AUTOMATIC PLA SYNTHESIS FROM A DDL-P DESCRIPTION

S̆. Kang
W. M. vanCleemput

Center for Integrated Systems
Department of Electrical Engineering
Stanford University
Stanford, California 94305

Abstract - This paper describes an automatic PLA synthesis (APLAS) system which automatically generates a PLA for the control function of a design from a DDL-P description of a digital system. APLAS can also minimize and partition the PLA to meet the design constraints.

This is a very convenient tool for designing finite state machines. The control circuit of any digital system for which a state diagram can be drawn can be designed easily using this system.

1. Introduction

Because of its regular structure, a programmable logic array (PLA) eases many difficult problems associated with hardware synthesis. This prompts more use of PLA's these days, especially in VLSI. The main objective of our work was to develop a practical PLA synthesis procedure including PLA optimization techniques. The emphasis was put on VLSI applications, rather than on FPLA's.

Considering the fact that most data operators are more or less standardized while the control circuitry varies from machine to machine, we use PLA's only for implementing the control, not for data manipulation. Also by doing so, the machine structure required by the design specification can be kept during the design process.

Since a PLA is a two-level structure, it has some inherent redundancy from the classical point of view, which put some constraints on its use. But by employing the design methodology of random logic, the PLA can be optimized, which can be done in three ways: minimization, partitioning and folding. We considered two of them, minimization and partitioning.

Although we have developed many algorithms, we can not properly present them in this paper due to limited space. Therefore the emphasis is placed on the results. For the algorithms, we advise the reader to refer to [1].

2. Overview of APLAS

Figure 1 shows the basic structure of APLAS. The first program, SALT (Stanford Automatic Logic Translator), translates the DDL-P description of a digital system into Boolean equations, which the second program, SPAM (Stanford Programmable Array Minimizer), minimizes and converts into PLA format. The third program, PAPA (Programmable Array Partitioner), is used to partition a large PLA into smaller PLA's to meet the design constraints.

These programs are relatively independent of each other and each can be used separately. Figure 2 shows the steps required in designing a digital circuit using APLAS.

In the next sections, DDL-P and the three subsystems will be discussed.

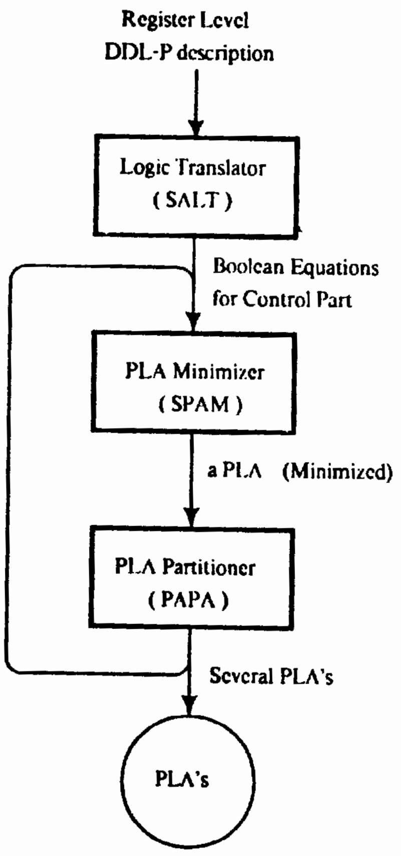


Figure 1. APLAS system

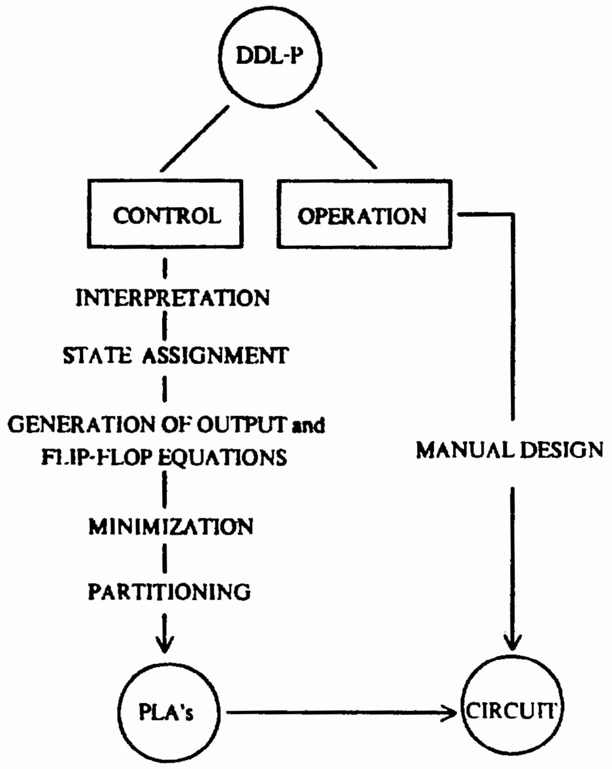


Figure 2. Design Steps of APLAS

3. DDL-P

DDL-P[9,10] is a Stanford version of DDL [6-8] (Digital Design Language), a register transfer language. In this section, the philosophy of DDL-P will be discussed.

3.1 DDL-P Machine Model

The DDL-P machine consists of two main parts, OPERATION and CONTROL (Figure 3). The OPERATION part consists of registers, ALU, memory, etc. which basically store and/or change data according to the instructions generated by the CONTROL part.

The CONTROL part is a system controller which goes through certain steps and issues command signals in each step, which control the actions in the OPERATION part. In turn, the OPERATION sends some information to the CONTROL, which is used to determine what next step the machine should take and/or what signals the CONTROL should generate in the next step. The reader can consider the CONTROL part to be a finite state machine. The DDL-P language also supports a more complicated machine structure (interpretively linked machines), which is not supported by APLAS.

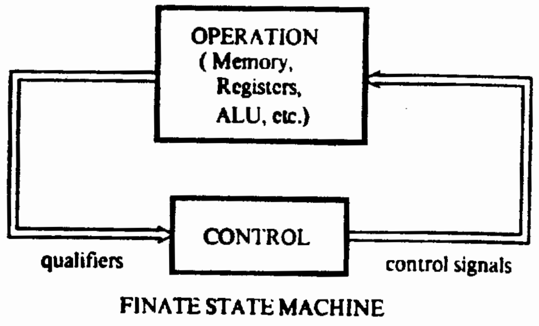


Figure 3. DDL-P Machine Model

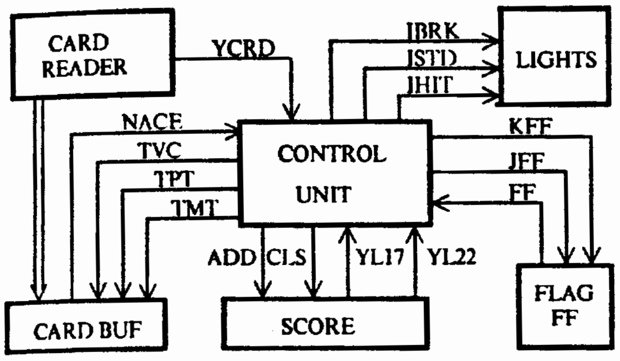
Example - BLACKJACK machine

```
''B L A C K J A C K   M A C H I N E .''

REGISTER  SCORE[5], CARDBUF[5], FF.
TERMINAL  HIT, BROKE, STAND,
   VALUE[1:5] = INPUT(1,VALUE),
   YCRD = INPUT(1,YCRD),
   YL17 = SCORE<17, YL22 = SCORE<22,
   NACE = CARDBUF#1.
OPERATION
   TPT = [CARDBUF <- 5D10], TMT = [CARDBUF <- 5D22],
   TVC = [CARDBUF <- VALUE], IHIT = [HIT=1B1],
   ISTD = [STAND=1B1], IBRK = [BROKE=1B1],
   CLS = [SCORE <- 5D0],
   ADD = [SCORE <-(SCORE(+)CARDBUF)TAIL 5],
   KFF = [FF<-1D0], JFF = [FF <- 1D1  ] .
CONTROL
   A:  CLS, KFF, -> B/
   B:  IHIT,TVC, IF YCRD THEN -> C ELSE -> B ENDIF/
   C:            IF YCRD THEN -> C ELSE -> D ENDIF/
   D:  ADD,    IF NACE*FF THEN -> F ELSE -> E ENDIF/
   E:  JFF, TPT, -> D/
   F:            IF YL17 THEN -> B ELSE -> G ENDIF/
   G:            IF YL22 THEN -> K ELSE -> H ENDIF/
   H:  KFF, TMT, IF  FF  THEN -> D ELSE -> J ENDIF/
   J:  IBRK,     IF YCRD THEN -> A ELSE -> J ENDIF/
   K:  ISTD,     IF YCRD THEN -> A ELSE -> K ENDIF/.$
```

In the above example, the CONTROL section in the description corresponds to the control unit in Figure 4. The other parts of the machine are described in TERMINAL and OPERATION which show implicitly or explicitly how the components are interconnected and at what points the flow of data is controlled. For example, some combinational networks are explicitly described in TERMINAL (e.g. YL17, YL22, etc.) while the ADD function described in OPERATION implicitly assumes the existence of a combinational network (adder) and a control point.

How all these actions are controlled to make this circuit a blackjack machine is described in the CONTROL section. This CONTROL section can be translated into Boolean equations by SALT, which can be directly mapped into physical hardware.



5 QUALIFERS
10 CONTROL SIGNALS
Figure 4. Blackjack Machine

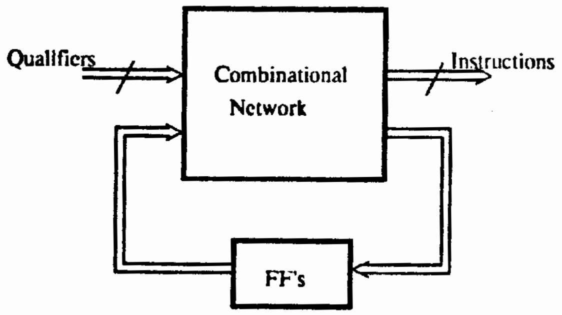


Figure 5. Control Realization

3.2 Characteristics of DDL-P

If carefully used, a DDL-P description can contain all the structural information as well as functional behavior. It is suitable for a digital system which is synchronized by one master clock. At the current stage, DDL-P does not support concurrency of modular blocks, which limits its use to a simple machine structure.

One of the most important and distinct characteristics of DDL-P is that it has a clear boundary between data flow and control flow. This can be painful to a designer who wants to describe a digital system in DDL-P. On the other hand, it is a convenient feature for a designer who wants to synthesize physical hardware from a DDL-P description.

4. SALT

The basic function of SALT is to translate the DDL-P description of a digital machine into Boolean equations that can be realized by physical hardware. The entire process is interactive and consists of three major steps.

First SALT interprets the input and sets up various tables for further processing. The next step is the state assignment. A user can do this manually, or ask SALT to do it automatically. Finally, SALT generates output equations for the control signals and flip-flop equations for the next state functions.

Currently SALT generates Boolean equations only for the control circuitry. The operation part (data paths) should be designed manually or by other design aids to make the design complete.

4.1 CONTROL Section Implementation

There are several ways of realizing CONTROL. One of them, which is used in SALT, is shown in Figure 5. CONTROL gets signals (qualifiers, e.g. YCRD, etc. in Figure 4) from the outside world (OPERATION) and combines them with the present state information which is internally stored in the state flip-flops. And then it generates control signals (instructions, e.g. TPT, ADD, etc. in Figure 4) to the outside world and next state function which will be stored in the state flip-flops.

SALT generates Boolean equations for the instructions and next state functions in terms of the qualifiers and the present state codes. The equations are used to build the hardware. Random logic or PLA's can be used as building blocks depending on the circumstances.

4.2 Decoder in TERMINAL

Conceptually it is possible to put everything in the CONTROL section of the DDL-P description, which should be included in the control circuit. But for practical reasons, it is not desirable.

For example, suppose we have 8 qualifiers which are not disjoint. Then we have up to 256 different conditions and should consider all or part of them in each state. If we use the 8 qualifiers directly in the CONTROL section of the description, we have to generate all those conditions in every state, which is quite inefficient. It is also difficult to read and understand that type of description.

As a way of avoiding that difficulty, we decode the qualifiers to generate the mutually exclusive signals, and consider them separately in the CONTROL section. This decoding section may be included in the TERMINAL section of the description.

If the designer wants the decoder in the real hardware, there is no problem. But if he merely uses the decoder to enhance the readability of the description, the description itself does not reflect the real hardware exactly. Therefore we should have a means to tell SALT that the decoder does not exist and should be included in the control circuit. This is done as a comment in the TERMINAL section.

4.3 State Assignment

State assignment is one of the important steps in SALT. Even though many methods have been proposed for the state assignment process [11–14], the practical method for finding the optimum state assignment has not been found, yet.

In real design, reducing the gate complexity is only one objective. In fact, for a proper circuit operation, race and hazard problems should be considered and they have to be eliminated at any cost.

SALT provides two ways of state assignment: manual and automatic. Automatic assignment is provided as a simple way of doing the assignment without too much concern with timing problems.

Manual state assignment

The user must select the number of flip-flops. After choosing the radix for state assignment, the user can type in the state codes for each state. SALT does not accept identical codes for two different states. During the process, the user can see the result up to the current position, change the previous code, or drop everything and restart the whole process.

Automatic state assignment

SALT always uses the minimum number of FF's. The user can choose between two types of codes: binary and Gray. The user may use these codes as a starting point for his search for better state state assignment. SALT assigns the selected code to the states in the written order of the description.

It should be noted that for an odd number of states, Gray code cannot be formed, i.e., the starting state and the last state can not be unit distance apart. SALT does not overcome this difficulty and is satisfied with distance two between the first and last states in the description.

4.4 Selection of Flip-Flop types

Together with state assignment, the selection of flip-flops affects the hardware cost. Since there are many kinds of flip-flops, it is not easy to choose any particular type of flip-flop for a given job. At the moment, it seems that only the designer's experience, intuition and/or preference can justify the choice of the flip-flops.

Currently three types of flip-flops are used in SALT: JK, D and T. JK flip-flops tend to result in less hardware if used with conventional combinational networks, but need twice the number of next state functions compared to D or T flip-flops. D flip-flops are preferable because of their simplicity if a PLA is used for the combinational network.

4.5 Example

The output of SALT is a set of Boolean equations. Most identifiers used in the equations come from the description. Some identifiers related to the state flip-flops are internally created by SALT. The state flip-flops are implicitly numbered by integers and their outputs and inputs are referenced as

```
Qn     : output of the n th FF
Dn     : input to the n th D FF
Tn     : input to the n th T FF
Jn,Kn  : inputs to the n th JK FF
```

When not all state codes are used, the unused codes can be don't-cares depending on the conditions. For that purpose, SALT puts out the number of flip-flops and the used codes for state assignment if the possible codes are not used up.

The following is an example output of the blackjack machine of which the DDL-P description is given in the previous section.

```
$ INPUT FILE  :  BLACK.JAK
      < OUTPUT EQUATIONS >
1.    CLS = -Q4*-Q3*-Q2*-Q1
2.    KFF = -Q4*-Q3*-Q2*-Q1 + Q4*-Q3*Q2*Q1
3.    IHIT = -Q4*-Q3*-Q2*Q1
4.    TVC = -Q4*-Q3*-Q2*Q1
5.    ADD = -Q4*-Q3*Q2*-Q1
6.    JFF = -Q4*Q3*Q2*-Q1
7.    TPT = -Q4*Q3*Q2*-Q1
8.    TMT = Q4*-Q3*Q2*Q1
9.    IBRK = Q4*-Q3*-Q2*Q1
10.   ISTD = Q4*-Q3*-Q2*-Q1
      < D  FF EQUATIONS >
1.    D1 = -Q4*-Q3*-Q2*-Q1 + -Q4*-Q3*-Q2*Q1*-YCRD
           + -Q4*-Q3*-Q2*Q1*YCRD + -Q4*-Q3*Q2*Q1*YCRD
           + Q4*Q3*Q2*-Q1*YL17 + Q4*-Q3*Q2*-Q1*-YL22
           + Q4*-Q3*Q2*Q1*-FF + Q4*-Q3*-Q2*Q1*-YCRD
2.    D2 = -Q4*-Q3*-Q2*Q1*YCRD + -Q4*-Q3*Q2*Q1*YCRD
           + -Q4*-Q3*Q2*Q1*-YCRD
           + -Q4*-Q3*Q2*-Q1*(NACE+FF)
           + -Q4*-Q3*Q2*-Q1*-(NACE+FF) + -Q4*Q3*Q2*-Q1
           + Q4*Q3*Q2*-Q1*-YL17 + Q4*-Q3*Q2*-Q1*-YL22
           + Q4*-Q3*Q2*Q1*FF
3.    D3 = -Q4*-Q3*Q2*-Q1*(NACE+FF)
           + -Q4*-Q3*Q2*-Q1*-(NACE+FF)
4.    D4 = -Q4*-Q3*Q2*-Q1*(NACE+FF)
           + Q4*Q3*Q2*-Q1*-YL17 + Q4*-Q3*Q2*-Q1*YL22
           + Q4*-Q3*Q2*-Q1*-YL22 + Q4*-Q3*Q2*Q1*-FF
           + Q4*-Q3*-Q2*Q1*-YCRD
           + Q4*-Q3*-Q2*-Q1*-YCRD
.& 4
0000
0001
0011
0010
0110
1110
1010
1011
1001
1000
.$
```

4.6 Results

Some results of SALT are shown in Table 1.

Table 1. SALT Execution

DDL-P Input	Quali-fiers	Control Signals	# of States	State Codes	FF	Ex. Time
Modulo 3/4 counter	1	2	4	Gray	JK	1.096 s
Traffic Light Controller	3	5	4	Gray	D	1.233 s
5-Bit Shift Registers	0	0	5	Binary	D	1.502 s
Decade counter	0	1	10	Binary	D	1.463 s
Blackjack Machine	5	10	10	Gray	JK	1.984 s
Prime Number Counter	3	1	19	Binary	D	2.643 s
Intel 8008 Microprocessor	26	41	18	Gray	D	4.425 s
Intel 8080 Microprocessor	60	61	19	Gray	D	6.112 s
PDP 11	95	54	112	Gray	D	61.201 s

5. SPAM

SPAM (Stanford Programmable Array Minimizer) is a program to minimize a multiple output Boolean switching function. The switching function is assumed to be implemented on a PLA. Since the number of products is important in a PLA implementation, the primary function of SPAM is to minimize the number of products which cover the given switching function.

SPAM accepts inputs of three different formats: Boolean equations, truth table, or SALT results. It then converts the function to sum-of-products form, minimizes the number of products and outputs the result in the PLA format.

After minimizing the number of products, SPAM tries to reduce the connections in the AND and OR arrays. The fewer connections in the AND and OR arrays, the smaller the AND and OR gates become in a random logic implementation. Therefore SPAM can be used to minimize two-level random logic (AND-OR, NOR-NOR, NAND-NAND) as well as PLA's.

SPAM does not guarantee an optimal solution, but most solutions are near optimal. Absolute minimality is often achieved especially, for relatively small problems.

The current version of SPAM can handle 72 inputs, 144 outputs and several thousand products. These limits may be changed easily as they are essentially a function of the available address space for the program.

5.1 Example

SPAM recognizes the input format with the first character. The truth table form must start with '#'. SALT result starts with '$'. Anything else will be considered to be Boolean equations. The input is essentially free-format. Blanks may be inserted at will to improve readability, except the blanks should not be embedded in identifiers of Boolean equations. Comments may also appear anywhere blanks are allowed.

The following output came from the SALT result of Blackjack machine given in the previous section.

```
SOLUTION :  8 INPUTS   14 OUTPUTS   18 PRODUCTS

        <INPUTS>             <OUTPUTS>

      1.  Q4               1.  CLS
      2.  Q3               2.  KFF
      3.  Q2               3.  IHIT
      4.  Q1               4.  TVC
      5.  YCRD             5.  ADD
      6.  YL17             6.  JFF
      7.  YL22             7.  TPT
      8.  FF               8.  TMT
      9.  NACE             9.  IBRK
                          10.  ISTD
                          11.  D1
                          12.  D2
                          13.  D3
                          14.  D4

        000000000 00000000011111
        123456789 12345678901234
    ----------------------------
    1.  01------- .....11.......
    2.  0--11---- ..........11..
    3.  1-11---0- .1.....1..1..1
    4.  0-01----- ..11......1...
    5.  1-00----- .........1....
    6.  0-00----- 11........1...
    7.  11---0--- ...........1.1
    8.  1-11---1- .1.....1...1..
    9.  11---1--- ..........1...
   10.  0-1------ ...........1..
   11.  0010----- ....1.......1.
   12.  1-010---- ..........1...
   13.  1-0-0---- .............1
   14.  1-01----- ........1.....
   15.  1010----- .............1
   16.  1010--0-- ..........11..
   17.  -010----1 .............1
   18.  -010---1- .............1
```

5.2 Results

Some results of SPAM are shown in Table 2.

Table 2. SPAM Execution

Source	Input Type	In/Out	# of Products		Execution Time
			Initial *	Final	
Modulo 3/4 counter	SALT	3/6	8/10	5	0.31 s
Traffic Light Controller	SALT	5/7	28/ 0	10	0.51 s
5-bit Shift Register	SALT	5/5	5/11	5	0.45 s
Decade counter	SALT	4/5	16/ 2	9	0.53 s
Blackjack Machine	SALT	9/14	40/ 2	18	1.30 s
Prime Number Counter	SALT	8/6	77/ 3	37	4.55 s
Intel 8008 Microprocessor	SALT	31/46	148/ 3	68	19.31 s
Intel 8080 Microprocessor	SALT	65/66	181/ 3	129	43.00 s
PDP 11	SALT	102/61	1651/ 1	348	156.00 s
4 x 4 -bit Multiplier	Tr.	8/8	225/ 0	129	92.12 s
4-bit Adder	Tr.	8/5	255/ 0	75	26.95 s
4-bit Adder (with carry-in)	Tr.	9/5	511/ 0	135	129.94 s
PLA 1	Tr.	9/15	33/29	29	5.42 s
PLA 2	Tr.	8/14	33/28	32	7.29 s
PLA 3	Tr.	23/40	273/273	165	390.00 s

* Initial number of products => normal products/ don't care products

Tr. : Truth table

6. PAPA

In a real circuit, a large PLA tends to be quite wasteful or not fast enough to support the other parts of the system. In this case, we can split it into several smaller PLA's to reduce the chip area and/or improve the speed.

PAPA is a program which does this job. It also has a redundancy removal routine which detects input and output redundancy [1] from a given PLA. The chip area and the overhead of using the many PLA's are the major factors considered in partitioning.

6.1 Example

PAPA can only accept the truth table input. The output is the partitioned truth table with some information about the result of partitioning such as the size of each partitioned PLA and the reduction ratio of chip area. The order of output lines is changed by PAPA.

The following example is the result of Blackjack machine whose truth table is given in the previous section. The ordering of outputs is omitted on purpose for clarity.

```
SOLUTION :  2 PLA's

PLA   1 : 9 INPUTS    8 OUTPUTS  14 PRODUCTS
PLA   2 : 4 INPUTS    6 OUTPUTS   4 PRODUCTS

TOTAL AREA : INITIAL        :  576
             FINAL          :  420

REDUCTION RATIO             :0.746

             000000000 00000000011111
             123456789 12345678901234
        ------------------------------
         1.  1-010---- 1.......
         2.  11---1--- 1.......
         3.  0-01----- 111.....
         4.  1010--0-- 1.....1.
         5.  0-00----- 1..11...
         6.  1-11---0- 1...11.1
         7.  0--11---- 1.....1.
         8.  1-11---1- ....111.
         9.  0-1------ ......1.
        10.  11---0--- ......11
        11.  -010---1- .......1
        12.  -010----1 .......1
        13.  1010----- .......1
        14.  1-0-0---- .......1

  ----------------

        15.  1-01                1.....
        16.  1-00                .1....
        17.  0010                ..11..
        18.  01--                ....11
```

6.2 Results

Some results of PAPA are shown in Table 3. All inputs are the PLA's minimized by SPAM. Overhead of using many PLA's is also accounted for calculating the reduction ratio. The partitioned PLA's are not minimized by SPAM, which may be possible.

Table 3. PAPA Execution

INPUT (in/out/product)	# of PLA's	Area		1 *	Execution Time
		Initial	Final	Reduction	
Blackjack (9/18/17)	3	612	314	1.89	0.17 s
8080 A (65/66/129)	16	25284	4893	4.33	5.83 s
8080 B (14/66/247)	8	23218	9354	2.23	4.32 s
4-bit Mult. (8/8/129)	4	3096	2460	1.18	0.39 s

* 1/Reduction = Initial area/(Final area + Overhead)

7. Conclusions

We have presented an automatic PLA synthesis system with some results on its performance. We successfully used this system to synthesize the control circuitry of microprocessors similar to the Intel 8008 and the 8080.

The system includes a PLA minimizer and a PLA partitioner for better use of PLA's. The PLA minimizer is powerful enough to accept a switching function with 72 inputs, 144 outputs and several thousand products. The performance of the partitioner largely depends on a PLA format. But it can reduce the chip area drastically (sometimes more than 50 %) in many cases.

REFERENCES

1. S. Kang, *Minimization, Partitioning and Synthesis of Programmable Logic Arrays,* Ph.D. dissertation, Dept. of EE, Stanford University, to be published.
2. S. Kang and W. M. vanCleemput, *SALT user's manual,* Computer Systems Lab. Tech. Report No. 203, Stanford University, Mar. 1981.
3. S. KANG and W. M. vanCleemput, *SPAM user's manual,* Computer Systems Lab. Tech. Report No. 204, Stanford University, Mar. 1981.
4. W. M. vanCleemput, *Design automation at Stanford,* Computer Systems Lab. Tech. Report No. 178, Stanford University, July 1979.
5. W. M. vanCleemput, *Design automation at Stanford,* Computer Systems Lab. Tech. Report No. 184, Stanford University, Feb. 1980.
6. J. R. Duley and D. L. Dietmeyer, "A digital system design language (DDL)," *IEEE Trans. Comput.*, vol. C-17, pp. 850–861, Sept. 1968.
7. J. R. Duley and D. L. Dietmeyer, " Translation of a DDL digital system specification to Boolean equations," *IEEE Trans. Comput.*, vol. C-18, pp. 305-313, Apr. 1969.
8. R. L. Arndt and D. L. Dietmeyer, "DDLSIM – A digital design language simulator," *Proc. National Electronics Conf.*, vol. 26, pp. 116–118, Dec. 1970.

9. W. E. Cory, J. R. Duley and W. M. vanCleemput, *An introduction to the DDL-P language*, Computer Systems Lab. Tech. Report No. 163, Stanford University, Mar. 1979.

10. W. E. Cory,J. R. Duley and W. M. vanCleemput, *DDL-P Command Language Manual*, Computer Systems Lab. Tech. Report No. 164, Stanford University, Mar. 1979.

11. D. B. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 466-472, Aug. 1962.

12. D. B. Armstrong, "On the efficient assignment of internal codes to sequential machines," *IRE Trans. Electron. Comput.*, vol. EC-11, pp. 611-622, Oct. 1962.

13. T. A. Dolotta and E. J. McCluskey, "The coding of internal states of sequential circuits," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 549-563, Oct. 1964.

14. J. R. Story, H. J. Harrison, and E. A. Reinhard, "Optimum state assignment for synchronous sequential circuits," *IEEE Trans. Comput.*, vol. C-21, pp. 1365-1373, Dec. 1972.

15. P. Bricaud and J. Campbell, "Multiple output PLA Minimization : EMIN," Wescon 78.

16. S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI : A heuristic approach for logic minimization," *IBM J. Res. Develop.*, vol. 18, pp. 443-458, Sept. 1974.

17. H. Langenbacher, "Boolean minimization for logic arrays," *M. S. Thesis*, Dept. of EE, San Diego State Univ. , Fall, 1979.