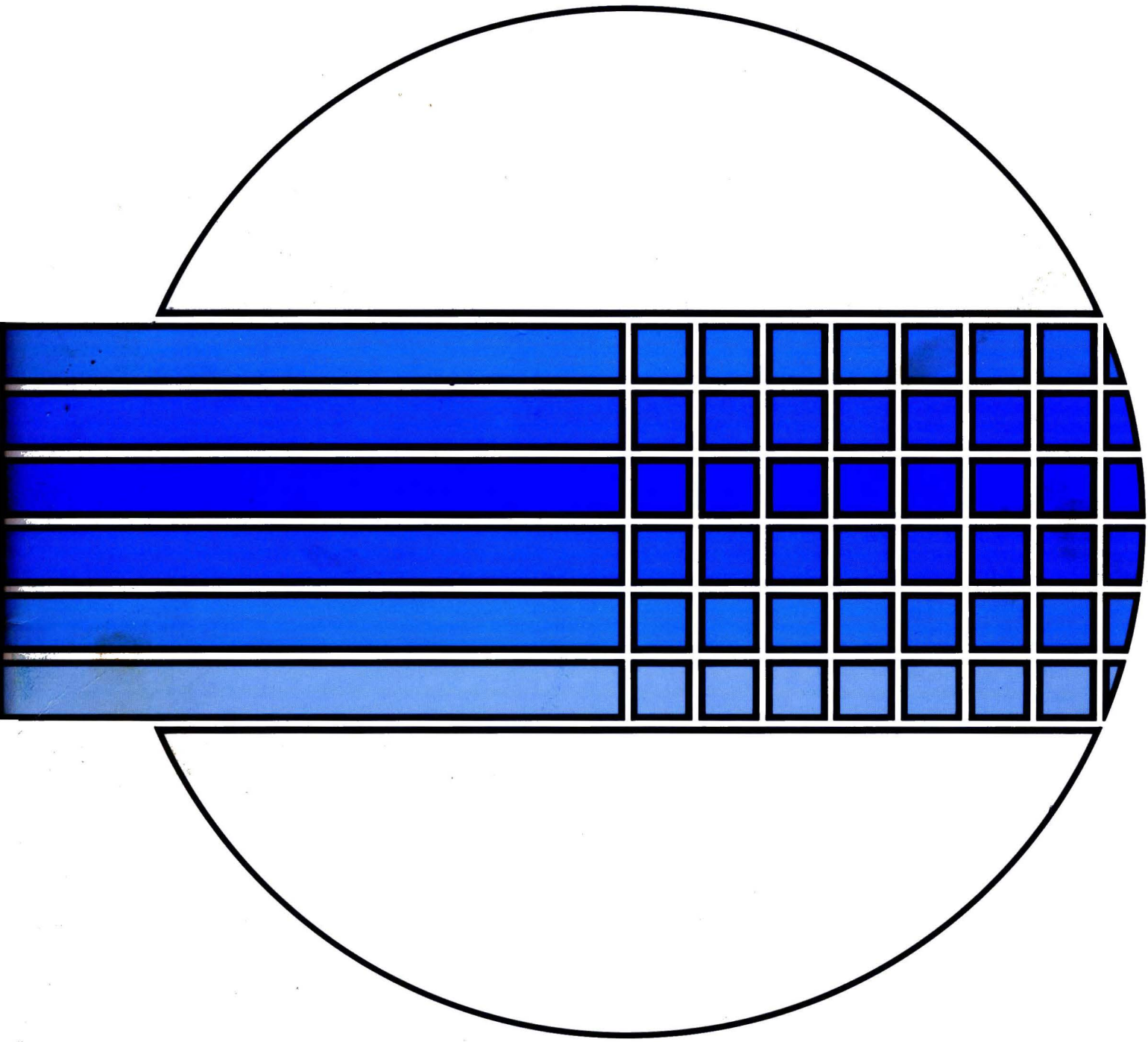
SIGNETICS FIELD PROGRAMMABLE LOGIC ARRAYS

$1.50



October 1977

Gentlemen:

Thank you for your interest in Signetics' family of field programmable logic arrays. The enclosed FPLA brochure contains device related information which should facilitate your understanding of the structure and applications of these new generation LSI devices.

As a preview of FPLA versatility, page 59 of the brochure contains the program table stored in a sample device for demonstrating that an FPLA is a general purpose logic tool, which could be called upon to emulate the function of an ALU . . . almost! A device thus programmed performs the OR, AND, Exclusive-OR, Mux and Add (with serial carry) of two 4-bit words. These functions are selected by connecting the FPLA as shown on page 60.

Both Signetics FPLAs are available now in either ceramic (I) or power plastic (N) packages, with unit price related to quantity as follows (commercial temperature range):

QUANTITY	1-24	25-99	100-999
N82S100/101N	$17.50	$14.50	$11.50
N82S100/101I	30.00	24.60	19.45

We have in-house programming capability to supply custom programmed parts within 7 days after receipt of your program table. Each custom pattern carries a one time charge of $25.00.

Field programming equipment is currently available to satisfy your programming needs as your usage develops.

PROGRAMMING ENVIRONMENT	MODEL	MANUFACTURER	PRICE
Prototyping and Qualification	LTC-F100	Signetics (408) 739-7700	$ 345.00
Pilot Production and Field Support	PR-100	Curtis Electro-Devices (415) 964-3136	1299.00
Volume Production	X	Data I/O (206) 455-3990	8000.00

In some applications, marginal design tradeoffs can be resolved in favor of FPLAs by compressing the logic truth table to a minimum number of product terms. Signetics has a computer program for executing a practical minimization algorithm, which it offers as a free service to its customers.

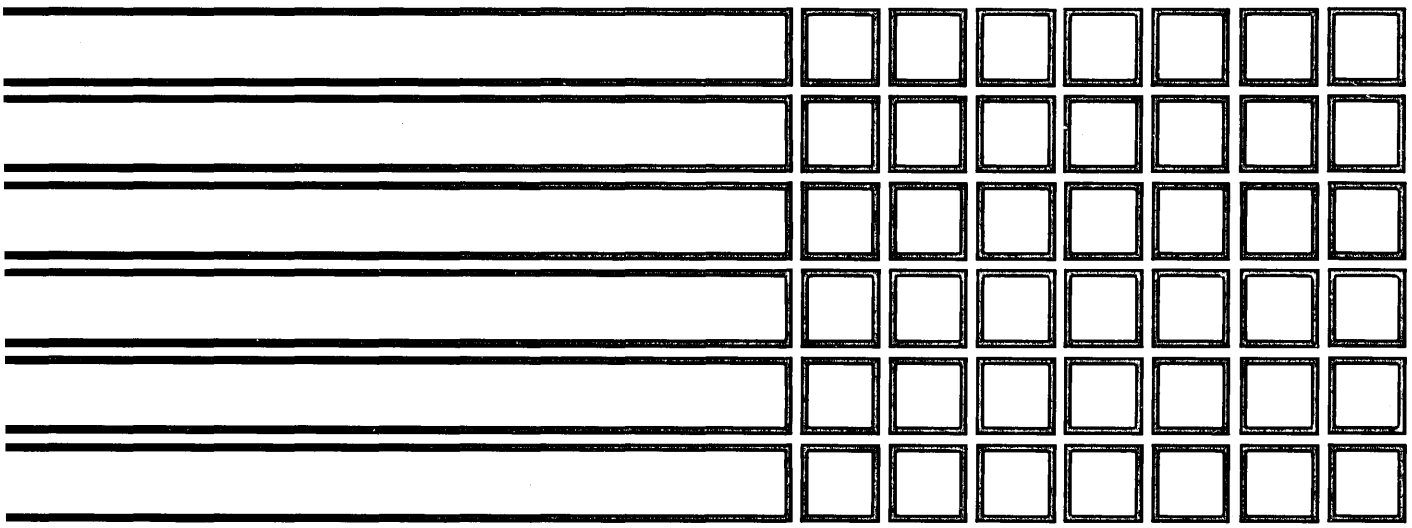
Both Signetics FPLAs have been designed for operation over the full Mil-Temp range, and military parts are available in standard S-grade and S/883B.

Please, do not hesitate to contact me if you need further information.

Napoleone Cavlan

Napoleone Cavlan
Applications Manager
Bipolar Memories

SIGNETICS FIELD PROGRAMMABLE LOGIC ARRAYS



Since the practical introduction of microprogramming in the last decade or so, microcode has progressively displaced random logic in step with the growing availability of user Programmable Read-Only Memories (PROMs). However, even with PROMs, designers soon realized that their rigid addressing structure made them unsuitable in a wide variety of applications which could greatly benefit from a structured logic approach.

Recently, microprocessors have provided a quantum jump in design flexibility in applications requiring about 30 IC packages, and beyond. When fewer packages are required, the inherent speed limitation, software requirements, and support circuitry of microprocessors place them out of range of a broad spectrum of applications.

These in general involve algorithms which require a high speed logic decision based on a large number of controlling variables. It is here that we step into the basic domain of Field Programmable Logic Arrays, encompassing applications in microprogramming, code conversion, random logic, look-up and decision tables, high speed character generators, etc. Moreover, when combined with a few storage elements (flip-flops), FPLAs can implement powerful logic machines of the Mealy/Moore form for the realization of finite state sequential controllers for traffic, process, peripheral devices, and other similar applications.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

NI-CR TECHNOLOGY MATURES

Nichrome was the first material to give rise to stable, low current fuses with excellent fusing characteristics, easily reproducible. However, as with all new developments, Nichrome technology had to undergo a learning curve, with each advance signaling the advent of more complex and higher performance devices, without a compromise in reliability. It soon became apparent that each incremental step in complexity implied a fuller understanding of the fusing phenomenon. Accordingly, fusible link technology has been intensively investigated by Signetics over the last 6 years (see Signetics' Prom Reliability page 39), giving rise to the broadest line of PROMs in the industry, and presently, the addition of a family of Field Programmable Logic Arrays (FPLAs), designed for both commercial and military temperature ranges.

WHAT IS AN FPLA?

Signetics' FPLAs are fast, user programmable, TTL logic elements with memory, which can streamline logic system design by integrating the equivalent of 528 TTL gates in 196 packages into a *single* IC package.

In terms of logic, the FPLA is a two level AND-OR, AND-NOR combinatorial logic element, consisting of a system of logic gates with programmable inputs and outputs as shown in Figure 1. These, by means of on-chip programmable connectors, enable the user to quickly implement 8 logic functions with a maximum of 48 product (AND) terms, involving up to 16 input variables.

FPLA AS A TWO LEVEL PROGRAMMABLE LOGIC ELEMENT

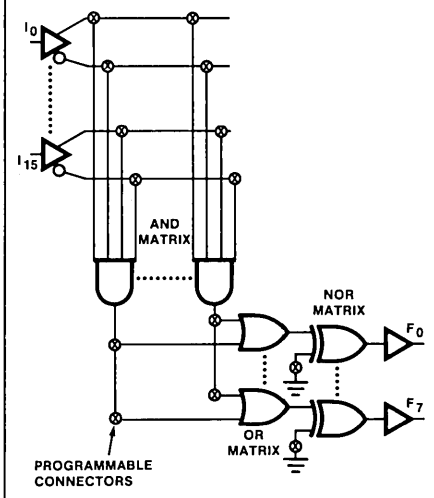


Figure 1

LOGIC STRUCTURE OF FPLA ILLUSTRATING AND, OR, AND EX-OR ARRAYS

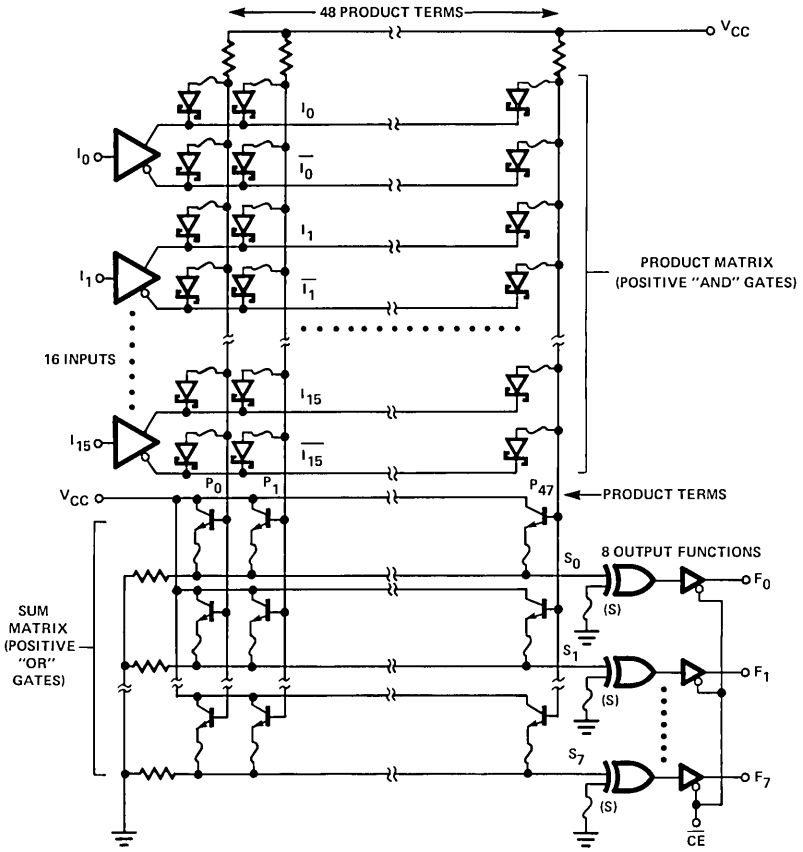


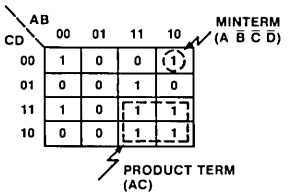
Figure 2

A more detailed organization of the FPLA is shown in Figure 2. The device consists of an upper resistor-diode AND matric containing 48 product term columns (P-terms), and a lower emitter-follower OR matrix containing 8 sum term rows (S-terms), one for each output function. Each P-term in the AND matrix is initially coupled to 16 true and complement input variables via 32 fusible Ni-Cr links for programming any desired input combination.

Each P-term is also coupled to each S-term in the OR matrix through an emitter fuse, for pulling the summing node to a high level when the P-term is activated. Each S-term in turn is coupled to its respective output via an Ex-OR gate, which has programmable transmission polarity by means of an input to ground through a fusible link. fusible link.

Selective programming of the internal links allows the user to create specific logic paths for producing any logic functions as a sum of products, defined in the typical Karnaugh Map of Figure 3.

TYPICAL KARNAUGH MAP

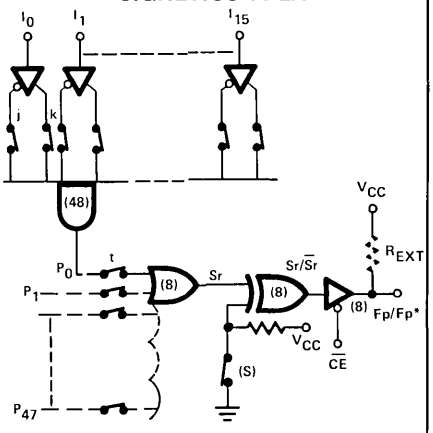


Minterms enclose a single cell. Product terms enclose 2 or more cells.

Figure 3

The transmission through the FPLA can be traced along the equivalent logic path shown in Figure 4. From this figure, it is apparent that Signetics' FPLA is basically a two level logic element. The first level produces 48 internal AND functions, P_0 through P_{47}, of up to 16 logic input variables, or their complement. The second level produces 8 OR output functions, F_0 through F_7, each involving up to 48 of the internally generated AND terms. Alternately, if desired, this second logic level can be programmed to provide 8 NOR output functions F_0^* through F_7^*. However, for

EQUIVALENT LOGIC PATH OF SIGNETICS' FPLA

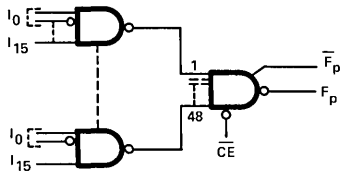


A typical set of equations describing the device logic transmission is shown on page 17.

Figure 4

each of the 8 outputs, either the function F_p (active-high), or F_p* (active-low) is available, but not both. The required output polarity is programmed by the user via link (S). Th overall logic function provided by each FPLA output is summarized in Figure 5.

TWO LEVEL LOGIC FUNCTION PROVIDED BY EACH OF THE 8 FPLA OUTPUTS



All input NANDs have 16 pairs of inputs for gating either true or complement states of logic variables. Fp or $\overline{Fp}$ are obtained with $\overline{CE}$ = "0."

Figure 5

When viewed strictly as a logic element, the FPLA can be used to implement sets of logic equations of the type:

$$F_0 = I_0 + I_1\,\overline{I}_5 + I_2\,\overline{I}_3\,I_7 \ldots$$
$$\overline{F}_1 = \overline{I}_0 + I_1\,\overline{I}_5 + I_6\,I_7\,I_8 \ldots$$
$$\overline{F}_2 = \overline{I}_2 + I_7\,I_5 + \ldots \text{ etc.}$$

or, by use of De Morgan's theorem, their equivalent as for F_0:

$$F_0 = \overline{(\overline{I}_0)\,(\overline{I}_1 + I_5)\,(\overline{I}_2 + I_3 + \overline{I}_7) \ldots}$$

This is readily shown in the logic equivalence of Figure 6.

LOGIC EQUIVALENCE OF F_0

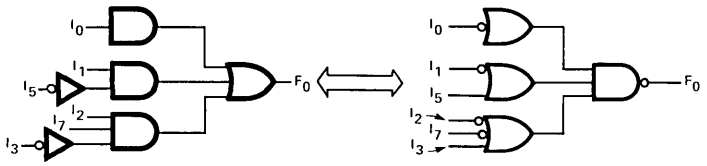


Figure 6

DISCRETE GATE LOGIC NETWORK FOR EQUATION SET F_{1-4}

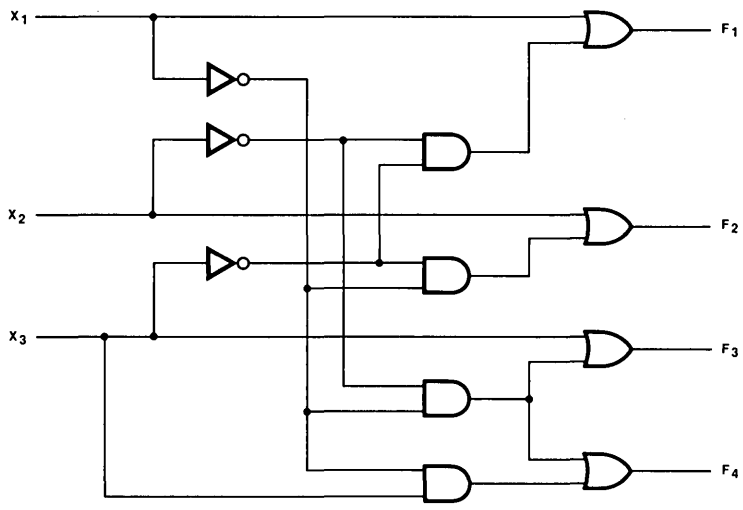


Figure 7

Generally FPLAs are effectively used in design situations involving many input variables and few active logic states; and, with a maximum access time of 50ns, the FPLA is a practical alternative to the long logic chains necessary when dealing with several input variables.

The following example is a brief, but concise, illustration of how to integrate random logic with discrete gates into a Signetics' FPLA. Given the set of logic equations F_{1-4} below:

$$F_1 = X_1 + \overline{X}_2\overline{X}_3$$
$$F_2 = X_2 + \overline{X}_1\overline{X}_3$$
$$F_3 = X_3 + \overline{X}_1\overline{X}_2$$
$$F_4 = \overline{X}_1X_3 + \overline{X}_1\overline{X}_2$$

These can be implemented with discrete gates as in the AND-OR-NOT logic network of Figure 7.

This method is practical for simple systems; but in more complex applications, it soon produces a distributed logic network with many IC packages and types, difficult to design, troubleshoot and modify.

On the other hand, the same set of equations can be easily coded in an FPLA Program Table (see page 23) and programmed in a device using inexpensive field equipment.

Typically, F_1 would require the FPLA to contain the fused link pattern shown in Figure 8, as specified in the accompanying Program Table slice. Overall, all four logic functions would use 3 inputs, 4 outputs and 7 product terms of the FPLA, leaving remaining resources spare for later modifications.

For example, if it becomes necessary to change the X_1 product term in F_1 to $\overline{X}_1$, deleting the wrong product term and adding the new one becomes a trivial task, as indicated in the modified pattern and revised Program Table of Figure 9.

These modifications can be made at any time in the field by the user, usually within the same device (as long as spare resources are available), by means of inexpensive programming equipment (as low as $350).

INTEGRATING LOGIC WITH FPLAs

PRODUCT TERM																ACTIVE LEVEL							
	INPUT VARIABLE (I_{in})																			H	H	H	H
NO	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OUTPUT FUNCTION							
																	7	6	5	4	3	2	1	0
0	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	H					•	•	•	A
1	—	—	—	—	—	—	—	—	—	—	—	—	—	L	L	—					•	•	•	A

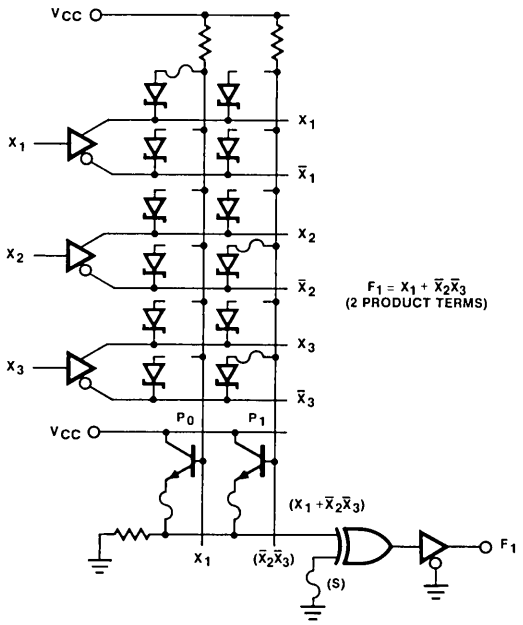
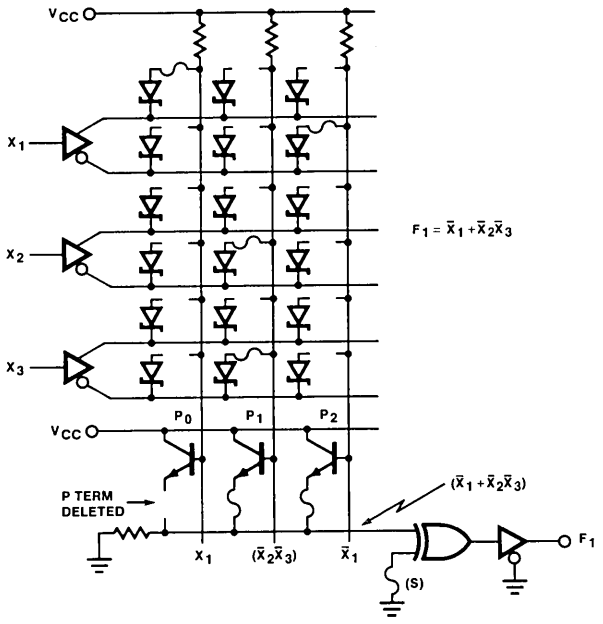


Figure 8

MODIFYING LOGIC WITH FPLAs

PRODUCT TERM																ACTIVE LEVEL							
	INPUT VARIABLE (I_{in})																			H	H	H	H
NO	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OUTPUT FUNCTION							
																	7	6	5	4	3	2	1	0
0	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	H					•	•	•	•
1	—	—	—	—	—	—	—	—	—	—	—	—	—	L	L	—					•	•	•	A
2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	L					•	•	•	A



F_1 modified by deleting term X_1 in the OR matrix and adding new term $\bar{X}_1$.

Figure 9

FPLA: LOGIC OR MEMORY?

In all practical applications, the view of Signetics' FPLAs as two level AND-OR (or AND-NOR) logic elements is perfectly adequate to manage all necessary logic manipulations. However, the use of FPLAs in certain types of applications can be better grasped by focusing on an alternate aspect of their internal structure. A useful insight is gained by comparing FPLAs to Programmable Read Only Memories (PROMs), and realizing that FPLAs can also be viewed as Conditionally Addressable Memories.

In the industry we refer to PROMs as 1K, 4K, etc. These usually imply standard organizations such as 256X4, 512X8, respectively. The larger in each pair of numbers refers to the number of **words** in a PROM, and the second represents the number of **bits** in **each** word. The product of both numbers (approximately 1K, 4K) gives the total number of storage bits contained in the PROM.

This aspect of PROMs carries over to FPLAs, such that Signetics' FPLAs can be described as 48X8, for a total **working** storage density of 384 bits. Thus, the FPLA is a relatively small PROM, but a much more useful one, due to a fundamental difference in input structure.

In a PROM (Figure 10), all internal words are reached by a **fixed** decoder internal to the device. The size of this decoder, as well as the storage matrix, doubles for each additional address input. In a 256X8 PROM, the internal decoder selects 1 of 256 words by examining 8 address inputs. For a 512X8 PROM, 1 of 512 words are selected by a decoder twice as large by examining 9 address inputs.

The presence of a fixed decoder renders PROM addressing exhaustive. This can never be avoided, and forces the utilization of PROMs in discrete chunks. This constraint is at the root of the inefficiency of PROMs in the type of application shown in Figure 11. Notice that if we define logic "1" as the active-true state of all output functions, it is not possible to compress the truth table by eliminating inactive minterms 2, 4, and 7. Moreover, with regard to minterms 0 and 1, it is necessary to allocate 2 distinct storage locations to activate output function O_3 with a single change in input variable A_0. In this case, A_0 represents a logical don't care (X) which cannot be directly programmed in a PROM. Instead, separate minterms $\overline{A}_2\overline{A}_1\overline{A}_0$ and $\overline{A}_2\overline{A}_1A_0$ must be programmed.

With an FPLA, both constraints are removed.

As shown in Figure 12, the FPLA does away with a fixed decoder in favor of a **programmable address matrix,** which offers, in place of forced exhaustive addressing, the flexibility to choose by "linear-select" any finite subset from a large number of input states. This is possible because each column of the address matrix functions essentially as a logic comparator programmed to recognize the simultaneous presence of (n) inputs, each either true, false, or both (don't care).

SIMPLIFIED PROM ORGANIZATION

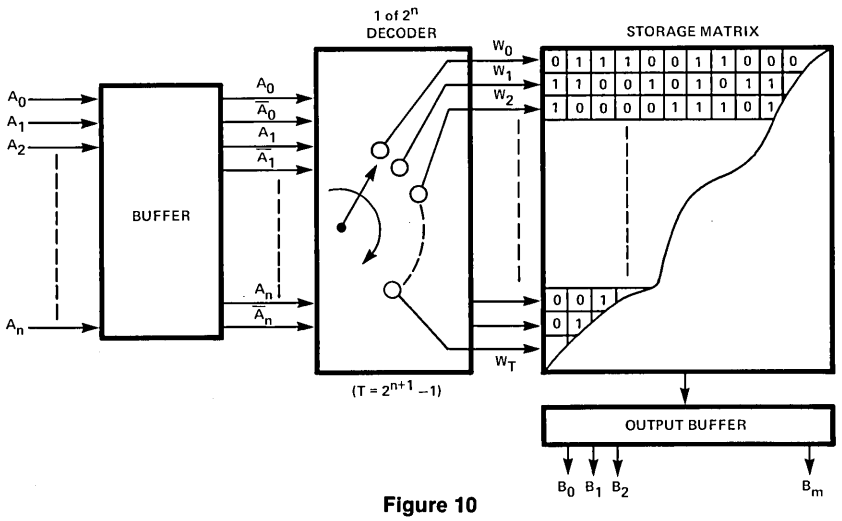


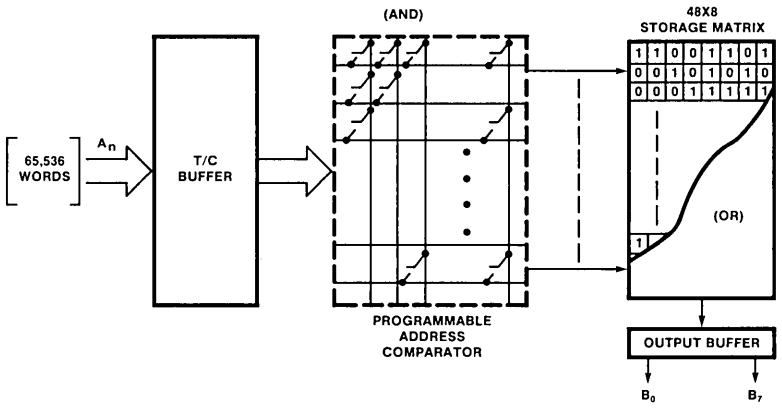
Figure 10

TYPICAL TRUTH TABLE STORED IN AN 8X4 PROM

	ADDRESS			OUTPUTS				
Mn	A_2	A_1	A_0	O_1	O_2	O_3	O_4	
0	0	0	0	1	1	1	1	
1	0	0	1	0	0	1	1	
2	0	1	0	0	0	0	0	INACTIVE MINTERMS
3	0	1	1	0	1	1	1	
4	1	0	0	0	0	0	0	
5	1	0	1	1	0	1	0	
6	1	1	0	1	1	0	0	
7	1	1	1	0	0	0	0	

Figure 11

TYPICAL FPLA ORGANIZATION



The input buffer drives a programmable address matrix, in which any one of 2^{n+1} input combinations can be programmed to select a stored word.

Figure 12

As a result, storage for unused minterms is no longer required. The necessary logic output for the inactive minterms occurs by "default." And, don't care states of input variables can be directly programmed in the FPLA. This allows to program the FPLA with either minterms, or the more general product terms (P-terms) of the input variables (addresses) to minimize logic "waste."

When any programmed logic combination is present at the FPLA inputs, the corresponding address matrix column (P-term) will be pulled high (logically active), forcing all (B) outputs to their true logic state programmed in the storage matrix. Conversely, for all **unprogrammed** logic combinations present at the FPLA inputs, all columns will remain low (logically inactive) forcing all (B) outputs to their false logic state by **default** (the complementary logic state of their programmed active level polarity).

Because it is programmable, the FPLA address matrix is not bound in size by the number of inputs it examines. Signetics' FPLA has 16 inputs to the matrix. If it were a PROM, this address matrix would have to be large enough to decode the address of 65,536 words. For the FPLA, the matrix has to be only large enough to store the address of 48 words: the FPLA's P-terms. The advantage comes about because here we have a choice to select a minimum of any 48 input words (or more, as determined by don't care input variables) from a total available pool of 65,536.

Due to the unique capability of FPLAs to store directly don't care (X) input states, each internal word (W) in the device storage matrix can be addressed by several logic input combinations (minterms), given by:

$$(M_n)T = 2^{m-r}$$

Where m = total number of input variables
r = number of active inputs (true or complement) contained in a programmed P-term column.

Thus, if $P_t = XXXI_0$, m = 4 and r = 1, for which $(M_n)T = 8$.

FPLA RESOURCES

Signetics' family of bipolar Field Programmable Logic Arrays includes both tri-state (82S100), and open collector devices (82S101), featuring the following characteristics:

- Field programmable (Ni-Cr link)
- 16-input variables
- 8 output functions
- 48 product terms
- 50ns max. access time (0-75°C)
- 600mW power dissipation (typical)
- TTL compatible
- 28-pin package
- $\overline{CE}$ input for expansion or inhibit
- Outputs individually programmable active "high" or "low"
- Single +5V power supply

These features and organization combine into an easy to use, high performance device, affording distinct user benefits:

A. 16-input variables
The 16X8 I/O configuration permits direct byte manipulations required by intelligent terminals, peripherals, microprocessor based emulators, minicomputers, and all the way up to the larger mainframes. Also, in address mapping applications, it provides the capability to scan an address field 65,536 words deep.

B. Chip Enable input
The Chip Enable input is a major improvement over alternate devices:

- Eases expansion of input variables and/or product terms.
- Permits application of tri-state device in bus organized systems.
- Provides logic inhibit or preconditional decoding functions.
- Provides a unique "default" logic state for all outputs, regardless of programmed output polarity.

C. Fastest access time
50ns maximum over the commercial temperature range renders the replacement of random logic feasible.

D. Fully buffered devices
All product terms can be utilized as many times as required, without affecting device speed and power dissipation.

E. 48 product terms (P-terms)
Allow the user to store in the FPLA 48 distinct words of 8 bits each. These 48 words can be addressed by a **minimum** of 48 input address combinations, chosen by the user among a total available pool of 2^{16} (65,536).

F. Polarity of all outputs individually programmable active-high or active-low
This feature is particularly useful in achieving further product term minimization in cases where the complement of an output function can be implemented with fewer product terms.

Example:
As shown in Figure 13, a 50% reduction in P-terms is obtained when the output of the logical structure of $\overline{F_Y}$ is inverted by means of a gate **external** to the elementary FPLA. The desired function F_Y is then realized with penalties in hardware, and circuit delays (however small). These are eliminated when using an FPLA with output polarity programmed active-low to realize the function 0's, rather than 1's.

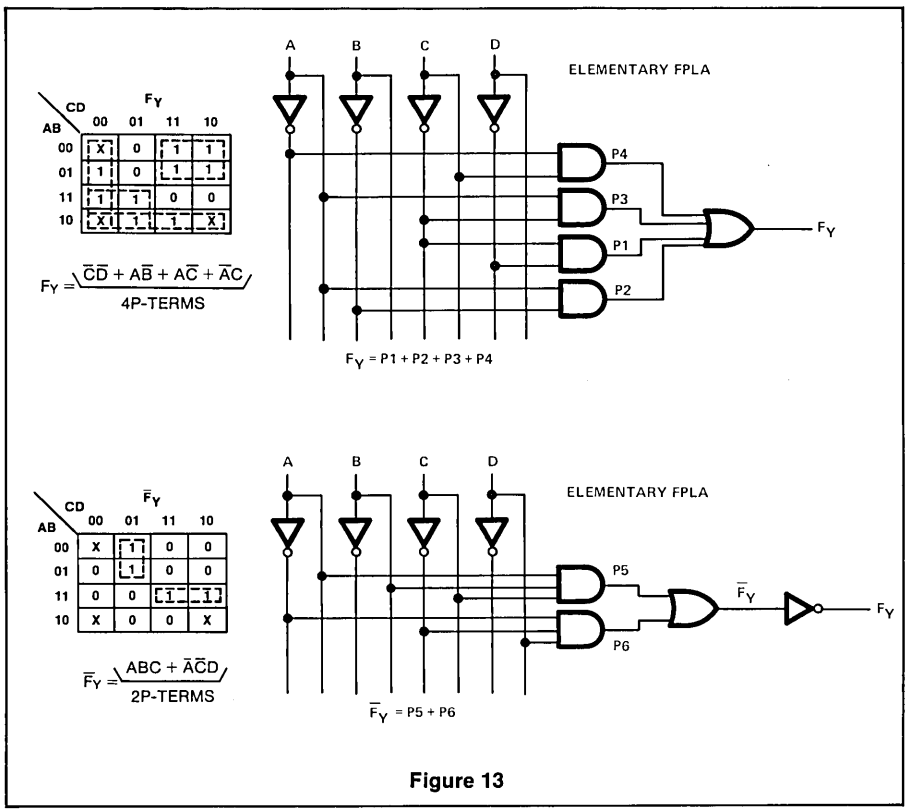


Figure 13

CHAPTER 2
DATA SPECIFICATIONS

DESCRIPTION

The 82S100 (tri-state outputs) and the 82S101 (open collector outputs) are Bipolar Programmable Logic Arrays, containing 48 product terms (AND terms), and 8 sum terms (OR terms). Each OR term controls an output function which can be programmed either true active-high (Fp), or true active-low ($F\dot{p}$). The true state of each output function is activated by any logical combination of 16-input variables, or their complements, up to 48 terms. Both devices are field programmable, which means that custom patterns are immediately available by following the fusing procedure outlined in this data sheet.

The 82S100 and 82S101 are fully TTL compatible, and include chip-enable control for expansion of input variables, and output inhibit. They feature either open collector or tri-state outputs for ease of expansion of product terms and application in bus-organized systems.

Both devices are available in commercial and military temperature ranges. For the commercial temperature range (0°C to +75°C) specify N82S100/101,I or N, and for the military temperature range (-55°C to +125°C) specify S82S100/101,I.

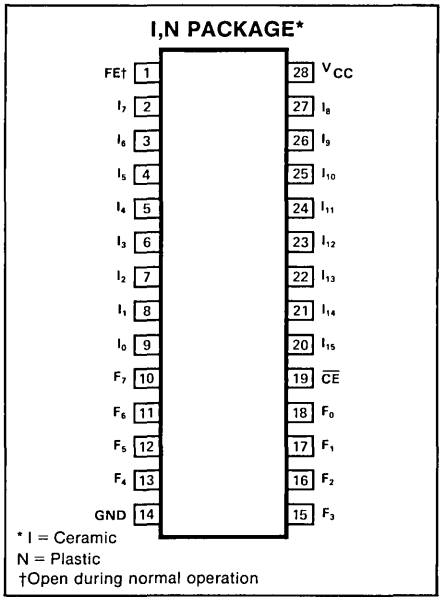
FEATURES

- **Field programmable (Ni-Cr link)**
- **Input variables: 16**
- **Output functions: 8**
- **Product terms: 48**
- **Address access time:**
  S82S100/101—80ns Max
  N82S100/101—50ns Max
- **Power dissipation: 600mW typ**
- **Input loading:**
  S82S100/101: -50μA Max
  N82S100/101: -100μA Max
- **Chip enable input**
- **Output option:**
  82S100: Tri-state
  82S101: Open collector
- **Output disable function:**
  Tri-state—Hi-Z
  Open collector—Hi

APPLICATIONS

- CRT display systems
- Random logic
- Code conversion
- Peripheral controllers
- Function generators
- Look-up and decision tables
- Microprogramming
- Address mapping
- Character generators
- Sequential controllers
- Data security encoders
- Fault detectors
- Frequency synthesizers

PIN CONFIGURATION

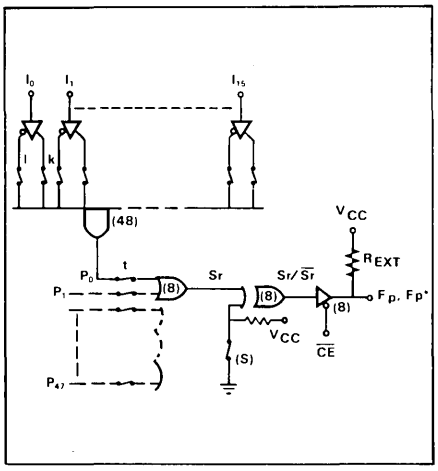


* I = Ceramic
N = Plastic
†Open during normal operation

TRUTH TABLE

MODE	Pn	$\overline{CE}$	Sr ? f(Pn)	Fp	$F\dot{p}$
Disabled (82S101)	X	1	X	1	1
Disabled (82S100)				Hi-Z	Hi-Z
Read	1	0	Yes	1	0
	0	0		0	1
	X	0	No	0	1

FPLA EQUIVALENT LOGIC PATH



LOGIC FUNCTION

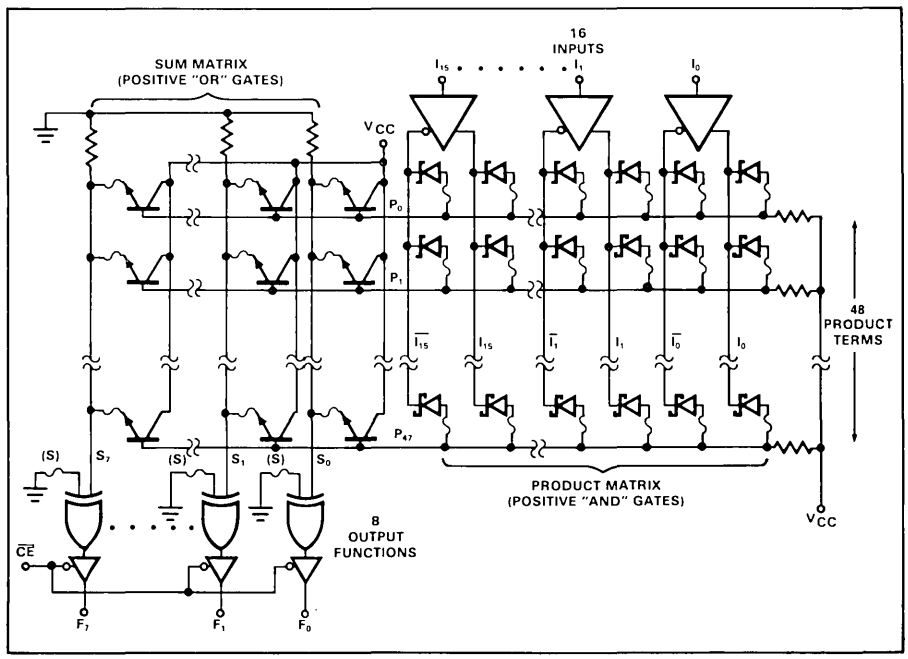
Typical Product Term:
$P_0 = I_0 \bullet I_1 \bullet \overline{I_2} \bullet I_5 \bullet \overline{I_{13}}$

Typical Output Functions:
$F_0 = (\overline{CE}) + (P_0 + P_1 + P_2)$ @ S = Closed
$F_0^* = (\overline{CE}) + (\overline{P_0} \bullet \overline{P_1} \bullet \overline{P_2})$ @ S = Open

NOTE

For each of the 8 outputs, either the function Fp (active-high) or $F\dot{p}$ (active low) is available, but not both. The required function polarity is programmed via link (S).

LOGIC DIAGRAM



ABSOLUTE MAXIMUM RATINGS[1]

PARAMETER		RATING Min	RATING Max	UNIT
V_{CC}	Supply voltage		+7	Vdc
V_{IN}	Input voltage		+5.5	Vdc
V_{OUT}	Output voltage		+5.5	Vdc
I_{IN}	Input currents	-30	+30	mA
I_{OUT}	Output currents		+100	mA
	Temperature range			°C
T_A	Operating			
	N82S100/101	0	+75	
	S82S100/101	-55	+125	
T_{STG}	Storage	-65	+150	

THERMAL RATINGS

TEMPERATURE	MILITARY	COMMERCIAL
Maximum junction	175°C	150°C
Maximum ambient	125°C	75°C
Allowable thermal rise ambient to junction	50°C	75°C

DC ELECTRICAL CHARACTERISTICS

N82S100/101: $0° \le T_A \le +75°C$, $4.75V \le V_{CC} \le 5.25V$

S82S100/101: $-55°C \le T_A \le +125°C$, $4.5V \le V_{CC} \le 5.5V$

PARAMETER		TEST CONDITIONS	N82S100/101 Min	N82S100/101 Typ[2]	N82S100/101 Max	S82S100/101 Min	S82S100/101 Typ[2]	S82S100/101 Max	UNIT
	Input voltage[3]								V
V_{IH}	High	V_{CC} = Max	2			2			
V_{IL}	Low	V_{CC} = Min			0.85			0.8	
V_{IC}	Clamp[3,4]	V_{CC} = Min, I_{IN} = -18mA		-0.8	-1.2		-0.8	-1.2	
	Output voltage	V_{CC} = Min							V
V_{OH}	High (82S100)[3,5]	I_{OH} = -2mA	2.4			2.4			
V_{OL}	Low[3,6]	I_{OL} = 9.6mA		0.35	0.45		0.35	0.50	
	Input current								μA
I_{IH}	High	V_{IN} = 5.5V		<1	25		<1	50	
I_{IL}	Low	V_{IN} = 0.45V		-10	-100		-10	-150	
	Output current	V_{CC} = Max							
I_{OLK}	Leakage[7]	V_{OUT} = 5.5V		1	40		1	60	μA
$I_{O(OFF)}$	Hi-Z state (82S100)[7]	V_{OUT} = 5.5V		1	40		1	60	μA
		V_{OUT} = 0.45V		-1	-40	-1		-60	
I_{OS}	Short circuit (82S100)[4,8]	V_{OUT} = 0V	-20		-70	-15		-85	mA
I_{CC}	V_{CC} supply current[9]	V_{CC} = Max		120	170		120	180	mA
	Capacitance[7]	V_{CC} = 5.0V							pF
C_{IN}	Input	V_{IN} = 2.0V		8			8		
C_{OUT}	Output	V_{OUT} = 2.0V		17			17		

AC ELECTRICAL CHARACTERISTICS

$R_1 = 470\Omega$, $R_2 = 1k\Omega$, $C_L = 30pF$

N82S100/101: $0°C \le T_A \le +75°C$, $4.75V \le V_{CC} \le 5.25V$

S82S100/101: $-55°C \le T_A \le +125°C$, $4.5V \le V_{CC} \le 5.5V$

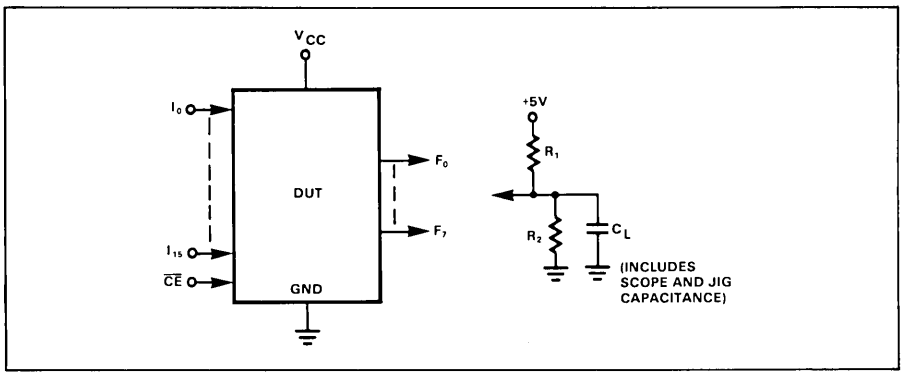
PARAMETER		TO	FROM	N82S100/101 Min	N82S100/101 Typ[2]	N82S100/101 Max	S82S100/101 Min	S82S100/101 Typ[2]	S82S100/101 Max	UNIT
	Access time									ns
T_{IA}	Input	Output	Input		35	50		35	80	
T_{CE}	Chip enable	Output	Chip enable		15	30		15	50	
	Disable time									ns
T_{CD}	Chip disable	Output	Chip enable		15	30		15	50	

NOTES on following page.

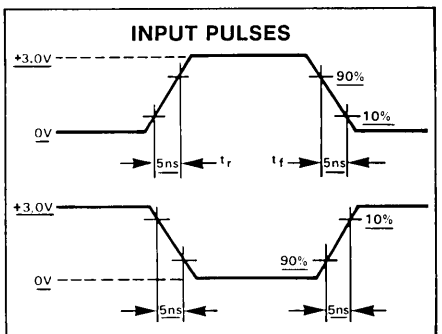
NOTES

1. Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only, and functional operation of the device of these or any other condition above those indicated in the operation of the device specifications is not implied.
2. All typical values are at V_{CC} = 5V, T_A = 25°C.
3. All voltage values are with respect to network ground terminal.
4. Test one at the time.
5. Measured with V_{IL} applied to $\overline{CE}$ and a logic high stored.
6. Measured with a programmed logic condition for which the output test is at a low logic level. Output sink current is applied thru a resistor to V_{CC}.
7. Measured with: V_{IH} applied to $\overline{CE}$.
8. Duration of short circuit should not exceed 1 second.
9. I_{CC} is measured with the chip enable input grounded, all other inputs at 4.5V and the outputs open.

TEST LOAD CIRCUIT

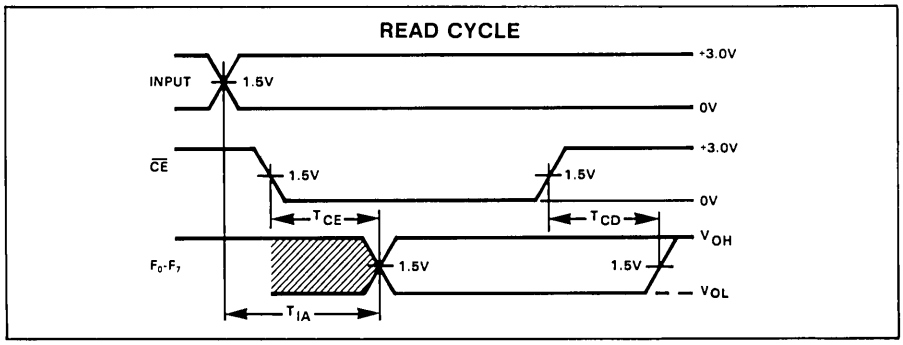


VOLTAGE WAVEFORM



Measurements: All circuit delays are measured at the +1.5V level of inputs and outputs.

TIMING DIAGRAM



TIMING DEFINITIONS

T_{CE} Delay between beginning of Chip Enable low (with Address valid) and when Data Output becomes valid.

T_{CD} Delay between when Chip Enable becomes high and Data Output is in off state (Hi-Z or high).

T_{IA} Delay between beginning of valid Input (with Chip Enable low) and when Data Output becomes valid.

VIRGIN DEVICE

The 82S100/101 are shipped in an unprogrammed state, characterized by:

1. All internal Ni-Cr links are intact.
2. Each product term (P-term) contains both true and complement values of every input variable I_m (P-terms always logically " alse").
3. The "OR" Matrix contains all 48-P-terms.
4. The polarity of each output is set to active high (Fp function).
5. All outputs are at a low logic level.

RECOMMENDED PROGRAMMING PROCEDURE

To program each of 8 Boolean logic functions of 16 true or complement variables, including up to 48 P-terms, follow the Program/Verify procedures for the "AND" matrix, "OR" matrix, and output polarity outlined below. To maximize recovery from programming errors, leave all links in unused device areas intact.

SET-UP

Terminate all device outputs with a 10K resistor to +5V. Set GND (pin 14) to 0V.

Output Polarity

PROGRAM ACTIVE LOW ($F\bar{p}$ FUNCTION)

Program output polarity before programing "AND" matrix and "OR" matrix. Program 1 output at the time. (S) links of unused outputs are not required to be fused.

1. Set FE (pin 1) to V_{FEL}.
2. Set V_{CC} (pin 28) to V_{CCL}.
3. Set $\overline{CE}$ (pin 19), and I_0 through I_{15} to V_{IH}.
4. Apply V_{OPH} to the appropriate output, and remove after a period t_p.
5. Repeat step 4 to program other outputs.

VERIFY OUTPUT POLARITY

1. Set FE (pin 1) to V_{FEL}; set V_{CC} (pin 28) to V_{CCS}.
2. Enable the chip by setting $\overline{CE}$ (pin 19) to V_{IL}.
3. Address a non-existent P-term by applying V_{IH} to all inputs I_0 through I_{15}.
4. Verify output polarity by sensing the logic state of outputs F_0 through F_7. All outputs at a high logic level are programmed active low ($F_{\bar{p}}$ function), while all outputs at a low logic level are programmed active high (F_p function).
5. Return V_{CC} to V_{CCP} or V_{CCL}.

"AND" Matrix

PROGRAM INPUT VARIABLE

Program one input at the time and one P-term at the time. All input variable links of unused P-terms are not required to be fused. However, unused input variables must be programmed as Don't Care for all programmed P-terms.

1. Set FE (pin 1) to V_{FEL}, and V_{CC} (pin 28) to V_{CCP}.
2. Disable all device outputs by setting $\overline{CE}$ (pin 19) to V_{IH}.
3. Disable all input variables by applying V_{IX} to inputs I_0 through I_{15}.
4. Address the P-term to be programmed (No. 0 through 47) by forcing the corresponding binary code on outputs F_0 through F_5 with F_0 as LSB. Use standard TTL logic levels V_{OHF} and V_{OLF}.
5a. If the P-term contains neither I_0 nor $\overline{I_0}$ (input is a Don't Care), fuse both I_0 and $\overline{I_0}$ links by executing both steps 5b and 5c, before continuing with step 7.
5b. If the P-term contains I_0, set to fuse the $\overline{I_0}$ link by lowering the input voltage at I_0 from V_{IX} to V_{IH}. Execute step 6.
5c. If the P-term contains $\overline{I_0}$, set to fuse the I_0 link by lowering the input voltage at I_0 from V_{IX} to V_{IL}. Execute step 6.
6a. After t_D delay, raise FE (pin 1) from V_{FEL} to V_{FEH}.
6b. After t_D delay, pulse the $\overline{CE}$ input from V_{IH} to V_{IX} for a period t_p.
6c. After t_D delay, return FE input to V_{FEL}.
7. Disable programmed input by returning I_0 to V_{IX}.
8. Repeat steps 5 through 7 for all other input variables.
9. Repeat steps 4 through 8 for all other P-terms.
10. Remove V_{IX} from all input variables.

VERIFY INPUT VARIABLE

1. Set FE (pin 1) to V_{FEL}; set V_{CC} (pin 28) to V_{CCP}.
2. Enable F_7 output by setting $\overline{CE}$ to V_{IX}.
3. Disable all input variables by applying V_{IX} to inputs I_0 through I_{15}.
4. Address the P-term to be verified (No. 0 through 47) by forcing the corresponding binary code on outputs F_0 through F_5.
5. Interrogate input variable I_0 as follows:
   A. Lower the input voltage at I_0 from V_{IX} to V_{IH}, and sense the logic state of output F_7.
   B. Lower the input voltage at I_0 from V_{IH} to V_{IL}, and sense the logic state output F_7.

The state of I_0 contained in the P-term is determined in accordance with the following truth table:

I_0	F_7	INPUT VARIABLE STATE CONTAINED IN P-TERM
0 1	1 0	$\overline{I_0}$
0 1	0 1	I_0
0 1	1 1	Don't Care
0 1	0 0	(I_0), $(\overline{I_0})$

Note that 2 tests are required to uniquely determine the state of the input variable contained in the P-term.

6. Disable verified input by returning I_0 to V_{IX}.
7. Repeat steps 5 and 6 for all other input variables.
8. Repeat steps 4 through 7 for all other P-terms.
9. Remove V_{IX} from all input variables.

"OR" MATRIX

PROGRAM PRODUCT TERM

Program one output at the time for one P-term at the time. All P_n links in the "OR" matrix corresponding to unused outputs and unused P-terms are not required to be fused.

1. Set FE (pin 1) to V_{FEL}.
2. Disable the chip by setting $\overline{CE}$ (pin 19) to V_{IH}.
3. After t_D delay, set V_{CC} (pin 28) to V_{CCS}, and inputs I_6 through I_{15} to V_{IH}, V_{IL}, or V_{IX}.
4. Address the P-term to be programmed (No. 0 through 47) by applying the corresponding binary code to input variables I_0 through I_5, with I_0 as LSB.
5a. If the P-term is contained in output function F_0 ($F_0 = 1$ or $F_0^* = 0$), got to step 6, (fusing cycle not required).
5b. If the P-term is **not** contained in output function F_0 ($F_0 = 0$ or $F_0^* = 1$), set to fuse the P_n link by forcing output F_0 to V_{OPF}.
6a. After t_D delay, raise FE (pin 1) from V_{FEL} to V_{FEH}.
6b. After t_D delay, pulse the $\overline{CE}$ input from V_{IH} to V_{IX} for a period t_p.
6c. After t_D delay, return FE input to V_{FEL}.
6d. After t_D delay, remove V_{OPF} from output F_0.
7. Repeat steps 5 and 6 for all other output functions.
8. Repeat steps 4 through 7 for all other P-terms.
9. Remove V_{CCS} from V_{CC}.

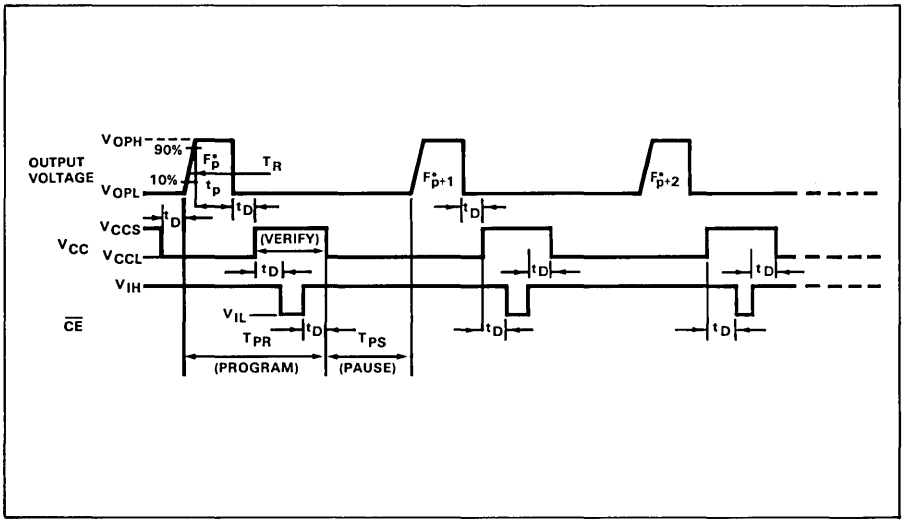
VERIFY PRODUCT TERM

1. Set FE (pin 1) to V_{FEL}.
2. Disable the chip by setting $\overline{CE}$ (pin 19) to V_{IH}.
3. After t_D delay, set V_{CC} (pin 28) to V_{CCS}, and inputs I_0 through I_{15} to V_{IH}, V_{IL}, or V_{IX}.
4. Address the P-term to be verified (No. 0 through 47) by applying the corresponding binary code to input variables I_0 through I_5.
5. After t_D delay, enable the chip by setting $\overline{CE}$ (pin 19) to V_{IL}.
6. To determine the status of the P_n link in the "OR" matrix for each output function F_p or F_p^*, sense the state of outputs F_0 through F_7. The status of the link is given by the following truth table:

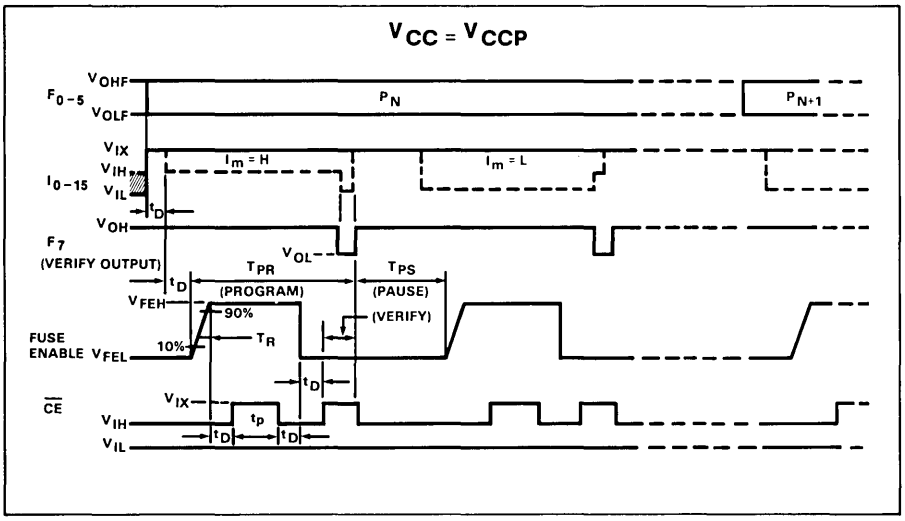
OUTPUT		P-TERM LINK
Active High (Fp)	Active Low (F_p^*)	
0 1	1 0	**Fused** **Present**

7. Repeat steps 4 through 6 for all other P-terms.
8. Remove V_{CCS} from V_{CC}.

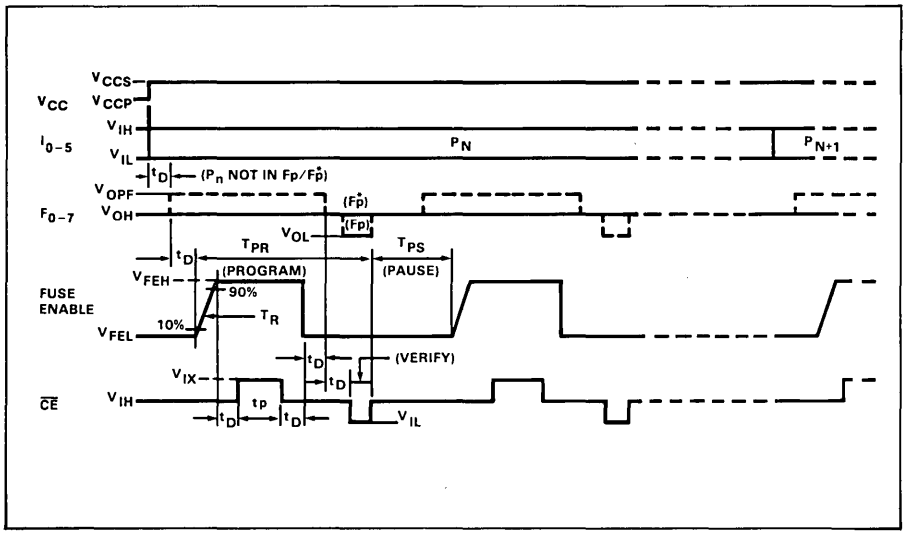
OUTPUT POLARITY PROGRAM-VERIFY SEQUENCE (TYPICAL)



"AND" MATRIX PROGRAM-VERIFY SEQUENCE (TYPICAL)



"OR" MATRIX PROGRAM-VERIFY SEQUENCE (TYPICAL)



PROGRAMMING SYSTEM SPECIFICATIONS[1] (T_A = +25°C)

PARAMETER		TEST CONDITIONS	LIMITS Min	LIMITS Typ	LIMITS Max	UNIT
V_{CCS}	V_{CC} supply (program/verify "OR", verify output polarity)[2]	I_{CCS} = 550mA, min, Transient or steady state	8.25	8.5	8.75	V
V_{CCL}	V_{CC} supply (program output polarity)		0	0.4	0.8	V
I_{CCS}	I_{CC} limit (program "OR")	V_{CCS} = +8.75 ± .25V	550		1,000	mA
	Output voltage					V
V_{OPH}	Program output polarity[3]	I_{OPH} = 300 ± 25mA	16.0	17.0	18.0	
V_{OPL}	Idle		0	0.4	0.8	
I_{OPH}	Output current limit (Program output polarity)	V_{OPH} = +17 ± 1V	275	300	325	mA
	Input voltage					V
V_{IH}	High		2.4		5.5	
V_{IL}	Low		0	0.4	0.8	
	Input current					µA
I_{IH}	High	V_{IH} = +5.5V			50	
I_{IL}	Low	V_{IL} = 0V			-500	
	Forced output voltage					V
V_{OHF}	High		2.4		5.5	
V_{OLF}	Low		0	0.4	0.8	
	Output current					
I_{OHF}	High	V_{OHF} = +5.5V			100	µA
I_{OLF}	Low	V_{OLF} = 0V			-1	mA
V_{IX}	$\overline{CE}$ program enable level		9.5	10	10.5	V
I_{IX1}	Input variables current	V_{IX} = +10V			2.5	mA
I_{IX2}	$\overline{CE}$ input current	V_{IX} = +10V			5.0	mA
V_{FEH}	FE supply (program)[3]	I_{FEH} = 300 ± 25mA, Transient or steady state	16.0	17.0	18.0	V
V_{FEL}	FE supply (idle)	I_{FEL} = -1mA, max	1.25	1.5	1.75	V
I_{FEH}	FE supply current limit	V_{FEH} = +17 ± 1V	275	300	325	mA
V_{CCP}	V_{CC} supply (program/verify "AND")	I_{CCP} = 550mA, min, Transient or steady state	4.75	5.0	5.25	V
I_{CCP}	I_{CC} limit (program "AND")	V_{CCP} = +5.0 ± .25V	550		1,000	mA
V_{OPF}	Forced output (program)		9.5	10	10.5	V
I_{OPF}	Output current (program)				10	mA
T_R	Output pulse rise time		10		50	µs
t_P	$\overline{CE}$ programming pulse width		0.3	0.4	0.5	ms[5]
t_D	Pulse sequence delay		10			µs
T_{PR}	Programming time			0.6		ms
$\frac{T_{PR}}{T_{PR} + T_{PS}}$	Programming duty cycle				50	%
F_L	Fusing attempts per link				2	cycle
V_S	Verify threshold[4]		1.4	1.5	1.6	V

NOTES

1. These are specifications which a Programming System must satisy in order to be qualified by Signetics.
2. Bypass V_{CC} to GND with a 0.01µf capacitor to reduce voltage spikes.
3. Care should be taken to ensure that the voltage is maintained during the entire fusing cycle. The recommended supply is a constant current source clamped at the specified voltage limit.
4. V_S is the sensing threshold of the FPLA output voltage for a programmed link. It normally constitutes the reference voltage applied to a comparator circuit to verify a successful fusing attempt.
5. These are new limits resulting from device improvements, and which supersede, but do not obsolete the performance requirements of previously manufactured programming equipment.

16X48X8 FPLA PROGRAM TABLE

THIS PORTION TO BE COMPLETED BY SIGNETICS

CF (XXXX) _____

CUSTOMER SYMBOLIZED PART # _____

DATE RECEIVED _____

COMMENTS _____

CUSTOMER NAME _____

PURCHASE ORDER # _____

SIGNETICS DEVICE # _____

TOTAL NUMBER OF PARTS _____

PROGRAM TABLE # _____

REV _____ DATE _____

PROGRAM TABLE ENTRIES						
INPUT VARIABLE			OUTPUT FUNCTION		OUTPUT ACTIVE LEVEL	
Im	$\overline{Im}$	Don't Care	Prod. Term Present in F_P	Prod. Term Not Present in F_P	Active High	Active Low
H	L	— (dash)	A	• (period)	H	L
NOTE: Enter (—) for unused inputs of used P-terms.			NOTES: 1. Entries independent of output polarity. 2. Enter (A) for unused outputs of used P-terms.		NOTES: 1. Polarity programmed once only. 2. Enter (H) for all unused outputs.	

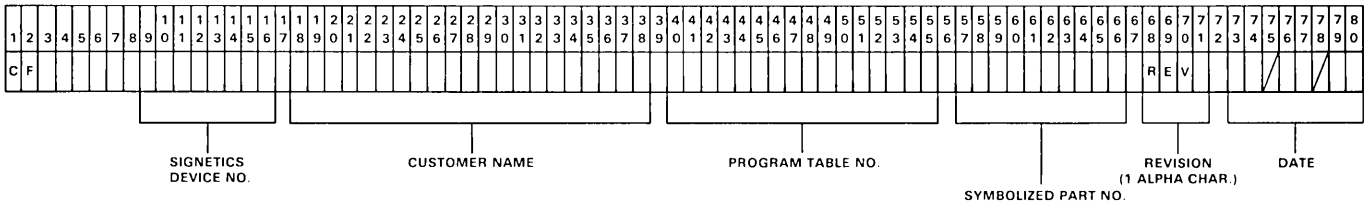
	PRODUCT TERM[1]																ACTIVE LEVEL[1]							
	INPUT VARIABLE[1]																OUTPUT FUNCTION[1]							
NO.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								
21																								
22																								
23																								
24																								
25																								
26																								
27																								
28																								
29																								
30																								
31																								
32																								
33																								
34																								
35																								
36																								
37																								
38																								
39																								
40																								
41																								
42																								
43																								
44																								
45																								
46																								
47																								

(1) Input and Output fields of unused P-terms can be left blank. Unused inputs and outputs are FPLA terminals left floating.

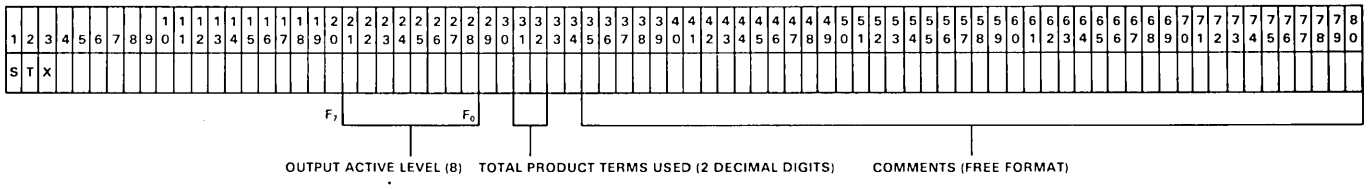
PUNCHED CARD CODING FORMAT

The FPLA Program Table can be supplied directly to Signetics in punched card form, using standard 80-column IBM cards. For each FPLA Program Table, the customer should prepare in input card deck in accordance with the following format. Product Term cards 3 through 50 can be in any order. Not all 48 Product Terms need to be present. Unused Product Terms require no entry cards, and will be skipped during the actual programming sequence:

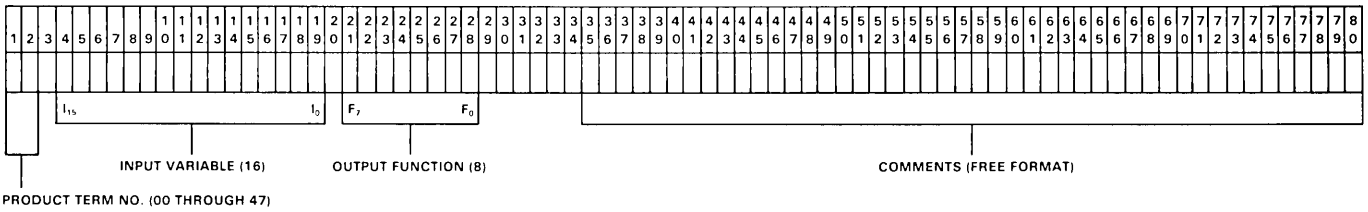
CARD NO.1—Free format within designated fields.



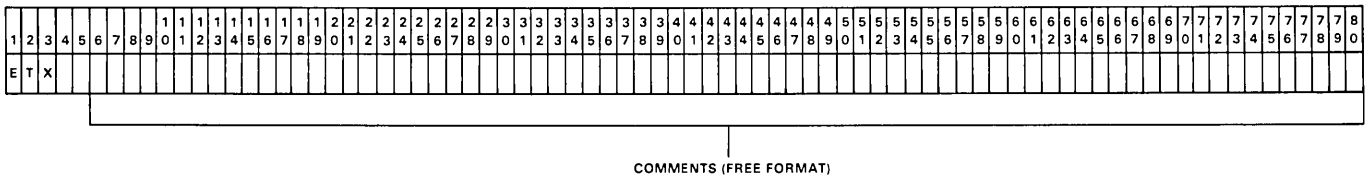
CARD NO. 2—



CARD NO. 3 through NO. 50



CARD NO. 51



Output Active Level entries are determined in accordance with the following table:

OUTPUT ACTIVE LEVEL	
Active high H	**Active low** L

NOTES

1. Polarity programmed once only.
2. Enter (H) for all unused outputs.

Input Variable entries are determined in accordance with the following table:

INPUT VARIABLE		
Im H	$\overline{\text{Im}}$ L	Don't care — (dash)

NOTE

Enter (—) for unused inputs of used P-terms.

Output Function entries are determined in accordance with the following table:

OUTPUT FUNCTION	
Product term present in F_P A	Product term *not* present in F_P • (period)

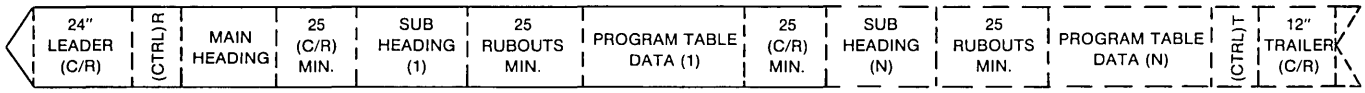
NOTES

1. Entries independent of output polarity.
2. Enter (A) for unused outputs of used P-terms.

TWX TAPE CODING FORMAT

The FPLA Program Table can be sent to Signetics in ASCII code format via airmail using any type of 8-level tape (paper, mylar, fanfold, etc.), or via TWX: just dial (910) 339-9283, tell the operator to turn the paper puncher on, and acknowledge. At the end of transmission instruct the operator to send tape to Signetics Order Entry.

A number of Program Tables can be sequentially assembled on a continuous tape as follows, however limit tape length to a roll of 1.75 inch inside diameter, and 4.25 inch outside diameter:



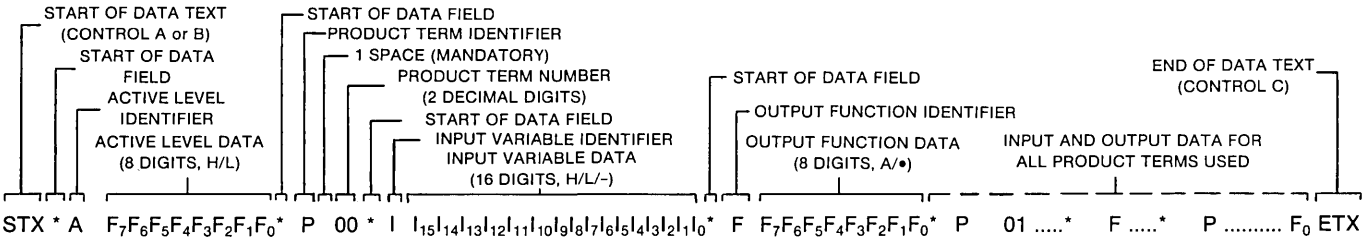
A. The MAIN HEADING at the beginning of tape includes the following information, with each entry preceded by a ($) character, whether used or not:

1. Customer Name _____
2. Customer TWX No. _____
3. Date _____
4. Purchase Order No. _____
5. Number of Program Tables _____
6. Total Number of Parts _____

B. Each SUB HEADING should contain specific information pertinent to each Program Table as follows, with each entry preceded by a ($) character, whether used or not:

1. Signetics Device No. _____
2. Program Table No. _____
3. Revision _____
4. Date _____
5. Customer Symbolized Part No. _____
6. Number of Parts _____

C. Program Table data blocks are initiated with an STX character, and terminated with an ETX character. The body of the data consists of Output Active Level, Product Term, and Output Function information separated by appropriate identifiers in accordance with the following format:



Entries for the 3 Data Fields are determined in accordance with the following Table:

INPUT VARIABLE		
I_m H	$\overline{I_m}$ L	Don't care — (dash)

NOTE

Enter (—) for unused inputs of used P-terms.

OUTPUT FUNCTION	
Product term present in Fp A	Product term not present in Fp • (period)

NOTES

1. Entries independent of output polarity.
2. Enter (A) for unused outputs of used P-terms.

OUTPUT ACTIVE LEVEL	
Active high H	**Active low** L

NOTES

1. Polarity programmed once only.
2. Enter (H) for all unused outputs.

Although the Product Term data are shown entered in sequence, this is not necessary. It is possible to input only one Product Term, if desired. Unused Product Terms require no entry. ETX signalling end of Program Table may occur with less than the maximum number of Product Terms entered.

NOTES

1. Corrections to any entry can be made by backspace and rubout. However, limit consecutive rubouts to less than 25.
2. P-Terms can be re-entered any number of times. The last entry for a particular P-Term will be interpreted as valid data.
3. Any P-Term can be deleted entirely by inserting the character (E) immediately following the P-Term number to be deleted, i.e., *P 25E deletes P-Term 25.
4. To facilitate an orderly Teletype print out, carriage returns, line feeds, spaces, rubouts etc. may be interspersed between data groups, but only preceding an asterisk (*).
5. Comments are allowed between data fields, provided that an asterisk (*) is not used in any Heading or Comment entry.

CHAPTER 3
PROGRAM/VERIFY PROCEDURE

PROGRAMMING SIGNETICS' FPLA

The FPLA is programmed by the user with the desired program table (or truth table) in 3 successive steps involving the AND matrix, the OR matrix and the transmission polarity of the output Exclusive-OR gates.

In its initial unprogrammed state, all Ni-Cr links are intact, such that:

- Each P-term contains both true and complement values of every input I_m. Hence, all P-terms are in the NULL state (unconditionally low).
- Each S-term contains all 48 P-terms.
- The polarity of each output is set to active-high (F_P function). Since all P-terms are inactive, all outputs will be at a low level when the chip is enabled ($\overline{CE}$ = low), regardless of input conditions.

The peripheral fusing circuitry inside the FPLA and a summary of the fusing requirements of the FPLA are shown in Figure 14 and Table 1, respectively. For a more detailed fusing procedure, refer to the data sheet in Chapter 2.

AND Matrix

Each P-term P_n is programmed by fusing the appropriate Ni-Cr links in all pairs that couple the P-term to each input variable. If P_n contains I_m, the $\overline{I_m}$ link is fused, and vice versa. If I_m is a don't care in P_n, both the I_m and $\overline{I_m}$ links must be fused. If fewer than 16 variables are used in any application, the unused variables represent don't care conditions for all used P-terms, and their corresponding I_m and $\overline{I_m}$ links **must** in general be fused (see **Editing,** below).

Since in a blank device all P-terms are in a logic null state, **unused** P-terms require no programming at all.

OR Matrix

The response of each output function to programmed P-terms is assigned in the OR matrix. If any product term P_n logically negates an output function, the link coupling that output function to the P-term must be fused. Conversely, if a P-term logically asserts (activates) an output function, the corresponding coupling link must remain intact.

No programming is required of OR matrix links coupling used or unused P-terms to S-terms servicing any unused output functions.

Output Active Level

The logic output transition (H → L, or L → H) required for each FPLA output function when activated by a selected P-term is programmed in the Ex-OR gates.

For an active-low output (H → L transition), the link grounding one input of the Ex-OR gate servicing the desired output must be fused. For an active-high output, the link must remain intact. No fusing is required of links servicing Ex-OR gates of unused outputs.

FUNCTIONAL FPLA BLOCKS ACTIVATED DURING ARRAY PROGRAM/VERIFY SEQUENCE

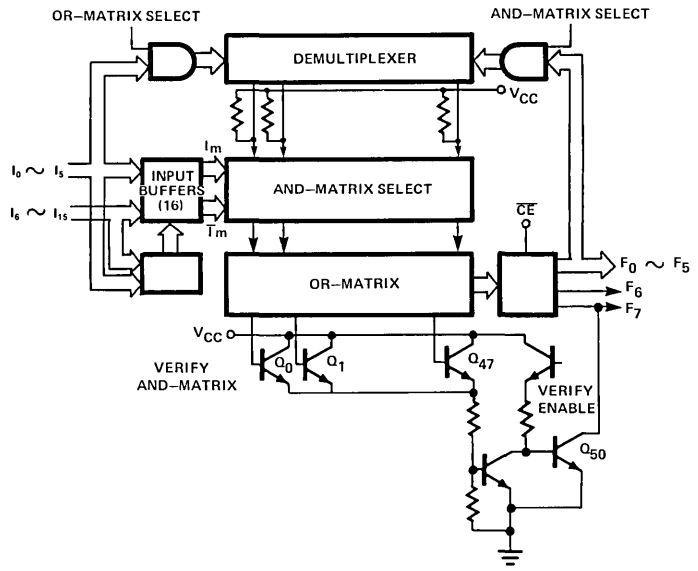


Figure 14

PROGRAM →		"AND" MATRIX	"OR" MATRIX	OUTPUT POLARITY
V_{CC}		+5.0V	+8.75V	"0"
INPUTS	Idle	+10.0V	Address P-Term with $I_0 \rightarrow I_5$	"1"
	Program I_m	"1"		
	Program $\overline{I_m}$	"0"		
OUTPUTS	Active-High	Address P-Term with $F_0 \rightarrow F_5$	N.A.	"0"
	Active-Low			+17.0V "0" 0.4MS
	(P_n) in		"0"	N.A.
	(P_n) out		+10.0V	
FE		+17.0V		+1.5V
$\overline{CE}$		+10.0V "1" 0.4MS		"1"

Entries "0" and "1" are standard TTL levels.

Table 1 SUMMARY OF FPLA INPUT REQUIREMENTS FOR PROGRAMMING RESPECTIVE AREAS IN THE DEVICE

EDITING SIGNETICS' FPLA

In contrast with PROMs, FPLAs have inherent program editing capabilities. After programming, the user can incorporate a number of program modifications in Signetics' FPLAs. These are tabulated in Table 2.

So, given a programmed function:

$$F_0 = I_0 + I_1 I_2 + I_3 \bar{I}_4 I_5$$

it is possible to modify it as:

$$\overline{F_0} = I_0 + \cancel{I_1 I_2} + I_1 \bar{I}_2 + I_3 \bar{I}_4 \cancel{I_5}$$

by:

1. Complement F_0 by reprogramming FPLA output active-low.
2. Delete P-term ($I_1 I_2$) from the OR matrix.
3. Program new P-term ($I_1 \bar{I}_2$) in the AND matrix.
4. Change input I_5 to Don't Care in P-term ($I_3 \bar{I}_4 I_5$) by fusing both I_5 links in the AND matrix.

GENERATING THE FPLA PROGRAM TABLE

In a typical application as in Figure 15, the symbolic statements, or the truth table, of a logic problem are first reduced to a minimum set of P-terms.

ELEMENTARY PROGRAM TO BE STORED IN FPLA

$\bar{F}_0 = P_0 + / + P_2$

$F_1 = / + P_1 + P_2$

a. Activity Map of elementary function set.

$P_0 = I_2 \bar{I}_1 \bar{I}_0$

$P_1 = I_2 \bar{I}_1 I_0$

$P_2 = \bar{I}_2 I_1 \bar{I}_0$

b. P-term List

Figure 15

The minimized output function set is expressed in the form of an activity map for tabulating those P-terms which are contained in an output function, and those which are not, designated by (P_n) or (/) in their respective positions.

The activity map eases the derivation of program table entries for the FPLA input variable field, output function field, and output active level polarity field.

The standard program table format adopted by Signetics is shown in Table 3. The term "Program Table" is used in favor of truth table, because the former allows don't cares (X) as a **direct** entry, and thus is more general, and conforms to FPLA structure.

Ideally the FPLA program table should contain entries formulated with a code which not only issues unambiguous fusing commands to a programming system, but also readily displays the actual logic state of the FPLA outputs. In dealing with logical statements or truth tables, most logic designers are used to either (1/0) or (H/L)

ADD	P-Term to F_p/F_p^*	YES	Program desired logic combination into any unused P-Term. Blow S-Term link(s) coupling P-Term to *inactive* output functions.
	$I_m/\overline{I_m}$ to P-Term	NO	Delete erroneous P-Term. Add new, corrected P-Term.
DELETE	P-Term from F_p/F_p^*	YES	Blow S-Term link coupling P-Term to F_p/F_p^*.
	$I_m/\overline{I_m}$ from P-Term	YES	Blow both links coupling the input variable to the P-Term.
CHANGE	$F_p \rightarrow F_p^*$	YES	Blow Ex-OR link of output to be inverted.
	$I_m/\overline{I_m} \rightarrow X$	YES	Delete $\overline{I_m}/I_m$ from P-Term.
	$I_m \leftrightarrow \overline{I_m}$	NO	Delete erroneous P-Term, and add a new P-Term.
	$F_p^* \rightarrow F_p$	NO	Use spare active-high output.

Table 2 SUMMARY OF "EDITING" FEATURES OF SIGNETICS' FPLA

NO.	PRODUCT TERM: INPUT VARIABLE (I_m) 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ACTIVE LEVEL: OUTPUT FUNCTION 7	6	5	4	3	2	1	0
0																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								
21																								
22																								
23																								
24																								
25																								
26																								
27																								
28																								
29																								
30																								
31																								
32																								
33																								
34																								
35																								
36																								
37																								
38																								
39																								
40																								
41																								
42																								
43																								
44																								
45																								
46																								
47																								

Used P-terms can be programmed anywhere. Unused P-terms require no programming, and can be left blank.

Table 3 STANDARD SIGNETICS' FPLA PROGRAM TABLE

symbols. An additional symbol (X) is generally used for don't care input states. Their widespread usage is a strong incentive to choose among these symbols for a suitable set to code the FPLA program table.

However, in many cases the program table will be transmitted to remote programming centers over commercial communication links, which normally employ an ASCII alpha-numeric code. Since the "distance" between the ASCII codes for "0" and "1" is only 1 bit, the risk of undetected transmission errors is large. Thus, the set (H, L, X) is more preferable, but it is still not ideal. Indeed, to enhance the production of low cost programming equipment, in which a low cost LED display is mandatory, one must forego the (X) in favor of a more realizable symbol such as a (—).

Therefore, the code for each state of the input variables in each P-term is coded as illustrated in Table 4. All entries clearly indicate the logic states of the input variables which activate a given P-term.

An additional symbol for the input variables is required to code the state in which both I_m and $\overline{I_m}$ links are intact. This code, chosen as (0) again for ease of display, is necessary to indicate the state of input variable(s) in a virgin device, or for unused or partially programmed P-terms. It is the **initial** state of all input variables, signifying their logic null state. **If any used P-term contains at least one variable in the null state, the P-term will never be selected by any logic input combination. Entry of a (0) in the program table is thus meaningless, and not allowed.** However, it does require to be displayed by a programming system to indicate blank check results, or program fail conditions.

While these symbols are appropriate to code the various states of the FPLA input variables for each P-term, as well as the output Active Level polarities, they give rise to some ambiguities when used to code the FPLA outputs, because of the user choice of output Active Level. To code the outputs of the FPLA, several alternatives are available. In all cases, derivation of each entry involves scanning the Activity Map to determine whether or not an output function contains a particular P-term. Regardless of chosen output polarity, a P-term activates F_p if it is **contained** in F_p. Accordingly, any F_p will be forced high, and $\overline{F_p}$ (defined as F_p^*) will be forced low. Conversely, if P_n is not contained in an output, all F_p and F_p^* functions will remain in their **default** logic state (low or high, respectively). A particularly convenient method for coding the FPLA output table is shown in Table 5.

This coding system utilizes an A (for Active) to indicate the presence of P_n in either F_p or F_p^*, and a • (period) to indicate absence. It has the advantage that the FPLA output table can be constructed directly from the activity map. Also, when retrieving the stored output table from a programmed device, the presence/absence of a P-term in an output function is readily detected, yielding the easiest array verification procedure. However, in order to relate the actual logic output of the FPLA to the above entries (especially when dealing with code conversion, or address translations), reference to Table 6 is necessary.

$P_n \stackrel{?}{=} f(I_m)$	INPUT STATE	PROGRAM TABLE ENTRY	FUSE COMMAND
Yes	I_m	Ⓗ	Fuse $\overline{I_m}$ link
	$\overline{I_m}$	Ⓛ	Fuse I_m link
No	Don't Care	⊖	Fuse both

Table 4 PROGRAM TABLE CODING OF INPUT VARIABLES

$F_p \stackrel{?}{=} f(P_n)$	ACTIVITY MAP	OUTPUT TABLE		FUSE COMMANDS
Yes	P_n	Ⓐ	Ⓐ	Do not Fuse P_n link in OR-Matrix
No	/	⊙	⊙	Fuse P_n link in OR-Matrix
	Active Level	H	L	
		F_p	F_p^*	Function Polarity

Entries, contained in ◯, are obtained by "multiplying" the contents of the activity map with the active level. Note that they are independent of output polarity.

Table 5 TABLE FOR FORMULATING OUTPUT TABLE ENTRIES FOR THE FPLA

FPLA OUTPUT TABLE	FPLA LOGIC OUTPUT		
Ⓐ	H	L	
⊙	L	H	
	Ⓗ	Ⓛ	Active Level
Function Polarity	F_p	F_p^*	

The FPLA output is obtained by "multiplying" output table entries with the active level.

Table 6 TABLE FOR CALCULATING THE FPLA LOGIC OUTPUT

	PRODUCT TERM: INPUT VARIABLE (I_m)																ACTIVE LEVEL: OUTPUT FUNCTION							
																	H						L	
NO.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0														H	L	L							•	A
1														H	L	H							A	•
2														L	H	L							A	A
3																								
4																								
5																								
6																								
7																								
8																								
9																								
44																								
45																								
46																								
47																								

Table 7 PARTIAL PROGRAM TABLE FOR THE EQUATION SET OF FIGURE 15

On the basis of the above coding system, a partial program table for the equation set in Figure 15 is shown in Table 7.

Note that only 3 P-terms and 2 outputs are used. Also note that the Active Level for FPLA outputs 0 and 1 has been set to L and H respectively to implement the required logic transition polarities of $\overline{F_0}$ and F_1.

To complete the table we must dispose of all its blank areas. The guiding concern here should be to leave intact as many as possible of the unused FPLA resources, for possible later use. Hence:

1. Leave blank Input Variable and Output Function fields of all unused P-terms (P_3 through P_{47}).
2. Enter H (initial virgin state) in the Active Level field of all unused output functions (F_2 through F_7).
3. Enter A (initial virgin state) in the Output Function field of all unused output functions (F_2 through F_7).
4. Enter a (—) for all unused input variables (I_3 through I_{15}) of used P-terms. In general this is not the best alternative. Another, more conservative option, is described below.

The complete program table for the above example, after applying rules 1 through 4, is shown in Table 8.

	PRODUCT TERM: INPUT VARIABLE (I_m)																ACTIVE LEVEL							
																	H	H	H	H	H	H	H	L
NO.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OUTPUT FUNCTION: 7	6	5	4	3	2	1	0
0	—	—	—	—	—	—	—	—	—	—	—	—	—	H	L	L	A	A	A	A	A	A	•	A
1	—	—	—	—	—	—	—	—	—	—	—	—	—	H	L	H	A	A	A	A	A	A	A	•
2	—	—	—	—	—	—	—	—	—	—	—	—	—	L	H	L	A	A	A	A	A	A	A	A
3																								
4																								
5																								
6																								
7																								
8																								
9																								
44																								
45																								
46																								
47																								

Table 8 COMPLETE PROGRAM TABLE FOR THE EQUATION SET OF FIGURE 15

DISPOSITION OF UNUSED INPUTS

When a particular application involves less than 16-input variables, if *unused* inputs are programmed as don't care in all *used* P-terms (M) of the FPLA, it is no longer possible to modify the logic structure of the (M) P-terms by reinstating any of the unused inputs as additional controlling variables to the FPLA.

While it is possible to recover from this condition by deleting P-terms requiring change, and adding any of the remaining (48-M) P-terms programmed with the desired number of input variables, this method ultimately fails once all 48 P-terms are exhausted.

This method can be combined with an alternate procedure to obtain a greater degree of flexibility in adding previously unused inputs to a preprogrammed FPLA. It requires that about one half of all originally unused inputs be programmed high and the remaining half low, in (M) P-terms only. These inputs are then normally tied to high and low logic levels respectively.

If at any time during function update or modification it becomes necessary to add high and/or low control variables to (N) of the (M) P-terms, any of the properly programmed idle inputs are disconnected from their voltage clamps and connected to their corresponding logic sources. These newly activated inputs must in turn be reprogrammed as don't care in (M-N) of the used P-terms.

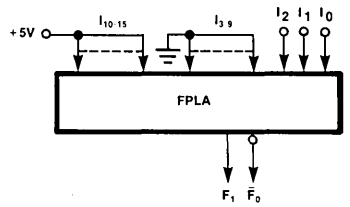
An illustration of the above concept is easily provided by recoding the previous problems as shown in Figure 16.

Suppose that later on in the design cycle a modification of system function is necessary, whereby:

FPLA WITH UNUSED INPUTS PROGRAMMED FOR LATER USE

	PRODUCT TERM: INPUT VARIABLE (I_m)																ACTIVE LEVEL							
																	H	H	H	H	H	H	H	L
NO.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	OUTPUT FUNCTION: 7	6	5	4	3	2	1	0
0	H	H	H	H	H	H	L	L	L	L	L	L	L	H	L	L	A	A	A	A	A	A	•	A
1	H	H	H	H	H	H	L	L	L	L	L	L	L	H	L	H	A	A	A	A	A	A	A	•
2	H	H	H	H	H	H	L	L	L	L	L	L	L	L	H	L	A	A	A	A	A	A	A	A
3																								
4																								
5																								
6																								
7																								
8																								
9																								
44																								
45																								
46																								
47																								

a. FPLA Program Table



b. FPLA Connections

Figure 16

$$\overline{F_0} = P_0 + / + P_2$$
$$F_1 = / + P_1 + P_2$$

and

$$P_0 = \overline{I_X}\, I_2\, \overline{I_1}\, \overline{I_0}$$
$$P_2 = I_Y\, \overline{I_2}\, I_1\, \overline{I_0}$$

The new high and low input variables in P_0 and P_2 can be readily included without resorting to adding new P-terms as shown in Figure 17.

VERIFYING THE STORED PROGRAM

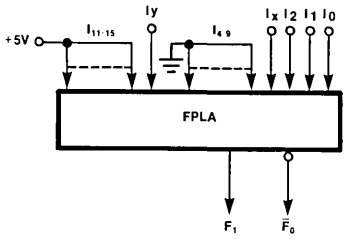
Unlike PROMs, verification of an FPLA after programming presents unique difficulties, posed by the large number of inputs to be manipulated and by the associative characteristic of FPLAs.

In general, the FPLA program table may bear little resemblance to the original truth table, yet, from a black box viewpoint, the logic function of the FPLA should match entry for entry the original truth table. This level of verification can only be obtained through a logic verification procedure, in which the logic transfer characteristic of the FPLA is exhaustively examined by

FPLA INCORPORATING ADDITIONAL INPUTS REQUIRED BY SYSTEM MODIFICATION

NO.	PRODUCT TERM: INPUT VARIABLE (I_m) 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ACTIVE LEVEL: H H H H H H L — OUTPUT FUNCTION 7	6	5	4	3	2	1	0
0	H	H	H	H	H	—	L	L	L	L	L	L	L	H	L	L	A	A	A	A	A	A	•	A
1	H	H	H	H	H	—	L	L	L	L	L	L	—	H	L	H	A	A	A	A	A	A	A	•
2	H	H	H	H	H	H	L	L	L	L	L	L	—	L	H	L	A	A	A	A	A	A	A	A
3																								
4																								
5																								
6																								
7																								
8																								
9																								
44																								
45																								
46																								
47																								

a. Modified FPLA Program Table incorporating additional active inputs $I_x = I_3$, and $I_y = I_{10}$



b. Modified FPLA Connections

Figure 17

exercising its inputs with a minterm generator.

But, while logic verification is the ultimate test of FPLA valid function, it is a useless tool for determining the FPLA stored program. This is readily apparent in Figure 18 which shows the output of an elementary FPLA to be the same (low) for 3 distinct internal programmed states, when its single input is toggled between high and low logic levels.

Since a non-ambiguous map of the status of every link in the device is a most essential tool required to monitor and manipulate the stored program (especially while interacting with an FPLA programming system or when duplicating from a master device), Signetics' FPLAs allow such map to be obtained via an array verify test sequence comprising 3 tests for examining the links in the output Ex-OR, the AND matrix, and the OR matrix.

ARRAY VERIFY

The peripheral fusing circuitry in Signetics' FPLAs incorporates additional networks and dedicated paths for the array verify test sequence. These are shown at the bottom of the composite FPLA diagram in Figure 14. Specifically, to sense the status of the AND matrix links, the OR matrix includes an extra row of non-fusible emitter followers Q_0 through Q_{47}, monitored via Q_{50} collector ORed with output F_7. This stage does not interfere with F_7 during normal operation because Q_{50} can only get base drive during verify mode.

The internal map of the FPLA is obtained by performing the sequence of tests summarized in Table 9, during which the Fuse Enable input is maintained at +1.5V. Verification of the active level polarity of the outputs is obtained by addressing a non-existent P-term in the device, and thus rely on the pull-down resistors in the OR matrix to yield a non-ambiguous result.

DISTINCT FPLA PROGRAMMED STATES RESULTING IN IDENTICAL LOGIC FUNCTION

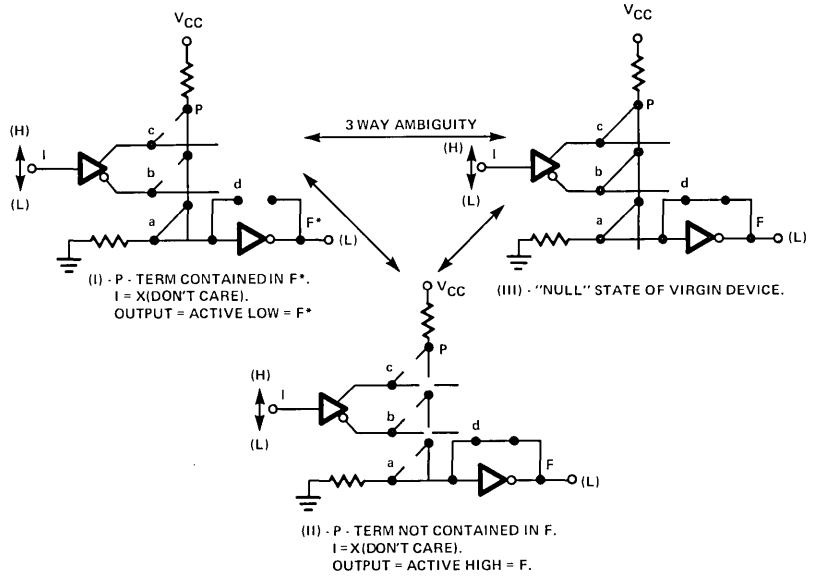


Figure 18

VERIFY →		"AND" MATRIX	"OR" MATRIX	OUTPUT POLARITY
V_{CC}		+5.0V	+8.75V	+8.75V
$\overline{CE}$		+10.0V	"0"	
INPUTS	Verify	(1): I_m = "0" (2): I_m = "1"	ADDRESS P-TERM WITH $I_0 \rightarrow I_5$	"1" (All)
INPUTS	Idle	+10.0V		
OUTPUTS	F_7	(1)— 0 | 0 | 1 | 1 (2)— 0 | 1 | 0 | 1 I_m = Null | I_m | $\overline{I_m}$ | Don't care	F_p | Act HI | Act LOW "0" | (P_n) out | (P_n) in	F_p | Polarity "0" | active-HIGH
OUTPUTS	Idle	ADDRESS P-TERM WITH $F_0 \rightarrow F_5$	"1" | (P_n) in | (P_n) out	"1" | active-LOW

The output active level test must be performed before the OR matrix test. Entries "0" and "1" are standard TTL levels.

Table 9 SUMMARY OF FPLA INPUT REQUIREMENTS FOR MAPPING THE STATUS OF ALL INTERNAL LINKS

To verify the AND matrix 2 tests are required for each input of all P-terms. The status of each I_m link coupling a P-term to the input buffer outputs is determined in accordance with Table 10.

I_m	F_7	INPUT VARIABLE STATE CONTAINED IN P-TERM	INPUT CODE
L	H	$\overline{I_m}$	L
H	L		
L	L	I_m	H
H	H		
L	H	Don't care	—
H	H		
L	L	(I_m), $(\overline{I_m})$	0
H	L		

Table 10 TABLE FOR DETERMINING THE STATUS OF EACH INPUT VARIABLE LINK IN THE AND MATRIX

Verification of the OR matrix requires prior knowledge of the output level polarities. The status of the OR matrix links coupling each P-term to the S-term is given by Table 11.

OUTPUT		
Active-high (F_p)	Active-low ($\overline{F_p}$)	P-term Link
L	H	FUSED
H	L	PRESENT

Table 11 TABLE FOR INTERPRETING THE STATUS OF OR MATRIX LINKS, BASED ON OUTPUT ACTIVE LEVEL TEST RESULTS

For a more detailed array verify procedure refer to the device data sheet.

LOGIC VERIFY

After an FPLA has been programmed, and its contents checked by array verify against hard-copy reference of the program table, there should be in most cases little reason to suspect that the device will not exhibit the correct logic function in a system environment. However, in some cases, device defects, programming equipment problems, user coding inexperience, as well as system logic races and other marginalities, may all contribute in creating a situation in which system failures are traced to an FPLA which nevertheless appears to contain the correct program table. In these cases, further device diagnostics are necessary to identify the source of the problem at hand, for which the actual operating system may be a slow and ineffective tool.

Also, at the end of the design cycle, some users may want to replace FPLAs with *mask* programmable PLAs for cost reduction. Since a PLA does not contain peripheral fusing circuitry, it is not possible to logically address each of its internal links to verify that the PLA contains the same program table as the master FPLA. In this case the only verification possible is a full logic verify of the PLA versus FPLA functions.

Ultimate verification of FPLA logic performance entails an exhaustive check of its logic function to compare the *expected* truth table with the *stored* truth table, obtained by cycling the FPLA inputs through all 2^{16} combinations with a minterm generator. This, however, involves dealing with a hardcopy reference of a table containing about 64,000 input entries, which is a totally impractical task in view of what may be required to generate and store such table.

A more feasible alternative consists of constructing a "hardwired" logic verify system which may be conveniently incorporated within the FPLA programming system. The programmer would then function as an FPLA emulator with the ability to produce and display the full truth table of the FPLA, viewed just as a logic box. This is extremely useful in code conversion, map translations, or when programming directly from a truth table.

In essence, the logic verify system must be able to compare the actual FPLA logic output with that computed on-the-fly by composite overlay and manipulation of the output table stored in the programmer, as activated by all concurrent and multiple address selections for each state of the input minterm generator.

The logic verify procedure presumes knowledge of the program table stored in the device; hence, it must necessarily follow an array verify operation to first scan and store in the system main memory the program table contained in the device under test. A comparison of the actual versus computed output tables in conjunction with a direct display of the FPLA logic output for each minterm input, will reveal all discrepancies.

To be useful, the logic verify procedure must also be fast. If should be complete within 5 to 10 seconds per device, and thus dictates use of a hardwired algorithm. The block diagram of a logic subsystem which executes a suitable algorithm, outlining basic hardware, controls, and data paths is shown in Figure 19.

The algorithm manipulates program table data stored in main memory and active level register, in the format contained in Table 12. Before loading the program table, M/M and the ALR are reset to "0," to clear all previously stored fusing commands. A binary counter, conditionally incremented, functions as minterm (M_n) generator, for addressing the FPLA with all 2^{16} input combinations. The FPLA output for each M_n input is stored in Register B. All 48 P-terms are fetched one at a time from the program table in M/M, and examined to determine whether they *logically contain* each M_n. The criteria which logically include or exclude M_n from a P-term are tabulated in Table 13 for all general programmed states. If the test fails, a new P-term is fetched, and the test repeated until all 48 P-terms have been examined, and all 2^{16} minterms are exhausted. On the other hand, if the test indicates that M_n is

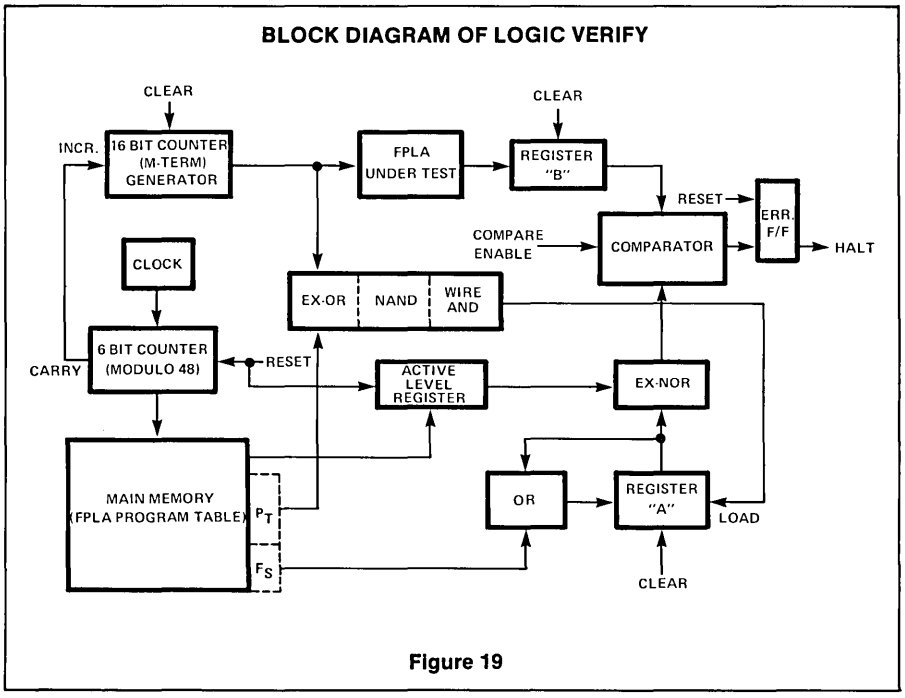


Figure 19

contained in the P-term, the F-set field associated with the addressed P-term is *overlaid* in Register A, while the M/M address of the P-term is stored in a stack containing the *concurrent* P-term list, and a presence flag set to indicate that the P-term address is a valid member of the list.

Testing continues until all 48 P-terms have been compared to the M_n count. At this point, Register A contains a *composite* FPLA output table obtained when all con ɹrrently selected P-terms are activated by M_n at the FPLA inputs. This table is merged through an EX-NOR with the contents of the ALR to produce a composite binary F-set, which is in turn compared with the contents of Register B. If they are equal, the M_n generator is incremented, and the test sequence repeated with M_n+1 until the last minterm. (Alternately, if in manual mode, before incrementing M_n one could observe the logic output of the FPLA with M_n as input by calling the contents of the display buffer). If the contents of Registers A and B differ, an error flag is set, and the M_n count halted. The following housekeeping displays occur, and the system will wait until a continue command:

- The concurrent P-term list is scanned and displayed in the designated field on the CRT.
- The contents of the M_n generator are displayed in the hexadecimal M-term count field, while its binary equivalent (presented to the FPLA inputs) is displayed in the Input field.
- Results of the EX-NOR of Register B with the contents of the ALR are displayed in the Output field. This yields the output table obtained from the *device* with M_n as input.
- The contents of the ALR are displayed in the Act Levl field.
- The contents of Register A are displayed in the Computed Output Table field. They indicate the composite output table *expected* from the FPLA with input M_n.
- The contents of Register B are displayed in the PLA Output field. They indicate the logic levels present at the FPLA outputs.

A suitable display of this information is shown in Table 14. All error conditions detected during logic verify will produce conflicting indications in the PLA output table versus the computed table. From Table 14, the presence of A in the PLA output table versus a • in the computed table suggests an illegal concurrency in the device. Conversely, the • in F_0 and F_6 in contrast with an A for the same bits in the computed table indicates inherent concurrencies absent in the device. Knowing all concurrent P-terms and the logic input to the FPLA, we can resort either to array verify or hardcopy reference of the program table and activity map for further diagnostics and isolation.

	ADDRESS	DATA									
	P-term #	P-term Field						F-Set Field			
		I_{15}	I_{14}	------	I_0	F_7	F_6	--------	F_0
Stored Format	Sequential	0 1	1 0	------	1 1	0	1	--------	0
Typical Entry	27	H	L	------	—	A	•	--------	A

a. M/M binary format and typical entry

	F_7	F_6	--------	F_0
Stored Format	0	1	--------	0
Typical Entry	H	L	--------	H

b. ALR binary format and typical entry

Table 12 BINARY ASSIGNMENT OF FPLA PROGRAM TABLE STORED IN MAIN MEMORY, AND ACTIVE LEVEL REGISTER

	I				II					III				
M_n	H	H	L	L	H	H	H	L	L	H	H	L	L	L
P-term	H	—	L	—	L	H	—	L	—	H	—	L	—	H
	M_n contained in P-term				M_n not contained in P-term									

(H◄►L) preclude logical inclusion

Table 13 CRITERIA FOR THE LOGICAL INCLUSION/EXCLUSION OF A MINTERM IN A P-TERM

```
M-TERM'          LOGIC VERIFY          ' ACT LEVL
[FA76]' ..............................  ' HHLHHHLH
' ' ' ' ' '      [INPUT VARIABLE]      ' ' ' ' ' ' ' '
  < P >          1 1 1 1 1 1           [ OUTPUT ]
  < L >          5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0
  < A >          H H H H H L H L L H H H H L H L   A • • A A • A •
[COMPUTED OUTPUT TABLE]                A A • A • • A A
[ERROR] ..............................  ↑   ↑   ↑
[PLA LOGIC OUTPUT] ..................  H H L H H H L H
```

ERROR: *P-TERM CONFLICT*
CONCURRENT P-TERM LIST: *0,1,2,3,4*

Output bits in error indicated by arrow.

Table 14 LOGIC VERIFY OF FPLA, YIELDING DEVICE TRUTH-TABLE FOR LOGIC INPUT FA76 (HEX).

CHAPTER 4
USAGE AND LIMITATIONS

LOGIC COMPRESSION

A concise illustration of the logic compression capabilities of FPLAs is obtained by using an FPLA to implement a small squaring matrix. As shown in Figure 20, this matrix generates a binary output A which is the square of a binary input B, over the range 0 to 15. This table, suitably coded, could be directly programmed in a Signetics' FPLA, without resorting to further manipulations.

However, here it will serve as a tractable case to outline a general systematic procedure involving:

1. *Formulating the logic problem.* This can be done using a set of Boolean equations, or a truth table as in Figure 20.
2. *Product term minimization.* This can be achieved by using any suitable means warranted by the complexity of the problems on hand. Useful tools are Karnaugh maps, Quine/McCloskey method, computerized algorithm, etc.

   In this respect, note that on a bit-slice basis all output functions, except F_0, contain a different number of 1's than their complements (obtained by complementing all output table entries). In general, but not always of course, it is reasonable to expect a function with the least number of 1's to collapse to the fewest number of product terms. Since the FPLA outputs can be programmed active-high or active-low, the designer has the freedom to implement either true or complement functions, with a view towards optimum minimization. But, since a minimum solution is obtained by a simultaneous minimization of all output functions, all combinations of true and complement outputs should be minimized, before a minimum solution can be chosen. Since there are 8 outputs, there are 2^8 output sets involving true and complement functions. These require the solution of 256 minimization problems, which could eventually be done only with the aid of a computer.

   Without such capability, as a best guess one may choose to minimize a table containing a minimum number of 1's, obtained by suitable assignment of output level polarities, and complementing table entries where necessary. In the case of the squaring matrix, the given table already contains the least number of 1's; therefore all FPLA outputs will be assigned active-high polarities to implement all true output functions. Also, for expediency sake, product term minimization will be done on a bit-slice basis by means of the Karnaugh maps in Figure 21.

Note that output F_1 is unconditionally 0. It doesn't contain any products terms, and so it will not be activated whenever

SQUARING MATRIX FOR WHICH THE OUTPUT IS THE SQUARE OF ALL 16-INPUT MINTERMS

INPUTS				OUTPUTS							
I3	I2	I1	I0	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	1	0	0	1
0	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	1	1	0	0	1
0	1	1	0	0	0	1	0	0	1	0	0
0	1	1	1	0	0	1	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	1	0	1	0	0	0	1
1	0	1	0	0	1	1	0	0	1	0	0
1	0	1	1	0	1	1	1	1	0	0	1
1	1	0	0	1	0	0	1	0	0	0	0
1	1	0	1	1	0	1	0	1	0	0	1
1	1	1	0	1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	0	0	0	0	1

B A

FUNCTION

A =B² (BINARY)

0 ≤ B ≤ 15

BIT SLICE

Figure 20

MINIMIZED OUTPUT FUNCTION SET AND P-TERM LIST OF SQUARING MATRIX

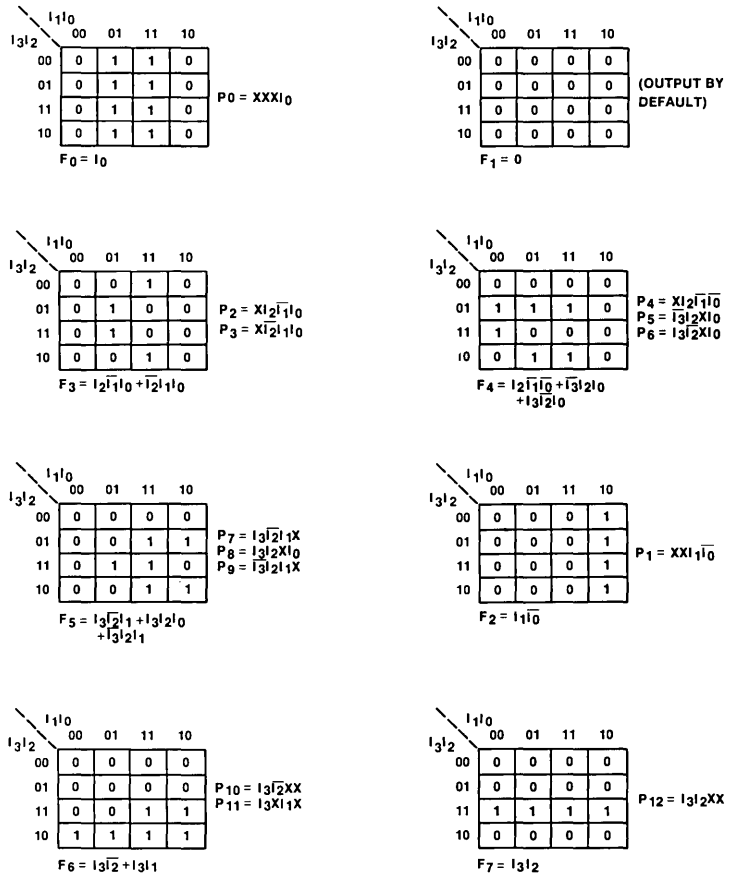


Figure 21

P-terms P_{0-12} are selected. The necessary logic output will be produced by default.

3. *Generating the activity map.*

   The activity map is a useful aid in generating the program table necessary to program an FPLA with the desired logic function. The activity map for the squaring matrix is shown in Figure 22. It lists the minimized output function set involving the ordered P-terms.

4. *Generating the FPLA program table.*

   This is shown in Table 15. The active level polarity of all output functions is entered first. Next, with each available P-term in sequence, the logic input structure of each P-term is assigned and, with the aid of the activity map, an (A) is entered for each activated function, and a period (•) otherwise. This table provides a direct source of programmable entries in the format established for commercially available FPLA programmers.

Pn		ACTIVE LEVEL H H H H H H H H
	INPUTS	OUTPUT FUNCTION
	3 2 1 0	7 6 5 4 3 2 1 0
0	- - - H	• • • • • • • A
1	- - H L	• • • • • A • •
2	- H L H	• • • • A • • •
3	- L H H	• • • • A • • •
4	- H L L	• • • A • • • •
5	L H - H	• • • A • • • •
6	H L - H	• • • A • • • •
7	H L H -	• • A • • • • •
8	H H - H	• • A • • • • •
9	L H H -	• • A • • • • •
10	H L - -	• A • • • • • •
11	H - H -	• A • • • • • •
12	H H - -	A • • • • • • •

Table 15 FPLA PROGRAM TABLE FOR SQUARING MATRIX

By comparing the program table with the original truth table, it can be seen that the squaring matrix has been compressed from 16 minterms to 13 P-terms. Since an FPLA allows direct storage of either 0, 1, or X logic states of input variables, the formal logic compression obtained via minterm to product term minimization of the squaring matrix has been readily translated into hardware.

A representation of the actual logic function programmed in the FPLA in terms of conventional logic symbols is shown in Table 16 with the set-up for verifying the logic function illustrated in Figure 23. Although it shows little resemblance to the original truth table, it must match the function of the squaring matrix. The desired function is obtained by the "Concurrent," "Selective," and "Multiple" addressing

ACTIVITY MAP OF MINIMIZED OUTPUT FUNCTION SET OF SQUARING MATRIX

$F_0 = P_0 + / + / + / + / + / + / + / + / + / + / + / + / + / + /$
$F_1 = / + / + / + / + / + / + / + / + / + / + / + / + / + / + /$
$F_2 = / + P_1 + / + / + / + / + / + / + / + / + / + / + / + /$
$F_3 = / + / + P_2 + P_3 + / + / + / + / + / + / + / + / + / + /$
$F_4 = / + / + / + / + / + P_4 + P_5 + P_6 + / + / + / + / + / + /$
$F_5 = / + / + / + / + / + / + / + / + / + P_7 + P_8 + P_9 + / + / + /$
$F_6 = / + / + / + / + / + / + / + / + / + / + / + / + / + P_{10} + P_{11} + /$
$F_7 = / + / + / + / + / + / + / + / + / + / + / + / + / + / + / + P_{12}$

Pn = Function activated by P-term
/ = Function ignores P-term

Figure 22

Pn	INPUTS				OUTPUTS							
	I3	I2	I1	I0	F7	F6	F5	F4	F3	F2	F1	F0
0	X	X	X	1	0	0	0	0	0	0	0	1
1	X	X	1	0	0	0	0	0	0	1	0	0
2	X	1	0	1	0	0	0	0	1	0	0	0
3	X	0	1	1	0	0	0	0	1	0	0	0
4	X	1	0	0	0	0	0	1	0	0	0	0
5	0	1	X	1	0	0	0	1	0	0	0	0
6	1	0	X	1	0	0	0	1	0	0	0	0
7	1	0	1	X	0	0	1	0	0	0	0	0
8	1	1	X	1	0	0	1	0	0	0	0	0
9	0	1	1	X	0	0	1	0	0	0	0	0
10	1	0	X	X	0	1	0	0	0	0	0	0
11	1	X	1	X	0	1	0	0	0	0	0	0
12	1	1	X	X	1	0	0	0	0	0	0	0

Table 16 CONVENTIONAL LOGIC REPRESENTATION OF PROGRAM TABLE IN THE FPLA

SET-UP FOR VERIFYING THE LOGIC FUNCTION OF THE FPLA

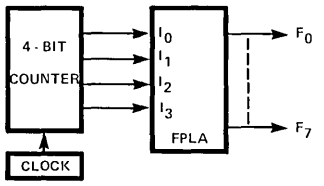


Figure 23

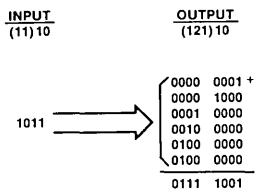
modes characteristic of FPLAs. These can be observed by listing the composite FPLA output while executing an exhausting logical scan at its inputs, as shown in Table 17.

By viewing each row of the program table as an FPLA word selected by the corresponding P-term address, concurrent addressing is shown by the simultaneous selection of words P_0, P_3, P_6, P_7, P_{10} and P_{11} which occurs with a binary 1011 input to the FPLA (Figure 24a). Similarly, multiple addressing is readily apparent by observing that word P_0 is selected by 8 different input combinations, in a manner reminiscent of virtual memory storage (Figure 24b).

Mn	INPUTS				CONCURRENT P-TERMS
	I3	I2	I1	I0	
0	0	0	0	0	None (Default state term)
1	0	0	0	1	P0
2	0	0	1	0	P1
3	0	0	1	1	P0,P3
4	0	1	0	0	P4
5	0	1	0	1	P0,P2,P5
6	0	1	1	0	P1,P9
7	0	1	1	1	P0,P5,P9
8	1	0	0	0	P10
9	1	0	0	1	P0,P6,P10
10	1	0	1	0	P1,P7,P10,P11
11	1	0	1	1	P0,P3,P6,P7,P10,P11
12	1	1	0	0	P4,P12
13	1	1	0	1	P0,P2,P8,P12
14	1	1	1	0	P1,P11,P12
15	1	1	1	1	P0,P8,P11,P12

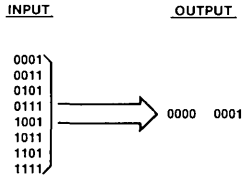
Table 17 SIMULTANEOUSLY SELECTED P-TERMS OBTAINED BY EXERCISING THE FPLA WITH A MINTERM GENERATOR

ADDRESS SELECTION



Internal OR performed by the FPLA.

a. Concurrent address selection by Mn = $(11)_{10}$.



b. Multiple address selection of P_0.

Figure 24

Selective addressing occurs when minterm "0" is presented at the FPLA input, but does not activate any of the programmed P-terms 0 thru 12, and thus none of the output functions.

At this point it is worth noting that the above implementation is not unique, since the program table is not unique. This results from the individual, rather than the *simultaneous* minimization of the output function set. For example:

$$F_4 = \bar{I}_3 I_2 X I_0 + I_3 \bar{I}_2 X I_0 + \bar{I}_3 I_2 \bar{I}_1 X + I_3 I_2 \bar{I}_1 \bar{I}_0$$
$$F_5 = \bar{I}_3 I_2 I_1 X + I_3 \bar{I}_2 I_1 X + I_3 I_2 X I_0$$
$$F_6 = I_3 \bar{I}_2 X X + I_3 I_2 I_1 X$$

is an equivalent form for $F_{4,5,6}$. This choice of expression, although it introduces an additional P-term in F_4, eliminates P_{12} for realizing F_7, since:

$$F_7 = \underbrace{I_3 I_2 X X}_{\text{contained in } F_{4,5,6}} = I_3 I_2 I_1 X + I_3 I_2 X I_0 + I_3 I_2 \bar{I}_1 \bar{I}_0$$

In this case no net reduction in number of P-terms is obtained. However, the method is at the root of the search for a minimum set of P-terms which will implement the desired logic function. Indeed, the reduction of a set of logic functions of several variables to a minimum set of prime implicants (P-terms) requires a simultaneous minimization process for which suitable algorithms have already been developed.

Signetics has successfully translated such an algorithm in an efficient software program for execution on an IBM 370/155 computer system.

ASYNCHRONOUS SEQUENTIAL LOGIC

FPLAs can be very effective tools in streamlining the design of asynchronous sequential networks by reducing package count, easing modification, and by providing more uniform logic delays which generally reduce, but do not entirely eliminate the incidence of logic hazards due to oscillations and critical races. And, when identified, they may be easier to eliminate by redundant usage of logic which is "already there," without additional hardware penalties.

The following example illustrates the general procedure:

> **Problem:** Design a network to provide an output Z = 1 when both inputs X and Y are 1, but only when X goes to 1 before Y. Control inputs X and Y can change only one at a time; Z should remain 1 as long as Y = 1.

As a first step, the primitive flow table is generated as shown in Table 18. By definition, each row of this table can contain only one stable state. Dashes, denoting don't care, are entered in each cell mapping a forbidden transition of the control inputs from a stable state (double simultaneous transitions). Since these, by definition, cannot occur, the dashes can be used to simplify the specification of the FPLA P-terms.

XY	00	01	11	10	Z
a	①	5	—	2	0
b	1	—	3	②	0
c	—	4	③	2	1
d	1	④	3	—	1
e	1	⑤	6	—	0
f	—	5	⑥	7	0
g	1	—	6	⑦	0

Circled entries are stable states. Uncircled entries signify unstable states.

Table 18 PRIMITIVE FLOW TABLE

The 7 rows in Table 18, corresponding to 7 stable states, can be reduced to the 3 shown in Table 19, by merging all rows with identical states in each column, independent of output state associated with each row. In merging rows, stable state entries override unstable states. Also regardless of merger, Z output values in the final output matrix are dictated by the stable states (circled entries) in the primitive flow table. The 3 rows of the merged flow table need at least 2 *secondary* variables to assign a total of $2^2 = 4$ secondary states involving 2 feedback loops. This leaves a spare secondary state (row) to be used for logic reduction, or resolution of critical races. These can in general be minimized by assigning a grey code to the feedback loops, which results in secondary state assignments involving a single variable change for transitions between rows.

XY	00	01	11	10	
R1	①	5	—	2	(a)
R2	1	④	③	②	(b,c,d)
R3	1	⑤	⑥	⑦	(e,f,g)

Table 19 MERGED FLOW TABLE FOR MINIMIZING FEEDBACK LOOPS

A primitive flow matrix suitable secondary state assignments is shown in Table 20. The empty cells of $R_0 = 00$ have been assigned suitable unstable states to simplify hardware implementation. These are explicitly indicated in the final flow matrix of Table 21. From the flow matrix, the excitation matrix for the feedback outputs J-K of the network is derived in Table 22 by ensuring that all stable states in each row are assigned the corresponding j-k input values for that row.

jk \ XY	00	01	11	10	
00					
01	1	④	③	②	R_1
11	①	5	—	2	R_2
10	1	⑤	⑥	⑦	R_3

Table 20 PRIMITIVE FLOW MATRIX WITH FEEDBACK INPUT VARIABLES j-k, AND CORRESPONDING SECONDARY ASSIGNMENTS

jk \ XY	00	01	11	10	
00	1	4	3	7	R_0
01	1	④	③	②	R_1
11	①	5	6	2	R_2
10	1	⑤	⑥	⑦	R_3

Table 21 FINAL FLOW MATRIX WITH OPTIMUM UNSTABLE STATES ASSIGNED TO SPARE ROW R_0

jk \ XY	00	01	11	10	
00	11	01	01	10	R_0
01	11	⑰	⑰	⑰	R_1
11	⑪	10	10	01	R_2
10	11	⑩	⑩	⑩	R_3

(JK)

Table 22 EXCITATION MATRIX OF FEEDBACK OUTPUT VARIABLES J-K (MAP ENTRIES)

An analysis of all transitions between rows of this matrix, as mapped in the transition matrix of Table 23, verifies that the choice of secondary state assignments does not produce logic hazards. The minimized logic equation set of the feedback outputs is obtained via separate Karnaugh maps for J and K, as in Table 24. The choice of a suitable set of P-terms for the J-K feedback variables must take into account again the possibility of logic hazards. As the control variables X and Y change, horizontal movements among the stable states in the transition matrix result in alternate deselection and selection of FPLA P-terms. Normal internal delay differences in this sequence may cause momentary deselection of all P-terms, deactivating all FPLA outputs. This is seen as negative or positive glitches (depending on active level polarity) on the FPLA outputs before they stabilize to the correct logic level. Glitches on feedback outputs may cause the circuit to settle in the wrong final state from any given initial stable state.

To remove these additional hazards, output spiking can be eliminated by choosing P-terms for J-K such that all legal intercolumn transitions from each stable state ensure "holding" of the J-K outputs. If necessary, redundant P-terms must be used to cover all possible cases ("minimal cover" technique). For example, from Tables 21 and 24 it is apparent that while in stable state 1, held by J-K feedback outputs via P-term $(\bar{X}\bar{Y})$, if input Y goes to "1," stable state ⑤ must be reached. But, if P-term $(\bar{X}\bar{Y})$ *deselects* before P-term (Yj) *selects,* the network may jump to R_0, state 4, and settle next in R_1, stable state ④. However, by including P-term $(\bar{X}j)$, the (j) input can be held steady while P-term $(\bar{X}\bar{Y})$ deselects.

It follows that, as a general rule, enough redundant P-terms should be used to prevent spurious transitions of the feedback variables for all horizontal transitions from stable states in each row. This requires modification of the J-K equations as follows:

$$J = \bar{X}\bar{Y} + \bar{X}j + Yj + X\bar{Y}\bar{k} + j\bar{k}$$
$$K = \bar{X}\bar{Y} + Y\bar{j} + \bar{j}k + X\bar{Y}k$$

Since the required logic is already available in the FPLA, no hardware penalties are incurred. As a final step, the network output Z must be realized. It is obtained from the output matrix of Table 25. By contrast, a discrete logic version of the same network with NAND gates would require 6 IC packages.

jk \ XY	00	01	11	10	
00	0	1	1	0	R_0
01	0	1	1	0	R_1
11	0	0	0	0	R_2
10	0	0	0	0	R_3

(Z)

Table 25 KARNAUGH MAP OF OUTPUT Z = Yj̄

jk \ XY	00	01	11	10	
00					R_0
01					R_1
11					R_2
10					R_3

Table 23 TRANSITION MATRIX MAPPING SECONDARY TRANSITIONS

jk \ XY	00	01	11	10
00	1	0	0	1
01	1	0	0	0
11	1	1	1	0
10	1	1	1	1

a. $J = \bar{X}\bar{Y} + Yj + X\bar{Y}\bar{k}$

jk \ XY	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	1	0	0	1
10	1	0	0	0

b. $K = \bar{X}\bar{Y} + Y\bar{j} + X\bar{Y}k$

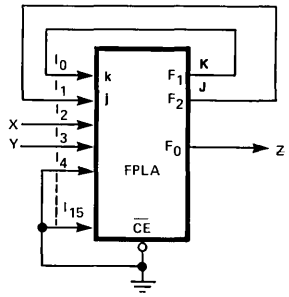
Table 24 KARNAUGH MAPS FOR DERIVING P-TERMS AND LOGIC EQUATIONS DEFINING THE FEEDBACK VARIABLES

FPLA PROGRAM TABLE AND CONNECTION FOR DESIRED NETWORK FUNCTION

NO.	PRODUCT TERM: INPUT VARIABLE (I_m) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	ACTIVE LEVEL: OUTPUT FUNCTION 7 6 5 4 3 2 1 0
0	L L — —	H H H A A •
1	L — H —	A • •
2	— H H —	A • •
3	H L — L	A • •
4	— — H L	A • •
5	— H L —	• A A
6	— — L H	• A •
7	H L — H	• A •
8	— L — H	• A •
9		
10		
44		
45		
46		
47		
I/O Assignment →	X Y j k	J K Z

Unused locations have been left blank for clarity.

a. Program Table.



Unused FPLA locations must be programmed in accordance with previously established criteria.

b. Circuit Connection

Figure 25

SYNCHRONOUS SEQUENTIAL LOGIC

When speed is not a problem, the difficulties posed by potential logic hazards in asynchronous logic designs can be altogether avoided by resorting to pulsed sequential networks. In these networks all secondary variables are allowed to settle following control input(s) transitions, before allowing the circuit to be locked in the next stable state. This is accomplished by periodic clocking of storage elements (flip-flops) used to hold the current state of the network in preparation for a new input condition. Essentially this is equivalent to solving a combinational problem at different times for determining the desired output, as well as which flip-flop control inputs, and under what conditions they must be enabled. Therefore, every network of this type can be generalized as containing 2 conceptually distinct memory and logic blocks, as shown in Figure 26.

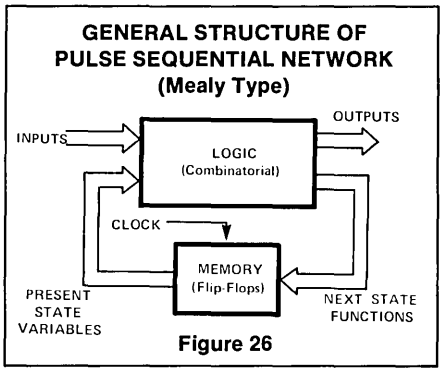


Figure 26

It is with respect to the latter block that FPLAs again provide an opportunity to streamline design, as well as decrease pressure on the designer for an all out effort (beyond simple Karnaugh maps) to minimize his logic for reducing package count.

The following example illustrates the general design method for typical applications.

Problem: Design a network for detecting a decimal 6 or 8 in a serial 4-bit BCD word (MSB first). Whenever a 6 or 8 occurs, output Z = "1."

In line with Figure 26 above, the desired network using an FPLA and D-flops to implement the respective blocks will have the organization shown in Figure 27. Starting with an initial reset state, designated ⓐ , a primitive stable state diagram is developed as shown in Figure 28, to take into account all valid and invalid input sequences.

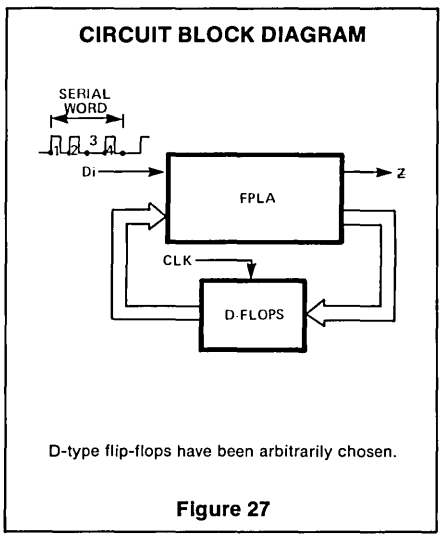


Figure 27

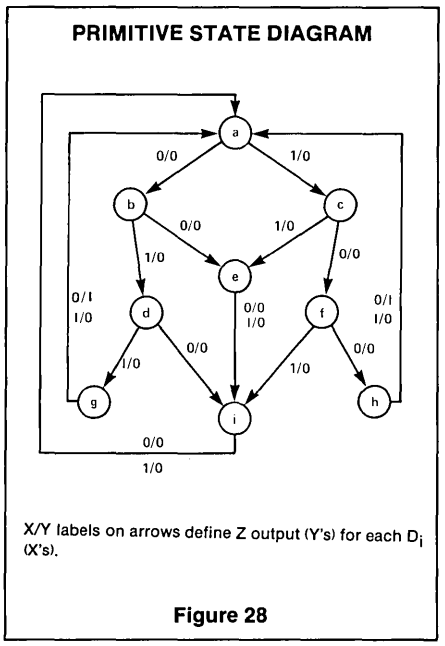


Figure 28

At this stage, an intuitive approach is perhaps the best recourse in developing a concise diagram excluding most redundant or duplicate states. These can be further eliminated by analysis of the primitive state table in Table 26 for combining all states which have *identical next states and outputs.* This step is necessary for minimizing secondary assignments to reduce the number of flip-flops required.

Since (m) stable states give rise to (n) flip-flops, where $2^n \geq m$, 3 flip-flops are necessary to define 8 secondary assignments corresponding to each state in the reduced table. These assignments in terms of D-flops Q_A, Q_B and Q_C are summarized in the transition table of Table 27. As the network moves through its stable states, the flip-flop transitions mapped in Table 27 must be ensured by suitable programming of the D input of each flip-flop, designated

PRESENT STATE	INPUT (D_i) "0"	"1"	"0"	"1"
a	b	c	0	0
b	e	d	0	0
c	f	e	0	0
d	i	g	0	0
e	i	i	0	0
f	h	i	0	0
g	a	a	1	0
h	a	a	1	0
i	a	a	0	0
	Next State		Z	

a. Primitive state table

PRESENT STATE	INPUT (D_i) "0"	"1"	"0"	"1"
a	b	c	0	0
b	e	d	0	0
c	f	e	0	0
d	i	g	0	0
e	i	i	0	0
f	g	i	0	0
g	a	a	1	0
i	a	a	0	0
	Next State		Z	

b. Reduced state table

Table 26 FLIP-FLOP MINIMIZATION BY COMBINING STATES g AND h IN THE PRIMITIVE STATE TABLE

PRESENT STATE	ASSIGNMENT Q_A	Q_B	Q_C	INPUT (D_i) "0"			"1"			"0"	"1"
a	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	0	0	0	1	1	0	0
c	0	1	0	1	0	1	1	0	0	0	0
d	0	1	1	1	1	1	1	1	0	0	0
e	1	0	0	1	1	1	1	1	1	0	0
f	1	0	1	1	1	0	1	1	1	0	0
g	1	1	0	0	0	0	0	0	0	1	0
i	1	1	1	0	0	0	0	0	0	0	0
				Next State						Z	

Table 27 STATE TRANSITION TABLE AND "NEXT" OUTPUT, WITH SECONDARY ASSIGNMENTS FOR D-FLOPS $Q_{A\text{-}C}$ FOR EACH STABLE STATE

D_A, D_B and D_C respectively. This is accomplished by using the transition table in conjunction with the excitation table for a D-flop (Table 28), to generate control matrices from which control equations for D_A, D_B and D_C are derived. As shown in Figure 29, control matrices are really Karnaugh maps in which 1/0 cell entries refer to the logic state of each D input in terms of the input Data D_i and flip-flop outputs (present state variables) Q_{A-C}, which are fed back in the network.

Q	→ Q^{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Table 28 EXCITATION TABLE FOR D-TYPE FLIP-FLOP

The logic equations of Figure 29 are readily programmed in an FPLA with the program table shown in Table 29. The final network is obtained as in Figure 30.

FINAL CIRCUIT CONNECTION, UTILIZING TRI-STATE FPLA AND EDGE-TRIGGER D-FLOPS

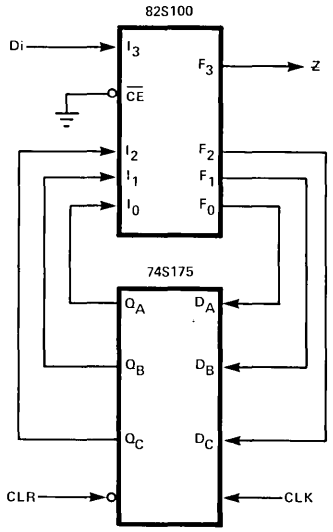


Figure 30

CONTROL MATRICES AND LOGIC EQUATIONS FOR FLIP-FLOP INPUTS D_{A-C} AND OUTPUT Z

$Q_A Q_B$ \ $Q_C D_i$	00	01	11	10
00	0	0	0	1
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

a. $D_A = \overline{Q}_A Q_B + Q_A \overline{Q}_B + \overline{Q}_A Q_C \overline{D}_i$

$Q_A Q_B$ \ $Q_C D_i$	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	0	0	0	0
10	1	1	1	1

b. $D_B = \overline{Q}_A \overline{Q}_B D_i + \overline{Q}_A Q_B Q_C + Q_A \overline{Q}_B$

$Q_A Q_B$ \ $Q_C D_i$	00	01	11	10
00	1	0	1	0
01	1	0	0	1
11	0	0	0	0
10	1	1	1	0

c. $D_C = \overline{Q}_A \overline{Q}_C \overline{D}_i + Q_A \overline{Q}_B \overline{Q}_C + \overline{Q}_B Q_C D_i + \overline{Q}_A Q_B \overline{D}_i$

$Q_A Q_B$ \ $Q_C D_i$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	0	0	0
10	0	0	0	0

d. $Z = Q_A Q_B \overline{Q}_C \overline{D}_i$

Figure 29

NO.	PRODUCT TERM: INPUT VARIABLE (I_m) 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	ACTIVE LEVEL: OUTPUT FUNCTION H H H H 7 6 5 4 3 2 1 0
0	— — H L	• • • A
1	— — L H	• • • A
2	L H — L	• • • A
3	H — L L	• • A •
4	— H L L	• • A •
5	— — L H	• • A •
6	L L — L	• A • •
7	— L L H	• A • •
8	H H L —	• A • •
9	L — H L	• A • •
10	L L H H	A • • •
11		
12		
43		
44		
45		
46		
47		
I/O Assignment →	Di QC QB QA	Z DC DB DA

Unused locations have been left blank for clarity.

Table 29 FPLA PROGRAM TABLE

DEALING WITH DEVICE LIMITATIONS

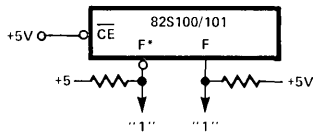
In some applications, a single FPLA cannot accommodate the full program table because it commands greater resources than the finite number of inputs, outputs, and P-terms available. In many cases this can only be overcome by resorting to design intuition and ingenuity in place of complex data manipulations which tend to obscure the problem on hand, and may render troubleshooting difficult.

Borderline cases can usually be resolved by judicious inspection of the program table to discover ways to further compression. Nevertheless, to increase design flexibility in these situations, Signetics' FPLAs are the only ones which feature a Chip Enable input which can be used for input and P-term expansions, preconditional input decoding, and output inhibit.

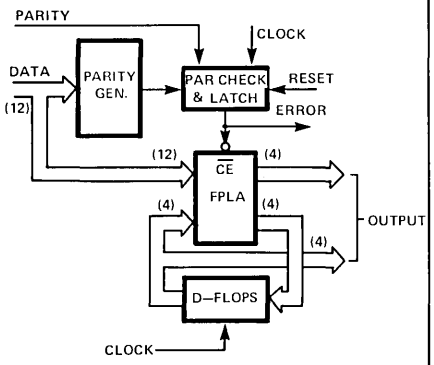
The output inhibit function of $\overline{CE}$ not only permits utilization of the tri-state device in bused organizations, but also provides a means to force all outputs to a unique logic state, regardless of their programmed polarity, without sacrificing FPLA inputs or entailing additional hardware.

This feature is essential in a number of applications involving system initialization from a known state, exit to "idle" following sequence error, synchronous clocking, etc. For example, in the typical sequencer of Figure 31b, if an input error occurs parity fails, forcing all outputs to logic "1" ("idle" state, by user definition).

CHIP ENABLE CONTROL



a. With $\overline{CE}$ = 1, both True (F) and Complement (F*) outputs are forced to logic "1."



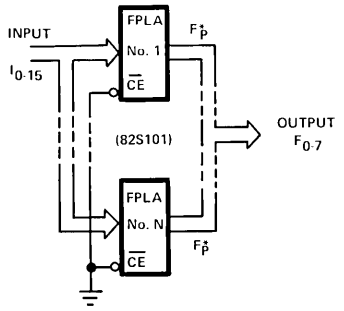
b. Sequential Controller forced into "idle" state by input parity error.

Figure 31

PRODUCT TERM EXPANSION

Expansion of P-terms involving up to 16 input variables is easily accomplished with open collector devices, as shown in Figure 32. It is only necessary to parallel respectively all inputs and outputs of several devices, operated with $\overline{CE}$ at ground (unless needed as additional control function). The composite logic output of the network is determined by P-terms activated in one or several FPLAs simultaneously.

P-TERM EXPANSION WITH OPEN COLLECTOR FPLAs, INVOLVING UP TO 16-INPUT VARIABLES



All outputs must be programmed active-low (F_P^*) to realize the wire-AND function. The total number pf P-terms available is 48N.

Figure 32

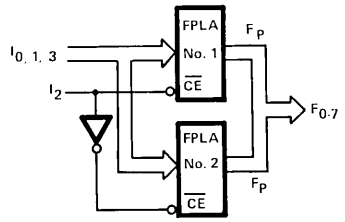
When using tri-state devices (82S100), P-term expansion cannot be readily achieved in the same way because of logic conflicts ensuing from the active pull-up outputs of FPLAs sharing the same output bus. To ensure enabling only one device at a time, P-term expansion *must* involve the $\overline{CE}$ input.

In most applications requiring more than 48 P-terms it should be a relatively simple task to partition the program table in 2 or more subtables, each containing less than 48 P-terms which in turn can be fitted in separate FPLAs. This partitioning is achieved by segmenting the original table about the 1's and 0's of suitable input variables. Since all P-terms P_n which contain a segmenting variable as don't care give rise to 2 P-terms P_{na} and P_{nb}, it is best to segment a program table about variables with the fewest don't care states.

The logic sources of segmenting variables are removed from the FPLA input field and made to drive instead the $\overline{CE}$ input of the required FPLAs, after proper decoding. As an example, if one were restricted to use tri-state FPLAs with only 10 P-terms each to incorporate the program table of Table 16 (page 40), a segmentation of this table about input I_2 yields the subtables shown in Table 30.

Each subtable contains less than 10 P-terms, and will fit in separate FPLAs which are operated in parallel and controlled by I_2 via their $\overline{CE}$ input, as shown in Figure 33.

SQUARING MATRIX FUNCTION RESIDENT IN 2 TRI-STATE FPLAs PROGRAMMED RESPECTIVELY WITH SUBTABLES A AND B.



Note that the inhibit function of $\overline{CE}$, renders unnecessary to program the outputs active-low.

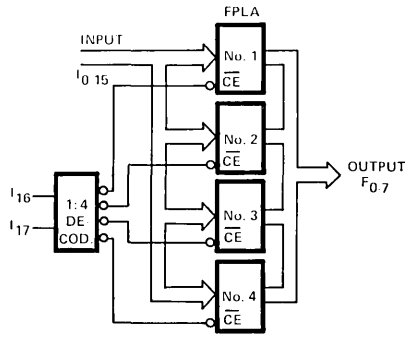
Figure 33

The feasibility of this procedure is strongly dependent on the contents of the original program table, and in some degenerate cases (too few or no 0's at all in the input field of the program table) it may not work. Also, note that in general the final number of P-terms used may increase due to expansion of input don't cares. However, this is preferable to no solution at all.

INPUT VARIABLE EXPANSION

This is the most difficult and cumbersome task with FPLAs. When the program table involves more than 16 inputs, the above partitioning technique by subtables segmented about any suitable variables will work as well with tri-state or open collector devices. This technique is shown applied to 18 input variables in Figure 34. In this case several devices are necessary, even though not all FPLA P-terms are used.

DIRECT MANIPULATION OF 18-INPUT VARIABLES USING $\overline{CE}$ WITH EITHER 82S100 OR 82S101 FPLAs



Note that here it is not necessary to program all output functions active-low (F_P^*) because of the disabling function of $\overline{CE}$.

Figure 34

Note that the expansion capability provided by $\overline{CE}$ input limits the total number of FPLAs required to 2^n, where (n) is the number of segmenting variables. Without $\overline{CE}$, the total number of FPLAs required would be 2^{n+1}.

With more than 20 or so inputs this approach may become too costly, and thus it may make more sense to review the program table in conjunction with the problem at hand for ways to multiplex the FPLA inputs. This also involves a sort of segmentation of the program table for grouping P-terms about input variables which are mutually exclusive.

The principle is illustrated in Table 31 when dealing with only 17 input variables and 5 P-terms, for simplicity. The original program table in (a) has been segmented about the 0's and 1's of variable I_n, and the P-terms regrouped as in (b). Note that it was necessary to create new P-terms 4a and 4b to expand the don't care for I_n in P-term 4. Here, when I_n = 0, the outputs are independent of I_{n-1}, and when I_n = 1 the outputs are independent of I_{n+1}. These inputs can be multiplexed in an FPLA with I_n as the steering condition, as shown in Figure 35. The FPLA program table contains upper P-terms with I_{n-1} variable

P-TERMS		INPUTS				OUTPUTS							
P_n	$\overline{P_n}$	I_3	I_2	I_1	I_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0a	0	X	0	X	1	0	0	0	0	0	0	0	1
1a	1	X	0	1	0	0	0	0	0	0	1	0	0
3	2	X	0	1	1	0	0	0	0	1	0	0	0
6	3	1	0	X	1	0	0	0	1	0	0	0	0
7	4	1	0	1	X	0	0	1	0	0	0	0	0
10	5	1	0	X	X	0	1	0	0	0	0	0	0
11a	6	1	0	1	X	0	1	0	0	0	0	0	0

a. Subtable A to be stored in FPLA #1 with I_2 removed. P_{11a} can be eliminated since it is "covered" by P_{10}.

P-TERMS		INPUTS				OUTPUTS							
P_n	$\overline{P_n}$	I_3	I_2	I_1	I_0	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
0b	0	X	1	X	1	0	0	0	0	0	0	0	1
1b	1	X	1	1	0	0	0	0	0	0	1	0	0
2	2	X	1	0	1	0	0	0	0	1	0	0	0
4	3	X	1	0	0	0	0	0	1	0	0	0	0
5	4	0	1	X	1	0	0	0	1	0	0	0	0
8	5	1	1	X	1	0	0	1	0	0	0	0	0
9	6	0	1	1	X	0	0	1	0	0	0	0	0
11b	7	1	1	1	X	0	1	0	0	0	0	0	0
12	8	1	1	X	X	1	0	0	0	0	0	0	0

b. Subtable B to be stored in FPLA # 2, with I_2 also removed.

Table 30 SUBTABLES OF SQUARING MATRIX

P_n	I_{16}	...	I_{n+1}	I_n	I_{n-1}	...	I_0	F_x	F_y
0	0	...	X	1	0	...	1	1	0
1	1	...	1	0	X	...	1	1	1
2	X	...	0	0	X	...	0	0	1
3	0	...	X	1	X	...	X	1	0
4	1	...	X	X	X	...	0	0	1
5	1	...	X	1	1	...	0	1	0

a. Initial Program Table involving 17 input variables, which cannot be directly examined by a single FPLA.

	P_n	I_{16}	...	I_{n+1}	I_n	I_{n-1}	...	I_0	F_x	F_y
UPPER	1	1	...	1	0 →	X	...	1	1	1
	2	X	...	0	0 →	X	...	0	0	1
	4a	1	...	X	0 →	X	...	0	0	1
LOWER	0	0	...	X	← 1	0	...	1	1	0
	3	0	...	X	← 1	X	...	X	1	0
	4b	1	...	X	← 1	X	...	0	0	1
	5	1	...	X	← 1	1	...	0	1	0

b. Variable I_{n+1} and I_{n-1} can be multiplexed on a single FPLA input because they are mutually exclusive "about" I_n (selector).

Table 31 PROGRAM TABLE

removed, and lower P-terms with I_{n+1} variable removed.

When this technique fails too, it may still be possible to factor out of the logic equation of each FPLA output common expressions involving the variables in excess. These can be externally combined with simple gating, or another FPLA, into first level P-terms generating *dummy* variables to be applied to a second-level FPLA.

OUTPUT EXPANSION

If an application requires more than 8 outputs, several FPLAs can be used with parallel inputs and separate outputs. In other cases, it may be more cost effective

MULTIPLEXING OF INPUTS I_{n+1} AND I_{n-1} WITH SELECTOR INPUT I_n ALLOWS 17 INPUTS TO BE HANDLED WITH ONE 16-INPUT FPLA

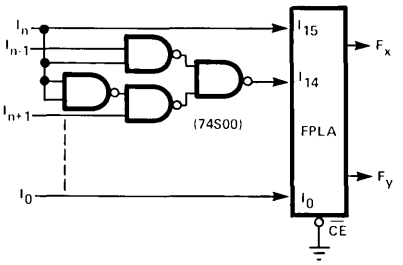
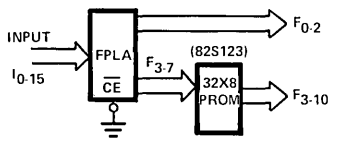
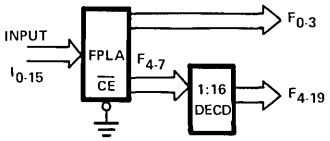


Figure 35

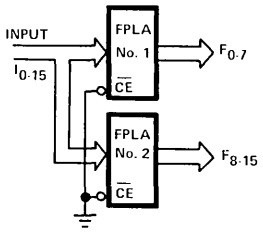
to encode the Output Table stored in a single device and then unscramble the desired output states via a 32X8 PROM or 1/N decoder as required. Both methods are shown in Figure 36. Some caution, however, is required in formulating the FPLA program table to ensure that either single or concurrent P-term selections will ultimately point to a unique decoder or PROM address.

OUTPUT EXPANSION





a. Output expansion by decoding outputs previously encoded in the FPLA Program Table.



b. Output expansion by utilizing additional FPLAs.

Figure 36

CHAPTER 5
APPLICATIONS

The recent surge in design activity involving microprocessors and microprogramming techniques reflects the growing trend to replace hardwired logic with microcode for gaining system flexibility at lower cost. In this respect, designers have come to rely on ever larger and denser PROMs to fit the demands of their applications, and today PROMs as large as 8K bits, organized as 1KX8 or 2KX4, are readily available. However, a PROM solution in general forces the user to allocate storage for all possible logic combinations of the input variables, whether needed or not. As a result, when dealing with the type of problem requiring the manipulation of more than about 10 logic input variables (or Addresses), several IC packages are usually necessary. This quickly renders a PROM solution marginal at best, in terms of speed, power, and cost, and in most cases impractical. Fortunately, many combinational and sequential logic designs involve logic functions which are True for only a small subset of the total logic states generated by the controlling variables. Typical examples are the 96 graphic characters, out of 2^{12} coding states, of a 12-bit Hollerith code; or the 50 (or so) subroutine-start addresses, from a total of 2^{16}, in a typical 16-bit microprogrammed machine. It is here that we step in the basic domain of Field Programmable Logic Arrays which, when viewed as associative memories, exhibit selective, concurrent, and multiple addressing modes that enable compressing a set of logic functions to the minimum required states, at substantial savings in hardware.

The areas of application in which FPLAs provide a more efficient design alternative span the whole spectrum of logic design. Many applications based on mask-programmable devices have been well documented [1,2,3,4,5]. However, since FPLAs can be readily programmed in the field by the user, they are more economical and easier to use, and should find their way quickly in a wider variety of design situations.

The typical design applications described in the following pages emphasize the conceptual aspects of FPLA usage, in order to focus the reader on the basic roles of FPLAs in logic design, and ease the transfers of these basic ideas to a variety of other practical applications.

An estimate of the savings and design advantages obtainable by using FPLAs can be gleaned by examining the recent experience of a Signetics' customer who used FPLAs in the design of an automatic landing system for aircrafts. By using a different design approach, he was able to replace 49 IC packages with 1 FPLA. The tradeoff in both design alternatives is shown in Table 32. In the discrete approach, $1 is about what it takes today to place one IC on a PC board.

QUANTITY	TYPE
12	7400 (Quad 2-NAND)
9	7402 (Quad 2-NOR)
8	7427 (Triple 3-NOR)
5	7442 (BCD/DEC Decoder)
2	74175 (Quad D-FLOP)
4	7404 (Hex Inverter)
2	7430 (8-Input NAND)
7	7408 (Quad 2-AND)

COMPARISON		
	Random Logic	FPLA
ICs	49	1
Power	3.3W	0.6W
Speed	65NS	50NS
Cost	$49	$11.50
Pins	700	28
Space	50in²	2in²

One FPLA replaces 49 ICs at less than 1/4 the cost.

Table 32 THE ECONOMICS OF LOGIC REPLACEMENT WITH FPLAs

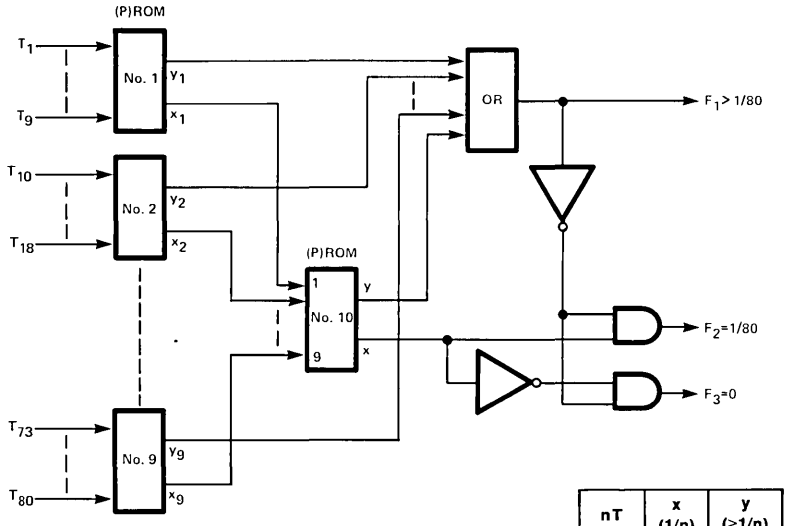
FAULT MONITOR NETWORKS

The dramatic savings in hardware which can be obtained by using FPLAs to manipulate a large number of logic variables is readily apparent when building 1/N detectors, as a special case of m/N decoder networks. These are useful in a variety of applications in computers, data communications, and fault monitor systems. For example, in a data multiplexing system it is not uncommon to find 80 or more channels time-division multiplexed onto a single transmission line. If a fault occurs in the multiplexer-control network, multiple or no connections on the line give rise to invalid transmission. These type of faults can be readily detected by using a 1/80 detector to monitor the normal selection status of only one multiplexer channel at a time. A 1/N detector could be implemented by using logic gates. Excluding inversion and ORing of partial results, the number of gates required is given by the number of logic states to be detected. For 80 status monitor terminals (1 for each data channel):

$$\#\ \text{of Gates}_{(AND)} = N!(N-1)! = 80$$

This approach, when complicated by the fact that each gate also requires 80 inputs, becomes quickly impractical. A more practical alternative involves partitioning the number of terminals in equal subsets which are applied to PROMs whose truth tables yield outputs x = 1/n and y > 1/n **[6]**. Each PROM is used as a basic building block in a cascaded array, to implement a general algorithm for detecting 0,1 or more True states (logic "1" = channel selected) of n variables. This is shown in Figure 37 for N = 80, using a 512X8 PROM organization. It requires 10 PROMs, plus some gating circuitry for status indication.

1/80 DETECTOR WITH (P)ROMs



nT	x (1/n)	y (>1/n)
0	0	0
1	1	0
>1	0	1

(ALGORITHM)

Figure 37

However, with FPLAs a more efficient solution is possible as shown in Figure 38. It requires only 6 devices.

1/80 DETECTOR WITH FPLAs

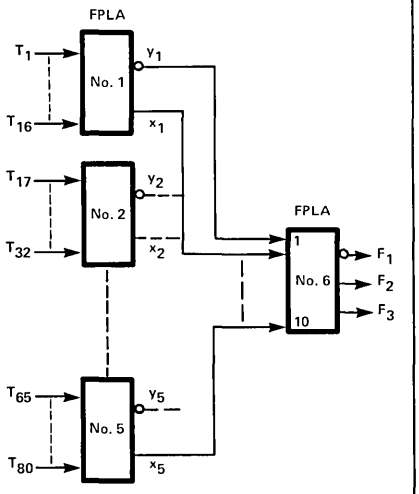


Figure 38

Since each FPLA can examine 16 terminals, 5 are sufficient to service all 80 terminals. Each FPLA utilizes 17 P-terms to detect the presence of zero, 1/16, or > 1/16 via outputs (x) and (y) as defined in the program table of Figure 33a. An additional FPLA is necessary to examine a total of 10 partial x and y results from the first level devices, and to give final indication of the number of selected terminals. The program for the last FPLA is contained in Table 33b.

INPUT																OUTPUT	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	F_1 x	F_2^* y
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

a. Program Table stored in FPLAs 1 through 5

INPUT										OUTPUT		
x = 1/N					y > 1/N					> 1/N	= 1/N	0
10	9	8	7	6	5	4	3	2	1	F_1^*	F_2	F_3
0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	1	0

b. Program Table stored in FPLA 6

Starred (*) outputs are programmed active-low. Conventional logic symbols are used for clarity.

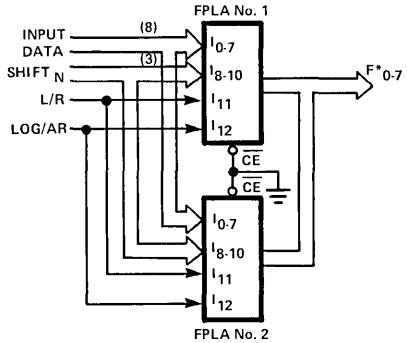
Table 33 PROGRAM TABLES OF FPLAs USED IN 1/80 DETECTOR

FAST MULTIBIT SHIFTER

Computer performance can be greatly increased by incorporating hardware capabilities to execute fast multibit shifts. This results in a considerable reduction in execution time for algorithms that involve a large number of arithmetic, logic, or circular shifts, such as divide/multiply, floating point operations, etc.

A multibit shifter implemented with 2 FPLAs is shown in Figure 39a. It provides arithmetic or logic shift of an 8-bit byte either left or right up to 7 places within 1

FAST MULTIBIT SHIFTER WITH FPLAs



a. Shifter shifts Left/Right, Arithmetic or Logic up to 7 places in 35ns.

I_m	LEFT SHIFT S_{LN}: I_{11} = 0								RIGHT SHIFT S_{RN}: I_{11} = 1								
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
0	0	1	2	3	4	5	6	7	0								
1	1	2	3	4	5	6	7		1	0							
2	2	3	4	5	6	7			2	1	0						
3	3	4	5	6	7				3	2	1	0					Logic/Arithmetic
4	4	5	6	7					4	3	2	1	0				I_{12} = X
5	5	6	7						5	4	3	2	1	0			
6	6	7							6	5	4	3	2	1	0		
7	7								7	6	5	4	3	2	1	0	
7	7	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	
7										7	6	5	4	3	2	2	
7											7	6	5	4	3	3	Arithmetic
7												7	6	5	4	4	I_{12} = 1
7													7	6	5	5	
7														7	6	6	

b. Logic equation set of shifter to be programmed in FPLAs.

Figure 39

clock cycle. Two FPLAs are necessary, for a total of 71 P-terms.

The program table to be stored in the devices is derived from the set of output equations tabulated in Figure 39b. The table entries represent output functions F_0 through F_7, which are true (1) at coordinate points ($I_m \bullet S_{LN}$) or ($I_m \bullet S_{RN}$). These are respectively the *ordered input data bits*, and the number of right or left shifts. A further subdivision of the table is given by I_{12} for arithmetic or logic shifts.

For example, for a logic shift of the input data, the P-terms which must be programmed in the FPLAs for say output bit 5 are:

$$F_5 = I_5S_{RO} + I_6S_{R1} + I_7S_{R2} + I_5S_{LO} + I_4S_{L1} + I_3S_{L2} + I_2S_{L3} + I_1S_{L4} + I_0S_{L5}$$

The P-terms in the equation are in turn converted in program table format, typically as shown in Table 34.

The wire-AND of the 2 FPLAs requires F_0 through F_7 to be programmed active-low (each designated as F*). Therefore, the shifter outputs the complement of the shifted input word, which must be in turn complemented if this inversion cannot be buried in the system. Both P-terms involving S_{RO} and S_{LO} can be combined as I_5S_{XO}, denoting a don't care for right or left shift. All 16 such terms appearing in F_0 through F_7 can be combined into 8 P-terms. It can be readily shown that all 64 P-terms implicit in the upper half of the table are needed for both arithmetic and logic shift, and require I_{12} = X (don't care) as conditional input. For the arithmetic shift selected by I_{12} = "1," 7 additional P-terms are necessary to ensure propagation of the sign bit to the right in a right shift, and retention of the sign bit in F_7 during a left shift. These additional P-terms can be obtained by listing the complete equation set summarized in the bottom half of the table.

For example, for an arithmetic shift output F_6 is given by:

$$F_6 = I_6S_{LO} + I_5S_{L1} + I_4S_{L2} + I_3S_{L3} + I_2S_{L4} + I_1S_{L5} + I_0S_{L6} + I_6S_{RO} + +I_7S_{R1} + (S_{R2} + S_{R3} + S_{R4} + S_{R5} + S_{R6} + S_{R7})\, I_7I_{12}$$

This application can be readily expanded to detect overflow, or to execute circular shifts. The capability for circular shifts is obtained by using and additional FPLA, for a total of 124 P-terms.

Note that here we can obtain a shift of 7 bit positions in 35ns, typical.

PRIORITY RESOLVER AND LATCH

FPLAs can perform the dual function of detecting and latching tri-state-bus data, on a priority basis. By using only 24 P-terms in a single FPLA, 3 priority functions can be selected via inputs $S_{0,1,2}$ as shown in Figure 40.

The reset pulse clears any previously latched priority, and must be at least 30ns wide to compensate for FPLA delay. Sampling of the system bus begins with the trailing edge of reset, and ends about 50ns after the detection of an input request (H → L transition). This delay is provided by the feedback chain of spare gates in the DM8097 buffers, and is required to allow the FPLA to latch the incoming request before releasing the bus. It is also the circuit's resolving time of nearly simultaneous requests. The FPLA program table is shown in Table 35. The function selected by S_0 provides a 1 of 8 priority in *time* by latching the first of eight signals occurring on the bus, and is useful in many polling applications in which a 50ns resolution is adequate. The functions selected by S_1 and S_2 provide 1 of 8 complementary priorities in *space* by latching the highest ranked signal on the bus.

Both functions are particularly useful in asynchronous multiport systems for transferring control of the main system bus. The concept illustrated is readily expanded with additional output circuitry to monitor up to 16 inputs with any assigned rank, or to implement a clocked revolving priority of N signals.

The primary advantage provided by the FPLA is that the reassignment of priority rank is facilitated by combining the external selection with FPLA programmability, without resorting to system *wire changes.*

MEMORY OVERLAYS

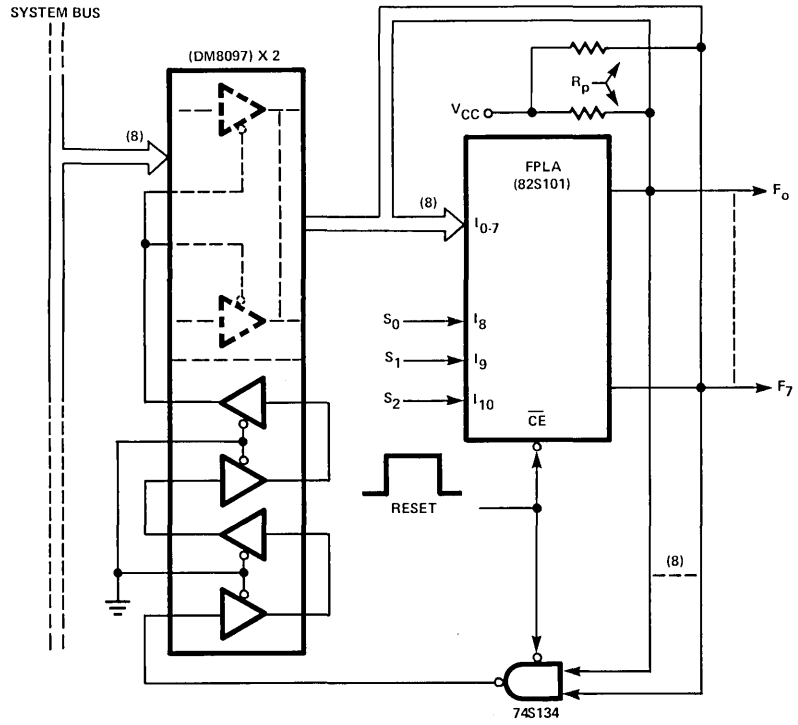
The storage and software efficiency of a computer can be improved by overlaying

	LOG/AR	L/R	S_N			I_m								F*
	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	5
I_5SR0 - - -	X	1	0	0	0	X	X	1	X	X	X	X	X	0
I_5SL0 - - -	X	0	0	0	0	X	X	1	X	X	X	X	X	0

(X) = Don't Care

Table 34 PROGRAM TABLE REPRESENTATION FOR $\bar{F}_5$ (RATHER THAN F5, DUE TO OUTPUT WIRE-ANDING)

PRIORITY RESOLVER AND LATCH WITH FPLA



The FPLA latched state must be reset prior to sampling new data.

Figure 40

INPUTS										OUTPUTS								FUNCTION
S2	S1	S0	17	16	15	14	13	12	11	10	F7	F6	F5	F4	F3	F2	F1	F0	
0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	↑ 1st of 8 Priority ↓
			1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	
			1	1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	
			1	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	
			1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	
			1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	
			1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	
0	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	
0	1	0	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	0	↑ 1 of 8 Priority (Ascending rank) ↓
			X	X	X	X	X	X	0	1	1	1	1	1	1	1	0	1	
			X	X	X	X	X	0	1	1	1	1	1	1	1	0	1	1	
			X	X	X	X	0	1	1	1	1	1	1	1	0	1	1	1	
			X	X	X	0	1	1	1	1	1	1	1	0	1	1	1	1	
			X	X	0	1	1	1	1	1	1	1	0	1	1	1	1	1	
			X	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	
0	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	↑ 1 of 8 Priority (Descending rank) ↓
			1	1	1	1	1	1	0	X	1	1	1	1	1	1	0	1	
			1	1	1	1	1	0	X	X	1	1	1	1	1	0	1	1	
			1	1	1	1	0	X	X	X	1	1	1	1	0	1	1	1	
			1	1	1	0	X	X	X	X	1	1	1	0	1	1	1	1	
			1	1	0	X	X	X	X	X	1	1	0	1	1	1	1	1	
			1	0	X	X	X	X	X	X	1	0	1	1	1	1	1	1	
1	0	0	0	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	

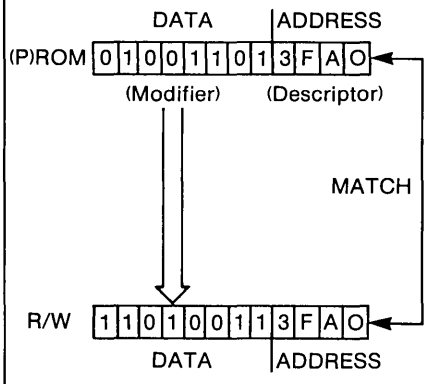
F_{0-7} must be programmed active-low. Unused inputs are programmed as Don't Care.

Table 35 FPLA PROGRAM TABLE FOR PRIORITY RESOLVER

Read/Write memory with (P)ROM memory in blocks of various sizes, including overlay on an individual word basis.

Overlay is a memory-conservation technique that permits several sets of information to share a block of storage. This allows several routines to occupy the same storage locations at different times. The method is also useful in incorporating special diagnostics, or for tailoring machine function to specific customer requirements while maintaining software compatibility.

OVERLAY IS DEFINED BY DESCRIPTOR AND MODIFIER

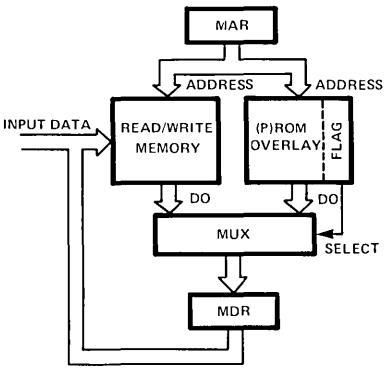


When (P)ROM and R/W memory addresses match, modifier data replaces data byte in R/W memory.

Figure 41

A memory overlay results in modification of a stored byte at a specific address in R/W memory, and is conceptually indicated in Figure 41. A typical memory overlay application is shown in Figure 42 in which a flag is used to conditionally transfer (P)ROM or R/W data in the MDR.

TYPICAL MEMORY OVERLAY SYSTEM



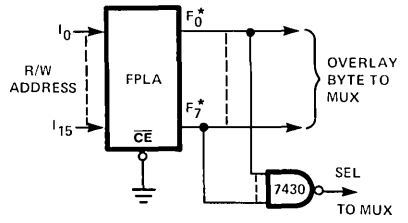
(P)ROM word is jammed in MDR when address "present" flag bit is true.

Figure 42

Since (P)ROMs are available in discrete chunks confined in standard IC configurations, a lot of storage can be wasted when the application requires overlay of many blocks of few words each, scattered throughout the address range of R/W main memory. All unused (P)ROM locations servicing a sparsely overlaid sector are forever inhibited access, and are therefore wasted. By using an FPLA instead of (P)ROM, the FPLA address matrix is programmed to recognize only the address of the RAM memory locations to be overlaid. The contents of the overlaid locations (the RAM modifier) are programmed in the FPLA storage matrix. This way, total PROM storage is compressed to the actual words used. Also, because of the large number of inputs to the FPLA, the overlaid locations can be scattered anywhere within a 64K address range. The chip enable feature readily extends this range to any practical size by allowing several FPLAs in parallel to examine a larger number of address inputs.

Note that with an FPLA it is not necessary to store flags at the overlay addresses for generating the Mux select signal. This signal can be generated by monitoring a $1 \rightarrow 0$ transition on any of the FPLA outputs. When no match exists, all FPLA outputs will be at logic "1" (assuming that they have been programmed active-low). If we exclude overlay of all 1's, the Mux select signal is obtained as in Figure 43.

MUX SELECT SIGNAL



SEL = 1 when at least 1 FPLA output goes low, indicating an address match, and causing the Mux to select FPLA data.

Figure 43

CORE MEMORY PATCH

The use of partially functional random access memory devices is a well known technique employed by manufacturers of add-on and other large memory systems to reduce overall memory cost per bit. This technique now can be extended to core memory systems by means of an FPLA. Modern core planes are available in many sizes, up to 16K X 18 or 32K X 9. A 64K X 9 memory would require 2 planes, each containing about 300K cores, in which it is not unusual to find as many as 100 broken or improperly tested cores.

Currently, cores are replaced by a hand "restringing" operation, at a cost of about $2/core. A better alternative to core replacement would be a dynamic repair routine, in which memory addresses containing bad bits are patched by an auxiliary memory. However, since bad cores can be scattered anywhere in the plane, this approach would in general be not cost

effective without an efficient address selector network.

The FPLA renders this technique economically feasible by providing an address "locator" function by virtue of its programmable address characteristics. The core memory addresses containing bad bits are mapped in the AND matrix of an FPLA, whose output OR matrix is programmed in turn with sequential address pointers to a small auxiliary RAM containing correct data.

This scheme is shown in Figure 44 and Table 36. A 16-input FPLA is used as an address map, and a 64 X 9 RAM as auxiliary memory, chosen to simplify control and to allow several bad core bits/word. The 48 P-terms of the FPLA allow dynamic repair of 48 memory addresses scattered anywhere in core. Correct data stored in the 82S09 is addressed by 6 FPLA outputs programmed as a binary table. Memory select control is provided by the F_6 output from the FPLA to jam the contents of auxiliary memory in the MDR when a faulty core location is addressed, and to enable writing in auxiliary memory only in the patched locations.

CORE MEMORY "PATCH" WITH FPLA REQUIRES ONLY 2 ICs

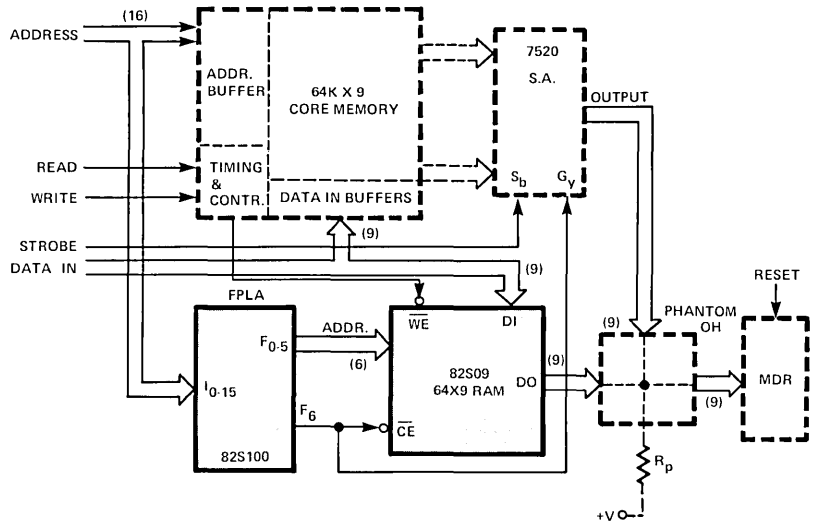


Figure 44

FPLA ADDRESS MAP	
CORE MEMORY	**AUXILIARY MEMORY**
A1	00_8
A2	01_8
A3	02_8
I	I
I	I
I	I
I	I
A48	57_8
Invalid Data	Valid Data

Table 36 FPLA TRANSLATES FAULTY 16-BIT INPUT ADDRESSES A1 THROUGH A48 INTO VALID AUXILIARY MEMORY LOCATIONS

The core memory system normally contains sockets and connections for both FPLA and auxiliary RAM. These are used only with partially functional planes. The FPLA input table is programmed immediately following final test with the addresses of core failures.

This technique could also be applied, with suitable modifications, to memory systems implemented with partially functional bipolar or MOS memory devices. It could also be extended to patch modifications in ROM memory systems, or utilize spare locations in PROM memory systems to avoid replacing several packages because of random or repeated changes.

SUBROUTINE ADDRESS MAP AND BRANCH LOGIC

In the design of microprogrammed computers considerable design flexibility is gained by complete freedom in allocating microprogram subroutines throughout microcontrol store, and by using variable formats in the instruction register op-code field.

To satisfy these requirements in an economical manner, an efficient means of address translation is mandatory. FPLAs are ideally suited for this application as shown in a typical system in Figure 45. The first FPLA translates the current op-code from a 16-bit instruction register into 48 subroutine-start addresses in microcontrol store. Variable op-code formats are easily handled by judicious programming of don't care states in the FPLA input table. The second FPLA is used to generate branch conditions based on the current microinstruction, as well as jump and status conditions in the machine. In particular, using tri-state FPLAs (82S100) saves a multiplexer in the address path of the ROM Address Register, while their 50ns access time minimizes overhead time in the instruction execution loop.

SUBROUTINE ADDRESS MAP AND BRANCH LOGIC WITH TRI-STATE FPLAs

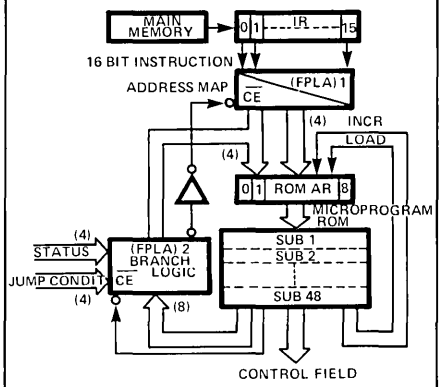


Figure 45

"VECTORED" PRIORITY INTERRUPT SYSTEM

In some applications, FPLAs are marginally cost effective when dedicated to a specific function which leaves spare most of the device resources. In such cases, the cost tradeoff may be resolved by a more efficient utilization through time-sharing the FPLA to perform separate functions.

This technique can be applied to the design of a "vectored" priority interrupt system for the Signetics 2650 microprocessor. The circuit in Figure 46 is all that is required to service 6 I/O devices via the conventional, single level, address vectoring interrupt mechanism of the 2650.

When one or more devices requests service, the CPU receives an $\overline{INTREQ}$ signal on its single interrupt pin. Program control is transferred to any of 128 possible memory locations as determined by an 8-bit vector supplied by the FPLA on the CPU data bus, in accordance with a preprogrammed priority. Since memory locations are expressed in 2's complement, the vector can point anywhere within -63 to +64 bytes of page zero, byte zero of memory. Also, both direct or relative indirect addressing modes can be specified

by the vector (bit $D_7 = 0/1$), hence program execution can be directed anywhere within addressable memory.

During the execution of the asynchronous CPU handshake the FPLA supplies at various times 3 distinct functions:

1. Interrupt request to the CPU, triggered by one or more service requests from devices 1 through 6.
2. Priority resolution of simultaneous requests by placing on the CPU data bus the vector of the highest ranked interrupting device.
3. Issue a request reset signal to 1 of 6 selected devices to acknowledge servicing its interrupt.

The 6 I/O devices have been assigned the arbitrary vectors tabulated in Table 37.

2's COMPLEMENT VECTOR	μP DBUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	D/I+/-	32	16	8	4	2	1	
+25 Direct	0	0	0	1	1	0	0	1
-39 "	0	1	0	1	1	0	0	1
+25 Indirect	1	0	0	1	1	0	0	1
-39 "	1	1	0	1	1	0	0	1
+55 Direct	0	0	1	1	0	1	1	1
+38 "	0	0	1	0	0	1	1	0

Table 37 VECTORS POINTING TO MEMORY LOCATIONS CONTAINING INSTRUCTIONS FOR SERVICING INTERRUPTING DEVICES

The FPLA program table in Table 38 shows the FPLA P-terms necessary to execute the above functions, with inputs $I_{15,14}$ used as function selectors under CPU control. Note that it was necessary to program the FPLA outputs with the *complement* of the vector, to compensate for the inversion with the 8T31.

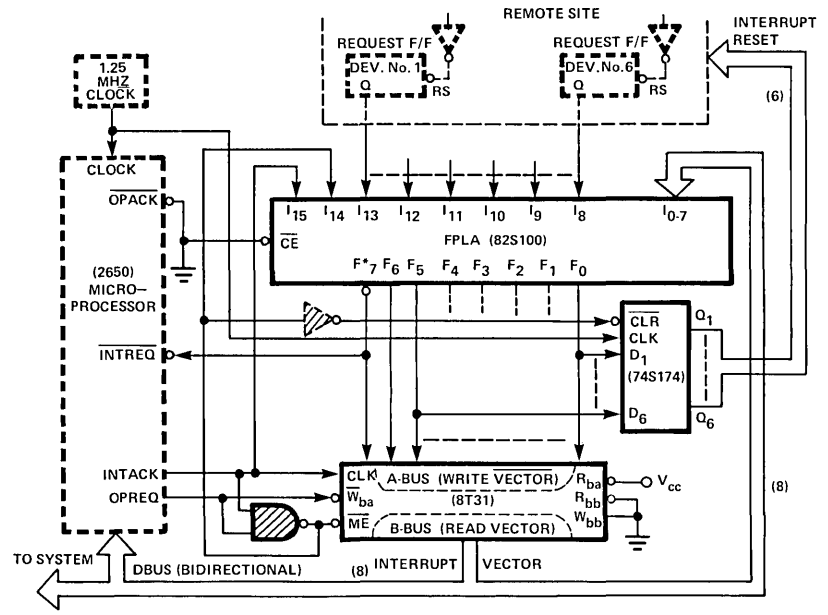
The timing diagram of the CPU handshake and FPLA response is shown in Figure 47.

In order to be immediately serviced, an $\overline{\text{INTREQ}}$ must be received by the CPU before the last cycle of the current instruction. When this occurs, the CPU finishes executing the current instruction, and in its last cycle, rather than fetching the next sequential instruction, it 1) sets the interrupt inhibit bit in the program status word to inhibit further interrupts, and 2) inserts the first byte of the "Zero Branch-to-Subroutine, Relative" instruction in the IR.

In the next cycle, the CPU gets ready to access the data bus to fetch the interrupt vector as the second byte of the ZBSR instruction, hence it generates the INTACK signal which is used to jam on the FPLA outputs the complement of the vector associated with the highest ranked device requesting service. The vector is latched, and placed on the CPU data bus following the leading edge of OPACK, after which the 8T31 A-bus is locked out. The CPU reads the D-bus on the trailing edge of T_2, and begins executing the interrupt routine. When the routine is completed, a return instruction clears the interrupt inhibit bit and links execution back to the interrupted program. Meanwhile, in order to communicate with the device being serviced by the interrupt routine, it is necessary to flag the device that its request has been acknowledged. This is done by issuing to the device a reset signal generated by the FPLA. The latched vector is fed back in the FPLA and decoded to issue a unique reset signal, which is in turn latched in the 74S174 on the leading edge of T_2 clock phase.

Several variants of this basic approach have been investigated. In particular, in a case where one needs to service 12 I/O devices and can tolerate to point the vector within a narrower memory address range, it is possible to substitute the 8T31 with 4 tri-state buffers, and use the FPLA in a wrap-around connection to latch the vector. The generation of the $\overline{\text{INTREQ}}$ and reset signals must however be reallocated outside the FPLA.

"VECTORED" PRIORITY INTERRUPT SYSTEM FOR THE 2650 MICROPROCESSOR REQUIRES 3 ICs, AND 2 SPARE GATES



Starred (*) FPLA outputs are programmed active-low.

Figure 46

FUNCTION	FUNCTION SELECTOR I_{15}	I_{14}	PRIORITY/REQUEST GENERATOR I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	RESET GENERATOR I_6	I_5	I_4	I_3	I_2	I_1	I_0	FPLA OUTPUT F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
INTERRUPT REQUEST TO μP	0	1	X	X	X	X	X	1	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
	0	1	X	X	X	X	1	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
	0	1	X	X	X	1	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
	0	1	X	X	1	X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
	0	1	X	1	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1
PRIORITY RESOLVER	1	1	X	X	X	X	X	1	X	X	X	X	X	X	X	X	1	1	1	0	0	1	1	0
	1	1	X	X	X	X	1	0	X	X	X	X	X	X	X	X	1	0	1	0	0	1	1	0
	1	1	X	X	X	1	0	0	X	X	X	X	X	X	X	X	0	1	1	0	0	1	1	0
	1	1	X	X	1	0	0	0	X	X	X	X	X	X	X	X	0	0	1	0	0	1	1	0
	1	1	X	1	0	0	0	0	X	X	X	X	X	X	X	X	1	1	0	0	1	0	0	0
	1	1	1	0	0	0	0	0	X	X	X	X	X	X	X	X	1	1	0	1	1	0	0	1
RESET REQUEST	1	0	X	X	X	X	X	X	0	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0
	1	0	X	X	X	X	X	X	0	1	0	1	1	0	0	1	1	0	0	1	0	0	0	0
	1	0	X	X	X	X	X	X	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	0
	1	0	X	X	X	X	X	X	1	1	0	1	1	0	0	1	1	0	0	0	0	1	0	0
	1	0	X	X	X	X	X	X	0	0	1	1	0	1	1	1	1	0	0	0	0	0	1	0
	1	0	X	X	X	X	X	X	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	1

Only 18 P-terms are necessary to perform three time-shared functions.

Table 38 FPLA PROGRAM TABLE

REFERENCES

1. D. Mrazek, and M. Morris, "How to Design with Programmable Logic Arrays," National Semiconductor Corp., app. note AN-89, 1973.
2. G. Reyling, "PLAs enhance digital processor speed and cut component count," Electronics, August 1974.
3. J. Maggiore, "PLA—A universal logic element," Electronic Products Magazine, April 1974.
4. W.N. Carr, and J.P. Mize, "MOS/LSI Design and Application," pp. 229-258, T.I. Electronics Series, McGraw-Hill Co., 1972.
5. J.C. Logue et al, "Hardware implementation of a small system in PLAs," IBM J. Res. Develop., March 1975.
6. A.W. Kobylar et al, "ROMs cut cost, response time of m/N detectors," Electronics, February 1973.

TIMING DIAGRAM OF I/O SERVICE REQUEST INTERRUPTING PROGRAM EXECUTION

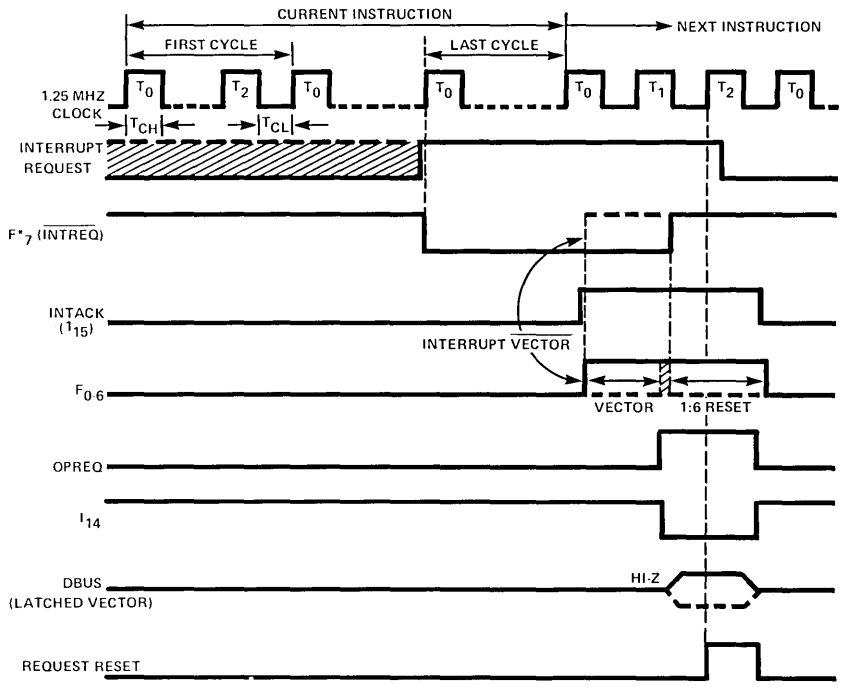


Figure 47

CHAPTER 6
APPENDICES

APPENDIX A PROGRAM TABLE OF SAMPLE DEVICE

(This portion to be completed by Signetics)

CF (XXXX) 1001 *OR, AND, EX-OR, MUX, ADD*

CUSTOMER SYMBOLIZED PART # _____ *WHEN YOU LIKE*

DATE RECEIVED _____

COMMENTS _____ *FOR ALL PURPOSE LOGIC APPLICATIONS*

CUSTOMER NAME *ADVANCED LOGIC DESIGNS*

PURCHASE ORDER # *NONE (FREE SAMPLE)*

SIGNETICS DEVICE # *82S100/101*

TOTAL NUMBER OF PARTS *1*

PROGRAM TABLE # _____ *REV* _____ *DATE* *7/22/75*

PROGRAM TABLE ENTRIES

INPUT VARIABLE			OUTPUT FUNCTION		OUTPUT ACTIVE LEVEL	
I_m	$\bar{I}_m$	Don't Care	Prod. Term Present in F_P	Prod. Term *Not* Present in F_P	Active High	Active Low
H	L	— (dash)	A	• (period)	H	L

NOTE
Enter (—) for *unused* inputs of *used* P-terms.

NOTES
1. Entries independent of output polarity.
2. Enter (A) for *unused* outputs of *used* P-terms.

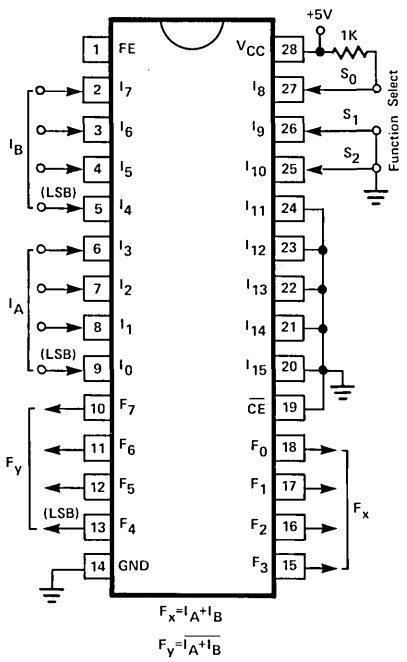
NOTES
1. Polarity programmed once only.
2. Enter (H) for all *unused* outputs.

	PRODUCT TERM*																ACTIVE LEVEL							
	INPUT VARIABLE																L	L	L	L	H	H	H	H
NO.	1	1	1	1	1	1											OUTPUT FUNCTION*							
	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	—	—	—	—	—	L	—	H	—	—	—	—	—	—	—	H	•	•	•	A	•	•	•	A
1	—	—	—	—	—	L	L	H	—	—	—	H	—	—	—	—	•	•	•	A	•	•	•	A
2	—	—	—	—	—	L	—	H	—	—	—	—	—	—	H	—	•	•	A	•	•	•	A	•
3	—	—	—	—	—	L	L	H	—	—	H	—	—	—	—	—	•	•	A	•	•	•	A	•
4	—	—	—	—	—	L	—	H	—	—	—	—	—	H	—	—	•	A	•	•	•	A	•	•
5	—	—	—	—	—	L	L	H	—	H	—	—	—	—	—	—	•	A	•	•	•	A	•	•
6	—	—	—	—	—	L	—	H	—	—	—	—	H	—	—	—	A	•	•	•	A	•	•	•
7	—	—	—	—	—	L	L	H	H	—	—	—	—	—	—	—	A	•	•	•	A	•	•	•
8	—	—	—	—	—	L	L	L	—	—	—	H	—	—	—	H	•	•	•	A	•	•	•	A
9	—	—	—	—	—	L	L	L	—	—	H	—	—	—	H	—	•	•	A	•	•	•	A	•
10	—	—	—	—	—	L	L	L	—	H	—	—	—	H	—	—	•	A	•	•	•	A	•	•
11	—	—	—	—	—	L	L	L	H	—	—	—	H	—	—	—	A	•	•	•	A	•	•	•
12	—	—	—	—	—	H	H	H	—	—	—	H	—	—	—	—	•	•	•	A	•	•	•	A
13	—	—	—	—	—	H	H	H	—	—	H	—	—	—	—	—	•	•	A	•	•	•	A	•
14	—	—	—	—	—	H	H	H	—	H	—	—	—	—	—	—	•	A	•	•	•	A	•	•
15	—	—	—	—	—	H	H	H	H	—	—	—	—	—	—	—	A	•	•	•	A	•	•	•
16	—	—	—	—	L	H	L	L	—	—	—	L	—	—	—	H	•	•	•	A	•	•	•	A
17	—	—	—	—	L	H	L	L	—	—	—	H	—	—	—	L	•	•	•	•	•	•	•	A
18	—	—	—	—	H	H	L	L	—	—	—	L	—	—	—	L	•	•	•	•	•	•	•	A
19	—	—	—	—	H	H	L	L	—	—	—	H	—	—	—	H	•	•	•	•	•	•	•	A
20	—	—	—	H	—	H	L	L	—	—	L	—	—	—	H	—	•	•	•	•	•	•	A	•
21	—	—	—	H	—	H	L	L	—	—	H	—	—	—	L	—	•	•	•	•	•	•	A	•
22	—	—	—	L	—	H	L	L	—	—	L	—	—	—	L	—	•	•	•	•	•	•	A	•
23	—	—	—	L	—	H	L	L	—	—	H	—	—	—	H	—	•	•	•	•	•	•	A	•
24	—	—	H	—	—	H	L	L	—	L	—	—	—	H	—	—	•	•	•	•	•	A	•	•
25	—	—	H	—	—	H	L	L	—	H	—	—	—	L	—	—	•	•	•	•	•	A	•	•
26	—	—	L	—	—	H	L	L	—	L	—	—	—	L	—	—	•	•	•	•	•	A	•	•
27	—	—	L	—	—	H	L	L	—	H	—	—	—	H	—	—	•	•	•	•	•	A	•	•
28	—	H	—	—	—	H	L	L	L	—	—	—	H	—	—	—	•	•	•	•	A	•	•	•
29	—		—	—	—	H	L	L	H	—	—	—	L	—	—	—	•	•	•	•	A	•	•	•
30	—	L	—	—	—	H	L	L	L	—	—	—	L	—	—	—	•	•	•	•	A	•	•	•
31	—	L	—	—	—	H	L	L	H	—	—	—	H	—	—	—	•	•	•	•	A	•	•	•
32	—	L	—	—	—	H	L	L	—	—	—	—	H	—	—	—	•	•	•	A	•	•	•	•
33	—	L	—	—	—	H	L	L	H	—	—	—	—	—	—	—	•	•	•	A	•	•	•	•
34	—	—	—	—	H	H	L	L	—	—	—	—	—	—	—	H	•	•	A	•	•	•	•	•
35	—	—	—	—	H	H	L	L	—	—	—	H	—	—	—	—	•	•	A	•	•	•	•	•
36	—	—	—	L	—	H	L	L	—	—	—	—	—	—	H	—	•	A	•	•	•	•	•	•
37	—	—	—	L	—	H	L	L	—	—	H	—	—	—	—	—	•	A	•	•	•	•	•	•
38	—	—	L	—	—	H	L	L	—	—	—	—	—	H	—	—	A	•	•	•	•	•	•	•
39	—	—	L	—	—	H	L	L	—	H	—	—	—	—	—	—	A	•	•	•	•	•	•	•
40	—	—	—	—	—	H	L	L	H	—	—	—	H	—	—	—	•	•	•	A	•	•	•	•
41	—	—	—	—	—	H	L	L	—	—	—	H	—	—	—	H	•	•	A	•	•	•	•	•
42	—	—	—	—	—	H	L	L	—	—	H	—	—	—	H	—	•	A	•	•	•	•	•	•
43	—	—	—	—	—	H	L	L	—	H	—	—	—	H	—	—	A	•	•	•	•	•	•	•
44																								
45																								
46																								
47																								

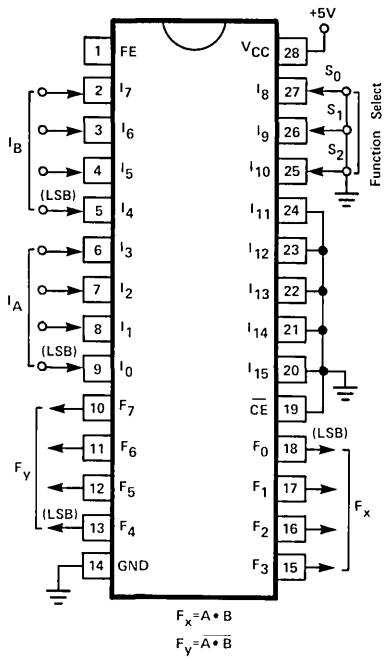
*Input and Output fields of *unused* P-terms can be left blank. Unused inputs and outputs are FPLA terminals left floating.

APPENDIX B CONNECTIONS FOR SAMPLE DEVICE

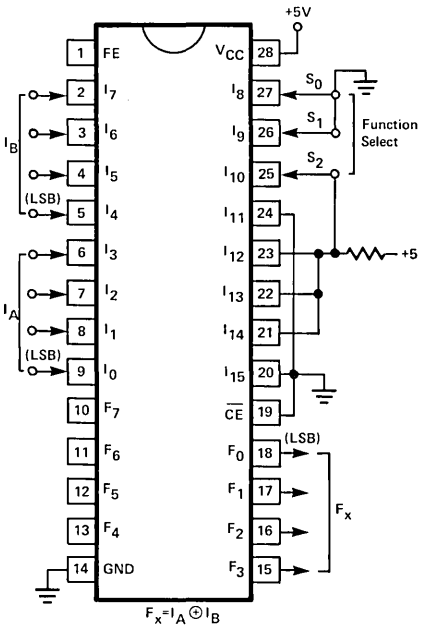
All device inputs may be toggled with manual switches, while all outputs can be monitored with an LED arrangement. To observe the 5 logic functions of the sample FPLA, connect the device as follows:



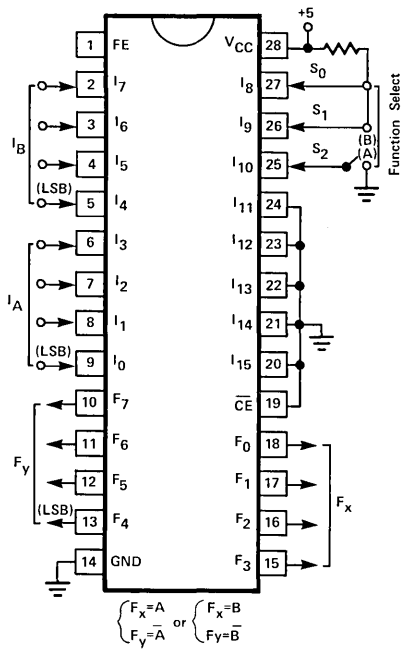
a. "OR" FUNCTION



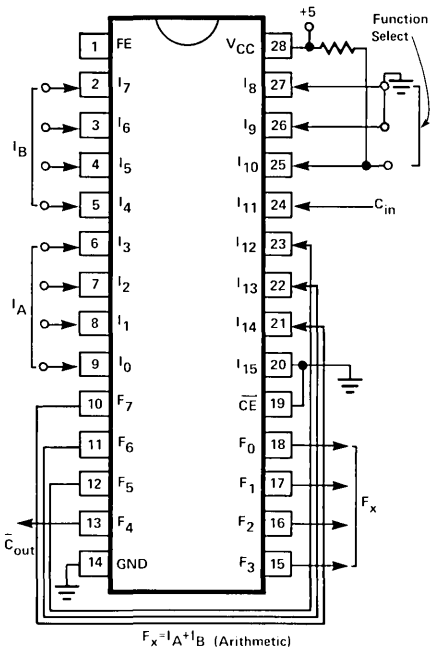
b. "AND" FUNCTION



c. "EX-OR" FUNCTION



d. "MULTIPLEX" FUNCTION



e. "ADD" FUNCTION (with Serial Carry)

SALES OFFICES

SIGNETICS

HEADQUARTERS

811 East Arques Avenue
Sunnyvale, California 94086
Phone: (408) 739-7700

ALABAMA
Huntsville
Phone: (205) 533-5800

ARIZONA
Phoenix
Phone: (602) 971-2517

CALIFORNIA
Inglewood
Phone: (213) 670-1101
Irvine
Phone: (714) 833-8980
(213) 924-1668
San Diego
Phone: (714) 560-0242
Sunnyvale
Phone: (408) 736-7565

COLORADO
Parker
Phone: (303) 841-3274

FLORIDA
Pompano Beach
Phone: (305) 782-8225

ILLINOIS
Rolling Meadows
Phone: (312) 259-8300

KANSAS
Wichita
Phone: (316) 683-5652

MARYLAND
Columbia
Phone: (301) 730-8100

MASSACHUSETTS
Woburn
Phone: (617) 933-8450

MINNESOTA
Edina
Phone: (612) 835-7455

NEW JERSEY
Cherry Hill
Phone: (609) 665-5071
Piscataway
Phone: (201) 981-0123

NEW YORK
Wappingers Falls
Phone: (914) 297-4074
Woodbury, L.I.
Phone: (516) 364-9100

OHIO
Worthington
Phone: (614) 888-7143

TEXAS
Dallas
Phone: (214) 661-1296

REPRESENTATIVES

ALABAMA
Huntsville
Murcota
Phone: (205) 539-8476

CALIFORNIA
San Diego
Mesa Engineering
Phone: (714) 278-8021
Sherman Oaks
Astralonics
Phone: (213) 990-5903

CANADA
Calgary, Alberta
Philips Electronics Industries Ltd.
Phone: (403) 543-5711
Montreal, Quebec
Philips Electronics Industries Ltd.
Phone: (514) 342-9180
Ottawa, Ontario
Phillips Electronics Industries Ltd.
Phone: (613) 237-3131
Scarborough, Ontario
Philips Electronics Industries Ltd.
Phone: (416) 292-5161
Vancouver, B.C.
Philips Electronics Industries Ltd.
Phone: (604) 435-4411

COLORADO
Denver
Barnhill Five, Inc.
Phone: (303) 426-0222

CONNECTICUT
Newtown
Kanan Associates
Phone: (203) 426-8157

FLORIDA
Altamonte Springs
Semtronic Associates
Phone: (305) 831-8233
Largo
Semtronic Associates
Phone: (813) 586-1404

ILLINOIS
Chicago
L-Tec Inc.
Phone: (312) 286-1500

INDIANA
Indianapolis
Enco Marketing
Phone: (317) 546-5511

KANSAS
Overland Park
Advanced Technology Sales
Phone: (913) 492-4333

MARYLAND
Glen Burni
Microcomp, Inc.
Phone: (301) 247-0400

MASSACHUSETTS
Reading
Kanan Associates
Phone: (617) 944-8484

MICHIGAN
Bloomfield Hills
Enco Marketing
Phone: (313) 642-0203

MINNESOTA
Edina
Mel Foster Tech. Assoc.
Phone: (612) 835-2254

MISSOURI
St. Louis
Advanced Technology Sales
Phone: (314) 567-6272

NEW JERSEY
Haddonfield
Thomas Assoc. Inc.
Phone: (609) 854-3011

NEW MEXICO
Albuquerque
The Staley Company, Inc.
Phone: (505) 292-0060

NEW YORK
Ithaca
Bob Dean, Inc.
Phone: (607) 272-2187

NORTH CAROLINA
Cary
Montgomery Marketing
Phone: (919) 467-6319

OHIO
Centerville
Norm Case Associates
Phone: (513) 433-0966
Fairview Park
Norm Case Associates
Phone: (216) 333-4120

OREGON
Portland
Western Technical Sales
Phone: (503) 297-1711

TEXAS
Austin
Cunningham Co.
Phone: (512) 459-8947
Dallas
Cunningham Co.
Phone: (214) 233-4303
Houston
Cunningham Company
Phone: (713) 461-4197

UTAH
West Bountiful
Barnhill Five, Inc.
Phone: (801) 292-8991

WASHINGTON
Bellevue
Western Technical Sales
Phone: (206) 641-3900

WISCONSIN
Greenfield
L-Tec, Inc.
Phone: (414) 545-8900

DISTRIBUTORS

ALABAMA
Huntsville
Hamilton/Avnet Electronics
Phone: (205) 533-1170

ARIZONA
Phoenix
Hamilton/Avnet Electronics
Phone: (602) 275-7851
Liberty Electronics
Phone: (602) 257-1272

CALIFORNIA
Costa Mesa
Avnet Electronics
Phone: (714) 754-6051
Schweber Electronics
Phone: (213) 556-3880
Culver City
Hamilton Electro Sales
Phone: (213) 558-2183
El Segundo
Liberty Electronics
Phone: (213) 322-8100
Mountain View
Elmar Electronics
Phone: (415) 961-3611
Hamilton/Avnet Electronics
Phone: (415) 961-7000
San Diego
Hamilton/Avnet Electronics
Phone: (714) 279-2421
Liberty Electronics
Phone: (714) 565-9171
Sunnyvale
Intermark Electronics
Phone: (408) 738-1111

CANADA
Downsview, Ontario
Cesco Electronics
Phone: (416) 661-0220
Zentronics
Phone: (416) 635-2822
Mississauga, Ontario
Hamilton/Avnet Electronics
Phone: (416) 677-7432
Montreal, Quebec
Cesco Electronics
Phone: (514) 735-5511
Zentronics Ltd.
Phone: (514) 735-5361

Ottawa, Ontario
Cesco Electronics
Phone: (613) 729-5118

Hamilton/Avnet Electronics
Phone: (613) 226-1700

Zentronics Ltd.
Phone: (613) 238-6411

Vancouver, B.C.
Bowtek Electronics Co., Ltd.
Phone: (604) 736-1141

Ville St. Laurent, Quebec
Hamilton/Avnet Electronics
Phone: (514) 331-6443

COLORADO

Commerce City
Elmar Electronics
Phone: (303) 287-9611

Denver
Hamilton/Avnet Electronics
Phone: (303) 534-1212

CONNECTICUT

Danbury
Schweber Electronics
Phone: (203) 792-3500

Georgetown
Hamilton/Avnet Electronics
Phone: (203) 762-0361

Hamden
Arrow Electronics
Phone: (203) 248-3801

FLORIDA

Ft. Lauderdale
Arrow Electronics
Phone: (305) 776-7790

Hamilton/Avnet Electronics
Phone: (305) 971-2900

Hollywood
Schweber Electronics
Phone: (305) 922-4506

Orlando
Hammond Electronics
Phone: (305) 241-6601

GEORGIA

Atlanta
Schweber Electronics
Phone: (404) 449-9170

Norcross
Hamilton/Avnet Electronics
Phone: (404) 448-0800

ILLINOIS

Elk Grove
Schweber Electronics
Phone: (312) 593-2740

Elmhurst
Semiconductor Specialists
Phone: (312) 279-1000

Schiller Park
Hamilton/Avnet Electronics
Phone: (312) 671-6082

Skokie
Bell Industries
Phone: (312) 965-7500

INDIANA

Indianapolis
Pioneer Elecrtonics
Phone: (317) 849-7300

KANSAS

Lenexa
Hamilton/Avnet Electronics
Phone: (913) 888-8900

MARYLAND

Baltimore
Arrow Electronics
Phone: (301) 247-5200

Gaithersburg
Pioneer Washington Electronics
Phone: (301) 948-0710

Hanover
Hamilton/Avnet Electronics
Phone: (301) 796-5000

Rockville
Schweber Electronics
Phone: (301) 881-2970

MASSACHUSETTS

Waltham
Schweber Electronics
Phone: (617) 890-8484

Woburn
Arrow Electronics
Phone: (617) 933-8130

Hamilton/Avnet Electronics
Phone: (617) 933-8000

MICHIGAN

Livonia
Hamilton/Avnet Electronics
Phone: (313) 522-4700

Pioneer Electronics
Phone: (313) 525-1800

MINNESOTA

Eden Prairie
Schweber Electronics
Phone: (612) 941-5280

Edina
Hamilton/Avnet Electronics
Phone: (612) 941-3801

Minneapolis
Semiconductor Specialists
Phone: (612) 854-8841

MISSOURI

Hazelwood
Hamilton/Avnet Electronics
Phone: (314) 731-1144

NEW MEXICO

Albuquerque
Hamilton/Avnet Electronics
Phone: (505) 765-1500

NEW YORK

Buffalo
Summit Distributors
Phone: (716) 884-3450

East Syracuse
Hamilton/Avnet Electronics
Phone: (315) 437-2642

Farmingdale, L.I.
Arrow Electronics
Phone: (516) 694-6800

Rochester
Hamilton/Avnet Electronics
Phone: (716) 442-7820

Schweber Electronics
Phone: (716) 461-4000

Westbury, L.I.
Hamilton/Avnet Electronics
Phone: (516) 333-5800

Schweber Electronics
Phone: (516) 334-7474

NORTHERN NEW JERSEY

Cedar Grove
Hamilton/Avnet Electronics
Phone: (201) 239-0800

Saddlebrook
Arrow Electronics
Phone: (201) 797-5800

SOUTHERN NEW JERSEY AND PENNSYLVANIA

Cherry Hill, N.J.
Milgray-Delaware Valley
Phone: (609) 424-1300

Moorestown, N.J.
Arrow/Angus Electronics
Phone: (609) 235-1900

Mt. Laurel, N.J.
Hamilton/Avnet Electronics
Phone: (609) 234-2133

CENTRAL NEW JERSEY AND PENNSYLVANIA

Somerset, N.J.
Schweber Electronics
Phone: (201) 469-6008

Horsham, PA
Schweber Electronics
Phone: (215) 441-0600

NORTH CAROLINA

Greensboro
Hammond Electronics
Phone: (919) 275-6391

Pioneer Electronics
Phone: (919) 273-4441

OHIO

Beechwood
Schweber Electronics
Phone: (216) 464-2970

Cleveland
Hamilton/Avnet Electronics
Phone: (216) 461-1400

Pioneer Standard Electronics
Phone: (216) 587-3600

Dayton
Hamilton/Avnet Electronics
Phone: (513) 433-0610

Pioneer Standard Electronics
Phone: (513) 236-9900

OKLAHOMA

Tulsa
Component Specialties
Phone: (918) 664-2820

TEXAS

Dallas
Component Specialties
Phone: (214) 357-4576

Hamilton/Avnet Electronics
Phone: (214) 661-8204

Quality Components
Phone: (214) 387-4949

Schweber Electronics
Phone: (214) 661-5010

Houston
Component Specialties
Phone: (713) 771-7237

Hamilton/Avnet Electronics
Phone: (713) 780-1771

Quality Components
Phone: (713) 772-7100

Schweber Electronics
Phone: (713) 784-3600

UTAH

Salt Lake City
Alta Electronics
Phone: (801) 486-7227

Hamilton/Avnet Electronics
Phone: (801) 972-2800

WASHINGTON

Bellevue
Hamilton/Avnet Electronics
Phone: (206) 746-8750

Seattle
Liberty Electronics
Phone: (206) 453-8300

WISCONSIN

New Berlin
Hamilton/Avnet Electronics
Phone: (414) 784-4510

FOR SIGNETICS PRODUCTS WORLDWIDE:

ARGENTINA
Fapesa I.y.C.
Buenos-Aires
Phone: 652-7438/7478

AUSTRIA
Osterreichische Philips
Wien
Phone: 93 26 11

AUSTRALIA
Philips Industries-ELCOMA
Lane-Cove, N.S.W.
Phone: (02) 427-0888

BELGIUM
M.B.L.E.
Brussels
Phone: 523 00 00

BRAZIL
Ibrape, S.A.
Sao Paulo
Phone: 284-4511

CANADA
Philips Electron Devices
Toronto
Phone: 425-5161

CHILE
Philips Chilena S.A.
Santiago
Phone: 39-4001

COLOMBIA
Sadape S.A.
Bogota
Phone: 600600

DENMARK
Miniwatt A/S
Kobenhavn
Phone: (01) 69 16 22

FINLAND
Oy Philips Ab
Helsinki
Phone: 1 72 71

FRANCE
R.T.C.
Paris
Phone: 355-44-99

GERMANY
Valvo
Hamburg
Phone: (040) 3296-1

HONG KONG
Philips Hong Kong, Ltd.
Hong Kong
Phone: 12-245121

INDIA
Semiconductors, Ltd.
(REPRESENTATIVE ONLY)
Bombay
Phone: 293-667

INDONESIA
P.T. Philips-Ralin Electronics
Jakarta
Phone: 581058

IRAN
Berkeh Company, Ltd.
Tehran
Phone: 831564

ISRAEL
Rapac Electronics, Ltd.
Tel Aviv
Phone: 477115-6-7

ITALY
Philips S.p.A.
Milano
Phone: 2-6994

JAPAN
Signetics Japan, Ltd.
Tokyo
Phone: (03) 230-1521

KOREA
Philips Elect Korea Ltd.
Seoul
Phone: 44-4202

MEXICO
Electronica S.A. de C.V.
Mexico D.F.
Phone: 533-1180

NETHERLANDS
Philips Nederland B.V.
Eindhoven
Phone: (040) 79 33 33

NEW ZEALAND
E.D.A.C., Ltd.
Auckland
Phone: 867119

NORWAY
Electronica A.S.
Oslo
Phone: (02) 15 05 90

PAKISTAN
Elmac Ltd
Karachi
Phone: 515-122

PERU
Cadesa
Lima
Phone: 628599

PHILIPPINES
Philips Industrial Dev., Inc.
Makata-Rizal
Phone: 868951-9

SINGAPORE/MALAYSIA
Philips Singapore Pte., Ltd.
Singapore
Phone: 538811

SOUTH AFRICA
E.D.A.C. (PTY), Ltd.
Johannesburg
Phone: 24-6701-3

SPAIN
Copresa S.A.
Barcelona
Phone: 329 63 12

SWEDEN
Elcoma A.B.
Stockholm
Phone: 08/67 97 80

SWITZERLAND
Philips A.G.
Zurich
Phone: 01/44 22 11

TAIWAN
Philips Taiwan, Ltd.
Taipei
Phone: (02) 551-3101-5

THAILAND
Saeng Thong Radio, Ltd.
Bangkok
Phone: 527195, 519763

UNITED KINGDOM
Mullard, Ltd.
London
Phone: 01-580 6633

UNITED STATES
Signetics International Corp.
Sunnyvale, California
Phone: (408) 739-7700

URUGUAY
Luzilectron SA
Montevideo
Phone: 9143 21

VENEZUELA
Industrias Venezolanas Philips S.A.
Caracas
Phone: 360-511

signetics

a subsidiary of **U.S. Philips Corporation**

Signetics Corporation
P.O. Box 9052
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

Printed in USA Oct. 1977