Logic Design of Programmable Logic Arrays

YAHIKO KAMBAYASHI, MEMBER, IEEE

Abstract—**Compared with random logic circuits, memory-type circuits are more suitable for LSI realization since their iterated structure of identical cells results in higher transistor density and higher yield. A programmable logic array (PLA) is a read only memory (ROM) with programmable addresses and it is suitable for realizing logic functions with many unspecified input combinations. For such a function, reduction of the number of input is possible in many cases. Since the cost of a PLA is mainly determined by the number of pins and the chip, both of which are affected by the number of inputs, the reduction of the number of inputs is very important in PLA design. On the other hand, the reduction of the number of product terms in a sum-of-product expression is important in conventional random logic synthesis. Since there is a tradeoff problem between the number of inputs and the number of product terms, we give a design procedure by the following preference order: 1) minimizing the number of pins, 2) minimizing the number of product terms, 3) minimizing the number of circuits used in the PLA. These factors also determine the area required for a chip.**

Index Terms—**Incompletely specified logic function, logic design, multiple-output function, programmable logic array, reduction of the number of inputs, two-level circuit.**

I. Introduction

DUE TO the recent development of LSI technology, a unit component for logic circuits can realize very complicated functions. Therefore, it is very important to develop a logical design procedure using such components. A circuit which has a simple or regular pattern is suitable for LSI realization since it requires less chip area (the chip area is mostly occupied by interconnections in the case of random logic circuits). A memory-type array is one example which has this property. Furthermore, a memory-type array is economical because of its large production volume.

Random access memory (RAM) and read only memory (ROM) can be used to realize arbitrary logic functions by storing function values at memory locations determined by the combinations of input variables. But if there are many unspecified combinations, memory cells corresponding to these addresses are redundant. Programmable logic arrays (PLA's) solve this problem since their address part can be programmed. PLA's can realize arbitrary logical functions and they can be used in microprogramming controllers as well as random logic circuits.

In this paper we will discuss design procedures for economical PLA's. Since the most important factors which determine the cost of an integrated circuit are the number of pins and the chip area of the circuit, we have to solve several problems which do not occur in minimization of the number of gates in traditional random logic synthesis.

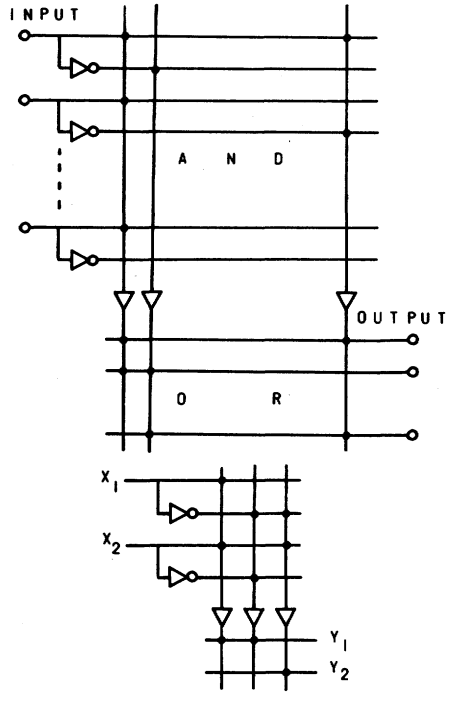

Fig. 1. Programmable logic array.

Fig. 1(a) shows the structure of a PLA, which consists of two matrices. In the first part (AND matrix) input signals and their negated signals are selectively connected to product term lines in such a way that certain combinations of input variables produce the logically true signal "1" on one or more product term lines. These lines are input to the second part (OR matrix) where other selecting connections transfer the signals to the output lines.

Fig. 1(b) shows an example of a PLA realizing the functions $y_1 = x_1x_2 + \bar{x}_1\bar{x}_2$ and $y_2 = \bar{x}_1x_2$. In this example the synthesis procedure is straightforward, that is, the selection of the connecting points is directly determined by the logical expressions. In existing hardware, functions of these matrices are NAND (NOR), so a PLA corresponds to a two-level NAND (NOR, respectively) and hence a two-level AND–OR (OR–AND, respectively) circuit. Since the AND–OR PLA structure for a given function is the same as the OR–AND PLA structure for the dual function, only synthesis procedures for AND–OR PLA's are considered in this paper. NOT circuits are used to generate negated input variables. There are PLA's which can be programmed electrically. These PLA's are called FPLA's (field programmable logic arrays).

Manuscript received February 28, 1979.

The author was with the Department of Information Science, Kyoto University, Sakyo, Kyoto, Japan. He is now with the School of Computer Science, McGill University, Montreal, P.Q., Canada.

0018-9340/79/0900-0609$00.75 © 1979 IEEE

The following factors must be considered in designing PLA's:

1) number of inputs
2) number of outputs
3) number of product term lines
4) number of NOT circuits
5) logic time delay.

The total number of pins is expressed as follows:
the total number of pins for the chip excluding power supply pins
= (the number of input variables)
+ (the number of outputs).
The chip area is mainly determined by
[(the number of inputs to the AND matrix)
+ (the number of outputs)]
× (the number of product term lines).
The number of inputs to the AND matrix is at most

$$2 \times \text{(the number of input variables)}.$$

The bound is attained when nonnegated and negated values of all inputs are added to the AND matrix. Normally there exist input variables which do not require both values. Since it may not be possible to reduce the number of outputs except when an encoding of outputs is permitted, the reduction of the number of input variables is very important to reduce both the pin count and the chip area. As previously shown, logic functions with many unspecified output values are suitable for the PLA realization. There are many kinds of representations for such functions. The selection of the minimum input representation is not trivial, and there is a tradeoff problem between the number of input variables and the number of product terms. The selection of the representation with the minimum number of negated input variables is also a nontrivial problem.

Another feature of our procedure is the use of NOT circuits in output portions. A simpler PLA may be obtained by introducing such NOT circuits, since the function f and its negation $\bar{f}$ may not require PLA's of the equal complexity. The NOT circuits in both input and output portions must be considered to obtain a PLA with the minimum number of NOT circuits. The PLA in Fig. 1(a) consists of two matrices and corresponds to a conventional two-level logic circuit. There exist PLA's with more than two matrices [8], which correspond to multilevel logic circuits. For such a realization we must consider the tradeoff problem between the circuit complexity and the circuit delay. In this paper this problem is avoided by handling only PLA's with two matrices.

Thus, there is a tradeoff problem among the above five factors. Our procedure uses the following preference order, since the reduction of the pin count is considered to be the most important factor to realize economical PLA's:

1) minimizing the number of pins
2) minimizing the number of product terms
3) minimizing the number of NOT circuits.

II. Basic Definitions

Let $\{x_1, x_2, \cdots, x_n\}$ and $\{y_1, y_2, \cdots, y_m\}$ be the set of input variables and the set of output variables of a PLA, respectively. Let $X_i = (a_{i1}, a_{i2}, \cdots, a_{in})$ and $Y_j = (b_{j1}, b_{j2}, \cdots, b_{jm})$ be an input vector and an output vector, respectively, where each element of the vectors is 0, 1, or *, where * is a DON'T CARE condition. $X_i^{(j)}$ and $Y_i^{(j)}$ denote the jth entries of input vector X_i and output vector Y_i, respectively. X_i and Y_i with *'s are regarded as sets of all input vectors and all output vectors consisting of 0's and 1's (without *'s) generated from X_i and Y_i by assigning 0's and 1's to *'s in all possible ways. For example,

$$(1, 0, *) = \{(1, 0, 0), (1, 0, 1)\}.$$

If outputs of two input vectors (1, 0, 0) and (1, 0, 1) are the same, we will use (1, 0, *) to represent both input vectors. If the third output can be either 1 or 0 when the first and the second outputs are 1 and 0, respectively, then it is denoted by (1, 0, *).

A multiple logic function L is specified by a set of pairs $\{(I_i, Y_i) \mid i = 1, \cdots, r\}$. I_i is a set of input vectors and Y_i is the output vector which is required to realize when any input vector contained in I_i is given. The following restrictions are assumed.

$$\text{For} \quad i \neq j,\ I_i \cap I_j = \emptyset, \tag{1}$$

$$Y_i \cap Y_j = \emptyset. \tag{2}$$

Equation (1) is obvious. If there exist i and j such that $Y_i \cap Y_j \neq \emptyset$, we can use $(I_i \cup I_j, Y \text{ in } Y_i \cap Y_j)$ instead of two pairs (I_i, Y_i) and (I_j, Y_j) since: 1) Y can replace both Y_i and Y_j because Y is obtained by assigning 0's and/or 1's to some *'s in Y_i and Y_j, and 2) the logical expression for $(I_i \cup I_j, Y)$ is not more complicated than the two logical expressions for (I_i, Y_i) and (I_j, Y_j).

I is defined by

$$I = \bigcup_{i=1,,r} I_i. \tag{3}$$

If L is incompletely specified, there are input vectors which are not contained in I. The set of all such vectors is denoted by I_o. I_o is the set of input vectors whose output values are not specified. For given L we define a characteristic function f_i of I_i as follows:

$$f_i(X) = 1 \qquad \text{for } X \in I_i,$$

$$f_i(X) = 0 \qquad \text{for } X \in I - I_i,$$

$$f_i(X) = 0 \text{ or } 1 \qquad \text{for } X \in I_o.$$

The cost of a PLA is assumed to be given by $AP + BT$ where P is the number of input–output pins and T is the number of product term lines. A and B are the cost coefficients and we assume $A \gg B$, that is, a reduction in the number of pins is more important than a reduction in the number of product term lines.

III. Reduction of the Number of Input Pins

In order to get a minimum PLA, we will first consider the problem of minimizing the number of input pins of a PLA. In random logic gate synthesis, a reduction in the number of gates is most important, so this problem was not treated since a reduction in the number of input variables may increase the cost of the minimum expression of the logic function (see Example 4). A special case is discussed in [3].

Definition 1: Input variable x_k is redundant with respect to I_i iff there exists a characteristic function f_i such that

$$f_i(X)|_{x_k=0} = f_i(X)|_{x_k=1} \qquad \text{for all } X \text{ in } I.$$

Definition 2: Input variable x_k is redundant in $L = \{(I_i, Y_i)\}$ iff x_k is redundant with respect to I_i for $i = 1, \cdots, r$.

If x_k is redundant in L, x_k is unnecessary to realize L.

The following theorems give necessary and sufficient conditions for redundancy.

Theorem 1: Input variable x_k is redundant in L iff there exist no I_i and I_j such that $(i \neq j, i \neq 0, j \neq 0)$

$$^3X_i \in I_i, \ ^3X_j \in I_j$$

for $h = 1, \cdots, n, h \neq k, X_i^{(h)} = X_j^{(h)}$ or $X_i^{(h)} = *$ or $X_j^{(h)} = *$, for $h = k$, $X_i^{(k)} \neq X_j^{(k)}$.

Proof: If there exist $X_i \in I_i$ and $X_j \in I_j$ satisfying the above condition, there exist no f_i and f_j satisfying Definition 1, since

$$f_i(X_i) \neq f_i(X_j) \text{ and } f_j(X_i) \neq f_j(X_j)$$
$$X_i^{(k)} \neq X_j^{(k)}, \ X_i^{(h)} = X_j^{(h)} \qquad \text{for all } h \neq k.$$

Conversely, if the condition is satisfied for all combinations of i and j, all f_i can satisfy Definition 1 and x_k is redundant. Q.E.D.

$X_{(\bar{k})}$ is defined as follows:

$$X_{(\bar{k})} = (a_1, \cdots, \bar{a}_k, \cdots, a_n)$$

where

$$X = (a_1, \cdots, a_k, \cdots, a_n).$$

Theorem 2: Input variable x_k is redundant in L iff there exists a logic function g of $n-1$ variables such that

$$a_k = g(a_1, \cdots, a_{k-1}, a_{k+1}, \cdots, a_n)$$

for all $X = (a_1, \cdots, a_{k-1}, a_k, a_{k+1}, \cdots, a_n)$ in I^+ where

$$I^+ = I - I^-, \ I^- = \{X \mid {}^3i, \ X \in I_i, \ X_{(\bar{k})} \in I_i\}.$$

The value of g for X in I^- can be determined arbitrarily.

There are two simple cases of this theorem.

Corollary 2.1: Input variable x_k can be expressed by a constant function $x_k = c$ iff the following condition is satisfied.

For all X in I_i $\quad (i = 1, \cdots, r)$,

1) $X^{(k)} = c$ or

2) $X^{(k)} = \bar{c}$ and $X_{(\bar{k})} \in I_i \cup I_0$.

Corollary 2.2: Input variable x_k can be expressed by a single variable function iff the following condition is satisfied.

For all X in I_i $\quad (i = 1, \cdots, r)$,

1) $X^{(k)} = X^{(h)}$ $\quad (X^{(k)} = \bar{X}^{(h)})$ or

2) $X_{(\bar{k})} \in I_i \cup I_0$.

Note that even if x_k is expressed by x_h, x_h may not be expressed by x_k.

Definition 3: Input variable x_k is essential in L iff it is not redundant in L.

If x_k is essential, x_k cannot be removed, but if x_k is redundant, x_k may become essential after removal of some redundant variables.

Example 1: Let us reduce the number of input variables in Table I(a). As there are no essential variables, any variable can be removed. x_3 is redundant since it satisfies Corollary 2.1 $(x_3 = 1)$. After removal of x_3, all variables are again redundant. By removing x_6, Table I(b) is obtained. Here all variables are essential since

x_1 is essential: $(1100) \in I_1$, $(0100) \in I_2$

x_2 : $(0000) \in I_1$, $(0100) \in I_2$

x_4 : $(0001) \in I_1$, $(0011) \in I_2$

x_5 : $(1100) \in I_1$, $(1101) \in I_2$.

There is, however, Table I(c) which has only three variables, and the vectors in I_1 and the vectors in I_2 are also distinguishable. Thus, the function in Table I(a) can be realized by only three variables x_2, x_4, and x_6.

As shown in the example, if we remove input variables sequentially, the result may not be minimal. There are, however, variables we can remove without affecting the results.

Theorem 3: If input variable x_k is expressed by a constant function, the removal of x_k does not restrict the removal of other variables. (Proof omitted.)

In order to obtain all minimum variable tables, we can use the minimum cover problem.

Algorithm 1:

1) Let all the essential variables be $x_{a1}, \cdots, x_{aq}$.

2) Define D_{kh} for all $X_k \in I_i$ and $X_h \in I_j$ $(i \neq j)$ if $X_k^{(a)} = X_h^{(a)}$ for $a = a1, \cdots, aq$.

3) A cover table is defined as follows:
Each row corresponds to one D_{kh}.
Each column corresponds to one input variable x_i.
Each entry of column x_i and row D_{kh} is b_{khi} where $b_{khi} = X_k^{(i)} \oplus X_h^{(i)}$ if none of $X_k^{(i)}$ and $X_h^{(i)}$ are $*$; otherwise, $b_{khi} = 0$.

4) Remove row $D_{k_1h_1}$ if there exists row $D_{k_2h_2}$ satisfying

$$b_{k_1h_1i} \geq b_{k_2h_2i} \qquad \text{for all } i.$$

5) Calculate the set S satisfying the following condition:

$$^\forall D_{kh}, \ ^3x_i \in S \text{ such that } b_{khi} = 1.$$

The set S with a minimum number of elements is called a minimum cover.

6) There exist characteristic functions of L which can be expressed using essential variables and input variables contained in S.

Proof: If essential variables differ in X_k and X_h, no additional variable is necessary to distinguish them. Otherwise, at least one variable is necessary in S which distinguishes X_k from X_h. Q.E.D.

Example 2: Let us apply Algorithm 1 to the function given by Table I(a). Since there are no essential variables, the cover table [see Table II(a)] is obtained in step 3), where only 1's are shown. After applying step 4), Table II(a) reduces to Table II(b). Covers of the table are given by the solutions of the following Boolean equation:

$$(x_1 + x_6)(x_5 + x_6)(x_2 + x_6)(x_4 + x_5)(x_2 + x_5)(x_4 + x_6)$$
$$(x_1 + x_2) = 1.$$

TABLE I

		x_1	x_2	x_3	x_4	x_5	x_6	x_1	x_2	x_4	x_5	x_2	x_4	x_6
I_1	X_1	1	1	1	0	0	0	1	1	0	0	1	0	0
	X_2	0	0	1	0	0	0	0	0	0	0	0	0	0
	X_3	0	0	1	0	1	1	0	0	0	1	0	0	1
	X_4	0	0	0	0	1	1	0	0	0	1	0	0	1
I_2	X_5	0	1	1	0	0	1	0	1	0	0	1	0	1
	X_6	0	0	1	1	1	0	0	0	1	1	0	1	0
	X_7	1	1	1	0	1	1	1	1	0	1	1	0	1
		(a)						(b)				(c)		

TABLE II

	x_1	x_2	x_3	x_4	x_5	x_6
D_{15}	1					1
D_{16}	1	1		1	1	
D_{17}					1	1
D_{25}		1				1
D_{26}				1	1	
D_{27}	1	1			1	1
D_{35}		1			1	
D_{36}				1		1
D_{37}	1	1				
D_{45}		1	1		1	
D_{46}			1	1		1
D_{47}	1	1	1			

(a)

	x_1	x_2	x_3	x_4	x_5	x_6
D_{15}	1					1
D_{17}					1	1
D_{25}		1				1
D_{26}				1	1	
D_{35}		1			1	
D_{36}				1		1
D_{37}	1	1				

(b)

By simplifying this we have

$$x_1x_5x_6 + x_2x_4x_6 + x_2x_5x_6 = 1.$$

Corresponding characteristic functions are as follows:

$$(x_1, x_5, x_6)\, f_1 = \bar{x}_5\bar{x}_6 + \bar{x}_1x_5x_6,$$

$$(x_2, x_4, x_6)\, f_1 = \bar{x}_4\bar{x}_6 + \bar{x}_2\bar{x}_4,$$

$$f_1 = \bar{x}_2\bar{x}_4 + x_2\bar{x}_6,$$

$$f_1 = \bar{x}_2x_6 + \bar{x}_4\bar{x}_6,$$

$$(x_2, x_5, x_6)\, f_1 = \bar{x}_5\bar{x}_6 + \bar{x}_2x_6.$$

After applying Algorithm 1, selection of minimum cost characteristic functions is necessary.

Example 3: Fig. 2 shows the seven-segment code for an LED display. Table III(a) shows the correspondence of the seven-segment code and the variables. We will design a PLA which produces different outputs for different input vectors among $X_0, X_1, \cdots, X_9$ [here, we do not consider $X_{6'}$, $X_{9'}$, and X_b (blank code)]. Essential input variables are a and g.

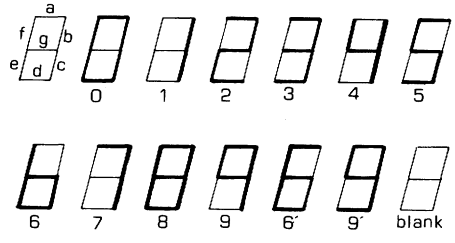

Fig. 2. Seven-segment code.

TABLE III

	a	b	c	d	e	f	g	a	d	e	f	g
X_0	1	1	1	1	1	1	0	1	1	1	1	0
X_1	0	1	1	0	0	0	0	0	0	0	0	0
X_2	1	1	0	1	1	0	1	1	1	1	0	1
X_3	1	1	1	1	0	0	1	1	1	0	0	1
X_4	0	1	1	0	0	1	1	0	0	0	1	1
X_5	1	0	1	1	0	1	1	1	1	0	1	1
X_6	0	0	1	1	1	1	1	0	1	1	1	1
X_7	1	1	1	0	0	0	0	1	0	0	0	0
X_8	1	1	1	1	1	1	1	1	1	1	1	1
X_9	1	1	1	0	0	1	1	1	0	0	1	1
$X_{6'}$	1	0	1	1	1	1	1					
$X_{9'}$	1	1	1	1	0	1	1					
X_b	0	0	0	0	0	0	0					
	(a)							(b)				

Input vectors are classified according to each combination of essential variables:

$a = 0, g = 0$ $\quad X_1$

$a = 0, g = 1$ $\quad X_4, X_6$

$a = 1, g = 0$ $\quad X_0, X_7$

$a = 1, g = 1$ $\quad X_2, X_3, X_5, X_8, X_9.$

Rows of the cover table consist of

TABLE IV

	x_1	x_2	x_3	x_4	x_5
I_1	0	0	1	1	1
	0	1	0	1	1
	1	1	0	0	1
	1	1	1	1	1
	1	0	0	1	1
I_2	0	0	0	0	1
	0	0	1	0	1
	0	1	0	0	1
	1	1	1	0	1
	1	0	0	0	1
	1	0	1	1	0

D_{07} D_{46} D_{23} D_{25} D_{28} D_{29} D_{35} D_{38} D_{39} D_{58} D_{59} D_{89}.

The corresponding equation is $bef + def = 1$.

The table consisting of variables a, d, e, f, g is shown in Table III(b). Even without variables b and c, all input vectors can be distinguished.

If there are many input variables, Algorithm 1 may be difficult to apply because of the size of the cover table. One heuristic algorithm which will give a near optimum solution in a shorter time is as follows.

Algorithm 2:

1) If both X and $X_{(\bar{k})}$ are in I_h in the given table, replace them by a single vector which is equal to X except the position of x_k being replaced by *.

2) Calculate all redundant variables in the table. If there are more than one, selection of one variable by step 3) is necessary.

3) Divide the set of I_i's into subsets according to each combination of essential variables.

3-1) For each subset T_i calculate N_{T_i}:

$$N_{Ti} = \left(\sum_k n_{kjo}\right) \cdot \left(\sum_k n_{kj1}\right) - \sum_k n_{kj0} n_{kj1}$$

$\sum_k$ is a summation over k such that $I_k \in T_i$.

Here, n_{kj0} and n_{kj1} are the numbers of 0's and 1's of $X^{(j)}$ for $X \in I_k$.

3-2) Select x_j whose value of $\sum_{T_i} N_{T_i}$ is largest.

4) Remove x_j from the table and for the new table apply step 1). Repeat this process until all variables become essential.

In order to obtain all optimum solutions, the branch and bound method must be combined with Algorithm 2.

IV. Reduction of the Chip Area

A. Reduction of the Number of Product Term Lines

It is obvious that reduction of the number of input variables will not reduce the number of product terms. There are, however, variables of which removal causes an increase in the number of product terms.

Example 4: The characteristic function of I_1 of Table IV is given by

$$f_1 = x_1 x_2 \bar{x}_3 + x_4 x_5.$$

After the removal of redundant variable x_5, the characteristic function will be

$$f_1 = x_1 x_2 \bar{x}_3 + \bar{x}_1 x_4 + x_2 x_4 + \bar{x}_3 x_4.$$

In this case the number of product term increases by the removal of an input variable.

Sufficient conditions for variables whose removal does not affect the number of product terms are as follows.

Theorem 4: 1) If input variable x_k is expressed by a constant function, the removal of x_k does not change the cost of minimum expressions of characteristic functions. 2) If input variable x_k is expressed by a single variable function, the removal of x_k does not change the cost of minimum expressions of characteristic functions.

This theorem is generalized as follows.

Theorem 5: The removal of input variable x_k does not change the minimum number of product terms of characteristic functions if there exist $a_1, \cdots, a_p (1 \leq a \leq n)$ and $b_1, \cdots, b_p (b_i = 0 \text{ or } 1)$ such that

1) $X^{(k)} = X^{(a_1)b_1} \cdot X^{(a_2)b_2} \cdots X^{(a_p)b_p}$ or

2) $X_{(\bar{k})} \in I_i \cup I_0$ for $X \in I_i$.

One of the above conditions must be satisfied for all X in I where $X^{(a)b} = X^{(a)}$ if $b = 1$, $X^{(a)b} = \bar{X}^{(a)}$ if $b = 0$.

In Example 4, x_5 is expressed by

$$\bar{x}_1 + x_2 + \bar{x}_3.$$

The converse of Theorem 5 is not always satisfied since the number of product terms will not be increased if x_5 appears only in the terms, each of which contains at least two of x_1, $\bar{x}_2$, x_3 (for example, $x_5 x_1 \bar{x}_2 = (\bar{x}_1 + x_2 + \bar{x}_3) x_1 \bar{x}_2 = \bar{x}_3 x_1 \bar{x}_2$).

By the above reason, after the reduction of input variables we need to find the optimum logical expression under the following criteria.

1) Minimize the number of product terms.

2) Minimize the number of literals (x_i and $\bar{x}_i$ are regarded as different literals).

3) Minimize the number of NOT circuits.

The area of the AND matrix is determined by the product of the number of product terms and the number of literals.

Example 5: For the function shown in Table I we have five logical expressions, each of which requires only three input variables. Table V shows the number of product terms, the number of literals, the area of the AND matrix, and the number of NOT circuits for each three-variable function. We will select $\bar{x}_4 \bar{x}_6 + \bar{x}_2 \bar{x}_4$ as the minimum cost expression.

B. Preprocessing of Input Vectors

There are cases when some preprocess of input variables will further decrease the complexity of a PLA.

1) Hash coding using logic circuits. Since, as in microprogramming, each input word (instruction) has a different word length, compression of input word using a simple logic circuit may be effective. A PLA will be simplified if we can make Hash codes for input vectors in the same set I_i be identical.

TABLE V

Expression	The number of product terms	The number of literals	The area of AND matrix	The number of NOT circuits
$\bar{x}_5\bar{x}_6+\bar{x}_1x_5x_6$	2	5	10	3
$\bar{x}_4\bar{x}_6+\bar{x}_2\bar{x}_4$	2	3	6	3
$\bar{x}_2\bar{x}_4+x_2\bar{x}_6$	2	4	8	3
$\bar{x}_2x_6+\bar{x}_4\bar{x}_6$	2	4	8	3
$\bar{x}_5\bar{x}_6+\bar{x}_2x_6$	2	4	8	3

2) Decoders. Fleisher and Maissel [8] have proposed a PLA with an input decoder.

3) Simple logic operation on essential input vectors. Simple logic operation (AND, OR, NAND, NOR) among input variables is sometimes effective. If we use Algorithm 2, essential variables generated later have a higher possibility of such simplification. Also, a similar selection algorithm can be applied to essential variables determined by Algorithm 1.

The preprocessing of input vectors by simple logic functions is realized by the following algorithm.

1) Remove redundant variables using Algorithm 1 or Algorithm 2.

2) Introduce new variables, each of which is formed by AND or OR of two essential variables obtained by step 1).

3) Repeat steps 1) and 2) until no further reductions are possible. Since we need to distinguish r sets $(I_1, \cdots, I_r)$, the minimum number of input variables is $\lceil \log_2 r \rceil$ where $\lceil k \rceil$ is the least integer not less than k.

Example 6: We will apply the above algorithm to the example shown in Table III(b). If we add a new variable $h = b \cdot g$, then variables d and g become redundant. We have the table shown in Table VI(a). Since we need to distinguish ten input vectors, the minimum number of inputs is four, and thus no further reduction is possible.

Using this result, we have a seven-segment-to-decimal converter shown in Fig. 3(a), which requires only one decoder and one AND gate (if wired-AND is possible, it can be replaced by an output tie). The simplest circuit previously known requires two NAND gates, three Exclusive-OR gates, and one decoder [6] (see also [5] and [7]). Fig. 3(b) shows a circuit with a binary-to-decimal decoder [see Table VI(b)]. By a similar method, we can design a converter which permits $X_{6'}$, $X_{9'}$, and X_b [see Table VI(c) and Fig. 3(c)].

TABLE VI

Code	a	e	f	b·g
0	1	1	1	0
1	0	0	0	0
2	1	1	0	1
3	1	0	0	1
4	0	0	1	1
5	1	0	1	0
6	0	1	1	0
7	1	0	0	0
8	1	1	1	1
9	1	0	1	1

(a)

Code	$\bar{a}\cdot\bar{e}$	e	a·f	b·g
0	0	1	1	0
1	1	0	0	0
2	0	1	0	1
3	0	0	0	1
4	1	0	0	1
5	0	0	1	0
6	0	1	0	0
7	0	0	0	0
8	0	1	1	1
9	0	0	1	1

(b)

Code	e	f	a·b	b⊕g
0	1	1	1	1
1	0	0	0	1
2	1	0	1	0
3	0	0	1	0
4	0	1	0	0
5	0	1	0	1
6(6')	1	1	0	1
7	0	0	1	1
8	1	1	1	0
9(9')	0	1	1	0
blank	0	0	0	0

(c)

C. Use of NOT Circuits on Output Lines

Since usually PLA's realizing a logic function f and its negation $\bar{f}$ require different complexity, we may be able to obtain a simpler PLA by introducing NOT circuits on output lines of a PLA.

Example 7: We have the following logical expressions for $\bar{f}_1$ of Table I:

$$x_5\bar{x}_6 + x_2x_6,\ x_2x_6 + x_4,\ x_5\bar{x}_6 + \bar{x}_5x_6 + x_1x_6,$$

$$x_5\bar{x}_6 + \bar{x}_5x_6 + x_1x_5.$$

$\bar{f}_1 = x_2x_6 + x_4$ is the simplest expression, since it requires three literals and only one NOT circuit. The corresponding PLA is shown in Fig. 4.

Another advantage of this method is that the possibility of the common product terms among different outputs will

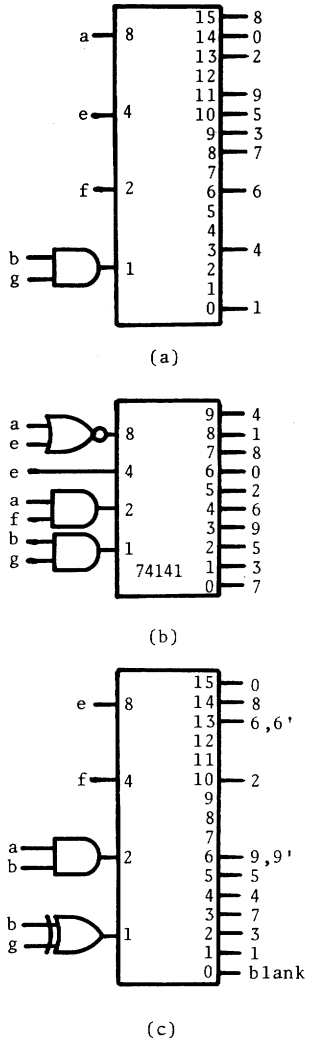

Fig. 3. Seven-segment-to-decimal converters.

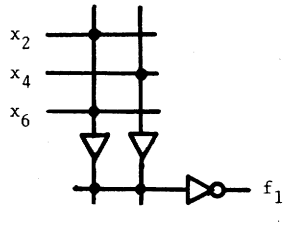

Fig. 4. PLA for the function of Table I.

increase. For example,

$$f_1 = x_1x_2x_3 + \bar{x}_1\bar{x}_2$$

$$f_2 = \bar{x}_1 + x_2$$

require four product terms, but

$$\bar{f}_1 = x_1\bar{x}_2 + \bar{x}_2x_3$$

$$\bar{f}_2 = x_1\bar{x}_2 + x_1\bar{x}_3$$

require three product terms, since $x_1\bar{x}_2$ is contained in both expressions.

In order to obtain the optimum solution by the introduction of NOT circuits on output lines, we need to select the minimum realization among minimum multiple-output expressions for $(f_1^{b_1}, f_2^{b_2}, \cdots, f_r^{b_r})$ (where $f_i^1 = f_i$ and $f_i^0 = \bar{f}_i$) for all possible combinations of $(b_1, b_2, \cdots, b_r)$ ($b_i = 0$ or 1). Usually this operation is time consuming. We have the following two simple methods which do not always produce the optimum solutions.

1) Instead of obtaining multiple-output logical expressions, obtain expressions for f_i and $\bar{f}_i$ for each $i (i = 1, 2, \cdots, r)$. Then find a combination of the expressions (for each i, exactly one of the expressions for f_i and $\bar{f}_i$ is selected) such that the combination corresponds to the lowest cost PLA.

2) When a multiple-output function $\{(I_i, Y_i)\}$ is given, we realize r functions corresponding to $i = 1, 2, \cdots, r$. The output of a PLA is always $(0, 0, \cdots, 0)$ for any input vector which is in I_0 and is not used to simplify the expression of any I_i (utilization of DON'T CARE conditions). Since such an input vector will be never added to a PLA, we can utilize the output $(0, 0, \cdots, 0)$ to simplify a PLA. If the realization of (I_k, Y_k) requires many product terms, then by introducing NOT circuits properly, the resulting PLA will produce Y_k when the original PLA produces $(0, 0, \cdots, 0)$. For this new PLA we need to consider $r - 1$ functions corresponding to $i = 1, \cdots, k - 1, k + 1, \cdots, r$. Discussions on this method is made in [9]. The following example shows these two methods.

TABLE VII

	x_1	x_2	x_3	x_4	f_1	f_2
I_1	1	1	1	0	1	1
I_2	1	0	0	0		
	0	0	1	1	1	0
I_3	1	0	1	1		
	1	1	0	1	0	1
	0	1	1	0		

Example 8: Consider the function shown in Table VII. There are the following expressions.

Expressions using x_1, x_2, and x_3:

1) $f_1 = x_1x_2x_3 + \bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2$

2) $\bar{f}_1 = x_2\bar{x}_3 + x_1\bar{x}_2x_3 + \bar{x}_1x_2$

3) $f_2 = x_1x_3 + x_2$

4) $\bar{f}_2 = \bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2$.

Expressions using x_1, x_2, and x_4:

5) $f_1 = x_1\bar{x}_4 + \bar{x}_1x_4$

6) $\bar{f}_1 = x_1x_4 + \bar{x}_1x_2$

7) $f_2 = x_2 + x_1x_4$

8) $\bar{f}_2 = \bar{x}_2\bar{x}_4 + \bar{x}_1x_4$.

Expressions using x_1, x_3, and x_4:

$$9)\quad f_1 = x_1\bar{x}_4 + \bar{x}_1 x_4$$
$$10)\quad \bar{f}_1 = x_1 x_4 + \bar{x}_1\bar{x}_4$$
$$11)\quad f_2 = x_1 x_4 + x_3\bar{x}_4$$
$$12)\quad \bar{f}_2 = \bar{x}_3\bar{x}_4 + \bar{x}_1 x_4.$$

The PLA realized by the combination of the expressions (6) and (7) is shown in Fig. 5(a).

The second method is as follows. Since I_3 contains the largest number of elements, we select I_3 as I_k. In order to convert (0, 0) to (0, 1), which is the required output for I_3, one NOT circuit is inserted to the second output. If we consider a realization using x_1, x_2, and x_3, the following input–output relation is required because of the existence of this NOT circuit.

x_1	x_2	x_3	f_1	$\bar{f}_2$
1	1	1	1	0
1	0	0	1	1
0	0	1	1	1
vectors in I_3			0	0

The resulting PLA is shown in Fig. 5(b). In this circuit all vectors in I_0 also produce output (0, 1).

V. Concluding Remarks

The following two characteristics make the design of PLA's different from the conventional logic design.

1) PLA's are suitable for realizing logic functions such that there are many input vectors whose output values are unspecified.

2) The reduction of the number of input pins is more important than the reduction of the number of product terms in order to realize an economical PLA.

For example, a 64 kbit ROM can realize an arbitrary logic function of up to 16 variables. For unspecified output values, the memory cells corresponding to them are not used. PLA's are more suitable to realize a function with many unspecified output values than ROM's since address parts can be programmed. A logic design procedure with many unspecified output values has been developed by McCluskey [1]. We have developed a Fortran program based on the idea in [10] which is an improved version of the procedure shown in [2].

The number of input pins will determine: 1) the total number of pins for a chip, and 2) the chip area. Thus, the reduction of the number of input pins is very important in designing PLA's, while for conventional random logic circuits, the gate count (i.e., the number of product term lines) mainly determines the cost. The reduction of input pins may be possible when there are many unspecified output values.

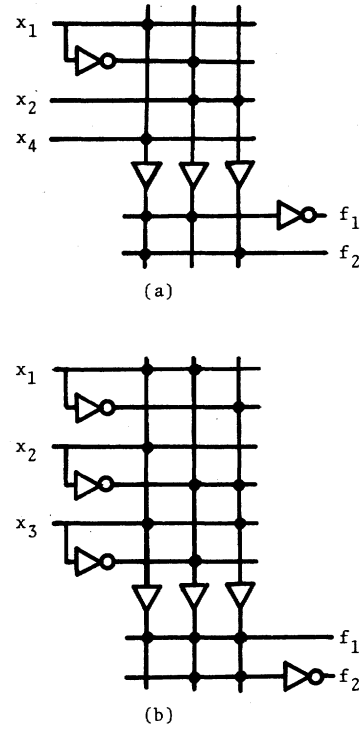

Fig. 5. Example 8.

There is a tradeoff problem between the computation time and the optimization of the result. For designing PLA circuits in high production volume, the design of an optimum circuit is more important than the cost of design. On the other hand, for circuits in low production volume, the reduction of the computation time for design is more important. In this paper we have presented optimum design procedures since there are many possible ways to change them to near optimum design procedures.

For designing circuits using PLA's or FPLA's available in the market, the design criteria are different from the criteria shown in this paper for the following reasons.

1) We need not minimize the number of input pins and the number of product terms. We need to design a circuit within the restriction for the maximum number of inputs and the maximum number of product terms.

2) For each input we cannot remove only one of an unnegated or a negated input line of the AND matrix even if it is not used in the matrix.

3) Some FPLA's do not have NOT circuits at the output portions and some FPLA's have programmable NOT circuits using Exclusive-OR gates.

In PLA's and FPLA's available in the market currently, the number of inputs is from 12 to 16 and the number of product terms is from 48 to 96. If a function with over the limit of the number of the inputs is given, Algorithms 1 or 2 (or an algorithm which removes redundant inputs in an arbitrary order) can be used. In this case it may be necessary to consider the tradeoff problem between the number of inputs and the number of product terms. By reasons 2) and 3), the reduction of NOT circuits may not be utilized in this case.

Acknowledgment

The author wishes to thank Prof. S. Yajima for valuable discussions and comments on the subject and J. L. Goodsell (now with the Amdahl Corporation) for his kind help in preparing the final manuscript.

References

[1] E. J. McCluskey, Jr., "Minimal sums for Boolean functions having many unspecified fundamental products," *AIEE Commun. Electron.*, Nov. 1962.
[2] J. R. Slagle, C. L. Chang, and R. C. Lee, "A new algorithm for generating prime implicants," *IEEE Trans. Comput.*, vol. C-19, pp. 304–310, Apr. 1970.
[3] R. C. De Vries, "Minimal set of distinct literals for a logically passive function," *J. Assoc. Comput. Mach.*, vol. 18, no. 3, pp. 431–443, July 1971.
[4] G. Reyling, "PLAs enhance digital processor speed and cut component count," *Electronics*, Aug. 8, 1974.
[5] J. Lambert, "Providing decimal output for a calculator chip," *Electronics*, p. 105, Aug. 8, 1974.
[6] J. Southway, "IC trio converts 7-segment code to decimal," *Electronics*, p. 113, Nov. 28, 1974.
[7] D. Howells, "Convert 7-segment numerical code to decimal or BCD outputs," *Electron. Design*, p. 96, Feb. 15, 1975.
[8] H. Fleisher and L. I. Maissel, "An introduction to array logic," *IBM J. Res. Develop.*, vol. 19, pp. 98–109, Mar. 1975.
[9] Y. Kambayashi, "Optimum logic design using memory-type arrays," in *Proc. IEEE Int. Symp. on Uniformly Structured Automata and Logic*, Tokyo, Japan, Aug. 1975, pp. 199–205.
[10] Y. Kambayashi, K. Okada, and S. Yajima, "Prime implicant generation of logic functions using clause selection method" (in Japanese), *Trans. IECEJ (Inst. Electron. Commun. Eng. Japan)*, vol. J62-D, pp. 89–96, Feb. 1979.

Yahiko Kambayashi (S'67–M'70) was born in Osaka, Japan, on February 15, 1943. He received the B.E., M.E., and Ph.D. degrees in electronic engineering from Kyoto University, Kyoto, Japan, in 1965, 1967, and 1970, respectively.

During 1970–1971 he was a Research Associate at Kyoto University. From 1971 to 1973 he was a Research Associate at the University of Illinois, Urbana. In 1973 he joined the Department of Information Science, Kyoto University, where he is currently an Associate Professor. His research interests include switching and automata theory, graph theory, and database theory.

Dr. Kambayashi is a member of the Association for Computing Machinery, the Institute of Electronics and Communication Engineers of Japan, and the Information Processing Society of Japan.

Fault Analysis and Test Generation for Programmable Logic Arrays (PLA's)

DANIEL L. OSTAPKO, MEMBER, IEEE, AND SE JUNE HONG, SENIOR MEMBER, IEEE

Abstract—**Programmable logic arrays (PLA's) are the logic implementation vehicle for many applications. Due to their regular structure, one is able to model and analyze many more of the likely physical faults than the conventional stuck faults considered for random combinational logic implementations. We investigate shorts between the lines and crosspoint defects (spurious absence or presence), as well as stuck faults in a PLA. It is shown that a complete crosspoint test set also detects most of all faults analyzed. The crosspoint-oriented test set is compact, easy to generate, and technology-invariant. For the test generation, the regularity of the PLA structure is utilized for ease of computation and for test set optimality. Groups of crosspoint defects are sensitized simultaneously. For each such fault group, a test configuration which contains the totality of the tests for the faults under consideration is efficiently generated. When the configuration is empty, there exists no test that detects the particular group of faults. A covering set of tests is then selected from the configuration. Our test generation method (TPLA) uses two basic and effective heuristics; they are the initial word ordering for processing and the use of look-ahead merit function whenever there is a free choice of values in a test input variables.**

Manuscript received February 28, 1979.
The authors are with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Introduction

PROGRAMMABLE logic array (PLA) is a structure that is finding increasing usage. The PLA, which is conceptually a two-level AND–OR, is attractive in LSI due to its memory-like regular structure. Fig. 1 shows an example of an eight-input, five-output PLA in a NOR–NOR implementation. The logic model in the figure shows that 0's denote connections in the input portion and 1's denote connections in the output portion. In general, PLA is a combinational logic device, comprised of the input decoders and AND and OR arrays. Fleisher and Maissel [1] describe various technology implementations and structures of PLA. This paper considers the generation and analysis of tests for PLA's.

Because of the increasing circuit density in LSI chips, faults other than the usual stuck variety are receiving increasing attention [2], [3]. As a result, memory is often tested by patterns generated by a special algorithm rather than by a general-purpose test generation procedure for stuck faults. The walking 0/1 tests [4] and the test set of Hayes [5] are examples of special-purpose test generation

0018-9340/79/0900-0617$00.75 © 1979 IEEE