Monolithic Memories MMI

The Idealogic Exchange (09-26)
Monolithic Memories Inc.
2175 Mission College Blvd.
Santa Clara, CA 95054
(408)970-9700

Dear PLEASM User,

PLEASM is the cousin of PALASM, as PROMs are akin to PALs. PALASM and PLEASM constitute MMI's current Programmable logic Support Software. Together, they demonstrate MMI's commitment to providing total support for fusible link logic. By making these tools available, we hope to encourage the design engineer to take advantage of the flexibility offered by semi-custom logic in general, and fusible link technology in particular.

As with PALASM, PLEASM is available for a variety of computers and operating systems. Please check that you have the correct disks or tapes for your particular machine.

By your order, you will have been enrolled in the Idealogic Exchange. To ensure that you will recieve new product announcements and any software updates, please fill out and return the Purchace Registration Card located at the front of this binder. For post-sale service, contact your local MMI representative or Field Applications Engineer or call the Software Services Group at MMI directly.

Be sure to back-up your PLEASM disks/tapes before proceeding any further.

Also, please check the Errata sheet at the back for late changes.

Thank you.

Very truly yours,

The Idealogic Exchange
Monolithic Memories Inc.

THE PLEASM MANUAL - FIRST RELEASE

To be used with PLEASM Version 1.2

Date February 15, 1984

Prepared by the Software Services Group

Product Planning and Applications Department

Monolithic Memories, Inc.

COPYRIGHT

TRADEMARKS

The following are registered trademarks of MMI: PAL, HAL, PLE. MMI also has the trademarks: PALASM, PLEASM.

The following are registered trademarks of Digital Equipment Corporation: VAX, VMS, PDP, RSX.

IBM Corporation has the trademarks: IBM PC, PC DOS.

The Osborne PC is a trademark of the Osborne Computer Corporation.

UNIX is a trademark of AT & T.

Intel/MDS is a registered trademark of Intel Corporation.

CP/M is a trademark of Digital Research, Inc.

MS-DOS is a trademark of Microsoft.

SSFORTRAN is a trademark of Supersoft Inc.

DISCLAIMER

Monolithic Memories Inc. makes no representations or warranties with respect to the contents within and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Monolithic Memories Inc. reserves the right to revise this publication and the product it describes and to otherwise make changes to the product without obligation of Monolithic Memories Inc. to notify any person or organization of such revision or changes.

Table of Contents

Introduction

PLEASM (Programmable Logic Element ASseMbler) is a software package developed by Monolithic Memories Inc. used for designing with PROM's as Programmable Logic Elements(PLE's). PLEASM is a FORTRAN IV program which assembles and simulates PLE Design Specifications. It also generates programming formats for direct download to PROM programmers and can therefore be regarded as a tool that reduces the design-to-production time considerably.

Key Features:

- Assembles Logic or Arithmetic equations into a PROM truth table.
- Provides INTEL HEX and ASCII HEX programming formats along with the hex check sum.
- Programming formats can be directly downloaded to standard PROM programmers.
- Simulates the Function Table, in the design equations.
- Reports design errors.

The purpose of this manual is to aid the user in running PLEASM and getting to know and understand all its capabilities It begins with an article that describes the wide range of PROM applications and the motivation for developing a software tool to aid in designing these applications. The next section states the system requirements for PLEASM. The next two sections give a detailed account of how to run the program with an explanation of what each of the options accomplish. This is strengthened by an example where all the operations have been performed on an input file that has the PLE specifications for basic logic gates. The PLEASM syntax is described next. This is best understood if this section is read in accompaniment with some of the design examples that come with the PLEASM program. A detailed PLE applications handbook will be available shortly.

The Appendix gives some machine specific information about PLEASM along with details on PROM programmers, the PLE design files supplied as examples, and user customization. An important part of the Appendix is the section on the errors detected by PLEASM. This should be very useful when creating your own PLE designs.

The new PLEASM user should read Sections 3 to 5 initially and then try to run the demonstration examples. Once the user is ready to create his own design, Sections 2 and 6 will provide ideas and details of the specification format. The Appendices serve to provide additional supporting information on a number of subtleties the user should be aware of in fully utilizing the software.

PROMs vs. PLEs

PROMs have grown steadily in size and speed since their introduction in 1971, when Monolithic Memories introduced the world's first 1K bit bipolar PROM. Today, 16K and 32K PROMs are readily available and their speeds have improved such that the maximum address time (address to output) of these devices is down to 60-70 nanoseconds, over the complete operating temperature and VCC range. This means that PROMs can be used effectively in both high speed memory and logic replacement applications.

The PROM implements a sum-of products boolean transfer function in which any possible input (address) combination can be transferred to any output variable (data out). Figure 1.1 shows logical structure of a typical PROM. The input Fixed-AND array is a decoder and the output Programmable-OR array is a decoder. It is this decoder area that is field programmable to implement any boolean transfer function.

Each output of the AND array is connected to an input of the OR array by thin metal wire (e.g. nichrome, titanium-tungsten, platinum-silicide) which can be selectively removed from the circuit by passing a current through it. This is referred to as "programming" or "blowing" a "fuse".

2.1 The PROM as a Memory Element

Because of their high speeds, bipolar PROMs are ideal for use in systems requiring fast Address-to Data access times. PROM applications can be found in both the data and control paths of a system.

In data paths, PROMs are used mainly as storage elements to implement different table look-up applications such as trigonometric functions, signal processing coefficients, bootstrapping and initialization programs, etc. In particular, the PROM can be used to advantage in the design of digital filters and Fast Fourier Transforms. In character generator applications, the PROM user has the flexibility of modifying the conventional fonts to his/her particular requirements.

In control paths, PROMs are used mainly to store microprograms. Microprogrammed controllers may be simple PROM-register finite state machines or they may be complex microprogrammed CPUs, where the complete instruction set of the system resides in PROM. It is thus possible, by using PROMs for microprogramming, to use the same hardware to emulate the characteristics of various processors.

Since most memory applications involve storing PROM memory data in a temporary register before it is used (pipelining), this has spawned a new generation of PROMs with on-board D-type edge-triggered registers. These registered PROMs operate faster than discrete PROM-register combinations, and the registered PROMs also occupy less space.

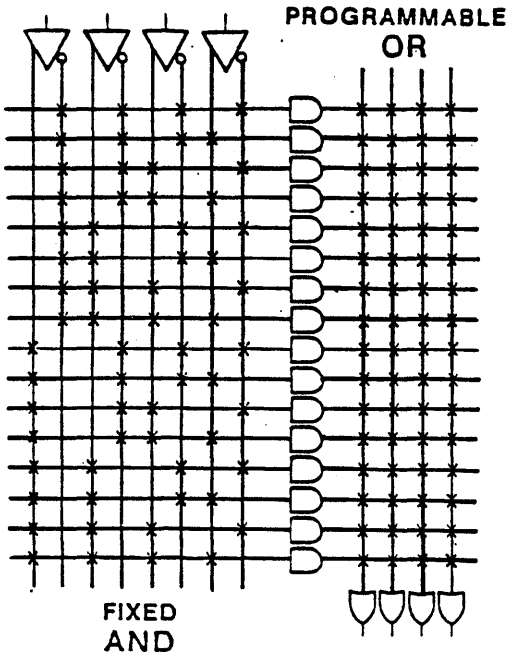


Figure 1.1 Typical PROM structure

$A \bullet B \quad C \bullet D = F_4$
$A + B + C + D = F_3$
$A \cdot B \cdot C \cdot D = F_2$
$A \cdot B + C \cdot D = F_1$

FUNCTION						F_1	F_2	F_3	F_4
ADDRESS	A_0	A_1	A_2	A_3	OUTPUT	O_1	O_2	O_3	O_4
0	0	0	0	0		1		1	
1	1	0	0	0		1		1	
2	0	1	0	0		1		1	
3	1	1	0	0		1		1	-
4	0	0	1	0				1	
5	1	0	1	0				1	1
6	0	1	1	0				1	1
7	1	1	1	0		1		1	
8	0	0	0	1				1	
9	1	0	0	1				1	1
10	0	1	0	1				1	1
11	1	1	0	1		1		1	
12	0	0	1	1				1	
13	1	0	1	1				1	
14	0	1	1	1				1	
15	1	1	1	1			1	1	

Figure 1.2 Combinatorial functions available in a 16x4 PROM

2.2 The PROM as a Programmable Logic Element.

The PROM implements a sum-of-products boolean transfer function so that any function of x inputs and y outputs may be generated in a PROM with x addresses and y data outputs. Figure 1.2 shows the combinational functions available in a 16 X 4 PROM.

The AND-OR structure of the PROM can be viewed as a two-level logic circuit. The fixed AND plane contains all possible input combinations. Each input combination is a product term and it is connected to the output in the OR plane.

In terms of a PLE, a product term is the equivalent of an AND gate equal in size to the number of inputs. Each output is equivalent to an OR gate connected to all the AND gates. Programming a fuse blows this connection between the AND gate and the OR gate. The PROM thus conveniently implements combinatorial logic when a large number of input combinations are required or a large number of product terms per output is desired.

Most applications of PLEs are in synchronous control systems where they replace random logic or customise logic functions. In data paths, they are used to generate complex functions such as pseudo random number generators, ALU operations, multiplications, reciprocals, etc.

3.1 Equipment/Set-Up for Load/Go system

PLEASM should run with minimal modifications on the following CPU's provided the minimum system requirements mentioned later in this section are satisfied.

Mainframe Computers
VAX-VMS, VAX-UNIX, or IBM.

Minicomputers
PDP-11/RSX or PDP-11/RT-11

Microcomputers
IBM-PC/MS-DOS or the CP-M system on Radio Shack, Apple, Kaypro or Osborne computers.

Other requirements are:

Removable Media : 5.25'' or 8'' disks, or tape.

Memory : 64K bytes minimum.

One EIA RS232 serial communications port.

PROM programmers suggested :
DATA I/O Model 19 Programmer with Unipak.
DATA I/O Model 29 A with Unipak.
DIGELEC Model UP-803 with FAM 12.
KONTRON Model MPP-80S

3.2 Equipment/Set-Up for Development system

For software development and user customization of the program, the requirements presented earlier are necessary.

In addition, a FORTRAN compiler/linker and another disk drive are necessary to create the executable version of the program. The compiler recommended is the Supersoft FORTRAN compiler which was used to develop and test the program on microcomputers at Monolithic Memories Inc.

PLEASM is compatible with both FORTRAN IV LEVEL G and FORTRAN 77 standards. Additionally, only standard FORTRAN constructs are used to ensure portability to many computer systems.

USER'S GUIDE TO PLEASM(tm) - PLE Assembler version 1.2A

To get started with PLEASM, turn the computer ON. Check your directory to make sure you have the files mentioned in Appendix A. Once this has been verified, run the program as explained below with one of the example files. Our suggestion is to start with the file P5000.TXT which contains the PLE Design Specifications for the basic logic gates. Once the capabilities of the program have been understood, you can work with any of the other examples or attempt to create your own designs.

Using PLEASM:

Type the system's execute command to run the program.
PLEASM will respond....

```
MONOLITHIC MEMORIES PLEASM(tm) VERSION 1.2A
(C) COPYRIGHT 1984 MONOLITHIC MEMORIES

WHAT IS THE SOURCE FILENAME (d:filename.ext) ?:P5000.TXT
```

At this point enter the name of the file containing the specifications for the PLE being designed. If you are using this package for the first time, we suggest you try out one of the design examples that was sent along with PLEASM. PLEASM next prompts you for the name of the file you could have the output sent to, defaulting to the console....

```
OUTPUT FILENAME - PRESS <ENTER> FOR NO OUTPUT FILE ?:<CR>
```

If you press <enter>/<return> the output will be sent to the console after each operation. At this point, the input.file is read, and a count of lines and characters in the file is written out to the screen. The next prompt is for the operation you want performed and is....

```
E=ECHO INPUT  S=SIMULATE  T=TRUTH TABLE  B=BRIEF TABLE
A=HEX TABLE   I=INTEL HEX H=ASCII HEX    C=CATALOG  Q=QUIT

ENTER OPERATION CODE:C
```

You can now enter the appropriate operation code, IN UPPER CASE!!.

The various options are briefly discussed below.

E ECHO INPUT - Prints the input PLE specifications file. Useful as a ready reference while working interactively with PLEASM.

S SIMULATE - Exercises the logic values in the optional function table in the logic equations provided. Errors in the function table are detected along with fairly explicit diagnostic messages. An important point to note is that all "don't care" conditions are treated as low logic values.
This option can be successfully invoked only when a function table is present in the input specifications.

T TRUTH TABLE - Prints out the entire binary truth table for all the input variables in the PLE by substitutions into the Boolean equations specified. The output has a tabular format for ease of reading. The program also provides a hex checksum for the entries in the truth table at the end.

B BRIEF TABLE - Prints out the truth table only for the used input addresses in the PLE, again by substitutions into the Boolean equations. The output is tabulated as before, this time with a partial hex checksum corresponding to the possibly shorter table.

A HEX TABLE - Prints out the entire truth table as before, except the inputs and outputs are translated into hex. Also generates a tabular format with a hex checksum.

I INTEL HEX - Generates the Intel Hex format for PROM programmers for both 4- and 8-bit data downloading. The format is shown below....

```
:AABBBB00CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCDD
```

```
: --> Starting colon marker
AA --> Record Length in hex
BBBB --> Record starting address in hex
C..C --> Data
DD --> Hex checksum
```

Here all data is sent in streams of 16 8-bit bytes starting at 0000H. 4-bit data is padded up with zeros in the most significant four places. The checksum is the negative of the sum of all 8-bit bytes starting at "AA" upto "DD", modulo 256. Transmission is terminated by the string ":00000001FF".

H ASCII HEX SPACE - Generates the ASCII Hex format for PROM programmers for 4- and 8-bit data downloading. The format is shown below....

```
A
BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB BB .
C
```

A --> Record start character (STX)
BB --> Data byte
C --> End of text character (ETX)

Data is sent in streams of 16 8-bit bytes separated by spaces. An execute character, the ASCII period ".", is sent at the end of each stream of data. 4-bit data is padded up with zeros in the most significant 4 places. In addition, a hex checksum is passed at the end of the transmission.

C CATALOG - Prints a one-line description of each option provided by PLEASM. Always displays to the console.

CATALOG OF OPERATION CODES:

MONOLITHIC MEMORIES PLEASM(tm) VERSION 1.2A

PLEASM --PLE ASSEMBLER-- PROVIDES THE FOLLOWING OPTIONS :

```
C  CATALOG        - PRINTS THE PLEASM CATALOG OF OPERATIONS
E  ECHO INPUT     - PRINTS THE PLE DESIGN SPECIFICATIONS
T  TRUTH TABLE    - PRINTS THE ENTIRE TRUTH TABLE
B  BRIEF TABLE    - PRINTS ONLY USED ADDRESSES IN THE TRUTH TABLE
A  HEX TABLE      - PRINTS THE TRUTH TABLE IN HEX FORM

S  SIMULATE       - EXERCISES THE FUNCTION TABLE  IN  THE  LOGIC
                    EQUATIONS

I  INTEL HEX      - GENERATES INTEL HEX PROGRAMMING FORMAT
H  ASCII HEX      - GENERATES ASCII HEX PROGRAMMING FORMAT

Q  QUIT           - EXITS PLEASM
```

Q QUIT - Exits the PLEASM program and prompts for restarting with another input specifications file.

```
ENTER OPERATION CODE: E
(     Echoes the input PLE Design Specifications file. This will
( help verify that the input file has been read in correctly.

PLE5P8                                          PLE DESIGN SPECIFICATION
P5000                                             VINCENT COLI 10/03/82
BASIC GATES
MMI SANTA CLARA, CALIFORNIA


O1 =  I0                                        ; BUFFER
O2 = /I0                                        ; INVERTER
O3 =  I0  *  I1  *  I2  *  I3                   ; AND GATE
O4 =  I0  +  I1  +  I2  +  I3                   ; OR GATE
O5 = /I0  + /I1  + /I2  + /I3                   ; NAND GATE
O6 = /I0  * /I1  * /I2  * /I3                   ; NOR GATE
O7 =  I0 :+: I1 :+: I2 :+: I3                   ; EXCLUSIVE OR GATE
O8 =  I0 :*: I1 :*: I2 :*: I3                   ; EXCLUSIVE NOR GATE


FUNCTION TABLE

I0 I1 I2 I3 O1 O2 O3 O4 O5 O6 O7 O8

;INPUT   - -  OUTPUTS  FROM  BASIC  GATES  - -
;0123    BUF  INV  AND  OR  NAND  NOR  XOR  XNOR   COMMENTS
-------------------------------------------------------------------
 LLLL     L    H    L   L    H     H    L    H     ALL ZEROS
 HHHH     H    L    H   H    L     L    L    H     ALL ONES
 HLHL     H    L    L   H    H     L    L    H     ODD CHECKERBOARD
 LHLH     L    H    L   H    H     L    L    H     EVEN CHECKERBOARD
-------------------------------------------------------------------

DESCRIPTION

THIS EXAMPLE ILLUSTRATES THE USE OF PLEs TO IMPLEMENT THE BASIC GATES:
BUFFER, INVERTER, AND GATE, OR GATE, NAND GATE, NOR GATE, EXCLUSIVE OR
GATE, AND EXCLUSIVE NOR GATE.

NOTE ALSO THAT THREE-STATE OUTPUTS ARE PROVIDED WITH ONE ACTIVE LOW
OUTPUT ENABLE CONTROL (/E).
```

```
ENTER OPERATION CODE: S
(     This  option  verifies  that  the  output  entries  in  the
( Function Table are correct for the given Boolean equations  and
( input  vectors.  Any  discrepancy between the  expected  output
( value as  given in the Function Table  and the output value  as
( computed from the Boolean equations is flagged as an error.
(
(     The  following are acceptable input entries in the Function
( Table:
(
(                H - High level
(                L - Low level
(                X - Irrelevant

FUNCTION TABLE

I0 I1 I2 I3 O1 O2 O3 O4 O5 O6 O7 O8

;INPUT    - -  OUTPUTS  FROM  BASIC  GATES  - -
;0123     BUF  INV  AND  OR  NAND  NOR  XOR  XNOR   COMMENTS
-------------------------------------------------------------------
 LLLL      L    H    L   L    H     H    L    H     ALL ZEROS
 HHHH      H    L    H   H    L     L    L    H     ALL ONES
 HLHL      H    L    L   H    H     L    L    H     ODD CHECKERBOARD
 LHLH      L    H    L   H    H     L    L    H     EVEN CHECKERBOARD
-------------------------------------------------------------------

PASS SIMULATION


ENTER OPERATION CODE: T
(    Generates an exhaustive binary truth table for all the given
( inputs by substitution in the Boolean equations.

BASIC GATES

 ADD   A0 A1 A2 A3 A4  O1 O2 O3 O4 O5 O6 O7 O8
-----------------------------------------------
   0   L  L  L  L  L   L  H  L  L  H  H  L  H
   1   H  L  L  L  L   H  L  L  H  H  L  H  L
   2   L  H  L  L  L   L  H  L  H  H  L  H  L
 . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . .
  29   H  L  H  H  H   H  L  L  H  H  L  H  L
  30   L  H  H  H  H   L  H  L  H  H  L  H  L
  31   H  H  H  H  H   H  L  H  H  L  L  L  H
-----------------------------------------------

HEX CHECK SUM = 00F48
```

```
ENTER OPERATION CODE: B
(      Prints the truth table for only the used input and  output
( pins.

BASIC GATES

 ADD   A0 A1 A2 A3  O1 O2 O3 O4 O5 O6 O7 O8
-----------------------------------------
   0   L  L  L  L   L  H  L  L  H  H  L  H
   1   H  L  L  L   H  L  L  H  H  L  H  L
   2   L  H  L  L   L  H  L  H  H  L  H  L
 . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . .
  13   H  L  H  H   H  L  L  H  H  L  H  L
  14   L  H  H  H   L  H  L  H  H  L  H  L
  15   H  H  H  H   H  L  H  H  L  L  L  H
-----------------------------------------

PARTIAL HEX CHECK SUM = 007A4


ENTER OPERATION CODE: A
(      Generates  the truth table with input and  output  vectors
( translated into hex.

BASIC GATES

 ADD       HEX ADDRESS       HEX DATA
-------------------------------------
   0           000             00B2
   1           001             0059
   2           002             005A
 . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . .
  29           01D             0059
  30           01E             005A
  31           01F             008D
-------------------------------------

HEX CHECK SUM = 00F48


ENTER OPERATION CODE: I
(     Generates the Intel Hex programming format for  downloading
( to a PROM programmer.

:10000000B2595A995A999A595A999A599A595A8D4C
:10001000B2595A995A999A595A999A599A595A8D3C
:00000001FF
```

```
ENTER OPERATION CODE: H
(      Generates the ASCII Hex Space programming format for
( downloading to a PROM programmer, with the required <STX>,
(  <SOH>, and <ETX> control characters delimiting the
( transmission.

^B^A
B2 59 5A 99 5A 99 9A 59 5A 99 9A 59 9A 59 5A 8D .
B2 59 5A 99 5A 99 9A 59 5A 99 9A 59 9A 59 5A 8D .
^C
00F48


ENTER OPERATION CODE: Q
(      Exits PLEASM and prompts for restarting with another input
( file.

RESTART PLEASM(Y/N) ?:
```

PLE Design Specification:

The PLE Design Specification is the input file used with PLEASM. It is also the recommended data sheet format for describing the function of a PROM, once it has acquired the unique personality of a particular fuse pattern. This format for creating an input specifications file can be best understood by studying the examples that come along with this package. The format for the PLE Design Specification is :

Line 1 PLE part number, starting in column 1, followed by the words PLE DESIGN SPECIFICATION.

Line 2 User's part number followed by the designer's name and the date, starting in column 1. This may be an identifier that defines the application and is used for reference.

Line 3 Device Application name, starting in column 1.

Line 4 User's company name and address, starting in column 1.

Lines 2-4 are without formal rules and are provided for documentation purposes.

Line 5 Address pin list, prefixed by .ADD, starting in column 1. The pin list should be ordered LSB first.

Line 6 Data pin list, prefixed by .DAT, starting in column 1. The pin list should be ordered LSB first.

The pin list is a sequence of symbolic names separated by one or more spaces on one or more lines in the order of the device pin numbers. Each symbolic name is unique (except the unused pins which may have the same name). All pins including power and ground must be named. Names may use any printable characters except the operators ';', '.', ',', '=', '*', and '+'. The prefix '/' is used to logically complement the name.

Line m EQUATIONS

The transfer function of the device is expressed in the following form :

SYMBOL = EXPRESSION

The following terms are used to construct the equations:

SYMBOL Pin Name with optional prefix '/'

EXPRESSION A sequence of SYMBOLS separated by operators.

OPERATORS (in hierarchy of evaluation)

;	Comment follows
.	Dot operator (pin list or arithmetic operator follows)
ADD	Address pins (Inputs)
DAT	Data Pins (Outputs)
,	Delimiter, separates binary bits (MSB first)
=	Equality (combinatorial)

BOOLEAN OPERATORS

/	Complement, prefix to a pin name
*	AND (PRODUCT)
+	OR (SUM)
:+:	XOR (EXCLUSIVE OR)
:*:	XNOR (EXCLUSIVE NOR)

ARITHMETIC OPERATORS

.*.	Multiply (Arithmetic multiplication)
.+.	PLUS (Arithmetic addition)

Line n FUNCTION TABLE

The function table begins with the key word FUNCTION TABLE, starting in column 1 of a line following the equation list. It is followed by a pin list which may be in a different order and polarity from the pin list in Lines 5 and 6. The pin list is followed by a line of dashes (-'s) which is in turn followed by a list of vectors, one vector per line. One state must be specified for each pin name and optionally separated by spaces.

A vector is a sequence of states listed in the same order as the pin list and followed by an optional comment. The vector list is followed by another dashed line.

An important restriction is that blank lines are not permitted in the body of the function table. To separate logically distinct parts of the function table, however, comments can be used. In other words, blank lines are permitted with a semicolon (;) in the first column. Additionally, comments can be placed in this line. Extra blank lines might result in the simulator scanning past the end of the function table and detect "-" as an error symbol, resulting in failure to pass simulation.

A function table is optional and need be present only for simulation to be performed. The keyword FUNCTION TABLE, however, is necessary in the input file. This should start in column 1, and should follow the equation list.

Definition of Function Table States:

Symbol	Definition	Input	Output
H	HIGH LEVEL	Drive High	Test High
L	LOW LEVEL	Drive Low	Test Low
X	IRRELEVANT	Don't Care Condition	Do Not Test

Line o DESCRIPTION (optional)

This begins with the keyword DESCRIPTION, starting in column 1. The device operation and application are described here. All the lines following the keyword DESCRIPTION are treated as comments. Even though the lines following may be blank, the keyword DESCRIPTION is necessary in the input file for assembly.

An informal grammar for the input specifications file is given below:

```
PLE_SPECIFICATIONS_FILE --> PLE_TYPE_LINE
                            PLE_PATTERN_LINE
                            APPLICATION_DESCRIPTION_LINE
                            COMPANY_INFORMATION_LINE
                            ADDRESS_PIN_LIST
                            DATA_PIN_LIST
                            EQUATION_LIST
                            FUNCTION TABLE
                            [FUNCTION_TABLE_PIN_LIST]
                            [FUNCTION_TABLE]
                            DESCRIPTION
                           [COMMENTS ]
```

PLE_TYPE_LINE --> PLE<PLE PART NUMBER> ... PLE DESIGN SPECIFICATIONS

PLE_PATTERN_LINE --> <REFERENCE NUMBER FOR APPLICATION, AUTHOR'S NAME>

APPLICATION_DESCRIPTION_LINE --> <APPLICATION NAME>

COMPANY_INFORMATION_LINE --> <COMPANY NAME AND ADDRESS>

ADDRESS_PIN_LIST --> .ADD <PIN NAME LIST FOR INPUT PINS>

DATA_PIN_LIST --> .DAT <PIN NAME LIST FOR OUTPUT PINS>

EQUATION_LIST --> <LIST OF BOOLEAN EQUATIONS>

FUNCTION_TABLE_PIN_LIST --> <LIST OF PINS TO BE SIMULATED>

FUNCTION_TABLE --> H|L|X <ENTRIES DEFINING LOGIC VALUES FOR THE PINS>

COMMENTS --> <COMMENTS DESCRIBING APPLICATION>

where, [..] denotes an optional expression,

<..> denotes an informal representation for the expression,

'___' denotes a keyword,

and ' | ' denotes the "OR" operator.

Important notes on PLEASM input specifications:

1. The specifications file must contain the keywords FUNCTION TABLE and DESCRIPTION starting in column 1 for assembly to occur.

2. All responses to PLEASM prompts should be in upper case.

3. The input specifications file should preferably be entirely in upper case.

4. The number of characters in the file, the number of lines in the file, and the number of characters in each line should be within the limits set in the I/O initialization package. The current limits set are 6000 chars/file, 250 lines/file, and 80 chars/line.

A.1 PLEASM on the VAX11/VMS

The following files should be in your tape if you have the Load/Go System :

PLEASM.EXE - This is the executable file that can be invoked to assemble your input PLE specifications.

P5000.TXT through P5017.TXT - These are the example files containing some applications. They are useful for studying how the input file should be written, and can be run with PLEASM to provide an on-line demonstration of how the program works. Details on the contents of these files can be found in Appendix D.

IOINIT.FOR - User customization package for array dimensions and I/O.

If you have ordered the Development system, you should have in addition the files :

PLEASM.FOR - This file contains the FORTRAN source for the PLEASM program and can be compiled by any FORTRAN compiler you have at your disposal.

IOLIB.FOR - This file contains the I/O features that make this version of PLEASM compatible with that on the IBM-PC. This will have to be linked in with the main program during compilation.

Unloading Your Mag Tape under VAX/VMS

Helpful Hints:

- The volume is labelled PLEASM and is recorded in Files-11 format, 1600 BPI and 9-track mag tape.

- For the neophytes who wish to learn everything there is to learn about mag tapes and more, Digital Equipment Corporation has published The Magnetic Tape Users' Guide (Order No. AA-M539A-TE).

- Your tape drive goes by many different names. For example, MTA0: or MSA0:. To list all devices on your installation, type SH DEV M<cr> when you see the $ prompt.

After loading your tape on the drive, when you see the $ prompt, type the following:

```
$ ALL MTA0: TAPE

( Allocates space for TAPE on device MTA0:

$MOUNT/OVER=IDENT TAPE

(   This mounts the tape and overrides any tape labels. Operator
( privileges are sometimes required to do this.
```

```
$CREATE/DIR [.PLEASM]
$SET DEFAULT [.PLEASM]

(   Sets up the directory appropriately.


$COPY TAPE:*.*;* []*

( This copies the tape files to your directory.

$DISMOUNT TAPE

( Dismounts tape and wraps things up.
```

DISREGARD THIS MARKED SECTION IF YOU HAVE THE LOAD/GO SYSTEM !!
****

Compile and link the source program using the following sequence of commands, to create the executable version :

```
$ FORTRAN PLEASM,IOLIB
$ LINK PLEASM,IOLIB
```

****

Using PLEASM:

Create your PLE Design Specification file using one of your system's editors.
Then type the following:

```
$ RUN PLEASM
```

The program PLEASM should now run . At this point, refer to Section 4. for step-by-step instructions on how to use the program.

Dumping a File From the VAX(VMS) to the Data I/O:

Cable Connections:

The RS-232C cable that connects the VAX-11 to the Data I/O has lines 2 and 3 reversed. The only other pins that must be connected are pins 1 and 7.

Operating Procedures:

1. Turn Data I/O power off.
2. Connect the Data I/O programmer to the modem and VT100 terminal as shown in Fig. A.1-1.
3. Turn the Data I/O programmer on.
4. Press the "SELECT" (Data I/O).

5. Enter "EB"(Data I/O)
6. Press the "START"(Data I/O).
7. Type "TY FILENAME.DAT" (VT100).
8. Press "RETURN" on the VT100.
9. Disconnect VT-100 terminal from the modem and Data I/O.
10. Reconnect Data I/O to VT-100 as shown in Fig. A.1-2.
11. Press the "SELECT" (Data I/O).
12. Press the "SELECT" (Data I/O).
13. Enter "E1" (Data I/O).
14. Press "START" (Data I/O).
15. Use VT-100 keyboard in order to communicate with the Data I/O.

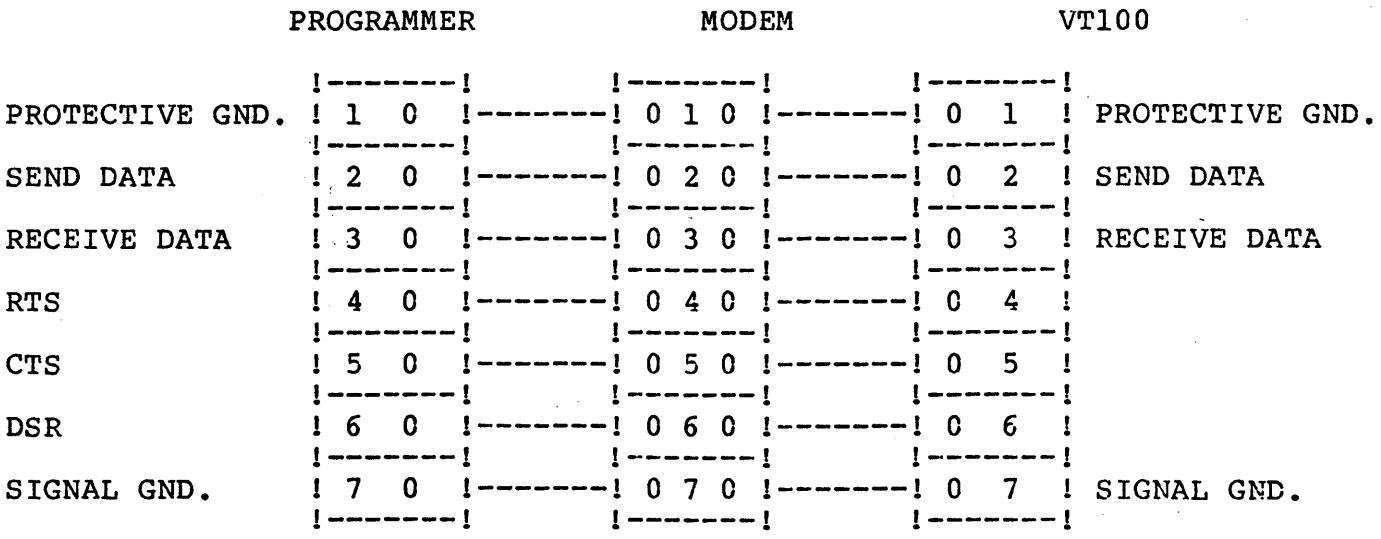


Fig. A.1-1: Downloading from host (VAX-11) to Programmer. (VAX talking to Programmer and VT100).

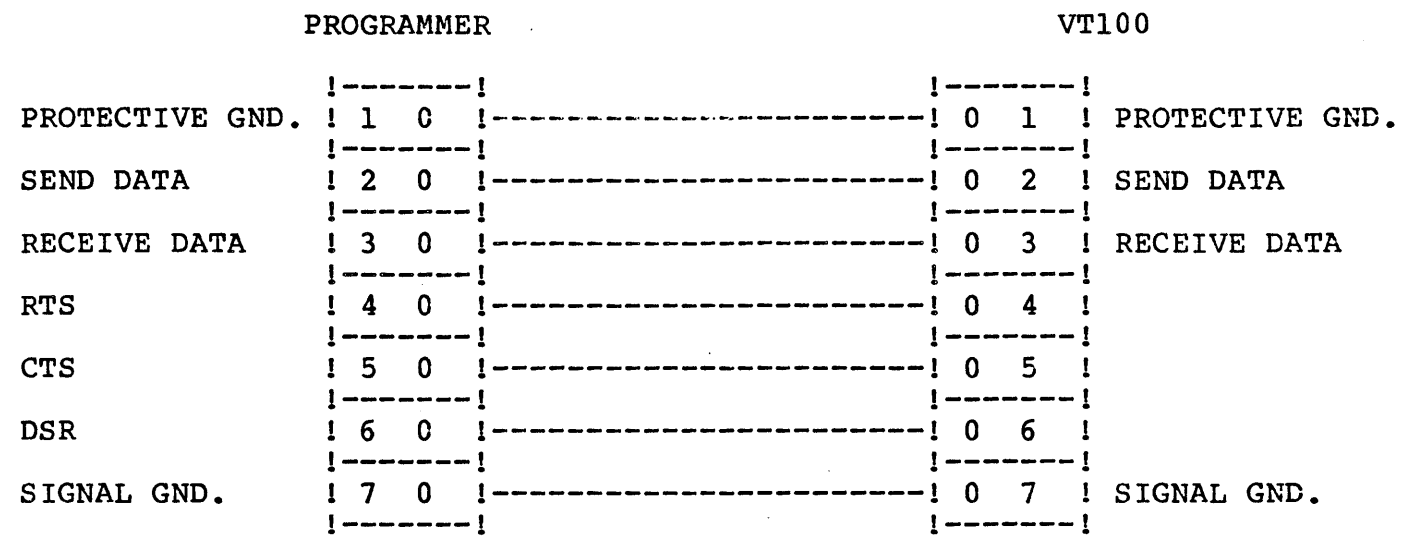


Fig. A.1-2: Using Programmer as host.

Please note that for the IBM PC, system requirements are as follows:

- 8088 based microprocessor system
- 64K bytes minimum of memory
- MS-DOS (PC-DOS) operating system
- Optional text printer
- 1 disk drive.

The IBM PC version comes with a diskette which contains the following files:

Disk #1: PLEASM.EXE (if the Load/Go system has been ordered)
PLEASM.FOR (if the Development system was ordered)
P5000.TXT - P5017.TXT
PALCOM.EXE & PALSETUP.EXE

PLEASM.EXE is the executable version of PLEASM.
PLEASM.FOR is the FORTRAN source for the program.
P5000.TXT - P5017.TXT contain the example applications.
PALCOM.EXE is the program used for downloading.
PALSETUP.EXE allows the user to specify communications protocol

DISREGARD THIS MARKED SECTION IF YOU HAVE THE LOAD/GO SYSTEM !!
****

To create the executable version for the source program you have to compile and link the source file using any FORTRAN compiler/linker you have at your disposal. The one recommended is the Supersoft FORTRAN compiler which was used during the development and testing of this program. For this compiler the sequence of commands to create the executable file would be (with the compiler/linker in drive B: and the source in drive A:) :

```
A> B:SFOR PLEASM.FOR PLEASM.REL
A> B:CNV PLEASM.REL PLEASM.OBJ/R
A> B:LINK S+SEMU+@PLEASM.RSP,PLEASM,NUL,SFLIB+MLIB,,
```

This creates the executable file PLEASM.EXE.

****

To use PLEASM on the IBM PC, two steps are necessary:

1. Create and edit the PLE Specification File.
2. Run PLEASM.

To run PLEASM with Disk #1 in drive A, type the following when you see the A> prompt:

```
A>PLEASM
```

At this point, PLEASM begins running. For step-by-step instructions on how to use the program, please refer to Section 4.

General:

1. Do NOT terminate the program abnormally by pressing CTRL-C, CTRL-BREAK, etc. when you are sending the output to a file rather that the screen. Use instead the QUIT option to terminate your session. If the session is terminated using CTRL-C, this may result in lost files on your disk. This is because your output file will not have been properly closed.

2. The approximate run-times for the programs vary depending on the size of the PLE and the length of the input specifications file. Simulation takes between 1 and 3 minutes, while generating programming formats could take anywhere between 3 and 10 minutes.

PROGRAMMER VENDOR LIST

Following is a list of MMI approved programmer vendors. Although the list below is mostly for the San Francisco Bay Area offices, you should be able to obtain information about their regional sales offices by calling the telephone numbers listed.

1. Data I/O
   473 Sapena Court/Suite 4
   Santa Clara, CA 95050 (408)727-0641

2. Structured Design
   1700 Wyatt Drive/Suite 3
   Santa Clara, CA 95054 (408)988-0725

3. Kontron
   630 Price Avenue
   Redwood City, CA 94063 (415)361-1012

4. Stag
   528-5 Weddell
   Sunnyvale, CA 94029 (408)745-1991

5. Digelec
   7335 E. Acoma Drive/Suite 103
   Scottsdale, AZ 85260 (602)991-7268

6. Storey Systems
   3213 N. Hwy 67/Suite 103
   Mesquite, TX 75150 (214)270-4135

7. Varix
   122 Spanish Village #608
   Dallas, TX 75248 (214)620-0925

8. Citel
   3060 Raymond Street
   Santa Clara, CA 95050 (408)727-6562

PROM/PLE Programmer Information:

DATA I/O Model 19 Programmer with UniPak:

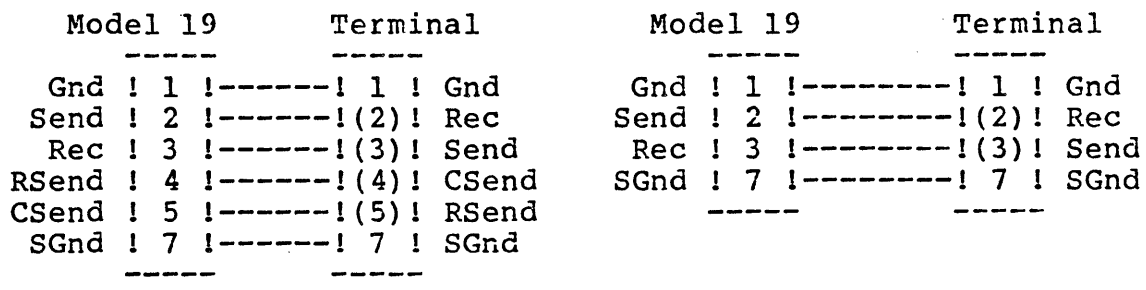
Key Features:

- Accepts PLE HEX programming format.
- Allows user limited communication via RS232 interface to computer development system.

Using the DATA I/O Model 19 with UniPak:

RS232 Serial Interface:

Prior to powering up the DATA I/O it is important that the RS232 interface be connected to the host computer correctly. The correct interface is described below:



Turn on the DATA I/O programmer. After it finishes its self check routine, type the following key sequence on the DATA I/O keyboard:

<LOAD> - Select device type

<SELECT>
<D1> - Prepares DATA I/O to receive data via RS232
<START> interface port

The DATA I/O Model 19 is now ready to accept data from the computer development system. After the file is entered, the data may be reviewed on the terminal by typing the following on the DATA I/O:

<KEYBOARD>
<ENTER> - Sends data to remote terminal

For additional information please refer to the DATA I/O Model 19 users manual.

DATA I/O Model 29A with UniPak:

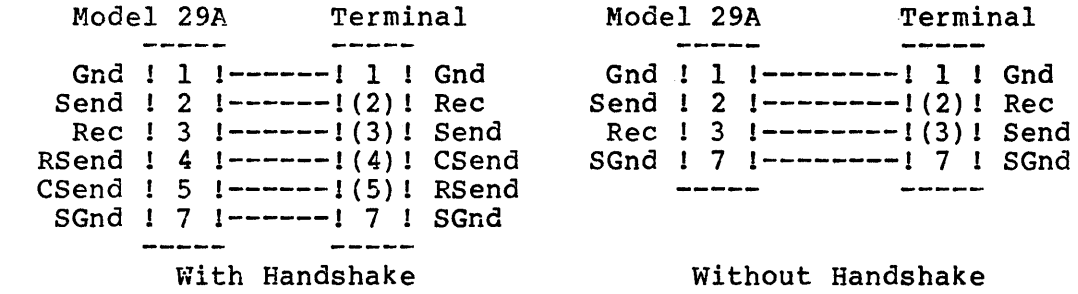
Key Features:

- Accepts PLE HEX programming format.
- Allows interactive communication via RS232 interface to computer development system.

Using the DATA I/O Model 29A with UniPak:

RS232 Serial Interface:

Prior to powering up the DATA I/O it is important that the RS232 interface be connected to the host computer correctly. The correct interface is described below:



Turn on the DATA I/O programmer. After it finishes its self check routine, type the following key sequence on the DATA I/O keyboard:

<COPY> <DEVICE> <RAM> <FFCC> - Enter family and pin code

<SELECT> <F7> - Configure for HEX input format

<COPY> <PORT> <RAM> - Prepares DATA I/O to receive data via RS232 interface port

The DATA I/O Model 29A is now ready to accept data from the computer development system. After the file is entered, the data may be reviewed on the terminal by typing the following on the DATA I/O:

<COPY> <RAM> <PORT> - Send data to remote terminal

Interactive communication may be achieved by typing on the DATA I/O:

<SELECT> <FB> - Enable output port for interactive communication

For additional information please refer to the DATA I/O Model 29A users manual.

DIGELEC Model UP-803 with FAM 12:

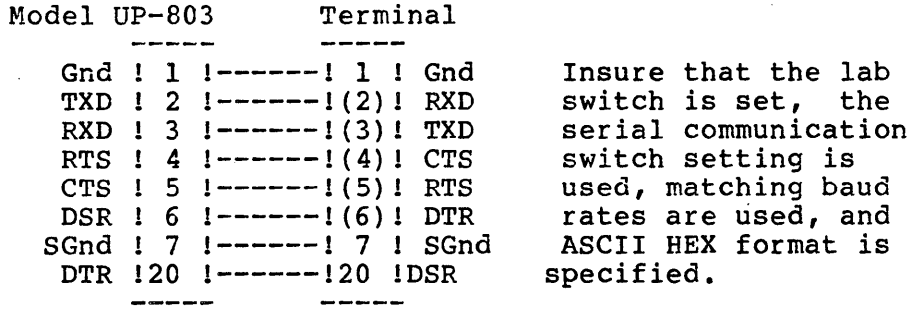
Key Features:

- Accepts PLE HEX programming format.
- Allows limited communication via RS232 interface to computer development system.

Using the DIGELEC Model UP-803 with FAM 12:

RS232 Serial Interface:

Prior to powering up the DIGELEC it is important that the RS232 interface be connected to the host computer correctly. The correct interface is described below:



Turn on the DIGELEC programmer. Prepare the DATA TRANSFER function of the UP-803 as follows:

SOURCE : SERIAL INPUT

DESTINATION : RAM

DESTINATION INITIAL ADDRESS : XXXX

BLOCK LENGTH : YYYY (FFFF if unknown)
Press the <EXECUTE> key on the UP-803.

The DIGELEC Model UP-803 is now ready to accept data from the computer development system. After the file is entered, the data may be reviewed on the remote terminal by selecting the following on the DIGELEC:

SOURCE : RAM

DESTINATION : SERIAL OUTPUT

Press the <EXECUTE> key on the UP-803.

For additional information on the use of the DIGELEC Model UP-803 please refer to the users manual.

KONTRON Model MPP-80S

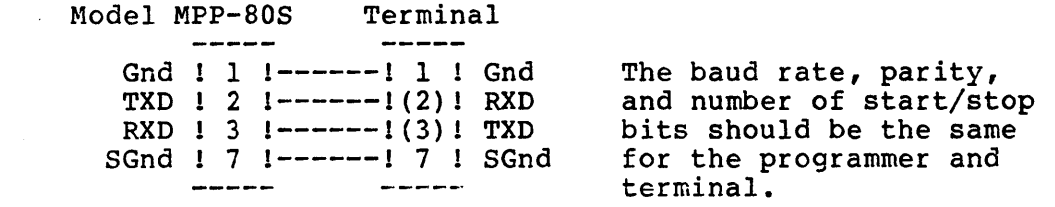
Key Features:

- Accepts PLE Intel HEX programming format.
- Allows limited communication via RS232 interface to computer development system.

Using the KONTRON Model MPP-80S:

RS232 Serial Interface:

Prior to powering up the KONTRON it is important that the RS232 interface be connected to the host computer correctly. The correct interface is described below:



Turn on the KONTRON programmer. Prepare the DATA TRANSFER function of the MPP-80S by performing the following:

- Select proper device type
- Press the white <IN> key (Select input port)
- Press the grey <A> key (Select port A)
- Enter <40> for Intel HEX format
- Press the grey <ENTER> key

The KONTRON Model MPP-80S is now ready to accept data from the computer development system. After the file is entered, the data may be reviewed on the remote terminal by selecting the following on the KONTRON:

- Press the white <OUT> key (Select output port)
- Press the grey <A> key (Select port A)
- Enter output format desired
- Press the grey <ENTER> key

For additional information on the use of the KONTRON model MPP-80S please refer to the users manual.

PLEASM Error messages:

PLEASM detects and reports the following errors encountered while running the program. These include incorrect file naming, syntax errors in the input specifications file, and errors in simulation.

1. Non-existent filename specified in response to the query for the name of the source file.

DISK I/O ERROR - MAYBE WRONG FILENAME ???

****

2. Open file named in response to query for the name of the output file.

DISK I/O ERROR - MAYBE WRONG FILENAME ???

****

3. Input file size in number of characters exceeds the maximum dimensions specified in the initialization subroutine. See Appendix I for details.

TOO MANY CHARACTERS IN INPUT FILE

This will probably lead to a run-time error with an output message appropriate to your system.

****

4. Keyword FUNCTION TABLE missing in input specifications.

*** KEYWORD "FUNCTION TABLE" MISSING. ASSEMBLY TERMINATED

****

5. Keyword DESCRIPTION missing in input specifications.

*** KEYWORD "DESCRIPTION" MISSING. ASSEMBLY TERMINATED

****

6. Invalid PLE name in line 1 of the input specifications.

PLE PART TYPE PLE$$$$ IS INCORRECT

****

7. Number of pin names specified in the address list exceeds the number of input pins available in the PLE being used.

*** TOO MANY PIN NAMES IN INPUT PIN LIST ***

****

8. Number of pin names specified in the data list exceeds the number of output pins available in the PLE being used.

*** TOO MANY PIN NAMES IN OUTPUT PIN LIST ***

****

9. A pin name specified in the equations or in the function table pin list does not match any of the names declared in the address and data pin lists.

ERROR SYMBOL = $$$$$$$$

****

10. The "SIMULATE" option has been invoked without a function table being present in the input specifications.

FUNCTION TABLE MUST BE SUPPLIED IN ORDER TO PERFORM SIMULATION

****

11. The number of pin names in the function table pin list exceeds the number of pins being used in the PLE.

*** TOO MANY PIN NAMES IN FUNCTION TABLE PIN LIST ***

****

12. A symbol other than H(high), L(low), or X(don't care) has been entered in the function table.

ERROR SYMBOL **$** IN LINE $$$ OF FUNCTION TABLE

****

13. Simulation error caused by an entry in the function table (expected) not agreeing with that evaluated from the Boolean equations (actual).

FUNCTION TABLE ERROR IN LINE $$$ PIN = $$$$$$$$ EXPECTED $ ACTUAL $

The offending line in the function table is then printed out with a question mark in place of the incorrect entry.

****

14. Overall count of function table errors if there are one or more simulation errors.

```
ERRORS IN FUNCTION TABLE = $$$$

*****************************************************************
```

15. PLEASM could not generate HEX programming formats for the PLE being used.

```
PLEASM DOES NOT SUPPLY HEX PROGRAMMING FORMAT FOR $$$$ BY $$ PLE'S

*****************************************************************
```

PLE DESIGN LIBRARY

PAT #	TITLE	PLE TYPE
P5000	BASIC GATES	PLE5P8
P5001	MEMORY ADDRESS DECODER	PLE8P8
P5002	6-BIT TRUE/INVERT AND CLEAR/SET LOGIC FUNCTION	PLE8P8
P5003	EXPANDABLE 3-TO-8 DEMULTIPLEXER	PLE5P8
P5004	DUAL 2:1 MULTIPLEXER	PLE10P4
P5005	QUAD 2:1 MULTIPLEXER	PLE10P4
P5006	HEXADECIMAL TO SEVEN SEGMENT DECODER	PLE5P8
P5007	5-BIT BINARY TO BCD CONVERTER	PLE5P8
P5008	4-BIT BCD TO GRAY CODE CONVERTER	PLE5P8
P5009	4-BIT GRAY CODE TO BCD CONVERTER	PLE5P8
P5010	8-BIT PRIORITY ENCODER	PLE8P4
P5011	4-BIT MAGNITUDE COMPARATOR	PLE8P4
P5012	6-BIT MAGNITUDE COMPARATOR	PLE12P4
P5013	4-BIT MULTIPLIER	PLE8P8
P5014	PARTIAL PRODUCTS ADDER	PLE10P4
P5015	PARTIAL PRODUCTS ADDER	PLE8P4
P5016	PARTIAL PRODUCTS ADDER	PLE12P8
P5017	PARTIAL PRODUCTS ADDER	PLE12P8

P5000.TXT
THIS EXAMPLE ILLUSTRATES THE USE OF PLEs TO IMPLEMENT THE BASIC GATES: BUFFER, INVERTER, AND GATE, OR GATE, NAND GATE, NOR GATE, EXCLUSIVE OR GATE, AND EXCLUSIVE NOR GATE.
NOTE ALSO THAT THREE-STATE OUTPUTS ARE PROVIDED WITH ONE ACTIVE LOW OUTPUT ENABLE CONTROL (/E).
PLEASM GENERATES THE PROM TRUTH TABLE FROM THE LOGIC EQUATIONS AND SIMULATES THE FUNCTION TABLE IN THE LOGIC EQUATIONS.

P5001.TXT
THIS PLE8P8 PROVIDES A SINGLE CHIP ADDRESS DECODER FOR USE WITH MANY POPULAR 8-BIT MICROPROCESSORS SUCH AS THE Z80 AND 8080. THE FIVE MSB ADDRESS LINES (A11-A15) AND THE MEMORY REQUEST LINE (/MREQ) FROM THE Z80 MICROPROCESSOR ARE DECODED TO PRODUCE EIGHT ACTIVE LOW CHIP ENABLES (/CE1-/CE8) TO SELECT A RANGE OF 2K BYTES FROM A BANK OF EIGHT 2Kx8 STATIC RAMS. THIS BANK OF STATIC RAMS WILL OCCUPY THE LOWEST 16K BYTES OF ADDRESS SPACE LEAVING THE UPPER 48K BYTE SPACE AVAILABLE FOR OTHER MEMORIES AND I/O. THE PLE8P8 HAS THREE ADDITIONAL INPUTS WHICH CAN BE RESERVED FOR FUTURE SYSTEM EXPANSION.

P5002.TXT
THIS PLE8P8 IS A 6-BIT TRUE/COMPLEMENT AND CLEAR/SET LOGIC FUNCTIONS. THE CONTROL LINES (I1 AND I2) SELECT ONE OF FOUR LOGIC FUNCTIONS FOR THE 6-BIT INPUT DATA (D1-D6) AND THE 6-BIT OUTPUT FUNCTION (Y1-Y6). WHEN I1 IS FALSE (I1=LOW) THE FUNCTION IS INVERT IF I2 IS FALSE (I2=LOW) OR TRUE IF I2 IS TRUE (I2=HIGH). WHEN I1 IS TRUE (I1=HIGH) THE FUNCTION IS CLEAR IF I2 IS FALSE (I2=LOW) OR SET IF I2 IS TRUE (I2=HIGH).
THE PLE8P8 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE LOW OUTPUT ENABLE CONTROLS (/E1 AND /E2).

I1	I2	D1-D6	Y1-Y6	OPERATION
L	L	D	/D	INVERT
L	H	D	D	TRUE
H	L	X	H	CLEAR
H	H	X	L	SET

P5003.TXT
THIS PLE5P8 IMPLEMENTS A 3-TO-8 DEMULTIPLEXER. THE DEVICE WILL DEMULTIPLEX THREE SELECT SIGNALS (S2-S0) INTO EIGHT OUTPUTS (Y7-Y0) USING THE INPUT DI WITH POLARITY SELECT PO. THIS CHIP ALSO HAS CASCADABLE CAPABILITY AND THREE-STATE OUTPUTS.
PIN ASSIGNMENTS:

1. PO HIGH INDICATES OUTPUT IS ACTIVE HIGH. LOW INDICATES OUTPUT IS ACTIVE LOW.
2. DI DATA INPUT (DEMULTIPLEXING INPUT). ACTIVE LOW IF PO IS LOW.

3. S2-S0 SELECT PINS. S2 IS THE MOST SIGNIFICANT BIT. ACTIVE HIGH REGARDLESS OF PO.
4. Y7-Y0 OUTPUTS. CAN BE ACTIVE HIGH OR ACTIVE LOW DEPENDING ON PO. ACTIVE LOW IF PO IS LOW.

P5004.TXT

THE DEVICE WILL SWITCH BETWEEN TWO PAIRS OF 2-BIT INPUTS (A, B AND C, D), AS DETERMINED BY THE TWO SELECT LINES (SX, SZ), FOR OUTPUT THROUGH TWO PAIRS OF 2-BIT OUTPUTS (X AND Z). THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH TWO ACTIVE LOW ENABLE PINS (/E1 AND /E2). THE FUNCTIONS OF THE DEVICE ARE SUMMARIZED IN THE TABLE BELOW:

SELECT LINES		INPUT A, B				INPUT C, D					OUTPUT X, Z				
S X	S Z	A 1	A 2	B 1	B 2	C 1	C 2	D 1	D 2	!	X 1	X 2	Z 1	Z 2	FUNCTION
L	L	A1	A2	X	X	C1	C2	X	X	!	A1	A2	C1	C2	SELECT A, B
L	H	A1	A2	X	X	X	X	D1	D2	!	A1	A2	D1	D2	SELECT A, D
H	L	X	X	B1	B2	C1	C2	X	X	!	B1	B2	C1	C2	SELECT B, C
H	H	X	X	B1	B2	X	X	D1	D2	!	B1	B2	D1	D2	SELECT B, D

P5005.TXT

THIS IS AN EXAMPLE OF A QUAD 2-TO-1 MULTIPLEXER IMPLEMENTED IN A PLE10P4. THE DEVICE FUNCTION IS TO SWITCH BETWEEN TWO 4-BIT INPUTS (A1-A4 AND B1-B4) TO ONE 4-BIT OUTPUT (Z1-Z4) AS DETERMINED BY TWO INPUT SELECT LINES (S2,S1). THE PLE10P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE LOW ENABLE PINS (/E1 AND /E2). THE FUNCTION IS SUMMARIZED BELOW:

SELECT LINES		INPUT A				INPUT B					OUTPUT Z				
S 1	S 2	A 1	A 2	A 3	A 4	B 1	B 2	B 3	B 4	!	Z 1	Z 2	Z 3	Z 4	FUNCTION
L	L	X	X	X	X	X	X	X	X	!	L	L	L	L	SELECT ALL LOWS
L	H	A1	A2	A3	A4	X	X	X	X	!	A1	A2	A3	A4	SELECT A1-A4
H	L	X	X	X	X	B1	B2	B3	B4	!	B1	B2	B3	B4	SELECT B1-B4
H	H	X	X	X	X	X	X	X	X	!	H	H	H	H	SELECT ALL HIGHS

P5006.TXT
THIS EXAMPLE ILLUSTRATES THE USE OF A PLE5P8 AS A HEXADECIMAL TO SEVEN SEGMENT DECODER. THE DEVICE DECODES A 4-BIT BINARY INPUT (D,C,B,A) INTO THE SEVEN SEGMENT OUTPUTS NEEDED TO DRIVE AN LED DISPLAY. NOTE THAT THIS DESIGN IS AN IMPROVEMENT FROM THE 74LS47 SINCE ALL SIXTEEN HEXADECIMAL DIGITS (0-F) CAN BE DISPLAYED. A LAMP TEST IS PROVIDED TO ILLUMINATE ALL SEVEN SEGMENTS AND THE DECIMAL POINT (IF DP IS CONNECTED) BY BRINGING LAMP TEST HIGH (LT=HIGH) REGARDLESS OF THE OTHER BINARY INPUTS. THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH ONE ACTIVE LOW ENABLE PIN (/E).

DIGIT	LT	INPUT D	C	B	A	SEGMENT ON	OUTPUT DISPLAY
0	L	L	L	L	L	ABCDEF	0
1	L	L	L	L	H	BC	1
2	L	L	L	H	L	ABDEG	2
3	L	L	L	H	H	ABCDG	3
4	L	L	H	L	L	BCDFG	4
5	L	L	H	L	H	ACDFG	5
6	L	L	H	H	L	ACDEFG	6
7	L	L	H	H	H	ABC	7
8	L	H	L	L	L	ABCDEFG	8
9	L	H	L	L	H	ABCFG	9
A	L	H	L	H	L	ABCEFG	A
B	L	H	L	H	H	CDEFG	b
C	L	H	H	L	L	ADEF	C
D	L	H	H	L	H	BCDEG	d
E	L	H	H	H	L	ADEFG	E
F	L	H	H	H	H	AEFG	F
G	H	X	X	X	X	ABCDEFG	8 *

* BLANK TEST OF DISPLAY

P5007.TXT
THIS 5-BIT BINARY TO 2-DIGIT BCD CONVERTER IS IMPLEMENTED IN A PLE5P8. THE DEVICE ACCEPTS A 5-BIT BINARY INPUT (BI) AND CONVERTS THIS INTO TWO 4-BIT BINARY CODED DECIMAL (BCD) OUTPUTS (B1 AND B0). THREE-STATE OUTPUTS ARE ALSO PROVIDED WITH ONE ACTIVE LOW ENABLE PIN (/E).

P5008.TXT
THIS PLE5P8 WILL CONVERT A 4-BIT BCD INPUT (B3-B0) INTO A 4-BIT GRAY CODE REPRESENTATION (G3-G0) FOR OUTPUT.

P5009.TXT
THIS PLE5P8 WILL CONVERT A 4-BIT GRAY CODE INPUT (G3-G0) INTO A 4-BIT BINARY REPRESENTATION (B3-B0) FOR OUTPUT.

P5010.TXT
THIS 8-BIT PRIORITY ENCODER SCANS FOR THE FIRST HIGH INPUT LINE (I7-I0) FROM I7 (WHICH HAS THE HIGHEST PRIORITY) TO I0 (WHICH HAS THE LOWEST PRIORITY). IT WILL GENERATE A BINARY ENCODED OUTPUT (S2-S0) WHICH WILL POINT TO THE HIGHEST PRIORITY INPUT WHICH IS AT A HIGH STATE. IF NO INPUT LINES ARE HIGH (I7-I0=LOW), THEN THE BINARY ENCODED OUTPUTS WILL BE ZERO (S2-S0=LOW) AND THE ENABLE OUTPUT WILL BE HIGH (EN=HIGH) INDICATING A CARRY OUT TO THE NEXT PRIORITY ENCODER. THE OUTPUT ENABLE WILL BE LOW (EN=LOW) IF ANY OF THE INPUT LINES ARE HIGH. THE PLE8P4 ALSO HAS THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

P5011.TXT
THIS PLE8P4 COMPARES TWO 4-BIT NUMBERS (A3-A0 AND B3-B0) TO ESTABLISH IF THEY ARE EQUAL (A = B => EQ=H), NOT EQUAL (A NOT = B => NE=H), LESS THAN (A < B => LT=H), OR GREATER THAN (A > B => GT=H) AND REPORTS THE COMPARISON STATUS ON THE OUTPUTS (EQ, NE, LT, GT) AS ILLUSTRATED IN THE OPERATIONS TABLE BELOW.
THE PLE8P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

INPUT NUMBERS A3-A0	B3-B0	COMPARISON STATUS EQ	NE	LT	GT	OPERATION
A	= B	H	L	L	L	COMPARE A EQUAL TO B
A NOT	= B	L	H	X	X	COMPARE A NOT EQUAL TO B
A	< B	L	H	H	L	COMPARE A LESS THAN B
A	> B	L	H	L	H	COMPARE A GREATER THAN B

P5012.TXT
THIS PLE12P4 COMPARES TWO 6-BIT NUMBERS (A5-A0 AND B5-B0) TO ESTABLISH IF THEY ARE EQUAL (A = B => EQ=H), NOT EQUAL (A NOT = B => NE=H), LESS THAN (A < B => LT=H), OR GREATER THAN (A > B => GT=H) AND REPORTS THE COMPARISON STATUS ON THE OUTPUTS (EQ, NE, LT, GT) AS ILLUSTRATED IN THE OPERATIONS TABLE BELOW.
THE PLE12P4 ALSO FEATURES THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

INPUT NUMBERS A5-A0	B5-B0	COMPARISON STATUS EQ	NE	LT	GT	OPERATION
A	= B	H	L	L	L	COMPARE A EQUAL TO B
A NOT	= B	L	H	X	X	COMPARE A NOT EQUAL TO B
A	< B	L	H	H	L	COMPARE A LESS THAN B
A	> B	L	H	L	H	COMPARE A GREATER THAN B

P5013.TXT
THIS PLE8P8 PERFORMS 4-BIT LOOK-UP TABLE MULTIPLICATION. THE DEVICE ACCEPTS TWO 4-BIT OPERANDS (X3-X0 AND Y3-Y0) TO PRODUCE THE 8-BIT PRODUCT (S7-S0). THE PLE8P8 ALSO HAS THREE-STATE OUTPUTS WITH TWO ACTIVE-LOW OUTPUT ENABLE CONTROL PINS (/E1 AND /E2).

```
                    X3 X2 X1 X0
              X     Y3 Y2 Y1 Y0
          ------------------------
          S7 S6 S5 S4 S3 S2 S1 S0
```

P5014.TXT
THIS PLE10P4 PERFORMS THE PARTIAL PRODUCTS REDUCTION FOR A WALLACE TREE ADDER. FIVE ROWS OF 2-BIT NUMBERS (A1-A0, B1-B0, C1-C0, D1-D0, AND E1-E0) ARE ADDED TO PRODUCE A 4-BIT SUM (P3-P0).

```
              A1 A0
              B1 B0
              C1 C0
              D1 D0
        +     E1 E0
       ------------
       P3 P2 P1 P0
```

P5015.TXT
THIS PLE8P4 PERFORMS THE PARTIAL PRODUCTS REDUCTION FOR A WALLACE TREE ADDER. SEVEN ROWS OF 1-BIT NUMBERS (A, B, C, D, E, F, AND G) ARE ADDED TOGETHER TO PRODUCE A 3-BIT SUM.

```
                 A
                 B
                 C
                 D
                 E
                 F
          +      G
         ---------
         P2 P1 P0
```

P5016.TXT
THIS PLE12P8 PERFORMS THE PARTIAL PRODUCTS REDUCTION FOR A WALLACE TREE ADDER. FOUR ROWS OF 3-BIT NUMBERS (A2-A0, B2-B0, C2-C0, AND D2-D0) ARE ADDED TOGETHER TO PRODUCE A 5-BIT SUM (P4-P0).

```
        A2 A1 A0
        B2 B1 B0
        C2 C1 C0
  +     D2 D1 D0
-----------------
P4 P3 P2 P1 P0
```

P5017.TXT
THIS PLE12P8 PERFORMS THE PARTIAL PRODUCTS REDUCTION FOR A WALLACE TREE ADDER. THREE ROWS OF 4-BIT NUMBERS (A3-A0, B3-B0, AND C3-C0) ARE ADDED TOGETHER TO PRODUCE A 6-BIT SUM.

```
        A3 A2 A1 A0
        B3 B2 B1 B0
  +     C3 C2 C1 C0
--------------------
P5 P4 P3 P2 P1 P0
```

HELP!! And where to get it:

Should you encounter any problems with PLEs or PLEASM, write or call:

The IdeaLogic Exchange
Mail-Stop (08-26)
Monolithic Memories Inc.
2175 Mission College Blvd.
Santa Clara, CA 95050

Tel: (408)970-9700

Input/Output:

Prior to running the program, the user has access to an I/O package which can be used to specify I/O unit numbers and array sizes without having to recompile the program. The variables asscciated with this I/O routine are explained below....

CPG(6000) - Maximum number of characters permitted in the input specifications file
LLN(250) - Maximum number of lines permitted in the input specifications file
CLN(250) - Maximum number of characters permitted per line in the input file
CONINP - The Logical Unit Number for <READ>'s from the console
CONOUT - The Logical Unit Number for <WRITE>'s to the console
FILINP - The Logical Unit Number for <READ>'s from a named file
FILOUT - The Logical Unit Number for <WRITE>'s to a named file

The Data Set Reference Numbers for INPUT and OUTPUT files are then assigned as variables at the begining of the program, as follows....

INPUT	PLE DESIGN SPECIFICATION	RPD=FILINP
INPUT	OPERATION CODES	ROC=CONINP
OUTPUT	ECHO AND TRUTH TABLE	POF=CONOUT
OUTPUT	HEX AND BINARY PROGRAMMING FORMATS	PDF=CONOUT/FILOUT
OUTPUT	PROMPTS AND ERROR MESSAGES	PMS=CONOUT/FILOUT

To perform the customization corresponding to your needs, you need to have access to the source code which means that you should have the Development system. These changes can help reduce the memory requirements and facilitate I/O.

The dimension of arrays CPG, LLN, and CLN can be modifed by making the appropriate changes in the subroutine IOINIT located at the top of the source code. This could be done to reduce memory requirements, since the arrays are statically allocated during compilation.

The logical unit numbers for the console are fixed for any given system and must be changed accordingly. For example, the logical unit number for reading from the console with VAX/VMS FORTRAN is 5 and for writing to the console is 6. These numbers are different with Supersoft FORTRAN used on the IBM-PC.

The logical unit numbers for writing and reading to and from files can be normally allocated to be between 0 and 19 excluding those reserved for the console.

HIGH SPEED BIPOLAR PROMS FIND NEW APPLICATIONS AS PROGRAMMABLE LOGIC ELEMENTS (PLEs)

Vincent J. Coli and Frank Lee
Product Planning and Applications
Monolithic Memories Inc.
2175 Mission College Boulevard
Santa Clara, California 95050
(408) 970-9700

Classic applications for bipolar PROMs include instruction storage for microprogram control store and software for microprocessor programs. However, due to a new design methodology and state of the art performance, PROMs are finding increasing numbers of applications as Programmable Logic Elements (or PLEs). This paper will cover the architecture, applications, and software support for PLEs.

Fuse-Programmable Logic Families

A typical combinatorial Boolean equation can be written in sum-of-product form, which consists of several AND gates summed at an OR gate. In general, a set of combinatorial Boolean equations with n inputs (I0,I1,...,In-1) and produces m outputs (O0,O1,...,Om-1) can be generated through one level of AND gates followed by one level of OR gates. Custom logic functions can be defined using programmable logic.

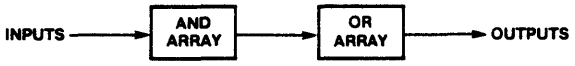


FIGURE 1. Structure of Programmable Logic Devices.

Fuse-programmable devices normally consist of two levels of logic -- AND-array and OR-array -- as suggested above. There are three basic types of fuse-programmable devices -- PROM (Programmable Read Only Memory), PLA (Programmable Logic Array), and PAL (Programmable Array Logic). Which arrays are fuse programmable distinguish these three types of devices.

PLAs offer the greatest flexibility since both the AND and OR arrays are programmable. This flexibility comes with the cost of lower performance, higher power dissipation, and generally higher price.

A PAL device has only the AND-array programmable; the OR array is fixed. Each output has an OR gate associated with it which sums a fixed number of product terms (AND combinations). Statistically there is only a limited number of product terms in any equation. So the flexibility of a PLA is normally not needed. This is a compromise between flexibility and cost and performance.

The OR-array is programmable in a PROM, but the fixed AND-array consists of all combinations of literals for each of the input variables. For example, there are 32 product terms available in a PROM with 5 inputs a,b,c,d,e (corresponding to words 0 through 31 in the PROM memory):

```
/a*/b*/c*/d*/e        (Word  0)
/a*/b*/c*/d* e        (Word  1)
/a*/b*/c* d*/e        (Word  2)
      .                   .
      .                   .
      .                   .
 a* b* c*/d* e        (Word 29)
 a* b* c* d*/e        (Word 30)
 a* b* c* d* e        (Word 31)
```

where '*' represents the Boolean AND operator and '/' represents the Boolean NOT or inverter operator. The fuses in the OR-array are programmed to select the desired AND combinations.

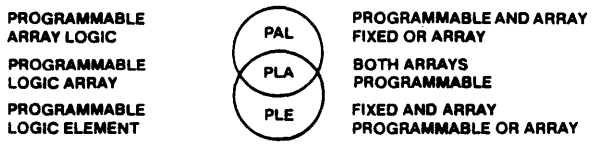


FIGURE 2. Structural differences between PLA, PAL, and PLE (PROM). Note that the PALs and PLEs complement each other. The PAL has many input terms while the PLE is rich in product terms.

The existence of all combinations of literals for all inputs make it possible to define functions which cannot be implemented in a PLA or a PAL. For example, a 5-input Exclusive-OR (XOR) function can be implemented using 16 product terms. This may exceed the number of product terms available in a PAL and will consume too many product terms in a PLA, but can be constructed quite efficiently in a PROM. It is important to realize that any combination of inputs can be decoded in a PROM as long sufficient input pins are provided since a PROM provides 2^n product terms (where n is the number of inputs). Another way of looking at this is that PROMs store the logic transfer function in a memory. The fixed AND-array (or AND-plane) consists of the row and column decoders while the fuses in the OR-array (or OR-plane) are the bits in the memory. In a memory, a fuse blown versus a fuse intact distinguishes a HIGH from a LOW.

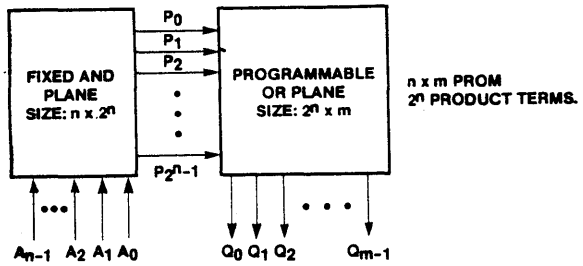


FIGURE 3. Block diagram of a PROM viewed as a PLE. Notice that the PLE provides many (2^n where n is the number of inputs) product terms. A by-product of this is programmable output polarity; either Active High or Active Low output polarity is available.

Due to this special characteristic of abundant product terms, PROMs are also often used as logic devices. In this paper, PROMs are referred to as PLEs (Programmable Logic Elements).

Advantages of PLEs

PLEs provide a cost-effective solution for many applications. Here are just some of the advantages of PLEs:

1) Customizable Logic - The designer is limited to standard functions if SSI/MSI devices are used. The designer can create his own logic chips using PLEs.

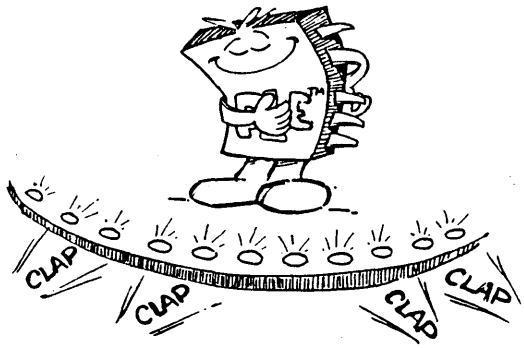
2) Design Flexibility - Modification of design is possible even without redesigning the PC board. For example, the address space of a microprocessor-based system can be reconfigured by merely programming a new PLE if the decoding is implemented in PLE. This feature comes in handy if you want to upgrade a system which originally uses 64K Dynamic RAMs to now use state-of-the-art 256K Dynamic RAMs.

3) Reduce Errors - Errors are sometimes unavoidable and often times quite expensive. Programmable devices make it easier and less expensive to correct errors.

4) Reduction of Printed Circuit Board Space - PLEs save PC board space since several SSI/MSI functions can be integrated into a single package.

5) Fast Turnaround Time - With existing commercial programmers and development software support, a prototype of the custom tailored PLE will be ready in just a few minutes.

A Great Performer!



PLE Applications

Applications of PLEs include random logic replacement, decoder/encoders, code converters, custom ALUs, error detection and correction, look-up tables (both trigonometric and arithmetic), data scaling, compression arithmetic like Wallace Tree adders, distributed arithmetic, and residue arithmetic.

Several levels of random logic chips can be replaced by one PLE. As discussed earlier, PLEs can implement logic in sum of products form.

Despite the existence of dedicated encoders and decoders, many of these functions are application dependent. A standard 3-to-8 decoder/demultiplexer (74S138) can be used for decoding applications. But the decoding scheme may require several 3-to-8 decoder/demultiplexers and additional SSI OR-gates. On the other hand, a PLE can be customized to perform the required decoding function with no additional gates. Simple decoders, such as those used for decoding memory chip selects from address lines can be implemented in a PLE with 5 to 10 inputs. More complex decoding may require 8 to 12 inputs.

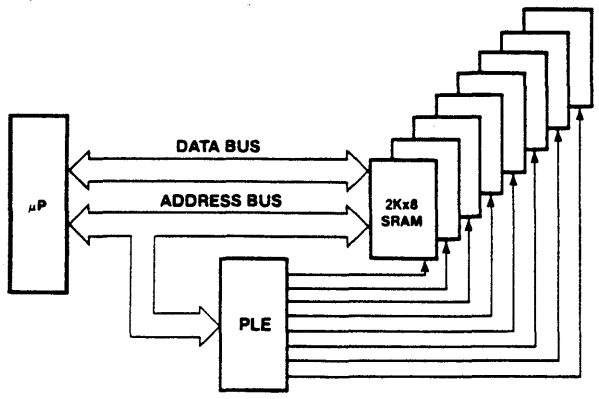


FIGURE 4. PLE address decoding application. The PLE select one of eight 2Kx8 Static RAMs by decoding several microprocessor address lines.

PLEs offer a very flexible solution for code conversion applications. Translations of codes such as from ASCII to EBCIDIC, Binary to BCD (Binary Coded Decimal), or BCD to Gray code can be implemented in PLEs. The 74S184 Binary-To-BCD Converter is actually a 32x8 PROM.

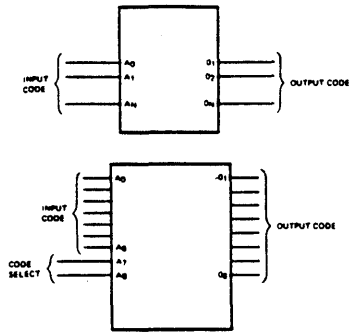


FIGURE 5. Two examples of PLE code converters. The second example illustrates how to use two inputs as code select lines so that four converters can be provided in one PLE.

Standard ALUs (such as the 74S181) may not provide a very specialized function which a particular system requires. In this case a PLE is again a good alternative. Although the PLE may be slower than a dedicated ALU, the presence of this specialized function is critical. For example, a 4-bit ALU can be constructed in a PLE with 12 inputs (A3-A0, B3-B0, I3-I0, Cin) and 8 outputs (F3-F0, /G, /P, Cout, A=B). Any 8 functions can be implemented.

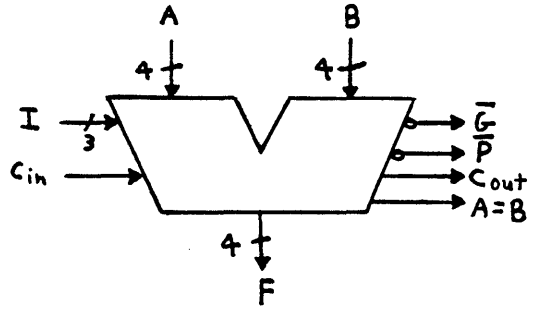
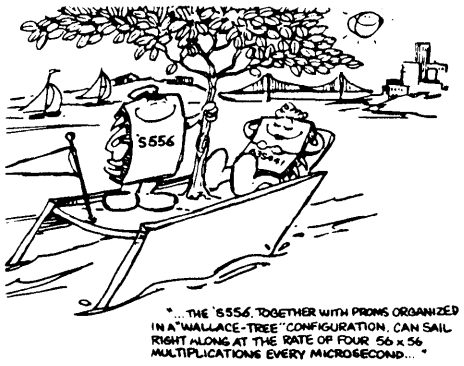


FIGURE 6. Block diagram for a 4-bit ALU which can be implemented in PLE.

Data scaling is another PLE application. A dedicated multiplier is not required if the scaling factor is a constant; the prescaled result can be stored in a PLE. Fixed-bit multipliers are typically implemented in PLEs.

Column compression technique (also called Wallace Tree Compression) is used when expanding an array of several smaller parallel multipliers to perform large wordlength multiplication. These smaller multipliers will generate partial products (intermediate results) which must be added together according to bit significance in order to calculate the final wordlength multiplication. Many levels of 2-input bus adders can be used to add these partial products, but the carry propagation delays may be too long. However, partial product adders implemented in PLEs can do compression of many levels without passing carries. Thus, the summation will be much faster.



Group Code Recoder (GCR) is an encoding/decoding scheme used for error detection on disk. During a WRITE operation, each 8-bit word is divided into two 4-bit nibbles. Both nibbles are then encoded into 5-bit codes before being recorded onto disk. Both 5-bit codes are decoded back to the original 4-bit nibbles and then combined during a READ operation. PLEs are exceptionally useful in mapping the 4-bit data to the 5-bit code and back.

Exclusive-OR gates, being half adders, are very prevalent in Error Detection and Correction (EDC) schemes. Many SSI chips are required to implement this function while PLAs and PALs may not provide sufficient product terms. PLEs are again an ideal solution.

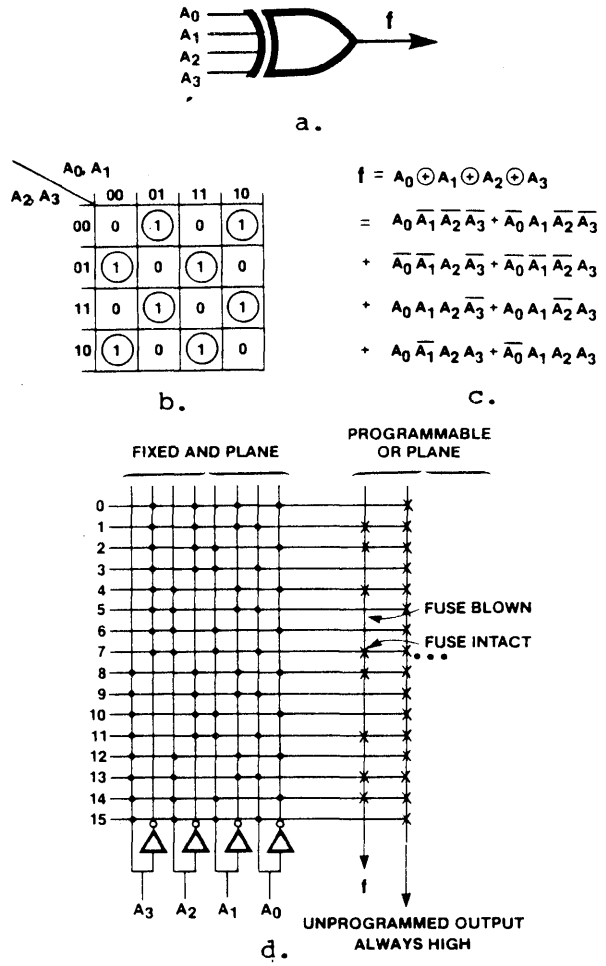


FIGURE 8. Exclusive-OR gates can be implemented in PLEs very efficiently. A 4-input XOR gate (a) maps into a checkerboard pattern in a Karnaugh Map (b) and requires eight products terms (c). The PLE implementation is shown in (d). An 8-input XOR gate requires sixteen product terms.

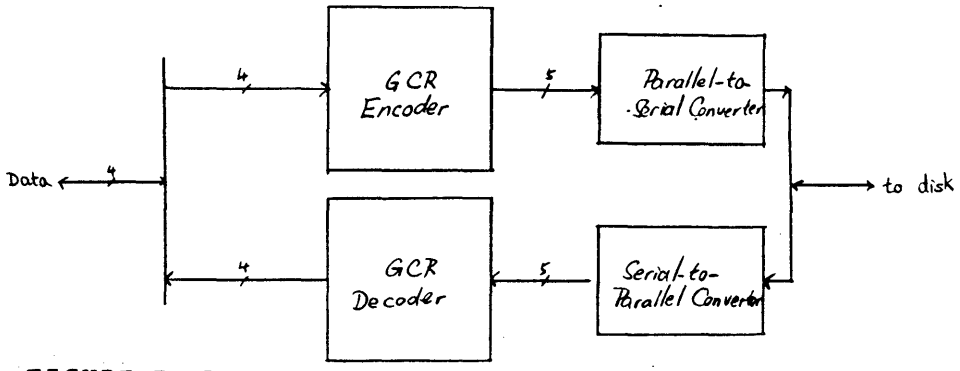


FIGURE 7. GCR encoder/decoder block diagram.

In many applications, the speed of the converging series used to generate the trigonometric functions is too slow and the accuracy obtained by direct look-up requires too much hardware. A good compromise between speed and hardware is to store an approximation to the function in a PLE. Then use this approximation as a starting point for an iterative algorithm (such as Newton-Raphson) to obtain additional accuracy. High-speed division, multiplication, and square root calculations can be performed in a similar manner.

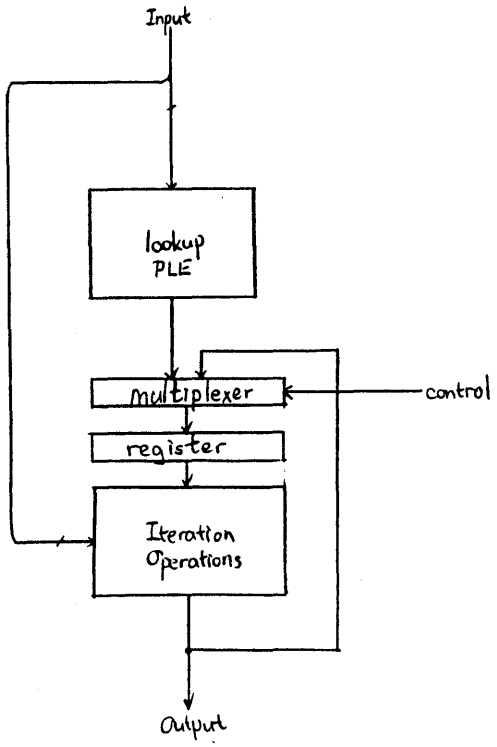


FIGURE 9. PLE look-up tables and iteration loops can be used to generate very accurate trigonometric and arithmetic functions. An approximation to the function is stored in a PLE and additional accuracy is gained using iteration operations.

Distributed arithmetic is used for performing convolution operations without using multiplier/accumulators. If the coefficients are constant, a look-up table for convolution can be stored in a PLE, thus replacing the multiplier.

Residue arithmetic (also called Carry-Independent arithmetic) is a technique used to perform very fast integer arithmetic. High speed is achieved by using numbers in residue representation so that the sequential delay of carries on digits of higher significance is eliminated. A residue numbering system (RNS) is determined using an optimum moduli when designing the system. Conversion to and from residue representation are basic mapping functions which can be conveniently done in PLE. Also, since operations in residue arithmetic are performed using modulo addition and multiplication without carries, these operations can also be done using PLEs. In general, residue arithmetic should only be used for arithmetic which requires intensive operations.

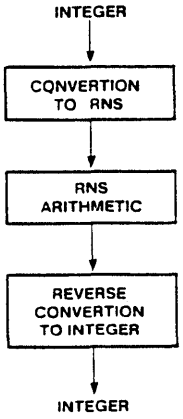


FIGURE 10. Architecture of a system based on RNS. An integer number is converted to RNS representation using PLEs, then the RNS arithmetic can be performed using PLEs, and finally the RNS result is converted back to integer representation again using PLEs.

Detailed application sessions on these topics are included in the Monolithic Memories PLE Handbook.

Restrictions

The basic restrictions for using PLEs to replace SSI/MSI parts are:

1) Since a memory element has a product term for every combination of literals of all the input terms, static hazard is normally unavoidable. For example, there are 5 inputs available in a 32x8 PROM. In order to generate a function like:

$$f = a^* b^* c^* d$$

The actual implementation inside the PROM will be:

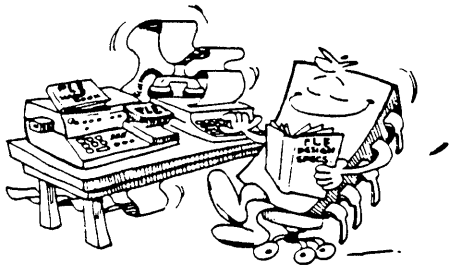
$$f = a^* b^* c^* d^*/e + a^* b^* c^* d^* e$$

If a=b=c=d=HIGH, according to the first equation, we shall expect f to remain HIGH independent of e changing. In the actual PROM implementation, there will be no hazard if e stays either HIGH or LOW. But if e changes, depending on whether e or /e will occur first, there exists the possibility that both product terms in the second equation will be LOW momentarily, which may cause a static logic hazard (HIGH to LOW to HIGH) for f. This hazard is commonly called a "glitch". Static hazards are not a problem for many applications, like those offered in this paper, but extreme care must be taken if the output of a PLE is used to strobe another device.

ADDRESS	e	d	c	b	a	f	
00	0	0	0	0	0	0	
01	0	0	0	0	1	0	
02	0	0	0	1	0	0	
				.			
				.			
				.			
0C	0	1	1	0	0	0	
0D	0	1	1	0	1	0	
0E	0	1	1	1	0	0	
0F	0	1	1	1	1	1	<=
10	1	0	0	0	0	0	
11	1	0	0	0	1	0	
12	1	0	0	1	0	0	
				.			
				.			
				.			
1D	1	1	1	0	1	0	
1E	1	1	1	1	0	0	
1F	1	1	1	1	1	1	<=

FIGURE 11. This Truth Table graphically illustrates the possible glitch (HIGH to LOW to HIGH hazard) for the function f = a * b * c * d implemented in a 32x8 PROM. Address 0F and 1F contain a 1 while all other locations contain a 0 for output f. If address input e should change, the PROM decoders could momentarily select a location containing a 0.

2) Although PROMs (or PLEs) are available with registered outputs, internal feedback from the outputs and buried registers are not yet available in PROMs. External connections from some outputs to inputs must be made for applications which require feedback (such as in state machines). However Registered PROMs without feedback are useful for pipelining (overlap instruction fetch and execution) in order to increase system throughput.



PLEASM Software Support

Monolithic Memories has developed a software tool to assist in designing and programming PROMs as PLEs. This package called "PLEASM" (PLE Assembler) is available for several computers including the VAX/VMS and IBM PC/DOS. PLEASM converts design equations (Boolean and arithmetic) into truth tables and formats compatible with PROM programmers. A simulator is also provided to test a design using a Function Table before actually programming the PLE. The PLEASM operators are listed below and the PLEASM catalog of operations is given on the next page. A sample PLE Design Specification (source code for PLEASM) with PLEASM outputs is given in FIGURE 12. PLEASM may be requested through the Monolithic Memories IdeaLogic Exchange.

;	Comment follows
.	Dot operator (Key operator or arithmetic follows)
ADD	Address pin names follow (inputs)
DAT	Data pin names follow (outputs)
,	Deliminator, separates binary bits (MSB first)
=	Equality (nonregistered)
:=	Equality (registered, replaced by after the clock)
/	Complement, prefix to a pin name
*	AND (product)
+	OR (sum)
:+:	XOR (Exclusive OR)
:*:	XNOR (Exclusive NOR) multiplication)

```
PLE5P8                                          PLE DESIGN SPECIFICATION
P5000                                              VINCENT COLI 10/03/82
BASIC GATES
MMI SANTA CLARA, CALIFORNIA
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8


O1 =  I0                                        ; BUFFER

O2 = /I0                                        ; INVERTER

O3 =  I0  *  I1  *  I2  *  I3  *  I4            ; AND GATE

O4 =  I0  +  I1  +  I2  +  I3  +  I4            ; OR GATE

O5 = /I0  + /I1  + /I2  + /I3  + /I4            ; NAND GATE

O6 = /I0  * /I1  * /I2  * /I3  * /I4            ; NOR GATE

O7 =  I0 :+: I1 :+: I2 :+: I3 :+: I4            ; EXCLUSIVE OR GATE

O8 =  I0 :*: I1 :*: I2 :*: I3 :*: I4            ; EXCLUSIVE NOR GATE


FUNCTION TABLE

I0 I1 I2 I3 I4 O1 O2 O3 O4 O5 O6 O7 O8

;INPUT   - -  OUTPUTS  FROM  BASIC  GATES  - -
;01234   BUF  INV  AND  OR  NAND  NOR  XOR  XNOR   COMMENTS
-------------------------------------------------------------------------
 LLLLL    L    H    L   L    H     H    L    L     ALL ZEROS
 HHHHH    H    L    H   H    L     L    H    H     ALL ONES
 HLHLH    H    L    L   H    H     L    H    H     ODD CHECKERBOARD
 LHLHL    L    H    L   H    H     L    L    L     EVEN CHECKERBOARD
-------------------------------------------------------------------------


DESCRIPTION

THIS EXAMPLE ILLUSTRATES THE USE OF PLEs TO IMPLEMENT THE BASIC GATES:
BUFFER, INVERTER, AND GATE, OR GATE, NAND GATE, NOR GATE, EXCLUSIVE OR
GATE, AND EXCLUSIVE NOR GATE.

NOTE ALSO THAT THREE-STATE OUTPUTS ARE PROVIDED WITH ONE ACTIVE LOW
OUTPUT ENABLE CONTROL (/E).

PLEASM GENERATES THE PROM TRUTH TABLE FROM THE LOGIC EQUATIONS AND
SIMULATES THE FUNCTION TABLE IN THE LOGIC EQUATIONS.
```

FIGURE 12a. PLE Design Specification. This is the source code for PLEASM. PLEASM generates the truth table and programming formats from the equations. PLEASM also exercises the Function Table in the equations and reports errors.

```
BASIC GATES
.ADD I0 I1 I2 I3 I4
.DAT O1 O2 O3 O4 O5 O6 O7 O8

 ADD   A0 A1 A2 A3 A4   O1 O2 O3 O4 O5 O6 O7 O8
-----------------------------------------------
   0   L  L  L  L  L    L  H  L  L  H  H  L  L
   1   H  L  L  L  L    H  L  L  H  H  L  H  H
   2   L  H  L  L  L    L  H  L  H  H  L  H  H
   3   H  H  L  L  L    H  L  L  H  H  L  L  L
   4   L  L  H  L  L    L  H  L  H  H  L  H  H
   5   H  L  H  L  L    H  L  L  H  H  L  L  L
   6   L  H  H  L  L    L  H  L  H  H  L  L  L
   7   H  H  H  L  L    H  L  L  H  H  L  H  H
   8   L  L  L  H  L    L  H  L  H  H  L  H  H
   9   H  L  L  H  L    H  L  L  H  H  L  L  L
  10   L  H  L  H  L    L  H  L  H  H  L  L  L
  11   H  H  L  H  L    H  L  L  H  H  L  H  H
  12   L  L  H  H  L    L  H  L  H  H  L  L  L
  13   H  L  H  H  L    H  L  L  H  H  L  H  H
  14   L  H  H  H  L    L  H  L  H  H  L  H  H
  15   H  H  H  H  L    H  L  L  H  H  L  L  L
  16   L  L  L  L  H    L  H  L  H  H  L  H  H
  17   H  L  L  L  H    H  L  L  H  H  L  L  L
  18   L  H  L  L  H    L  H  L  H  H  L  L  L
  19   H  H  L  L  H    H  L  L  H  H  L  H  H
  20   L  L  H  L  H    L  H  L  H  H  L  L  L
  21   H  L  H  L  H    H  L  L  H  H  L  H  H
  22   L  H  H  L  H    L  H  L  H  H  L  H  H
  23   H  H  H  L  H    H  L  L  H  H  L  L  L
  24   L  L  L  H  H    L  H  L  H  H  L  L  L
  25   H  L  L  H  H    H  L  L  H  H  L  H  H
  26   L  H  L  H  H    L  H  L  H  H  L  H  H
  27   H  H  L  H  H    H  L  L  H  H  L  L  L
  28   L  L  H  H  H    L  H  L  H  H  L  H  H
  29   H  L  H  H  H    H  L  L  H  H  L  L  L
  30   L  H  H  H  H    L  H  L  H  H  L  L  L
  31   H  H  H  H  H    H  L  H  H  L  L  H  H
-----------------------------------------------

HEX CHECK SUM = 00F3C
```

FIGURE 12b. Truth Table. PLEASM generates this truth table which can be used for verifying your design.

```
32 D9 DA 19 DA 19 1A D9 DA 19 1A D9 1A D9 DA 19 .
DA 19 1A D9 1A D9 DA 19 1A D9 DA 19 DA 19 1A CD .

00F3C
```

FIGURE 12c. PLEASM also generates this ASCII Hex programming format with Hex check sum. Control characters are included so that the information can be down loaded directly to a PROM programmer.

ECHO (E) - Prints the PLE Design Specification.
SIMULTE (S) - Exercises the Function Table in the equations.
TRUTH TABLE (T) - Prints the entire truth table.
BRIEF TABLE (B) - Prints only used address in the truth table.
HEX TABLE (H) - Prints the truth table in Hex form.
INTEL HEX (I) - Generates Intel Hex programming format.
ASCII HEX (A) - Generates ASCII Hex programming format.
SHORT HEX (P) - Generates ASCII Hex programming format without spaces.
BHLF (L) - Generates BHLF programming format.
BNPF (N) - Generates BNPF programming format.
CATALOG (C) - Prints the PLEASM catalog of operations.
QUIT (Q) - Exists PLEASM.

PLE Family

Monolithic Memories carries a family of fast PROMs which can be used as Memory or PLE devices. Since the critical parameter for logic applications is speed, our series of fast PROMs have worst-case memory access times (or propagation delays) ranging from 15ns for small PROMs to 40ns for large PROMs. The Logic Symbols for four of the PLEs are given in FIGURE 13 and a summary of the PLE family is given below:

PLE No.	Inputs	Outputs	Type
PLE5P8	5	8	nonreg
PLE8P4	8	4	nonreg
PLE8P8	8	8	nonreg
PLE9P4	9	4	nonreg
PLE9P8	9	8	nonreg
PLE10P4	10	4	nonreg
PLE10P8	10	8	nonreg
PLE11P4	11	4	nonreg
PLE11P8	11	8	nonreg
PLE12P4	12	4	nonreg
PLE12P8	12	8	nonreg
PLE9R8	9	8	reg
PLE10R8	10	8	reg *
PLE11R8	11	8	reg *

nonreg=NonRegistered and reg=Registered
* Several versions available.

Summary

There are many interesting applications for high-speed PROMs are PLEs. A software package called "PLEASM" is available as a development tool.

Acknowledgements

Several of the designs discussed in this paper were proposed by our good friend and colleague Ulrik Mueller, who is now studying Computer Science in his native country Denmark and our MMI Pal Zahir Ebrahim. Special thanks also go to Ranjit Padmanabhan for writing the PLEASM simulator.

References

1. "PLE Programmable Logic Element Handbook", Monolithic Memories, Inc. (available May 1984)

2. "PAL Programmable Array Logic Handbook", 3rd edition, J. Birkner, V. Coli, Monolithic Memories, Inc.

3. "Systems Design Handbook", Monolithic Memories, Inc.

4. "Bipolar LSI 1984 Databook", 5th edition, Monolithic Memories, Inc.

5. "PROMs and PLEs: An Application Perspective", Z. Ebrahim, Monolithic Memories Application Note AN-126.

6. "An Introduction to Arithmetic for Digital Designers", S. Waser, M. J. Flynn, Holt, Rinehart & Winston, N.Y., 1982.

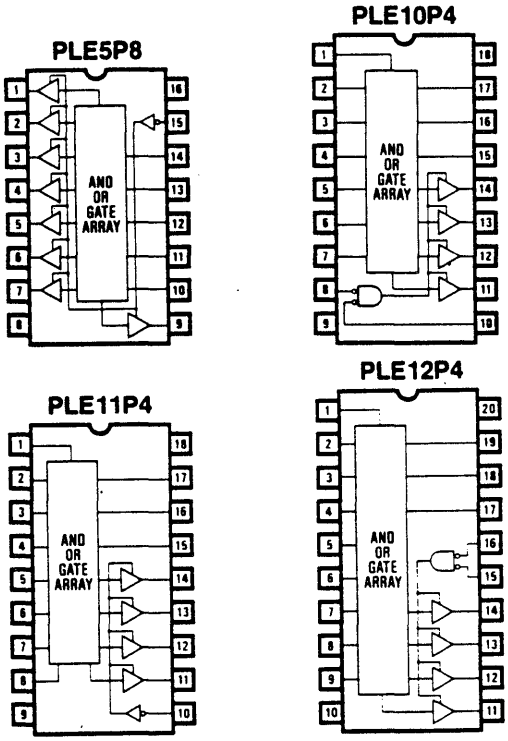


FIGURE 13. Four Sample PLE Logic Symbols.