Input Variable Assignment and Output Phase Optimization of PLA's

TSUTOMU SASAO, MEMBER, IEEE

Abstract — **A PLA minimization system having the following features is presented:**

1) minimization of both two-level PLA's and PLA's with two-bit decoders;

2) optimal input variable assignment to the decoders;

3) optimal output phase assignment; and

4) essential prime implicants detection without generating all the prime implicants.

By using this system, 16 control circuits for microprocessors and 12 arithmetic functions were minimized under five conditions. PLA's with two-bit decoders were 12 percent smaller than two-level PLA's when the input variables were trivially assigned, and 25 percent smaller when optimally assigned. For the control circuits, more than half of the prime implicants in the solutions were essential. For the arithmetic functions, the output phase optimized PLA's were 10 percent smaller than output phase trivial ones. The number of terms required to realize n-bit adders is $6 \cdot 2^n - 4n - 4$ for two-level PLA's and $n^2 + 1$ for PLA's with two-bit decoders.

Index Terms — **Adder, complexity of logic circuits, decoder assignment, essential prime implicants, logic design, output phase optimization, programmable logic array, switching theory.**

I. Introduction

As LSI systems become complex, time and cost for the logic design increases sharply. In order to shorten the design time and to minimize design errors, automatic design has become vitally important. Programmable logic arrays (PLA's) are suitable for the automatic design because of their regular structure [1]–[3].

Several useful tools for the PLA design have been reported.

1) Minimization of logic expressions [4]–[8].
2) High-level language to PLA transformation [9]–[12].
3) Folding [13].

In this paper, we will consider the minimization of two types of PLA's: two-level PLA's (Fig. 2) and PLA's with two-bit decoders (Fig. 6). It is known that the PLA's with two-bit decoders generally requires smaller arrays than the two-level PLA's. Table I shows the number of columns of PLA's for various functions [14].

This paper describes a minimization system for PLA's which has the following features. 1) It treats both two-level PLA's and PLA's with two-bit decoders. 2) It finds a near-optimal assignment of the input variables to the decoders. 3) It finds a near-optimal output phase assignment. 4) The minimization program for logical expressions finds all the essential prime implicants without generating all the prime implicants.

Manuscript received July 11, 1983; revised March 14, 1984.

The author was with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. He is now with the Department of Electronic Engineering, Osaka University, Osaka 565, Japan.

TABLE I
NUMBER OF COLUMNS OF PLA'S ($n = 2r$)

	Two-level PLA	PLA with Two-bit Decoders
Arbitrary function (Worst Case)	2^{n-1}	2^{n-2}
Symmetric function (Worst Case)	2^{n-1}	$\sqrt{3^{n-2}}$
Parity function	2^{n-1}	$\sqrt{2^{n-2}}$
Random function of 10-variables (Average)	163	120

Section II describes a design method for minimal two-level PLA's and PLA's with two-bit decoders using characteristic functions. The two-bit adder (Table IV) is realized by a two-level PLA (Fig. 2) and a PLA with two-bit decoders (Fig. 6).

Section III describes an assign method for input variables to the decoders. The two-bit adder is further reduced by making an optimal bit pairing (Fig. 8).

Section IV describes a method for output phase assignment. The two-bit adder can be further optimized by complementing the most significant output (Fig. 9).

Section V describes the experimental results (Tables VI and VII).

Appendix A explains detail of the output phase assignment algorithm.

Appendix B describes the fast essential prime implicant detection method without generating all the prime implicants.

Appendix C derives the number of terms of PLA's for n-bit adders (Table VIII).

II. PLA and Characteristic Functions

In this section, we describe a design method for two-level PLA's and PLA's with two-bit decoders using characteristic functions.

A. Positional Cube Notation

A positional cube notation [4], [15], [16] is quite convenient for manipulating logical expressions.

Example 1: Consider a three-variable (x_1, x_2, x_3) universe. The relation of implicants, their positional cube notations, and their meaning are shown in Table II. (End of Example)

In the positional cube notation, each variable is denoted by a binary pair: x_i is denoted by 01, $\bar{x}_i$ is denoted by 10, and

0018-9340/84/1000-0879$01.00

TABLE II
POSITIONAL CUBE NOTATION

Implicant	Positional Cube Notation	Meaning
$\bar{x}_1 \cdot x_2$	x_1 x_2 x_3 10 – 01 – 11	minterms with $x_1 = 0$, $x_2 = 1$, $x_3 = 0$ or 1
$x_2 \cdot x_3$	11 – 01 – 01	minterms with $x_1 = 0$ or 1, $x_2 = 1$, $x_3 = 1$
$\bar{x}_1 \cdot x_3$	10 – 11 – 01	minterms with $x_1 = 0$, $x_2 = 0$ or 1, $x_3 = 1$
U = (universe) (constant 1)	11 – 11 – 11	minterms with $x_1 = 0$ or 1, $x_2 = 0$ or 1, $x_3 = 0$ or 1
ϕ = (null) (constant 0)	10 – 10 – 00	no minterms

DON'T CARE (missing variable) is denoted by 11. This representation has the meaning that 10 is the first of the two values (0) of the variable, 01 is the second value (1), and 11 is the first or the second or both values. The code 00 represents no value of the variable, and any cube containing a 00 for any variable position denotes a null cube.

With this notation, multiple-valued input binary functions, which will be explained in Section II-F, can be represented in a straightforward manner.

A list of cubes represents the union of the vertices covered by each cube and is called a cover. The covers exclusively covering 1's, 0's, and unspecified points are called, respectively, the ON cover, the OFF cover, and the DC cover.

B. Characteristic Functions for Multioutput Functions

For an n-input m-output function, consider an $(n + 1)$ input two-valued output function where the output part is treated as an additional input variable. The variable which represents the output part takes m values.

Example 2: A 2-input 2-output function shown in Table III(a) can be represented as Table III(b). ***A*** denotes that "if $x_1 = 0$ and $x_2 = 0$, then $f_0 = 1$ and $f_1 = 0$," whereas ***B*** denotes that "the combination $x_1 = 0$, $x_2 = 0$, $f_0 = 1$, and $f_1 = 0$ is permitted." The combination $x_1 = 0$, $x_2 = 0$, and $f_1 = 1$ is not contained in Table III(b), so it is not permitted. Let F be the function which is represented by Table III(b), then F will show all the permitted combination of inputs and outputs. F is called *characteristic function* because it contains all and only the permitted combinations of inputs and outputs.

C. Design of Two-Level PLA's

By using characteristic functions, we can design PLA's for multiple-output functions.

Example 3: Consider the function of Table III. The cover for the characteristic function F can be simplified by using a minimization program (which will be described in Section II-E) as follows:

x_1 x_2 $f_0 f_1$		x_1 x_2 $f_0 f_1$
10 — 10 — 10		
10 — 01 — 01	→	11 — 10 — 10
01 — 10 — 10		10 — 01 — 01
01 — 01 — 10		01 — 11 — 10
input output		input output
Original Cover		Minimized Cover

Fig. 1 is a two-level PLA realization. Each cube in th minimized cover corresponds to each term of the PLA. No that 0's in the input parts of the cover denote the AN connection, whereas 1's in the output part denote c connections. (End of Exampl

Example 4: Consider the two-bit adder (ADR2)

$$\begin{array}{cccc} & & x_1 & x_2 \\ & +) & x_3 & x_4 \\ \hline & f_0 & f_1 & f_2 \end{array}$$

The truth table for ADR2 is shown in Table IV. The charac teristic function is represented by

x_1	x_2	x_3	x_4	f_0	f_1	f_2
10	10	10	01	0	1	
10	10	01	10	0	1	
10	10	01	01	0	1	1
10	01	10	10	0	1	
10	01	10	01	0	1	0
10	01	01	10	0	1	1
10	01	01	01	1	0	0
01	10	10	10	0	1	0
01	10	10	01	0	1	1
01	10	01	10	1	0	0
01	10	01	01	1	0	1
01	01	10	10	0	1	1
01	01	10	01	1	0	0
01	01	01	10	1	0	1
01	01	01	01	1	1	0

It can be minimized to

01	11	01	11	1	0	0
11	01	01	01	1	0	0
01	01	11	01	1	0	0
11	10	11	01	0	0	1
11	01	11	10	0	0	1
10	01	10	01	0	1	0
01	01	01	01	1	1	0
10	01	01	10	0	1	0
01	10	10	01	0	1	0
01	11	10	10	0	1	0
10	10	01	11	0	1	0

Fig. 2 shows the two-level PLA for ADR2.

(End of Exampl

TABLE III
CHARACTERISTIC FUNCTION

x_1	x_2	$f_0 f_1$		x_1 x_2	$(f_0 f_1)$	
0	0	1 0	$\leftarrow A$	10 – 10	– 10	$\leftarrow B$
0	1	0 1		10 – 01	– 01	
1	0	1 0		01 – 10	– 10	
1	1	1 0		01 – 01	– 10	

(a) Truth Table (b) Positional Cube Notation

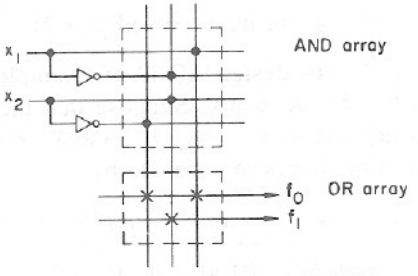


Fig. 1. Two-level PLA for Table III(a).

TABLE IV
TWO-BIT ADDER (ADR2)

x_1	x_2	x_3	x_4	f_0	f_1	f_2
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

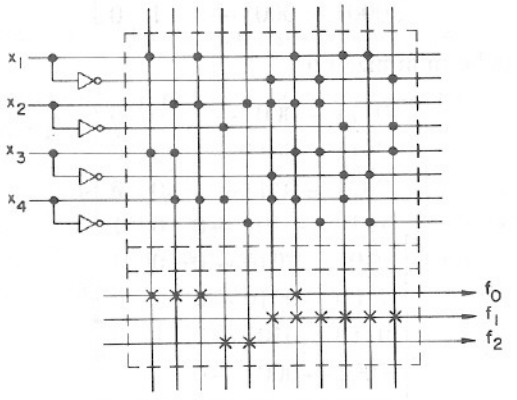


Fig. 2. Two-level PLA for ADR2.

D. PLA's with Two-Bit Decoders

As shown in Table I, PLA's with two-bit decoders generally require smaller arrays than two-level PLA's. The two-bit decoder shown in Fig. 3 generates all the maxterms of the two variables, i.e., $(x_1 \vee x_2, x_1 \vee \bar{x}_2, \bar{x}_1 \vee x_2$, and $\bar{x}_1 \vee \bar{x}_2)$. Note that the decoder here generates maxterms instead of minterms. An arbitrary two-variable function can be represented by a product of maxterms (a canonical product-of-sums expression)

TABLE V
NUMBER OF CUBES WHICH IS SUFFICIENT TO REALIZE FOR EACH OUTPUT PHASE ASSIGNMENT

Assignmemt vector	Output Assignment	Number of cubes	List of cubes
0 0 0	$\bar{f}_0\ \bar{f}_1\ \bar{f}_2$	5	P_2, P_4, P_6, P_7, P_8
0 0 1	$\bar{f}_0\ \bar{f}_1\ f_2$	5	P_2, P_4, P_5, P_6, P_7
0 1 0	$\bar{f}_0\ f_1\ \bar{f}_2$	4	P_2, P_3, P_6, P_8
0 1 1	$\bar{f}_0\ f_1\ f_2$	4	P_2, P_3, P_5, P_6
1 0 0	$f_0\ \bar{f}_1\ \bar{f}_2$	4	P_1, P_4, P_7, P_8
1 0 1	$f_0\ \bar{f}_1\ f_2$	4	P_1, P_4, P_5, P_7
1 1 0	$f_0\ f_1\ \bar{f}_2$	5	P_1, P_2, P_3, P_4, P_8
1 1 1	$f_0\ f_1\ f_2$	5	P_1, P_2, P_3, P_4, P_5

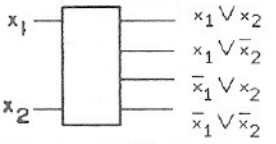


Fig. 3. Two-bit decoder.

$$f(x_1, x_2) = (c_0 \vee x_1 \vee x_2) \cdot (c_1 \vee x_1 \vee \bar{x}_2) \cdot (c_2 \vee \bar{x}_1 \vee x_2) \cdot (c_3 \vee \bar{x}_1 \vee \bar{x}_2)$$

where c_i $(i = 0, 1, 2, 3)$ is a constant 0 or 1.

An arbitrary two-variable function can be uniquely specified by a vector (c_0, c_1, c_2, c_3).

Example 5: When $(c_0, c_1, c_2, c_3) = (1, 0, 0, 1)$.

$$f(x_1, x_2) = (x_1 \vee \bar{x}_2) \cdot (\bar{x}_1 \vee x_2) = x_1 x_2 \vee \bar{x}_1 \bar{x}_2 .$$

(End of Example)

By making the AND of the maxterms for $c_i = 0$, we can realize a logic function represented by (c_0, c_1, c_2, c_3).

Example 6: Fig. 4 realizes $f(x_1, x_2) = x_1 x_2 \vee \bar{x}_1 \bar{x}_2$.

(End of Example)

Example 7: Fig. 5 realizes a function

$$f = f_1(x_1, x_2) \cdot f_2(x_3, x_4) \cdot f_3(x_5, x_6)$$

where

$$f_1(x_1, x_2) = x_1 x_2 \vee \bar{x}_1 \bar{x}_2 ,$$
$$f_2(x_3, x_4) = x_3 x_4 \vee \bar{x}_3 \bar{x}_4 ,$$
$$f_3(x_5, x_6) = x_5 x_6 \vee \bar{x}_5 \bar{x}_6 .$$

In other words,

$$f = (x_1 x_2 \vee \bar{x}_1 \bar{x}_2) \cdot (x_3 x_4 \vee \bar{x}_3 \bar{x}_4) \cdot (x_5 x_6 \vee \bar{x}_5 \bar{x}_6) .$$

This is a coincidence function for three bits. If we realize this function by a two-level PLA, we need 8 columns.

(End of Example)

E. Positional Cubes for PLA's with Two-Bit Decoders

Consider the function f in Example 7. Let the input variables be partitioned into $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$, and $X_3 = (x_5, x_6)$. Then, X_i $(i = 0, 1, 2)$ takes four values 00, 01, 10, and 11. Now, f can be represented by a positional cube

X_1				X_2				X_3			
00	01	10	11	00	01	10	11	00	01	10	11
1	0	0	1	— 1	0	0	1	— 1	0	0	1

This shows that $f = 1$ if ($X_1 = 00$ or 11) and ($X_2 = 00$ or 11)

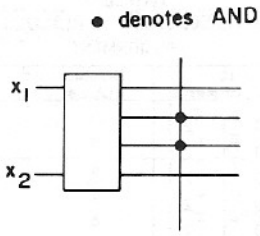


Fig. 4. Example 6.

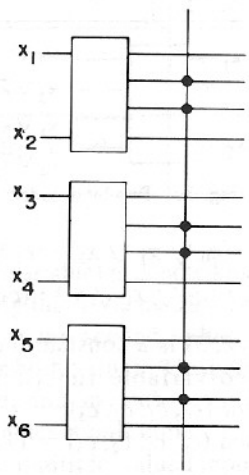


Fig. 5. Coincidence function for three bits.

and (X_3 = 00 or 11). This positional cube representation is a generalization of that for an ordinary two-valued logic function.

Lemma 2.1: Suppose that $n = 2r$, where r is a positive integer. Let the input variables be partitioned into $X_1 = (x_1, x_2), X_2 = (x_3, x_4), \cdots, X_r = (x_{n-1}, x_n)$.

1) Each column of a PLA with two-bit decoders realizes a function which has a form

$$f_1(x_1, x_2) \cdot f_2(x_3, x_4) \cdots f_r(x_{n-1}, x_n)$$

where $f_i(x_{2i-1}, x_{2i})$ is an arbitrary function of two variables.

2) A function realized by each column can be represented by a cube

$$c_0^1 c_1^1 c_2^1 c_3^1 - c_0^2 c_1^2 c_2^2 c_3^2 - \cdots\cdots\cdots\cdots - c_0^r c_1^r c_2^r c_3^r .$$

Proof: Let $I = 2i$.

1) Each decoder for $X_i = (x_{I-1}, x_I)$ generates all the maxterms of the two variables, i.e.,

$$(x_{I-1} \vee x_I), (x_{I-1} \vee \bar{x}_I), (\bar{x}_{I-1} \vee x_I), \quad \text{and} \quad (\bar{x}_{I-1} \vee \bar{x}_I) .$$

Each column of the PLA realizes a function P

$$\bigwedge_{i=1}^{r} (c_0^i \vee x_{I-1} \vee x_I) \cdot (c_1^i \vee x_{I-1} \vee \bar{x}_I) \cdot (c_2^i \vee \bar{x}_{I-1} \vee x_I) \cdot (c_3^i \vee \bar{x}_{I-1} \vee \bar{x}_I)$$

where

$$c_j^i = \begin{cases} 0 & \text{if there is an AND connection} \\ 1 & \text{if there is no connection.} \end{cases}$$

An arbitrary function $f_i(x_{I-1}, x_I)$ can be represented by a canonical product-of-sums expression

$$(c_0^i \vee x_{I-1} \vee x_I) \cdot (c_1^i \vee x_{I-1} \vee \bar{x}_I) \cdot (c_2^i \vee \bar{x}_{I-1} \vee x_I) \cdot (c_3^i \vee \bar{x}_{I-1} \vee \bar{x}_I)$$

where $c_j^i = 0$ or 1.

Hence, $P = \bigwedge_{i=1}^{r} f_i(x_{I-1}, x_I)$, and we have proved part 1).

2) Because for each part $f_i(x_{I-1}, x_I)$ can be uniquely represented by

$$(c_0^i, c_1^i, c_2^i, c_3^i), \text{ we have proved part 2).} \qquad \text{(Q.E.D.)}$$

Example 8: Let us design ADR2 of Example 4 by using PLA's with two-bit decoders. Suppose that the input variables are partitioned as $X_1 = (x_1, x_2)$, and $X_2 = (x_3, x_4)$. The characteristic function is represented by

X_1	X_2	f_0	f_1	f_2
1000	0100	0	0	1
1000	0010	0	1	0
1000	0001	0	1	1
0100	1000	0	0	1
0100	0100	0	1	0
0100	0010	0	1	1
0100	0001	1	0	0
0010	1000	0	1	0
0010	0100	0	1	1
0010	0010	1	0	0
0010	0001	1	0	1
0001	1000	0	1	1
0001	0100	1	0	0
0001	0010	1	0	1
0001	0001	1	1	0

This can be minimized to

0111	0001	1	0	0
0011	0011	1	0	0
0001	0111	1	0	0
1010	0101	0	0	1
0101	1010	0	0	1
1100	0010	0	1	0
0110	0100	0	1	0
1001	0001	0	1	0
0011	1000	0	1	0

Fig. 6 shows the PLA with two-bit decoders for ADR2. Note that 0's in the input parts denote the AND connections while 1's in the output part denote OR connection.

(End of Example)

Theorem 2.1: Suppose that the assignment of the input variables to the decoders is fixed. The necessary and sufficient number of columns of the PLA for the function $(f_0, f_1, \cdots, f_{m-1})$ is equal to the number of products in a minimum sum-of-products expression for the characteristic function F.

Proof: Let $\mathcal{F}_1$ be a minimum sum-of-product expres-

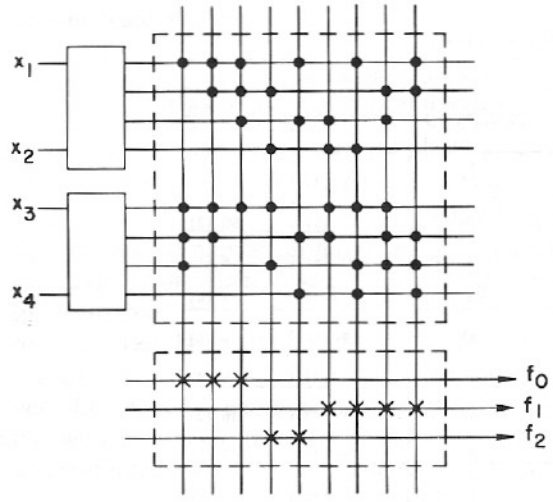


Fig. 6. PLA for ADR2 (input variable assignment nonoptimized).

sion for F, and $t_1 = t(\mathcal{F}_1)$ be the number of products in $\mathcal{F}_1$. We can realize a PLA for the functions with t_1 columns in the way shown in Section II-D. Suppose that a minimum PLA which realizes the function $(f_0, f_1, \cdots, f_{m-1})$ has s columns. Then, we can make a sum-of-products expression $\mathcal{F}_2$ with s products which represents the characteristic function F. Because $\mathcal{F}_1$ is a minimum sum-of-products expression for F, we have $t_1 \leqq s$. On the other hand, because the minimum PLA has s columns, $s \leqq t_1$. Hence, $t_1 = s$. (Q.E.D.)

F. Minimization of Characteristic Functions

From Theorem 2.1, we have the following. When the assignment of the input variables to the decoders is fixed, in order to minimize the size of the PLA, it is sufficient to obtain a minimum sum-of-products expression of the characteristic function.

Mathematically, the characteristic functions are defined as follows.

1) Characteristic function f of a single output function is a mapping $f: P_1 \times P_2 \times \cdots \times P_r \rightarrow B$ where $P_i = \{0, 1, \cdots, p_{i-1}\}$ and $i = 1, 2, \cdots, r$. $p_i = 2$ $(i = 1, 2, \cdots, r)$ if PLA's with one-bit decoders (i.e., two-level PLA's) are used, and $p_i = 4$ $(i = 1, 2, \cdots, r)$ if PLA's with two-bit decoders are used. In general, we can design PLA's where various sizes of decoders are used. In such case, if part X_i has an n_i-input decoder, then $p_i = 2^{n_i}$.

2) Characteristic function F of m-output function

$$f_j: P_1 \times P_2 \times \cdots \times P_r \rightarrow B$$

where

$$j = 0, 1, \cdots, m - 1,$$

is a mapping

$$F: P_1 \times P_2 \times \cdots \times P_r \times M \rightarrow B$$

where

$$M = \{0, 1, \cdots, m - 1\}$$

and

$$F(X_1, X_2, \cdots, X_r, j) = f_j(X_1, X_2, \cdots, X_r).$$

A class of functions which can be represented by

$$F: \overset{n}{\underset{i=1}{\times}} P_i \rightarrow B$$

where

$$P_i = \{0, 1, \cdots, p_i - 1\}$$

is called *a multiple-valued input binary function.*

Theorem 2.2: An arbitrary multiple-valued input binary function can be represented by a sum-of-products expression

$$\mathcal{F}(X_1, X_2, \cdots, X_n) = \bigvee_{(S_1, S_2, \cdots, S_n)} X_1^{S_1} \cdot X_2^{S_2} \cdot \cdots \cdot X_n^{S_n}$$

where $S_i \subseteq P_i$.

The authors have developed a program which obtains minimum sum-of-products expressions for multiple-valued input binary functions [18]. The algorithm is similar to Quine–McCluskey's method for two-valued input logic functions. The only difference is that the input variables take multiple values. We used Tison's algorithm [16] to generate all the prime implicants. Unfortunately, in the case of multiple-valued input binary functions (which correspond to PLA's with decoders), the number of prime implicants is much larger than that of the corresponding two-valued input logic function (which corresponds to two-level PLA's). Therefore, minimization of expressions for PLA's with two-bit decoders is more time consuming than that of expression for two-level PLA's. We have confirmed that our minimization program obtains minimum solutions in a reasonable computation time by using a mainframe computer, up to 10-variable problems in the case of two-valued input binary functions (for two-level PLA's), and up to 5-variable problems in the case of four-valued input binary functions (for PLA's with two-bit decoders).

Because the computation time and memory storage for the minimization program increase exponentially with the number of inputs, it is impractical to try to find absolute minimum solutions for larger problems.

MINI [4] is a heuristic program which obtains near-minimum sum-of-products expressions for multiple-valued input binary functions. Because the computation time for MINI is in most cases, roughly speaking, proportional to the square of the number of the cubes in the final solution, MINI can minimize practical PLA's with many inputs. Most PLA's shown in Tables VI and VII were minimized within a few minutes by using APL MINI II on IBM 3081K with 2 megabytes of workspace. (MINI II is an improved version of MINI developed by the author; see Section V-C.) The authors have also developed a Fortran version of MINI II, which is about 5 times faster than the APL version.

III. Assignment of the Input Variables to the Decoders

The size of PLA's with two-bit decoders can be reduced by optimizing the assignment of the input variables [14]. For the assignment problem, 16 different heuristic algorithms have

TABLE VI
NUMBER OF COLUMNS FOR ARITHMETIC PLA'S

Circuit Name In–Out–Products		Two-level PLA		PLA with two-bit decoders		
				Trivial Assignment	Assignment	near optimal
		Output Phase original	Output Phase near optimal	Output Phase original	Output Phase original	Output Phase near optimal
$f=x*y$	$n=3$ 6–6–63	31 (3)	31 (4)	27 (1)	22 (4)	19 (2)
	$n=4$ 8–8–255	127 (12)	112 (9)	108 (3)	89 (3)	80 (3)
$f=x+y$	$n=3$ 6–4–63	31 (16)	25 (10)	23 (10)	10 (8)	8 (5)
	$n=4$ 8–5–255	75 (35)	61 (24)	64 (20)	17 (12)	14 (8)
$f=\sqrt{x^2+y^2}$	$n=3$ 6–4–63	36 (18)	29 (7)	33 (9)	19 (6)	19 (6)
	$n=4$ 8–5–255	121 (23)	106 (22)	102 (13)	74 (13)	68 (4)
$f=x^2$	$n=6$ 6–12–63	49 (3)	43 (2)	44 (3)	41 (3)	36 (2)
	$n=8$ 8–16–255	185 (7)	180 (4)	163 (3)	156 (3)	147 (0)
$f=\sqrt{x}$	$n=6$ 6–4–63	23 (10)	21 (6)	18 (5)	17 (5)	14 (7)
	$n=8$ 8–5–255	57 (9)	52 (13)	42 (8)	38 (7)	33 (9)
$f=[(2^n-1)\times \log_2(x+1)/n]$	$n=6$ 6–6–63	40 (15)	35 (15)	33 (6)	34 (6)	32 (4)
	$n=8$ 8–8–255	129 (11)	118 (11)	115 (5)	109 (4)	99 (4)

Numbers in the parenthesis denote that of essential prime implicants.

TABLE VII
NUMBER OF COLUMNS FOR CONTROL PLA'S

Circuit Name In–Out–Products	Two-level PLA		PLA with Two-bit Decoders		
			Trivial Assignment	Assignment	near optimal
	Output Phase original	Output Phase near optimal	Output Phase original	Output Phase original	Output Phase near optimal
D1					
4–7–15	9 (3)	9 (3)	9 (0)	9 (0)	9 (0)
D2					
8–7–58	39 (18)	35 (16)	37 (12)	33 (10)	29 (7)
R1					
8–31–74	28 (22)	23 (12)	28 (22)	27 (20)	23 (12)
I0					
15–11–138	107 (57)		103 (36)	92 (41)	92 (41)
I1					
16–17–110	104 (54)	104 (54)	97 (31)	85 (30)	85 (30)
I2					
19–10–137	135 (85)	116 (33)	124 (68)	85 (30)	85 (30)
I3					
35–29–75	74 (44)	74 (44)	71 (38)	62 (38)	
I4					
32–20–234	212 (118)		194 (94)	152 (70)	
I5					
24–14–162	62 (53)	62 (53)	59 (45)	52 (42)	
I6					
33–23–54	54 (40)	54 (40)	52 (41)	51 (7)	
I7					
27–10–84	54 (31)	43 (31)	52 (30)	44 (7)	37 (10)
T1					
47–72–241	215 (46)		197 (48)	195 (0)	
S1					
7–3–29	29 (2)	27 (5)	21 (0)	21 (0)	20 (2)
A1					
12–8–20	20 (20)	16 (16)	20 (20)	20 (20)	16 (16)
A2					
10–8–91	68 (36)	37 (9)	48 (13)	38 (8)	26 (0)
X1					
27–6–110	110 (100)		88 (80)	80 (48)	80 (48)

Numbers in the parenthesis denote that of essential prime implicants.

been developed. Even the simplest one produced about 10 percent smaller PLA's than trivially assigned ones [17].

This section describes a simple algorithm which finds a near-optimal assignment of the input variables to the decoders. For simplicity, it is assumed that $n = 2r$, where r is a positive integer.

Definition 3.1: Let $\boldsymbol{I} = \{1, 2, \cdots, n\}$ be the set of indexes of variables in $\{X\}$. The partition of $\boldsymbol{I}$ which corresponds to the partition of $\{X\}$ is denoted by Π. The number of terms in a minimum sum-of-products expression for $f(X)$ under the partition Π is denoted by $t(f:\Pi)$.

Definition 3.2: Let an expression which represents $f(x_1, x_2, \cdots, x_n)$ be $\mathcal{F}$. The number of distinct terms which are obtained by deleting literals of x_i and x_j from $\mathcal{F}$ is denoted by $q(i,j)$.

Example 9: Let

$$\mathcal{F} = \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2x_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1x_2\bar{x}_3\bar{x}_4 .$$

The terms which are obtained by deleting the literals of x_3 and x_4 are

$$\bar{x}_1\bar{x}_2, \quad \bar{x}_1x_2, \quad x_1\bar{x}_2, \quad x_1\bar{x}_2, \quad \text{and} \quad x_1x_2.$$

The number of distinct terms is 4. So, we have $q(3,4) = 4$. Similarly, we have $q(1,3) = q(2,4) = 3, q(1,2) = q(1,4) = q(2,3) = 4$. (End of Example)

$t(f:\Pi_{ij})$ denotes the number of terms in a minimum sum-of-products expression when x_i and x_j are paired to form a four-valued variable.

Lemma 3.1: Let $\Pi_{ij} = \{[1], [2], \cdots, [j,j], \cdots, [n]\}$. $t(f:\Pi_{ij}) \leqq q(i,j)$.

Proof: Let $\mathcal{F}$ be an expression for f. Without loss of generality, we can assume that $i = 1$ and $j = 2$. Suppose that $\mathcal{F}$ is represented by a sum-of-products expression

$$\mathcal{F} = \bigvee_{(S)} x_1^{S_1} \cdot x_2^{S_2} \cdots x_n^{S_n}$$

where

$$S = (S_1, S_2, \cdots, S_n)$$

and

$$S_i \in \{*, 1, 0\}.$$

$x_i^{S_i}$ represents a literal of x_i, where

$$x_i^{S_i} = \begin{cases} 1 & \text{if } S_i = * \\ x & \text{if } S_i = 1 \\ \bar{x}_i & \text{if } S = 0. \end{cases}$$

By combining the products which have the factor $x_3^{S_3} \cdot x_4^{S_4} \cdots x_n^{S_n}$, we have

$$\mathcal{F}_1 = \bigvee_{(S^*)} \mathcal{G}(x_1, x_2, S^*) \cdot x_3^{S_3} \cdot x_4^{S_4} \cdots x_n^{S_n}$$

where

$$S^* = (S_3, S_4, \cdots, S_n)$$

and $\mathcal{G}(x_1, x_2, S^*)$ is an expression which contains no other variables than x_1 and x_2.

Let $t(\mathcal{F}_1)$ be the number of products in $\mathcal{F}_1$. Note that $t(\mathcal{F}_1)$ is equal to the number of distinct factors which have form $x_3^{S_3} \cdot x_4^{S_4} \cdots x_n^{S_n}$ in $\mathcal{F}_1$. By Definition 3.2, we have $t(\mathcal{F}_1) = q(i,j)$. Suppose that x_1 and x_2 are paired to form a variable $X_1 = (x_1, x_2)$, and that $X_1^{T_1}$ is replaced by $\mathcal{G}(x_1, x_2, S^*)$ where $T_1 \subseteq \{00, 01, 10, 11\}$, then $\mathcal{F}_1$ becomes a sum-of-products expression for f under the partition Π_{ij}. Because $t(f:\Pi_{ij})$ denotes the number of products in a minimum sum-of-products expression, we have $t(f:\Pi_{ij}) \leqq t(\mathcal{F}_1)$.

Hence, $t(f:\Pi_{ij}) \leqq q(i,j)$. (Q.E.D.)

The smaller $t(f:\Pi_{ij})$, the simpler the expression for f becomes when x_i and x_j are paired to form a four-valued variable. Because it takes much computation time to obtain $t(f:\Pi_{ij})$, we use an upper bound $q(i,j)$ instead.

Definition 3.3: An assignment graph for an n-variable function $f(x_1, x_2, \cdots, x_n)$ is a complete graph satisfying the following conditions:

1) G has n nodes ($n = 2r$);
2) the weight of the edge (i,j) is $q(i,j)$.

Algorithm 3.1 (near-optimal assignment of the input variable to the decoders):

1) Obtain a near-minimal sum-of-products expression for f.
2) Obtain the assignment graph for f.
3) Cover every node by disjoint edges so as to minimize the sum of the weights of the edges.
4) Obtain the partition of the variables corresponding to the edges.

Because Algorithm 3.1 is a heuristic one, it has no guarantee for optimality.

Example 10: Consider the function in Example 4.

1) The given expression is minimum.
2) Fig. 7 shows the assignment graph for the function of Example 4.
3) Edges (1, 3) and (2, 4) cover all the nodes of G. The sum of the weight is 7 + 8 = 15 and is the minimum.
4) The partition of $\{X\}$ is $X = (X_1, X_2)$, where $X_1 = (x_1, x_3)$ and $X_2 = (x_2, x_4)$.
5) Fig. 8 shows the PLA with the input assignment optimized. (End of Example)

In Algorithm 3.1, the most time is spent for obtaining a near-minimal sum-of-products expressions; the other time is relatively short.

IV. Output Phase Optimization

When realizing a multiple-output function $(f_0, f_1, \cdots, f_{m-1})$ by PLA's, we often have the option to realize either f_i or $\bar{f}_i$ for each output. The freedom comes from the acceptability of either form as input to the next level. Since there are 2^m different output phase assignments for m-output functions, a nonexhaustive heuristic method is desired [4]. In this section, an efficient heuristic method for the problem is described [22].

A. Double-Phase Characteristic Function

First, we will introduce a double-phase characteristic function which represents a PLA with $2m$ outputs $f_0, f_1, \cdots, f_{2m-1}$, $\bar{f}_0, \bar{f}_1, \cdots$, and $\bar{f}_{m-1}$.

Definition 4.1: Consider a set of m binary functions

$$f_j : \overset{n}{\underset{i=1}{\times}} P_i \rightarrow B$$

where

$$(j = 0, 1, \cdots, m-1)$$

and

$$P_i = \{0, 1, \cdots, p_i - 1\}.$$

$p_i = 2$ if a two-level PLA is used and $p_i = 4$ if a PLA with two-bit decoders is used. *A double-phase characteristic function* is defined as

$$F_D : \overset{n}{\underset{i=1}{\times}} P_i \times M_D \rightarrow B$$

where

$$M_D = \{0, 1, \cdots, 2m-1\}$$

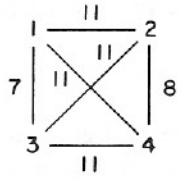


Fig. 7. Assignment graph.

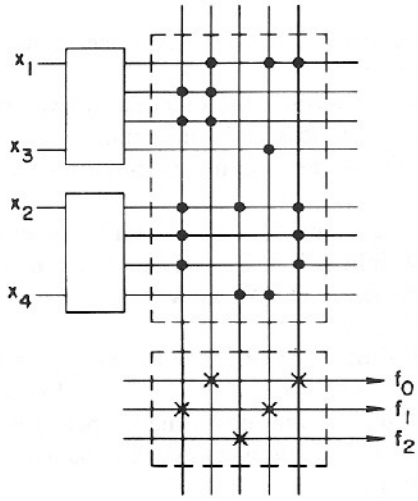


Fig. 8. PLA for ADR2 (optimal assignment of input variables).

and

$$F_D(X_1, X_2, \cdots, X_n, j) = \begin{cases} f_j(X_1, X_2, \cdots, X_n) & (j = 0, 1, \cdots, m-1) \\ \bar{f}_{j-m}(X_1, X_2, \cdots, X_n) & (j = m, m+1, \cdots, 2m-1). \end{cases}$$

Lemma 4.1: F_D can be represented by the following expression:

$$\mathcal{F}_D(X_1, X_2, \cdots, X_n, Y) = \bigvee_{(S_1, S_2, \cdots, S_n, R)} X_1^{S_1} \cdot X_2^{S_2} \cdots X_n^{S_n} \cdot Y^R$$

where $S_i \subseteq P_i$, and $R \subseteq M_D$.

Example 11: Let us optimize the output phase of the two-bit adder ADR2 in Example 10. Let the partition of the input variables $X = (x_1, x_2, x_3, x_4)$ be $X_1 = (x_1, x_3)$ and $X_2 = (x_2, x_4)$. An array for the double-phase characteristic function is

$$\mathcal{F}_D = \begin{bmatrix} 0001-1111-1 & 0 & 0 & 0 & 0 & 0 \\ 0111-0001-1 & 0 & 0 & 0 & 0 & 0 \\ 0110-1110-0 & 1 & 0 & 0 & 0 & 0 \\ 1001-0001-0 & 1 & 0 & 0 & 0 & 0 \\ 1111-0110-0 & 0 & 1 & 0 & 0 & 0 \\ 1111-1001-0 & 0 & 0 & 0 & 0 & 1 \\ 1000-1111-0 & 0 & 0 & 1 & 0 & 0 \\ 0110-1110-0 & 0 & 0 & 1 & 0 & 0 \\ 0110-0001-0 & 0 & 0 & 0 & 1 & 0 \\ 1001-1110-0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$\mathcal{F}_D$ can be minimized to

$$\mathcal{F}_D^m = \begin{array}{c} \begin{matrix} X_1 & X_2 & f_0 & f_1 & f_2 & \bar{f}_0 & \bar{f}_1 & \bar{f}_2 \end{matrix} \\ \begin{bmatrix} 0001-1111-1 & 0 & 0 & 0 & 0 & 0 \\ 0110-1110-0 & 1 & 0 & 1 & 0 & 0 \\ 1001-0001-0 & 1 & 0 & 0 & 0 & 0 \\ 0110-0001-1 & 0 & 0 & 0 & 1 & 0 \\ 1111-0110-0 & 0 & 1 & 0 & 0 & 0 \\ 1000-1111-0 & 0 & 0 & 1 & 0 & 0 \\ 1001-1110-0 & 0 & 0 & 0 & 1 & 0 \\ 1111-1001-0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{matrix}$$

Note that $\mathcal{F}_D^m$ represents a minimal PLA which realizes $f_0, f_1, f_2, \bar{f}_0, \bar{f}_1, \bar{f}_2$ simultaneously. (End of Example)

B. Idea of the Output Phase Optimization

The idea of the near-optimal output phase assignment algorithm will be illustrated by using the example.

Consider $\mathcal{F}_D^m$ of Example 11.

1) In order to realize f_0 only, two cubes are sufficient.

Proof: c_1 shows that if $X_1 = 11$ and ($X_2 = 00$, 01, 10, or 11), then $f_0 = 1$. c_4 shows that if ($X_1 = 01$ or 10) and $X_2 = 11$, then $f_0 = 1$. No other input combination makes $f_0 = 1$. (Q.E.D.)

In a similar way, we can see the number of cubes to realize other functions. The following table shows the number of cubes which is sufficient to realize each function:

Function	f_0	f_1	f_2	$\bar{f}_0$	$\bar{f}_1$	$\bar{f}_2$
Number of cubes	2	2	1	2	2	1

2) In order to realize both f_0 and $\bar{f}_1$ simultaneously, three cubes are sufficient.

Proof: c_1 shows that if $X_1 = 11$ and ($X_2 = 00$, 01, 10, or 11), then $f_0 = 1$. c_4 shows that if ($X_1 = 01$ or 10) and $X_2 = 11$, then $f_0 = \bar{f}_1 = 1$. c_7 shows that if ($X_1 = 00$ or 11) and ($X_2 = 00$, 01, or 10), then $\bar{f}_1 = 1$. No other input combination makes f_0 or $\bar{f}_1$ one. (Q.E.D.)

In a similar manner, we can know the number of cubes which is sufficient to realize two functions simultaneously.

3) In order to realize f_0, $\bar{f}_1$, and $\bar{f}_2$ simultaneously, four cubes are sufficient.

Proof: c_1 shows that if $X_1 = 11$ and ($X_2 = 00$, 01, 10, or 11), then $f_0 = 1$. c_4 shows that if ($X_1 = 01$ or 10) and $X_2 = 11$, then $f_0 = \bar{f}_1 = 1$. c_7 shows that if ($X_1 = 00$ or 11) and ($X_2 = 00$, 01, or 10), then $\bar{f}_1 = 1$. c_8 shows that if ($X_1 = 00$, 01, 10, or 11) and ($X_2 = 00$ or 11) then $\bar{f}_2 = 1$. No other input combination makes f_0 or $\bar{f}_1$ or $\bar{f}_2$ equal to one. (Q.E.D.)

4) In order to realize $f_0, f_1, f_2, \bar{f}_0, \bar{f}_1$, and $\bar{f}_2$ at the same time, all 8 cubes are sufficient.

Proof: Theorem 2.1. (Q.E.D.)

From 1)–4), we can see as follows.

Proposition 4.1: The number of products which is sufficient to realize a set of functions is equal to the number of cubes which have 1's in the corresponding outputs in $\mathcal{F}_D^m$.

Definition 4.3: An assignment vector $\boldsymbol{v} = (v_0, v_1, \cdots, v_{m-1})$

is a binary vector which denotes the output phase assignment $F^{\boldsymbol{v}} = (f_0^{v_0}, f_1^{v_1}, \cdots, f_{m-1}^{v_{m-1}})$ where

$$f_j^{v_j} = \begin{cases} f_j & \text{if } v_j = 1 \\ \bar{f}_j & \text{if } v_j = 0 \end{cases} \quad \text{and } j = 0, 1, \cdots, m-1.$$

From Proposition 4.1 and Definition 4.3, we can formulate the near-optimal output phase assignment problem as follows.

Problem 4.1: Let $\mathcal{F}_D^m$ be a minimum sum-of-products expression for F_D, $\boldsymbol{v}$ be an assignment vector, and $\mathcal{F}_D^*$ be a set of the cubes of $\mathcal{F}_D^m$ which have 1's in the corresponding output of $F^{\boldsymbol{v}}$. Find a vector $\boldsymbol{v}$ which makes $t(\mathcal{F}_D^*)$ minimum.

When m (the number of output functions) is small, then we can solve Problem 4.1 by exhaustion. But when m is large, we need an algorithmic way to solve it.

Note that only the output part of $\mathcal{F}_D^m$ contains all the information necessary to find an optimal solution for Problem 4.1.

Definition 4.4: Let $\mathcal{F}_D^m$ be a minimal sum-of-products expression for F.

An output matrix: $G = \{g_{ij}\}$ for $\mathcal{F}_D^m$ is the output part of $\mathcal{F}_D^m$. $g_{ij} = 1$ iff the jth output of the ith cube is one, and $g_{ij} = 0$, otherwise.

Example 12: The output matrix G for $\mathcal{F}_D^m$ of Example 11 is

$$G = \begin{array}{c} \begin{matrix} f_0 & f_1 & f_2 & \bar{f}_0 & \bar{f}_1 & \bar{f}_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{matrix}$$

(End of Example)

By using the output matrix, Problem 4.1 can be restated as follows.

Problem 4.2: Let $G = \{g_{ij}\}$ be an output matrix. Find an assignment vector

$$\boldsymbol{v} = (v_0, v_1, \cdots, v_{m-1})$$

which makes

$$s = \sum_{i=1}^{t} \left\{ \left(\bigvee_{j=0}^{m-1} g_{ij} \cdot v_j \right) \vee \left(\bigvee_{j=m}^{2m-1} g_{ij} \cdot \bar{v}_j \right) \right\}$$

minimum.

From here, we will consider a method to find an optimal solution for Problem 4.2 by using covering expressions. By inspecting the output matrix G of Example 12, we can see as follows.

In order to realize each function, the following cubes are necessary:

For f_0, p_1 and p_4 : For $\bar{f}_0, p_2$ and p_6 :

For f_1, p_2 and p_3 : For $\bar{f}_1, p_4$ and p_7 :

For f_2, p_5 : For $\bar{f}_2, p_8$.

The above six statements can be combined into following three statements.

In order to realize $(f_j$ or $\bar{f}_j), (j = 0, 1, 2)$, the following cubes are necessary:

For $(f_0$ or $\bar{f}_0)$, $(p_1$ and $p_4)$ or $(p_2$ and $p_6)$:

For $(f_1$ or $\bar{f}_1)$, $(p_2$ and $p_3)$ or $(p_4$ and $p_7)$:

For $(f_2$ or $\bar{f}_2)$, p_5 or p_2.

From the above three statements, we have the following statement. In order to realize $(f_0$ or $\bar{f}_0)$ and $(f_1$ or $\bar{f}_1)$ and $(f_2$ or $\bar{f}_2)$, we need $\{(p_1$ and $p_4)$ or $(p_2$ and $p_6)\}$ and $\{(p_2$ and $p_3)$ or $(p_4$ and $p_7)\}$ and $\{p_5$ or $p_8\}$.

We can simply represent the condition by the following expression:

$$Q = (\overset{f_0}{p_1} \cdot p_4 \vee \overset{\bar{f}_0}{p_2} \cdot p_6) \cdot (\overset{f_1}{p_2} \cdot p_3 \vee \overset{\bar{f}_1}{p_4} \cdot p_7) \cdot (\overset{f_2}{p_5} \vee \overset{\bar{f}_2}{p_8})$$

This is called *a covering expression,* which will be formally defined in Definition 4.5. If we realize (f_0, f_1, f_2), we need $(p_1 \cdot p_4) \cdot (p_2 \cdot p_3) \cdot p_5 = p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5$, i.e., the product of the three first terms in the parenthesis of the expression. In this case, we need five cubes. If we realize $(f_0, \bar{f}_1, f_2)$, we need $(p_1 \cdot p_4) \cdot (p_4 \cdot p_7) \cdot (p_5) = p_1 \cdot p_4 \cdot p_5 \cdot p_7$, i.e., the product of the first, the second, and the first terms in the parenthesis of the expression. In this case, we need four products. For other output phase assignments, we can obtain the number of cubes in a similar way. It is easy to see that by expanding Q into sum-of-products expression, we can obtain the number of cubes necessary to realize for all possible output phase assignments. Table V shows the number of cubes which are sufficient to realize for each output phase assignment.

Definition 4.5: Let G be an output matrix. A *covering expression* Q of $G = \{g_{ij}\}$ is

$$Q(p_1, p_2, \cdots, p_t) = \bigwedge_{j=0}^{m-1} \left[\left\{ \bigwedge_{i=1}^{t} (p_i \vee \bar{g}_{i,j}) \right\} \vee \left\{ \bigwedge_{i=1}^{t} (p_i \vee \bar{g}_{i,j+m}) \right\} \right]$$

where t is the number of rows in G.

Example 13: The covering function of G in Example 12 is

$$Q(p_1, p_2, \cdots, p_8) = (p_1 \cdot p_4 \vee p_2 \cdot p_6) \cdot (p_2 \cdot p_3 \vee p_4 \cdot p_7) \cdot (p_5 \vee p_8) = p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_8 \vee p_1 \cdot p_4 \cdot p_5 \cdot p_7 \vee p_1 \cdot p_4 \cdot p_7 \cdot p_8 \vee p_2 \cdot p_3 \cdot p_5 \cdot p_6 \vee p_2 \cdot p_3 \cdot p_6 \cdot p_8 \vee p_2 \cdot p_4 \cdot p_5 \cdot p_6 \cdot p_7 \vee p_2 \cdot p_4 \cdot p_6 \cdot p_7 \cdot p_8.$$

(End of Example)

C. Output Phase Optimization Algorithm

Algorithm 4.1 (near-optimal output phase assignment):

1) Obtain the double-phase characteristic function F_D, and minimize it.

2) Obtain the output matrix G.

3) Obtain a (near-) optimal assignment vector $\boldsymbol{v}$ for G as follows.

Let m be the number of outputs.

When $m \leq 10$, obtain $\boldsymbol{v}$ by expanding covering expression.

When $10 < m < 30$, obtain $\boldsymbol{v}$ by a branch and bound method.

(See Algorithm A.1 and Example A.1 in Appendix A.)

When $m \geq 30$, obtain $\boldsymbol{v}$ by a heuristic method which produces a near-optimal solution. (See Algorithm A.2 and Example A.2 in Appendix A.)

4) $F^{\boldsymbol{v}} = (f_0^{v_0}, f_1^{v_1}, \cdots, f_{m-1}^{v_{m-1}})$ is a (near-) optimal output phase assignment.

5) Let s be the number of terms to realize $F^{\boldsymbol{v}}$.

Theorem 4.1: Multiple-output function $F^{\boldsymbol{v}} = (f_0^{v_0}, f_1^{v_1}, \cdots, f_{m-1}^{v_{m-1}})$ can be realized with at most s products, where $\boldsymbol{v}$ and s are obtained in Algorithm 4.1.

Proof: Clear from the explanation after Example 11. (Q.E.D.)

Example 14: In Example 13, the product term $p_2p_3p_5p_6$ has four letters and it is minimum. The double-phase characteristic function corresponding to $p_2p_3p_5p_6$ is

$$\mathcal{F}_D^* = \begin{bmatrix} 0110 — 1110 — 010100 \\ 1001 — 0001 — 010000 \\ 1111 — 0110 — 001000 \\ 1000 — 1111 — 000100 \end{bmatrix}$$

Thus, $F^{\boldsymbol{v}}$ can be realized with at most four terms and the assignment vector for it is $\boldsymbol{v} = (0, 1, 1)$.

Obtained output phase assignment is $(\bar{f}_0, f_1, f_2)$.

The characteristic function for $(\bar{f}_0, f_1, f_2)$ is

$$\mathcal{F}^{\boldsymbol{v}} = \begin{bmatrix} 0110 — 1110 — 110 \\ 1001 — 0001 — 010 \\ 1111 — 0110 — 001 \\ 1000 — 1111 — 100 \end{bmatrix}$$

Fig. 9 shows the realization of ADR2. (End of Example)

D. Functions with Don't Cares

For the functions with don't cares, step 1) of Algorithm 4.1 should be modified as follows.

1) Obtain the double-phase characteristic function F_D and double-phase don't care characteristic function H_D (Definition 4.6). Minimize F_D by using H_D.

Definition 4.6: Let a set of m binary functions $h_j : \times_{i=1}^{n} P_i \rightarrow B (j = 0, 1, \cdots, m-1)$ denote the unspecified part of the functions; i.e., jth function is undefined iff $h_j = 1$. *A double-phase don't care characteristic function* is defined as

$$H_D : \overset{n}{\underset{i=1}{\times}} P_i \times M_D \rightarrow B$$

where

$$H_D(X_1, X_2, \cdots, X_n, j) = \begin{cases} h_j(X_1, X_2, \cdots, X_n) & \text{when } j = 0, 1, \cdots, \text{or } m-1. \\ h_{j-m}(X_1, X_2, \cdots, X_n) & \text{when } j = m, m+1, \cdots, 2m-1. \end{cases}$$

Example 15: Consider the following 3-input 3-output function with don't cares:

$$F = \begin{bmatrix} 10–10–10–100 \\ 10–10–01–100 \\ 10–01–10–011 \\ 10–01–01–101 \\ 01–10–10–011 \\ 01–10–01–110 \\ 01–01–10–001 \end{bmatrix}$$

$$DC = \begin{bmatrix} 10–10–01–011 \\ 10–01–01–011 \\ 01–10–10–100 \\ 01–10–01–001 \\ 01–01–01–111 \end{bmatrix}$$

1) The double-phase characteristic functions are

$$F_D = \begin{bmatrix} 10–10–10–100000 \\ 10–10–01–100000 \\ 10–01–10–011000 \\ 10–01–01–101000 \\ 01–10–10–011000 \\ 01–10–01–110000 \\ 01–01–10–001000 \\ 10–10–10–000011 \\ 10–10–01–000011 \\ 10–01–10–000100 \\ 10–01–01–000010 \\ 01–10–10–000100 \\ 01–01–01–000111 \\ 01–01–10–000110 \end{bmatrix}$$

$$H_D = \begin{bmatrix} 10–10–01–011011 \\ 10–01–01–011011 \\ 01–10–10–100100 \\ 01–10–01–001001 \\ 01–01–01–111111 \end{bmatrix}$$

Note that the output part of H_D is doubled by the simple concatenation of the copy.

2) F_D is minimized to $\mathcal{F}_D^m$

$$\mathcal{F}_D^m = \begin{bmatrix} 11–11–01–101000 \\ 10–01–11–100011 \\ 01–10–11–011000 \\ 10–01–10–011100 \\ 01–01–11–001110 \end{bmatrix}$$

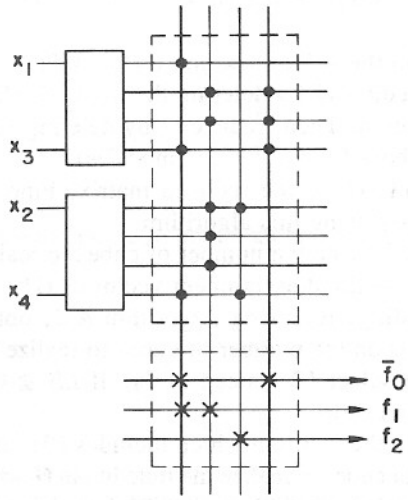


Fig. 9. PLA for ADR2 (output phase optimized).

3) The output matrix for $\mathcal{F}_D^m$ is

$$G = \begin{bmatrix} 101000 \\ 100011 \\ 011000 \\ 011100 \\ 001110 \end{bmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix}$$

4) The covering expression is

$$\begin{aligned} &Q(p_1, p_2, p_3, p_4, p_5) \\ &\quad = (p_1 \cdot p_2 \vee p_4 \cdot p_5) \cdot (p_3 \cdot p_4 \vee p_2 \cdot p_5) \\ &\qquad \cdot (p_1 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_2) \\ &\quad = p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_1 \cdot p_2 \cdot p_3 \cdot p_4 \\ &\qquad \vee p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_1 \cdot p_2 \cdot p_5 \\ &\qquad \vee p_1 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_2 \cdot p_3 \cdot p_4 \cdot p_5 \\ &\qquad \vee p_1 \cdot p_2 \cdot p_3 \cdot p_4 \cdot p_5 \vee p_2 \cdot p_4 \cdot p_5 . \end{aligned}$$

5) The product $p_1 \cdot p_2 \cdot p_5$ has three letters. The double-phase characteristic function corresponding to $p_1 \cdot p_2 \cdot p_5$ is

$$\mathcal{F}_m^* = \begin{bmatrix} 11\text{–}11\text{–}01\text{–}101000 \\ 10\text{–}10\text{–}11\text{–}100011 \\ 01\text{–}01\text{–}11\text{–}001110 \end{bmatrix}$$

6) The assignment vector is $\boldsymbol{v} = (1, 0, 0)$.

7) The obtained output phase is $(f_0, \bar{f}_1, \bar{f}_2)$.

(End of Example)

V. Experimental Results

A. Assignment of Input Variables and Output Phase Optimization

Table VI shows the number of columns of PLA's for arithmetic functions. For example, the third row shows the number of columns for 3-bit adders. It is a 6-input 4-output function, and originally has 63 product terms. For the two-level PLA, it requires 31 columns when the output phase is trivial, and 25 columns when the output phase is near-optimal. For the PLA's with two-bit decoders, it requires 23 columns when the assignment of the input variables to the decoders is trivial, ten columns when the assignment is near-optimal, and eight columns when both the assignment of the input variables and output phase are near-optimal.

From Table VI, we can see that PLA's with two-bit decoders are, on the average, 15 percent smaller than two-level PLA's when the assignments of the input variables are trivial, and 30 percent smaller when the assignments of the input variables are near-optimal. The output phase optimized PLA's are, on the average, 10 percent smaller than output phase trivial ones.

Table VII shows the number of columns for control circuit for microprocessors. In this case, PLA's with two-bit decoders are, on the average, 10 percent smaller than two-level PLA's when the assignments of the input variables are trivial, and 20 percent smaller when the assignments of the input variables are near-optimal. However, the output phase optimization reduced sizes only 3–6 percent.

Note that the size for the logarithm function ($n = 6$) for Table VI is worse than expected. This is due to the heuristics used in Algorithm 3.1 and the minimization algorithm MINI II. This shows that the heuristics or MINI II are not perfect, but they usually produce good solutions in a reasonable time.

Incidentally, for randomly generated functions of eight-variables ($d = 40$ percent; the number of the minterms is, on the average, 102.4), PLA's with two-bit decoders are, on the average, 24 percent smaller than two-level PLA's when the assignments of the input variables are trivial, and 32 percent smaller when the assignments of the input variable are optimal [14]. In this case, the optimal input variable assignments were obtained by exhaustion. There are 105 different ways for partitioning eight-input variables into four groups. We minimized 105 different expressions for each functions. Minimization of the expressions was done by using Quine–McCluskey method which obtains absolute minimum solutions.

B. Minimization Program

A PLA minimization program MINI [4] have been enhanced for the new system. The new program MINI II has the following features.

1) It uses a fast recursive complementation algorithm [20] instead of the disjoint sharp algorithm [4].

2) It detects all the essential prime implicants without generating all the prime implicants (algorithm is shown in Appendix B).

3) It has a special slim operation which will reduce the connections of both the AND and the OR array. This operation is vitally important for both the input variable assignment and the output phase optimization.

4) It is about 4–10 times faster than original MINI.

The numbers in the parenthesis in Tables VI and VII show the number of essential prime implicants. In the case of the control circuits, more than a half of the prime implicants in the solutions were essential in most cases. This fact

considerably speeds up the minimization process for the control PLA's.

C. Computation Time

All the programs are written in APL and run on IBM 3081k with 2 Mbytes of storage. Computation time depends on the size of the problem. For example, the 4-bit multiplier took about 3 min to obtain a PLA with near-optimal input assignment (89 products), and an additional 3 min to obtain a PLA with output phase near-optimal (80 products). In this case, almost all the computation time were spent for minimizing logic expressions.

VII. Conclusion

Sixteen control circuits and 12 arithmetic functions were minimized under five conditions.

1) When the assignment of the input variables to the decoders were not considered, PLA's with two-bit decoders were, on the average, 12 percent smaller than two-level PLA's.

2) When the assignment of the input variables to the decoders were near-optimal, PLA's with two-bit decoders were, on the average, 25 percent smaller than two-level PLA's.

3) In the control circuits, more than half of the prime implicants in the solutions were essential in most cases. Thus, the detection of the essential prime implicants seems to be useful for these kind of problems.

4) In the arithmetic functions, the output phase near-optimized PLA's were, on the average, 10 percent smaller than nonoptimized ones.

The number of the columns for n-bit adders is obtained as follows:

$$\begin{aligned} &6 \cdot 2^n - 4n - 5 && \text{for two-level PLA's.} \\ &n^2 + 1 && \text{for PLA's with two-bit decoders.} \end{aligned}$$

Appendix A
Near-Optimal Output Phase Assignment

Algorithm A.1 (Optimal assignment for G):

1) Let $2m$ be the number of columns of G. If $m \leq 10$, then obtain $\boldsymbol{v}$ by exhaustion (using covering expression).

2) If G has a row with all 0's, delete it from G.

3) If G has a column with all 0's, then let it be $I1$, and do step 6) and Stop.

4) (When we can select a column without loosing the optimality, select it.) Let $AG[i]$ denote the sum of the ith column of G ($i = 0, 1, \cdots, 2m-1$). If the row(s) which have 1's in the $I1$th column, have singleton 1 in the row(s), and if the row(s) which have 1's in the $I2$th column have singleton 1 in the row(s), where $I2 = I1 + m \pmod{2m}$, then select either $I1$ or $I2$, which has smaller $AG[I]$. Let it be $I1$. Do step 6) and then Stop. (For example, we can arbitrary choose either f_2 or $\bar{f}_2$, in Example 12.)

5) (Branching) Find i such that the difference of $AG[i]$ and $AG[i+m]$ is maximum, where $0 \leq i \leq m-1$. If $AG[i] \leq AG[i+m]$ then $I1 = i$: (choose to realize f_i). If $AG[i] > AG[i+m]$ then $I1 = i + m$: (choose to realize $\bar{f}_i$).

6) (Obtain the solution which includes $I1$).

6.1) Reduce G by deleting the rows which have 1's in the $I1$th column. Then, reduce G by deleting $I1$th and $I2$th columns, where $I2 = I1 + m \pmod{2m}$.

6.2) Let $G1$ be the reduced matrix. Find the optimal assignment by using this algorithm.

7) Let COST1 be the number of cubes to realize function by using the optimal assignment vector of $G1$.

8) (Bounding) By using Algorithm A.2, obtain LB, the lower bounds on the number of cubes to realize functions in G when we select $I2$ instead of $I1$. If $LB \geqq$ COST1 then choose $I1$ and Stop.

9) (Obtain the solution which includes $I2$) Let COST2 be the number of cubes to realize the function in G when we select $I2$ instead of $I1$. If COST1 > COST2 then choose the solution which include $I2$ and Stop. Otherwise, choose the other solution obtained by step 6), and then Stop.

Example A.1: Consider the output matrix G in Example 11.

1) This step is skipped for illustration.

2) There is no row with all 0's.

3) There is no column with all 0's.

4) $AG = [2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 1]$, $m = 3$. Let $I1 = 2$, and $I2 = 5$. Rows p_5 and p_8 have singleton 1's. Because $AG[2] = AG[5]$, we can arbitrary choose $I1 = 2$, which means we choose to realize f_2. Then, we will do step 6). Reduced matrix is

$$G1 = \begin{array}{c} \begin{matrix} 0 & 1 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

1) Now we are going to obtain an optimal assignment for $G1$.

2) The last row is all 0's, so delete it.

$$G1 = \begin{array}{c} \begin{matrix} 0 & 1 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

3) There is no column with all 0's.

4) $AG = [2 \quad 2 \quad 2 \quad 2]$.

5) We can arbitrarily choose $I1 = 0$ (choose to realize f_0).

6) Reduced matrix is

$$G2 = \begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \left[\begin{array}{cc} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{array}\right] \end{array}$$

1) Now, we are going to obtain an optimal assignment for $G2$.

2)

$$G2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

3) There is no column with all 0's.

4) $AG2 = [2 \quad 1]$. We select the second column, i.e., $I1 = 4$ and $I2 = 1$ (choose to realize $\bar{f}_1$).

7) Obtained assignment is $(f_0\bar{f}_1 f_2)$. COST1 = 4.

8) If we select $I2 = 3$ instead of $I1 = 0$ (i.e., if we choose to realize $\bar{f}_0$), the reduced matrix is

$$G2 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

In Algorithm A.2

1) $AG = [1 \quad 2]$

2) Lower bound on the cost of $G2$ is 1.

8) $LB = 1 + 2 + 1 = 4$. Because COST1 = LB, $(f_0, \bar{f}_1, f_2)$ is an optimal assignment. (End of Example)

Algorithm A.2 (Lower bound on the cost of G):

1) Let $AG[i]$ be the sum of ith column. $(i = 0, 1, \cdots, 2m - 1)$.

2) If $m = 1$ then $LB = \min\{AG[i], AG[m + 1]\}$ and Stop.

3) If $m > 2$, then let $I1$ be the argument which makes $\min\{AG[i], AG[i + m]\}$ maximum, where $i = 0, 1, \cdots, m - 1$. Reduce G by deleting the rows which have 1's in the $I1$th or $I2$th columns, where $I2 = I1 + m \pmod{2m}$. Then, reduce G by deleting $I1$th and $I2$th columns. Let $G1$ be the reduced matrix. Find the lower bound on the cost of $G1$ by using this algorithm. Let it be $LB1$. Let $LB = LB1 + \max_{(i)} \min\{AG[i], AG[i + m]\}$. Stop.

Example A.2: Let us obtain a lower bound on the cost of G in Example 12 by using Algorithm A.2.

1) $AG = [2 \quad 2 \quad 1 \quad 2 \quad 2 \quad 1]$.

2) $I1 = 0$ (1st column). Reduced matrix $G1$ is

$$G1 = \begin{array}{c} \begin{array}{cccc} 1 & 2 & 4 & 5 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

3) Now, we will obtain the lower bound on the cost of $G1$.

1) $AG1 = [1 \quad 1 \quad 1]$.

2) $I1 = 1$ (choose 1st column).

3) Reduced matrix $G2$ is

$$G2 = \begin{array}{c} \begin{array}{cc} 2 & 5 \end{array} \\ \left[\begin{array}{cc} 1 & 0 \\ 0 & 1 \end{array}\right] \end{array}$$

The lower bound on the cost of $G2$ is $LB2 = 1$. The lower bound on the cost of $G1$ is $LB1 = 1 + 1 = 2$. Hence, the lower bound on the cost of G is $LB = 2 + 2 = 4$.

(End of Example)

Algorithm A.3 (near-optimal assignment for G):

1) If the number of columns of G is smaller than 20, obtain $\boldsymbol{v}$ by exhaustion.

2) If G has a row with all 0's, delete the row from G.

3) Let $WG[i]$ denote the number of 1's in ith row $(i = 1, 2, \cdots, t)$. Let $AG[j] = \sum_i G[i, j]/WG[i]$.

4) Find J such that $AG[J]$ is minimum. If $0 \leqq J \leqq m - 1$ then choose to realize f_i. If $m \leqq J \leqq 2m - 1$ then choose to realize $\bar{f}_i$.

5) Let $I1 = J$. Do step 6.1) of Algorithm A.1.

6) Let $G1$ be the reduced matrix. Apply this algorithm for $G1$.

Example A.3: Let us obtain a near-optimal assignment for G of Example 12 by Algorithm A.3.

1) This step will be skipped for illustration.

2) There is no row with all 0's.

3) $WG = [1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1]$

$$AG = \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ [1.5 \quad 1.5 \quad 1.0 \quad 1.5 \quad 1.5 \quad 1.0] \end{array}$$

4) $J = 2$ (choose to realize f_2).

5) Reduced matrix is

$$G1 = \begin{array}{c} \begin{array}{cccc} 0 & 1 & 3 & 4 \end{array} \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

6) Now, we will obtain a near-optimal assignment for $G1$.

2) There is no row with all 0's.

3) $WG1 = [1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 1]$

$$AG1 = \begin{array}{c} \begin{array}{cccc} 0 & 1 & 3 & 4 \end{array} \\ [1.5 \quad 1.5 \quad 1.5 \quad 1.5] \end{array}$$

4) $J = 0$ (choose to realize f_0).

5) Reduced matrix is

$$G2 = \begin{array}{c} \begin{matrix} 1 & 4 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \end{array}$$

6) Now, we will obtain a near-optimal assignment for $G2$.
2)

$$G2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

3) $WG2 = [1 \quad 1 \quad 1]$
$AG2 = [2 \quad 1]$
4) $J = 4$ (choose to realize $\bar{f}_1$). Hence, obtained assignment is $(f_0, \bar{f}_1, f_2)$. (End of Example)

Appendix B
Detection of the Essential Prime Implicants without Generating all the Prime Implicants

In this section, we will show a fast essential prime implicant detection method without generating all the prime implicants. Although this method cannot detect the secondary essential prime implicants (prime implicants which correspond to the secondary essential rows [25]; also called secondary extremals [7]), our experiments show that this method is much faster than the local extraction algorithm [21]. This algorithm is also faster than that of [19].

Definition A.1: A product $P = X_1^{S_1} \cdot X_2^{S_2} \cdots X_n^{S_n}$ is called an *implicant* of F if F is equal to one whenever P is equal to one, and denoted by $P \leqslant F$. P is called a *prime implicant* of F if $P \leqslant F$ and S_i $(i = 1, 2, \cdots, n)$ are maximal. When $P \leqslant F$, F is said to *cover* P.

Definition A.2: Let c be a cube of F, and let v be a minterm of c. If the prime implicant which covers v is unique, then c is *an essential prime implicant*, and v is *distinguished minterm*.

Definition A.3: A sum-of-products expression is said to be *minimum* if it consists of the minimum number of prime implicants.

Lemma A.1: A minimum sum-of-products expression for F contains all the essential prime implicants of F, if any.

Definition A.4: Let c_1 and c_2 be cubes where

$$c_1 = X_1^{S_1} \cdot X_2^{S_2} \cdots X_n^{S_n} \quad \text{and} \quad c_2 = X_1^{T_1} \cdot X_2^{T_2} \cdots X_n^{T_n}.$$

A *consensus* of c_1 and c_2 is

$$\text{cons}(c_1, c_2) = \bigcup_{i=1}^{n=m} X_1^{S_1 \cap T_1} \cdot X_2^{S_2 \cap T_2} \cdots X_i^{S_i \cup T_i} \cdots X_n^{S_n \cap T_n}.$$

Definition A.5: Let c be a cube and $\mathcal{G}$ be an array. A consensus of c and $\mathcal{G}$ is defined as $\text{cons}(c, \mathcal{G}) = \cup_{c_i \in \mathcal{G}} \text{cons}(c, c_i)$.

Theorem A.1: Suppose that $\mathcal{F}$ can be written as $\mathcal{F} = c \vee \mathcal{G}$, where c is a prime implicant. Let $\mathcal{H} = \text{cons}(c, G)$. If $c \not\leqslant \mathcal{H}$, then c is essential.

Proof: Let c be a prime implicant where $c = X_1^{S_1} \cdot X_2^{S_2} \cdot \cdots \cdot X_n^{S_n}$ and $c \not\leqslant \mathcal{H}$. Because $c \not\leqslant \mathcal{H}$, there exists a minterm, $v = X_1^{a_1} \cdot X_2^{a_2} \cdots X_k^{a_k} \cdots X_n^{a_n}$ such that $v \in c \cdot \overline{\mathcal{H}}$, where $a_i \in S_i$ $(i = 1, 2, \cdots, n)$. Suppose that a prime implicant c' which is different from c covers v, where $c' = X_1^{T_1} \cdot X_2^{T_2} \cdots X_k^{T_k} \cdots X_n^{T_n}$. Because $c \cap c' \neq \phi$ and $c \not\subseteq c'$, we can assume that $T_k - S_k \neq \phi$, and that there is a minterm in c' such that $v' = X_1^{a_1} \cdot X_2^{a_2} \cdots X_{k-1}^{a_{k-1}} \cdot X_k^{b_k} \cdot X_{k+1}^{a_{k+1}} \cdots X_n^{a_n}$, where $b_k \in T_k - S_k$.

Because $v' \notin c$ and $v' \in c'$, v' is a minterm of $\mathcal{G}$. Therefore, there exists a cube d in $\mathcal{G}$ which contains v'. Let $d = X_1^{D_1} \cdot X_2^{D_2} \cdots X_k^{D_k} \cdots X_n^{D_n}$. Note that $a_i \in D_i$ $(i = 1, 2, \cdots, n, i \neq k)$ and $b_k \in D_k$. Consider a consensus of c and d: $h_k = \text{cons}(c, d) \supseteq X_1^{S_1 \cap D_1} \cdot X_2^{S_2 \cap D_2} \cdots X_k^{S_k \cup D_K} \cdots X_n^{S_n \cap D_n}$. Because $a_i \in S_i \cap D_i$ $(i \neq k)$ and $a_k \in S_k \cup D_k$, we have $v \in h_k$.

However, this contradicts the hypothesis that $v \in c \cdot \overline{\mathcal{H}}$ because $h_k \leqslant \mathcal{H}$. Hence, the prime implicant which covers c is unique. In other words, v is distinguished minterm and c is an essential prime implicant. (Q.E.D.)

Example A.4: Consider an array consisting of prime implicants

$$\mathcal{F} = \begin{bmatrix} 01\text{—}01\text{—}1110 \\ 01\text{—}10\text{—}0111 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix} \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{matrix}$$

Let us find the essential prime implicants of the array. $\mathcal{F}$ is written as $\mathcal{F} = c_1 \vee \mathcal{G}_1$, where

$$c_1 = \{01\text{—}01\text{—}1110\}$$

and

$$\mathcal{G}_1 = \begin{bmatrix} 01\text{—}10\text{—}0111 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix} \begin{matrix} c_2 \\ c_3 \\ c_4 \end{matrix}$$

First, make a consensus of c_1 and $\mathcal{G}_1$.

$$\mathcal{H}_1 = \text{cons}(c_1, \mathcal{G}_1) = \begin{bmatrix} 01\text{—}11\text{—}0110 \\ 11\text{—}01\text{—}0110 \end{bmatrix}$$

Because $c_1 \not\leqslant \mathcal{H}_1$, c_1 is an essential prime implicant.

$\mathcal{F}$ is written as $\mathcal{F} = c_2 \vee \mathcal{G}_2$ where

$$c_2 = \{01\text{—}10\text{—}0111\}$$

and

$$\mathcal{G}_2 = \begin{bmatrix} 01\text{—}01\text{—}1110 \\ 10\text{—}01\text{—}0111 \\ 10\text{—}11\text{—}0001 \end{bmatrix} \begin{matrix} c_1 \\ c_3 \\ c_4 \end{matrix}$$

Similarly, make a consensus of c_2 and $\mathcal{G}_2$

$$\mathcal{H}_2 = \text{cons}(c_2, \mathcal{G}_2) = \begin{bmatrix} 01\text{—}11\text{—}0110 \\ 11\text{—}10\text{—}0001 \end{bmatrix}$$

Because $c_2 \leqslant \mathcal{H}_2$, c_2 is not essential. Similarly, we can see that neither c_3 nor c_4 are essential. (End of Example)

In Theorem A.1, we have to check whether $c \leqslant \mathcal{H}$ or not. Checking it by the sharp operation [9] is quite time consuming. To check it, we have developed special

algorithms [20], [23], which are much faster than the sharp operation.

Appendix C
Complexity of Adders

In this section, we consider the number of columns of PLA with two-bit decoders for n-bit adders.

Let us design the following n-bit adder by using a PLA with two-bit decoders:

$$\begin{array}{lllll} & x_{n-1} & x_{n-2}\cdots x_1 & x_0 & \\ +) & y_{n-1} & y_{n-2}\cdots y_1 & y_0 & \\ \hline z_n & z_{n-1} & z_{n-2}\cdots z_1 & z_0 & \text{(sum)} \\ & c_{n-1} & c_{n-2}\cdots c_1 & c_0 & \text{(carry)} \end{array}$$

Let the partition of the input variables be $X_i = (x_i, y_i)$, where $(i = 0, 1, \cdots, n-1)$. We have the following relations where $\oplus$ denotes EXCLUSIVE OR. Note that $x_i \oplus y_i$, $x_i \oplus \bar{y}_i$, $x_i \vee y_i$, and $\bar{x}_i \vee \bar{y}_i$ can be realized by single term using the two-bit decoders.

$$z_i = (x_i \oplus y_i) \cdot \bar{c}_{i-1} \vee (x_i \oplus \bar{y}_i) \cdot c_{i-1}$$

$$c_i = x_i \cdot y_i \vee (x_i \vee y_i) \cdot c_{i-1}$$

$$\bar{c}_i = \bar{x}_i \cdot \bar{y}_i \vee (\bar{x}_i \vee \bar{y}_i) \cdot \bar{c}_{i-1}$$

$$z_n = c_{n-1}.$$

Let $t(f)$ be the number of terms in the expression for f; we have

$$t(z_i) = t(\bar{c}_{i-1}) + t(\bar{c}_{i-1}),$$

$$t(c_i) = 1 + t(c_{i-1}), \quad \text{and}$$

$$t(\bar{c}_i) = 1 + t(\bar{c}_{i-1}).$$

Because $t(c_0) = t(\bar{c}_0) = t(z_0) = 1$, we have

$$t(c_i) = t(\bar{c}_i) = i + 1$$

$$t(z_i) = 2i, \quad \text{where } i = 1, 2, \cdots, n-1.$$

Note that $t(z_n) = t(c_{n-1}) = n$. Let W be the number of columns for the PLA. Then,

$$W = \sum_{i=0}^{n} t(z_i) = 1 + \sum_{i=1}^{n-1} t(z_i) + n = n^2 + 1.$$

If we realize $(\bar{z}_n, z_{n-1}, \cdots, z_0)$ instead of $(z_n, z_{n-1}, \cdots, z_0)$, i.e., complement the most significant output, the size of the PLA becomes smaller. Note that

$$\bar{z}_n = \bar{c}_{n-1} = \bar{x}_{n-1} \cdot \bar{y}_{n-1} \vee (x_{n-1} \oplus y_{n-1}) \cdot \bar{c}_{n-2}$$

and

$$z_{n-1} = (x_{n-1} \oplus \bar{y}_{n-1}) \cdot c_{n-2} \vee (x_{n-1} \oplus y_{n-1}) \cdot \bar{c}_{n-2}.$$

$\bar{z}_n$ and z_{n-1} share a term $(x_{n-1} \oplus y_{n-1}) \cdot \bar{c}_{n-2}$. Therefore, we have

$$W = \sum_{i=0}^{n} t(z_i) = 1 + \sum_{i=1}^{n-1} t(z_i) + 1 = n^2 - n + 2.$$

TABLE VIII
NUMBER OF COLUMNS OF PLA'S FOR n-BIT ADDERS

Two-level PLA		
	Output Phase Non-Optimized	Output Phase Optimized
Without Carry Inputs	$6 \cdot 2^n - 4n - 5$	$6 \cdot 2^n - 4n - 3$
With Carry Inputs	$10 \cdot 2^n - 4n - 9$	$8 \cdot 2^n - 4n - 7$

PLA with decoders		
	Output Phase Non-Optimized	Output Phase Optimized
Without Carry (2,2,..., 2)	$n^2 + 1$	$n^2 - n + 2$
With Carry (2,2,..., 2,1)	$(n+1)^2$	$n^2 + n + 1$
With Carry (2,2,..., 2,3)	$n^2 + 1$	$n^2 - n + 2$

In a similar way, we obtain the size of the PLA's with carry inputs, and that of two-level PLA's. The number of columns which are sufficient to realize n-bit adders is summarized in Table VIII. In Table VIII, the number in the parenthesis shows the numbers of bits for each decoder. $(2, 2, 2, \cdots, 2, 1)$ shows that a 1-bit decoder is used for the carry input, whereas $(2, 2, 2, \cdots, 3)$ shows that a 3-bit decoder is used for x_0, y_0, and the carry input.

Although the minimality of the number for two-level PLA's in Table VIII has not been proved, the number is conjectured to be minimal for arbitrary n. The minimality for small n has been verified by using exhaustive methods. The minimality for PLA's with two-bit decoders in Table VIII has been proved [24].

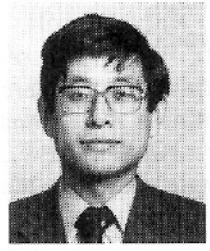
Acknowledgment

Dr. R. K. Brayton provided all the data for the control circuits as well as the results of his program ESPRESSO. Dr. S. J. Hong supplied the original APL MINI program and showed his unpublished works on "input variable assignment" and "output phase optimization" which were done many years ago with his colleague. Dr. K. Ishikawa, who was a Ph.D. degree student of the author, patiently worked with the author for developing a preliminary version of this system at Osaka University, Osaka, Japan.

References

[1] H. Fleisher and L. I. Maissel, "An introduction to array logic," *IBM J. Res. Devel.*, vol. 19, pp. 98–109, Mar. 1975.
[2] S. Muroga, *VLSI System Design*. New York: Wiley, 1982.
[3] M. Davio, J. P. Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation*. New York: Wiley, 1983.
[4] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Devel.*, pp. 443–458, Sept. 1974.
[5] P. Bricaud and J. Campbell, "Multiple output PLA minimization: EMIN," in *Proc. WESCON '78*, paper 33/3.
[6] A. Svoboda and D. E. White, *Advanced Logical Circuit Design Techniques*. New York: Garland, 1979.

[7] J. P. Roth, *Computer Logic, Testing and Verification*. Rockville, MD: Computer Science Press, 1980.
[8] R. K. Brayton, G. D. Hachtel, L. A. Hemachandra, A. R. Newton, and A. L. M. Sangiovanni-Vincentelli, "A comparison of logic minimization strategies using ESPRESSO: An APL program package for partitioned logic minimization," in *Proc. 1982 Int. Symp. on Circuit and Systems*, May 1982, pp. 42–48.
[9] D. L. Dietmeyer, *Logic Design of Digital Systems, 2nd ed.* Boston, MA: Allyn and Bacon, 1978.
[10] D. W. Brown, "A state-machine synthesizer—SMS," in *Proc. 18th Design Automat. Conf.*, June 1981.
[11] S. Kang and W. M. vanCleemput, "Automatic PLA synthesis from a DDL-P description," in *Proc. 18th Design Automat. Conf.*, June 1981, pp. 391–397.
[12] L. I. Maissel and D. L. Ostapko, "Interactive design language: A unified approach to hardware simulation, synthesis and documentation," in *Proc. 19th Design Automat. Conf.*
[13] G. D. Hachetel, A. R. Newton, and A. L. Sngiovanni-Vinventelli, "An algorithm for optimal PLA folding," *IEEE Trans. Comput. Aided Design Integrated Circuits Systems*, vol. CAD-1, no. 2, pp. 63–77, Apr. 1982.
[14] T. Sasao, "Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays," *IEEE Trans. Comput.*, vol. C-30, pp. 635–643, Sept. 1981.
[15] Y. H. Su and P. T. Cheung, "Computer minimization of multi-valued switching functions," *IEEE Trans. Comput.*, vol. C-21, pp. 995–1003, 1972.
[16] P. L. Tison, "An algebra for logic systems—Switching circuits application," *IEEE Trans. Comput.*, vol. C-20, pp. 339–351, Mar. 1971.
[17] K. Ishikawa, T. Sasao and H. Terada, "An assignment method for programmable logic arrays with decoders," *Trans. IECE Japan* (in Japanese), vol. J65-D, pp. 797–804, June 1982.
[18] ——, "A minimization algorithm for logical expressions and its bounds of application," *Trans. IECE Japan* (in Japanese), vol. J65-D, pp. 797–804, June 1982.
[19] ——, "A simplification algorithm for logical expressions: A5" *Trans. IECE Japan* (in Japanese), vol. J66-D, pp. 41–48, Jan. 1983.
[20] T. Sasao, "A fast complementation algorithm for sum-of-products expressions of multiple-valued input binary functions," in *Proc. 13th Int. Symp. on Multiple-Valued Logic*, May 1983, pp. 103–110.
[21] J. P. Roth, "Algebraic topological methods in synthesis," in *Proc. Int. Symp. on Theory of Switching*, Apr. 1957 (in *Annals Computat. Lab.*, Harvard University, Cambridge, MA, vol. 29. pp. 57–73, 1959).
[22] T. Sasao, "An application of multiple-valued logic to a design of master-slice gate array LSI," in *Proc. 12th Int. Symp. on Multiple-Valued Logic*, May 1982, pp. 45–54.
[23] ——, "Tautology checking algorithm for multiple-valued input binary functions and their application," in *Proc. 14th Int. Symp. on Multiple-Valued Logic*, May 1984.
[24] S. J. Hong, D. L. Ostapko, and H. Fleisher, private communication.
[25] S. Muroga, *Logic Design and Switching Theory*. New York: Wiley-Interscience, 1979.



Tsutomu Sasao (S'72–M'77) was born in Osaka, Japan, on January 26, 1950. He received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively.

Since 1977 he has been with Osaka University. His research interests include design automation of digital systems, switching theory, and application of microprocessors. He specializes in the design of PLA and application of multiple-valued logic to the design automation. From February 1982, he spent a year as a Visiting Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he developed a PLA minimization system.

Dr. Sasao served as Asia Area Program Chairman of the 1984 International Symposium on Multiple-Valued Logic, and is currently a member of the Executive Committee of the IEEE Computer Society Technical Committee on Multiple-Valued Logic. He has published three books on switching theory and logical design in Japanese. He is a member of the Institute of Electronics and Communication Engineers of Japan. He received the NIWA Memorial Award in 1979.