# ABEL SPECIFICATION

Advanced Boolean Expression Language (ABEL) is a tool the design engineer will use to develop logic systems with programmable logic.

## MAJOR OBJECTIVES

Convert an input description into a logic device fuse pattern.

Support all existing logic devices. Expandable to new devices.

Operate on computer systems available to Data I/O's customers.

## BACKGROUND

Programable Logic Array was developed as an alternative to enormous effort required to design a random logic IC. The logical organization of PLA's consists of an "and" array and an "or" array. (Figure 1a) The designer customizes the PLA to his needs by connecting the desired nodes of the "and" and "or" arrays. This may be done when the device is manufactured or in the field. (Similar to a mask programmable ROM and a fuse-link ROM.)

Field programmable logic consists of three types of devices: PROMS, PALs, and FPLAs. Both arrays are programmable in the FPLA while the PAL has a programmable "or" array and the PROM has a programmable "and" array. (Figure 1) The FPLA is a larger part and can hold more functions while the PAL is simple and fast. Many tasks can use either part. The simplicity of the PAL plus its development tools (PALASM) make it a very attractive part for the first time user.

## EXISTING LOGIC DEVELOPMENT LANGUAGES

There are many different logic development programs: PALASM, H & L, in-house programs , and third party software.

PALASM is the most widely used language for programmable logic. It was developed by Monolithic Memories Inc for use on their PALs. PALASM has become an "industry standard" development tool and is being expanded to handle additional devices such as PROMS. The boolean equation input is translated into a fuse pattern that is transfered to the PAL programmer (Data I/O's LogicPak). This translation is a simple one to one mapping of the boolean equation to the fuse numbers. No logic reduction or design automation is performed. PALASM does simplify the engineers task and is a major part of the PAL's success. Figure 2 shows an example of PALASM input and output.

H & L is a program developed by Data I/O to simplify programming the Signetics Integrated Fuse Logic (IFL). It is a simple text editor with data entry error checking. The input is in the form of a high (H) or low (L) or don't care (X). The created fuse map is then programmed in the device (see example in Figure 3).

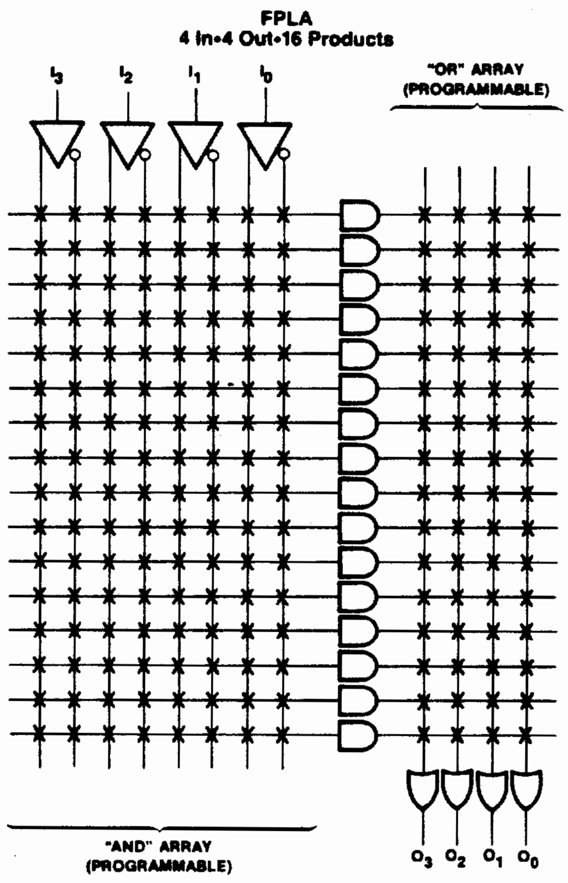
Computer companies and universities have developed in-house languages which aid the programming of digital logic devices. The "in-house" languages of IBM, DEC, etc will remain proprietary but some of the languages developed by universities are in the public domain. The active research programs are directed toward automated VLSI logic design, however the programmable logic is a subset of this work.

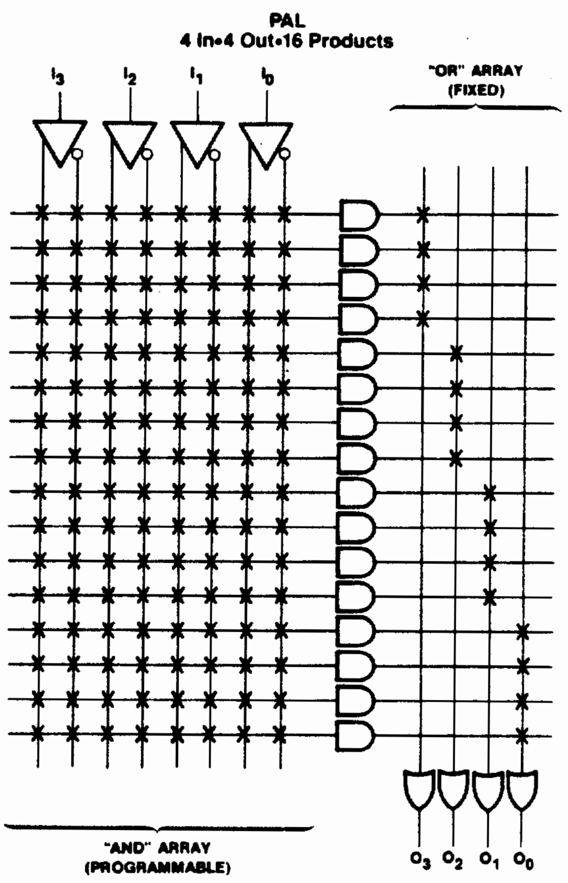
Couple, expected to be released soon by Assisted Technology, Inc., is a universal logic development language similar to PALASM. This language has many additional features, such as macros, and supports FLPA's and FPLS's as well as PAL's. Couple will run on personal computers such as the IBM PC and CP/M systems.

## COMPARISON OF ABEL

PALASM is a widely used language that will be used for comparison to the proposed ABEL language. PALASM was best suited for this comparison because of its industry wide acceptance and the similarities in the targeted marketplace.

Both PALASM and ABEL are languages which translate boolean equations into a fuse map pattern. ABEL however, takes into consideration the behavior of the input. ABEL has a more advanced structure internally and externally. Included in the language is the ability to handle all logic families and handle macro definitions or library functions. ABEL is to be designed in itself more like a high level language (see example in Figure 4).


FPLA
4 In•4 Out•16 Products
I3
I2
I1
I0
"OR" ARRAY (PROGRAMMABLE)
"AND" ARRAY (PROGRAMMABLE)
O3
O2
O1
O0


PAL
4 In•4 Out•16 Products
I3
I2
I1
I0
"OR" ARRAY (FIXED)
"AND" ARRAY (PROGRAMMABLE)
O3
O2
O1
O0

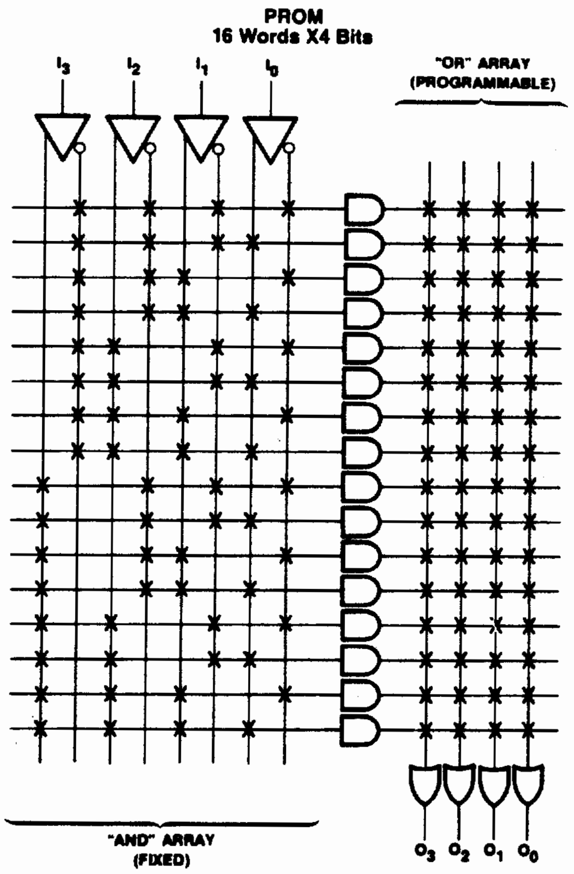

PROM
16 Words X4 Bits
I3
I2
I1
I0
"OR" ARRAY (PROGRAMMABLE)
"AND" ARRAY (FIXED)
O3
O2
O1
O0

FIG 1

```
PAL20L10                                          PAL DESIGN SPECIFICATION
VIDLOG                                              HARRY HUGHES 02/18/81
VIDEO LOGIC
MMI ENGLAND
CB2 CB3 CB4 CB5 CB6 CB7 BLK0 BLK1 G1 H1 H2 GND
H4 EN1 R G B X LOAD1 LOAD0 B0 B1 BLK VCC

IF (VCC) /LOAD0 = /B0* H1* H2* H4

IF (VCC) /LOAD1 =  H1* H2* H4*/B1
                +  H1* H2* H4* B1* B0

IF (VCC) /EN1   =  B1* B0

IF (VCC) /R     =  CB4* BLK1
                +  CB2* BLK0

IF (VCC) /G     =  CB5* BLK1
                +  CB3* BLK0

IF (VCC) /B     =  CB6* BLK1
                + /CB2*/CB3* BLK0

IF (VCC) /X     =  CB7* BLK1

IF (VCC) /BLK   = /G1*/BLK1*/BLK0
                + /G1*/BLK0*/CB4*/CB5*/CB6*/CB7



VIDEO LOGIC

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

 0 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
 1 ---- ---- ---- ---- ---- -X-- -X-- -X-- ---- ---- /G1*/BLK1*/BLK0
 2 ---- -X-- -X-- -X-- -X-- -X-- ---- -X-- ---- ---- /G1*/BLK0*/CB4*/CB5*/CB-


24 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
25 ---- ---- ---X ---- ---- ---- ---- ---- X--- X-X- /B0*H1*H2*H4

32 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
33 ---- ---X ---- ---- ---- ---- ---- ---- X--- X-X- H1*H2*H4*/B1
34 ---- --X- --X- ---- ---- ---- ---- ---- X--- X-X- H1*H2*H4*B1*B0

40 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
41 ---- ---- ---- ---- X--- ---- X--- ---- ---- ---- CB7*BLK1

48 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
49 ---- ---- ---- X--- ---- ---- X--- ---- ---- ---- CB6*BLK1
50 -X-X ---- ---- ---- ---- X--- ---- ---- ---- ---- /CB2*/CB3*BLK0

56 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
57 ---- ---- X--- ---- ---- ---- X--- ---- ---- ---- CB5*BLK1
58 X--- ---- ---- ---- ---- X--- ---- ---- ---- ---- CB3*BLK0

64 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
65 ---- X--- ---- ---- ---- ---- X--- ---- ---- ---- CB4*BLK1
66 --X- ---- ---- ---- ---- X--- ---- ---- ---- ---- CB2*BLK0

72 ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
73 ---- --X- --X- ---- ---- ---- ---- ---- ---- ---- B1*B0

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

NUMBER OF FUSES BLOW =  801
```

FIG 2

```
*A  HHHHHHHH
*P  ØØ     *I  H--H-H-HH---LLLL     *F  A.A.A..A
*P  Ø1     *I  H--H-H-LH---LLLL     *F  A.A..AA.
*P  Ø2     *I  H--H-H-HL---LLLL     *F  A..AA.A.
*P  Ø3     *I  H--H-H-LL---LLLL     *F  .AA.A..A
*P  Ø4     *I  L-----------LLLL     *F  A..AA..A
*P  Ø5     *I  H--HH-------LLLL     *F  A..A.A.A
*P  Ø6     *I  H--H-----------H     *F  .AA..A.A
*P  Ø7     *I  -----L------LLLH     *F  A.A.A.A.
*P  Ø8     *I  ------H-----LLLH     *F  A.A..A.A
*P  Ø9     *I  -----L------LLHL     *F  A.A.A.A.
*P  1Ø     *I  ------H-----LLHL     *F  A.A..A.A
*P  11     *I  -----L------LHLL     *F  A.A.A.A.
*P  12     *I  -H----------LHLL     *F  .A.AA.A.
*P  13     *I  -----L------HLLL     *F  A.A.A.A.
*P  14     *I  -H----------HLLL     *F  .A.AA.A.
*P  15     *I  H--H-H-HL-L-LLHH     *F  A..AA.A.
*P  16     *I  H--H-H-LL-L-LLHH     *F  .AA.A.A.
*P  17     *I  H--HH-----L-LLHH     *F  A..A.A.A
*P  18     *I  H--H------LHLLHH     *F  .AA..A.A
*P  19     *I  ----------H-LLHH     *F  .AA..A.A
*P  2Ø     *I  H--H-H-HH--ØHHLL     *F  A.A.A..A
*P  21     *I  H--H-H-LH---HHLL     *F  A.A..AA.
*P  22     *I  H--HH-------HHLL     *F  A.A..A.A
*P  23     *I  H--H-------HHHLL     *F  .AA..A.A
*P  24     *I  H-----------LHLH     *F  A..A.A.A
*P  25     *I  -H----------LHHH     *F  .A.A.AA.
*P  26     *I  --H---------HHHL     *F  A..A.AA.
*P  27     *I  H--H-H-LH---LHHL     *F  A.A..AA.
*P  28     *I  H--H-H-HH---LHHL     *F  A.A..AA.
*P  29     *I  H--H-H-HL---LHHL     *F  A..AA.A.
*P  3Ø     *I  H--H-H-LL---LHHL     *F  .AA.A.A.
*P  31     *I  H--H-------HLHHL     *F  .AA..A.A
*P  32     *I  -H----------LHHL     *F  .A.AA..A
*P  33     *I  ---------L--HHLH     *F  .AA.A..A
*P  34     *I  L-----------HLLH     *F  A.A.A.A.
*P  35     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  36     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  37     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  38     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  39     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  4Ø     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  41     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  42     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  43     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  44     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  45     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  46     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
*P  47     *I  ØØØØØØØØØØØØØØØØ     *F  AAAAAAAA
```

```
/****************************************/
/* A conceptual ABEL example           */
/*                                     */
/* 06 Dec 1982                         */
/****************************************/

HEADER
 DEVICE WIMP88R199;

 PINS
     MA0, MA1, MA2, A0, A1, A2, A8, A9, A10,
     ROW, HOLD, TOGGLE
 END_PINS;
END_HEADER.

FUNCTION MUX( A, B, SELECT);
    MUX = A * SELECT + B * /SELECT;
END_FUNCTION MUX.

EQUATIONS
    MA0 = MUX( A0, A8,  ROW);
    MA1 = MUX( A1, A9,  ROW);
    MA2 = MUX( A2, A11, ROW);
END_EQUATIONS.


STATE_LIST    /* A SIMPLE STATE MACHINE */
    FIRST;
    SECOND;
    THIRD;
END_STATE_LIST.

STATE_DIAGRAM
    STATE FIRST;
    IF /HOLD THEN SECOND
    ELSE FIRST;             /* LOOP UNTIL HOLD LOW */

    STATE SECOND;
    ALWAYS {  TOGGLE = TRUE ; THIRD; }

    STATE THIRD;
    ALWAYS {  TOGGLE = FALSE ; FIRST; }

END_STATE_DIAGRAM.

FUNCTIONAL_TEST

 TEST_NODES
    A0, A1, A2, A8, A9, A10, ROW, MA0, MA1, MA2;
 END_TEST_NODES.

 VECTORS
    H H H L L L   L  L L L ;
    H H H L L L   H  H H H ;
 END_VECTORS.

END_FUNCTIONAL_TEST.
```

```
                    **********************
                    *                    *
                    * PROBLEM DEFINITION *
                    *                    *
                    **********************

*****************     *****************     *****************
*               *     *               *     *               *
* BOOLEAN       *     *  STATE        *     *  TRUTH TABLE  *
* EQUATIONS     *     *  MACHINE      *     *  TEST VECTORS *
*               *     *               *     *               *
*****************     *****************     *****************

*****************     *****************
*               *     *               *
*  EQUATION     *     *  EQUATION     *
*  TRANSFORM    *     *  CALCULATOR   *
*               *     *               *
*****************     *****************

                      *****************
                      *               *
                      *  EQUATION     *
                      *  REDUCER      *
                      *               *
                      *****************

                      *****************     *****************
                      *               *     *               *
                      *  FUSE         *     *  SIMULATION   *
                      *  TRANSLATOR   *     *               *
                      *               *     *               *
                      *****************     *****************

*****************     *****************     *****************
*               *     *               *     *               *
*  CONTROL      *     *  FUSE         *     *   TEST        *
*  DOCUMENTS    *     *  PATTERN      *     *   VECTORS     *
*               *     *               *     *               *
*****************     *****************     *****************
```

Logic Development flow for ABEL

## DIGITAL DESIGN

R.C. Clare defined the three processes of logic design as DEFINITION, DESCRIPTION, and SYNTHESIS. (Reference __) The DEFINITION phase is the creative portion that the human engineer must perform. The engineer expresses his solution in the DESCRIPTION. The SYNTHESIS phase is a time consuming chore to implement the solution, a task well-suited for automation. The ABEL program will take the engineers description and synthesize the boolean equations and the fuse map for the programmable device.

The present versions of PALASM require the engineer to describe the problem in a restrictive "sum of products" format. This requires considerable effort to transform the DEFINITION into the DESCRIPTION. The level of pain can be reduced by adding MACRO processors and libraries of standard functions but the restrictive equation entry will always hinder the PALASM type of language.

All logic systems can be represented by a state machine, the combinational logic is just a special case (one state looping on itself). The synthesis of the device equations or patterns from a state machine is done in two steps, an equation calculator, and an equation reducer. The equation calculator produces a non-minimum (very non-minimum) set of equations that perform the desired function. These equations must be reduced before they will fit into any device. In the past this reduction required a mainframe computer, but improved algorithms will run on desktop computers in acceptable times. (A reasonable guess would be 10 to 300 seconds on a 16 bit microprocessor for present logic devices.)

## DESCRIPTION

All logic designs can be described with boolean equations, state diagrams, and truth tables. The boolean equations are normally expressed in sum of product form but other forms should be allowed. The state diagrams could be entered graphically but this would be very implementation dependent. Most design languages use the IF THEN ELSE construct to enter state diagrams. The truth table is an effective method for describing the behavior of a system for verfication and simulation.

The description will be aided by predefined and user defined macros and functions. If a logical construct was used often, such as a shift register, the ABEL compiler could pull a predefined set of equations from a library much faster then it could synthesize and reduce the equations from scratch.

ABEL will allow multiple sections of equations and state diagrams in a single file. This will allow the designer to define the problem in several modules (blocks) and then include as many as possible in each device. (Figure __) For example a problem can be defined in 5 blocks but all 5 won't fit into any single logic device. The engineer could partition blocks 1, 2, and 4 into a FPLS and blocks 3 and 5 into a PAL. This partitioning would be defined in the CONTROL block. The partitioning would be done by trial and error.

## EQUATION CACULATOR

Normal compiler techniques would be used to parse the input file into a intermediate form. If a sum of products set of equations were to be used in a PAL, this form could be mapped directly into fuse pattern. This is what PALASM does today. A state diagram would require processing to derive the flip-flop equations and the output tables. This is a straightfoward process. An existing program requires 100 or so lines of code. (Appendix __)

Truth tables or non sum of product equations could be transformed at this stage.

## EQUATION REDUCTION

One equation calculator example produced 50 equations for a 8 state machine that was to go in a PAL16R6. (Appendix __) Some form of logic reduction is required. Svoboda developed two programs, PRESTO and OPTIMA, that are widely used for logic reduction. As their names imply, PRESTO will do a fair job in a short time while OPTIMA finds the absolute minimum set of equations. Douglas Brown of Tektronics developed a State-Machine Synthesizer (SMS) using PRESTO and he claims a microprocessor version is practial. (Appendix __)

ABEL would have the classical compiler trade-off, efficeincy vs convenience. Given enough time and skill an engineer could fit more function into a given logic device than ABLE could.

## FUSE TRANSLATOR

The fuse translator would require a table of the individual device fuse maps. The translator just maps the reduced equations to the corresponding fuses. This is presently done with PALASM and H&L so existing techniques can be used.

## SIMULATION

The designer will describe, via truth table, the correct operation of the programmed device(s). ABEL will compare the operation of the simulated device against the truth table. The internal nodes as well as the external pins may be simulated. Other forms of description (besides truth table) may be added in furture versions.

## CONTROL DOCUMENTS

ABEL would provide the control documents for normal configuration management. This includes a source listing, printed fuse map, and diagram of the device pinout.

## FUSE PATTERNS

The fuse pattern output would conform to the JEDEC data transfer standard (Appendix __).

## TEST VECTORS

The test vector output file would meet the JEDEC standard for data transfer to logic programmers. The output would only include the stimulus and output for external nodes (pins).

## BIBLIOGRAPHY

Ayres,Ron, IC Specification Language, Design Automation Conference, 16, (1979)

Ayres, Ron, Silicon Compilation - A Hierarchical use of PLAs, Design Automation Conference, 16 (1979)

VanCleemput, W. M., Computer Hardware Description Languages and Their Application, Design Automation Conference, 16, (1979)

Bening, Lionel, Developments in computer Simulation of Gate Level Physical Logic, Design Automation Conference, 16, (1979)

Evangelisti, C. T., Goettzel, G., Ofek, H., Designing with LCD: Language for Computer Design, Design Automation Conference, 14, (1977) VanCleemput, W. M., An Hierarchical Language for the Structural Description of Digital Systems, Design Automation Conference, 14, (1977)

Bechtolsheim, Andreas, Interactive Specification of Structured Designs, Design Automation Conference 15, (1978)

McWilliams T., Widdoes L. Jr., SCALD: Structured Computer-Aided Logic Design, Design Automation Conference, 15, (1978)

Brown, Douglas, A state-Machine Synthesizer-SMS, Design Automation Conference, 18, (1981)

Darringer,John, Joyner William Jr., Design Automation Conference, (1980)

Northcutt, Duane, The Design and Implementation of Fault Insertion Capabilities for ISPS, Design Automation Conference, 17, (1980)

Cory, W. E., vanCleemput W. M., Developments in Verification of Design Correctness, Design Automation Conference, 17, (1980)

Cory, W. E., Symbolic Simulation for Functional Verification with ADLIB and SDL, Design Automation Conference, 18, (1981)

Shiva Sajjan, Combinational Logic Syntheses from an HDL Description, Design Automation Conference, 17, (1980)

Hightower, David, Roberts, Martin, Automated Logic Arrays and the Customer Interface, IEEE, (1981) Signetics IFL Development System, Data I/O Corporation, (1982)

Miller, Warren, The Philosophies of Fuse Programmable Logic Advanced Micro Devices

Osann, Robert, A Universal Language for Programmable Logic, Assisted Technology Inc.

Smith, Bill, High Level Logic Design, Signetics Corporation,

Schmookler, Martin, Design of Large ALUs Using Multiple PLA Macros, IBM Journal of Research and Development, Volume 24, No. 1, (January, 1981)

Eichelberger, E. B., Lindbloom, E., A Heuristic Test-Pattern Generator for Programmable Logic Arrays, IBM Journal of Research and Development, (January 1981)

Golden, R. L., Latus, P. A., Lowy, P., Design Automation and the Programmable Logic Array Macro, IBM Journal of Research and Development, (January 1981)

Kang, S., vanCleemput, W. M., Automatic PLA Synthesis from a DDL-P Description, Design Automation Conference, 18, (1981)

Garcia, Sal, Sriram, K. S., A Survey of IC CAD Tools for Design , Layout, and Testing , VLSI Design (Sept./Oct. 1982)

Goel, Prabhakar, Podem-X: An Automatic Test Generation System for VLSI Logic Structures, Design Automation Conference, 18,, (1981)

Holt, Dan, Sapiro, Steve, BOLT- A Block Oriented Design Specification Language, Design Automation Conference, 18, (1981)

Goates, Gary, ABLE: A LISP-Based Layout Modeling Language with User-Definable Pocedural Models for Storage/Logic Array Design, Design Automation Conference, 18, (1981)

Suwa, I., Kubitz, W. J., A Computer-Aided-Design System for Segmented-Folded PLA Macro-Cells, Design Automation Conference, 18, (1981)

Johnson, William, Crowley, Jeri, Steger, Mark, Woosley, Ellen, Mixed-Level Simulation from a Hierarchical Language, Journal of Digital Systems, Volume 5, Issue 3, (1980)

Hightower, David, Roberts, Martin, Automated Logic Arrays and the Customer Interface, IEEE, (1981)

## APPENDIX