EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions

Tsutomu Sasao, *Senior Member, IEEE*

***Abstract*—Minimization of AND–EXOR programmable logic arrays (PLA's) with input decoders corresponds to minimization of the number of products in Exclusive-OR Sum-Of-Products expressions (ESOP's) for multiple-valued-input two-valued-output functions. This paper presents a simplification algorithm for ESOP's. It iteratively reduces the number of the products in ESOP's as the first objective, and then reduces the number of the literals as the second objective. Various rules are used to replace a pair of products with another one. We simplified many AND–EXOR PLA's for arithmetic circuits. In most cases, AND–EXOR PLA's required fewer products than AND–OR PLA's.**

I. Introduction

AN ORDINARY programmable logic array (PLA) has an AND–OR structure, as shown in Fig. 1. Because PLA's can be designed automatically, easily tested, and easily modified, they are extensively used in modern LSI's. By replacing the OR array with the EXOR array in the PLA, we have an AND–EXOR PLA shown in Fig. 2. AND–EXOR PLA's have several advantages over AND–OR PLA's. First, AND–EXOR PLA's often require fewer products than AND–OR PLA's. Table I compares the number of products of various classes of functions [37]. Second, AND–EXOR PLA's are easier to test than AND–OR PLA's. Similar to AND–OR PLA's [12], AND–EXOR PLA's can be made to be universal testable. However, AND–EXOR PLA's require a smaller amount of hardware and shorter test sequence [33].

Although AND–EXOR PLA's have such merits, several problems must be solved before they are used in practical designs. The first problem is that EXOR's are more expensive and slower than OR's. The second problem is that the design of AND–EXOR PLA's is more difficult than that of AND–OR PLA's.

In this paper, we consider the design problems of AND–EXOR PLA's. An AND–OR PLA is represented by a set of sum-of-products expressions (SOP's). Similarly, an AND–EXOR PLA is represented by a set of exclusive-or sum-of-products expressions (ESOP's). In both cases, the number of products in a PLA is equal to the number of different products in the expressions. Therefore, in order to minimize the size of PLA's, it is sufficient to minimize the number of different products in the expressions.

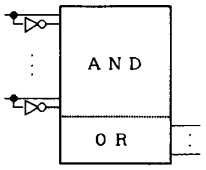

Fig. 1. AND–OR PLA with 1-bit decoders.

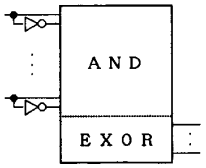

Fig. 2. AND–EXOR PLA with 1-bit decoders.

Minimization of SOP's has been studied for more than 30 years. Various algorithms have been developed to obtain minimum [21] and near-minimum PLA's [15], [4], [31]. However, the minimization of ESOP's is much more difficult than that of SOP's. No efficient method is known to obtain a minimum ESOP for a given function except for very small problems [3], [16], [17], [23], [25]. We have developed simplification algorithms for both SOP's and ESOP's, and designed various PLA's [37]. Our computer experiments show that ESOP's require fewer products than SOP's for most functions. Therefore, AND–EXOR PLA's usually require fewer products than AND–OR PLA's.

It is well known [30] that AND–OR PLA's with decoders, as shown in Fig. 3 require fewer products than AND–OR PLA's without decoders (or AND–OR PLA's with 1-bit decoders), as shown in Fig. 1. By replacing the OR array with the EXOR array in the PLA in Fig. 3, we have an AND–EXOR PLA with decoders shown in Fig. 4. This PLA structure usually requires fewer products than AND–EXOR PLA's without decoders (or AND–OR PLA's with 1-bit decoders). Similar to AND–EXOR PLA's with decoders, AND–OR PLA's with decoders are represented by ESOP's for

Manuscript received November 11, 1991; revised May 5, 1992. This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, and Culture of Japan. This paper was recommended by Associate Editor R. K. Brayton.

The author is with the Department of Computer Science and Electronic Engineering, Kyushu Institute of Technology, Iizuka 820, Japan.

IEEE Log Number 9202997.

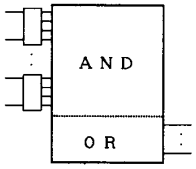

Fig. 3. AND–OR PLA with 2-bit decoders.

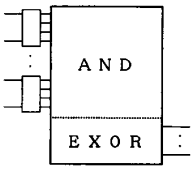

Fig. 4. AND–EXOR PLA with 2-bit decoders.

TABLE I
NUMBER OF PRODUCTS TO REALIZE VARIOUS FUNCTIONS

	AND–OR PLA (SOP)		AND–EXOR PLA (ESOP)	
	with 1-bit decoders	with 2-bit decoders	with 1-bit decoders	with 2-bit decoders
Arbitrary functions	2^{n-1}	$\frac{1}{2} \cdot 2^{n-1}$	$\frac{1}{2} \cdot 2^{n-1}$	$\frac{1}{2} \cdot 2^{n-1}$
Symmetric functions	2^{n-1}	$\frac{1}{3} \cdot 3^{n/2}$	$\frac{2}{3} \cdot 3^{n/2}$	$\frac{1}{3} \cdot 3^{n/2}$
Parity functions	2^{n-1}	$\frac{1}{2} \cdot 2^{n/2}$	n	$\frac{n}{2}$
n-bit adders	$6 \cdot 2^n - 4 \cdot n - 5$	$n^2 + 1$	$2^{n+1} - 1$	$\frac{1}{2} \cdot (n^2 + 3n - 2)$
$x_1y_1 \vee x_2y_2 \vee \cdots \vee x_ny_n$	n	n	$2^n - 1$	2

($n = 2r$, $n \geq 6$)

multiple-valued-input two-valued-output functions [34]. Table I also compares the number of products necessary to realize various classes of functions by PLA with 2-bit decoders.

Our recent research has shown that ESOP's require fewer products than SOP's to realize randomly generated functions and symmetric functions [37], [40]. To realize an arbitrary function of six variables, an ESOP requires at most 16 products, whereas an SOP requires 32 products [17]. As for the four-variable functions, ESOP's require, on the average, 3.66 products, whereas SOPs require 4.13 products [16]. Although there exists a class of functions whose ESOP realizations require more products than SOP's [40], we believe that the optimization technique of ESOP's will be an important tool in efficient logic design.

Many simplification algorithms for ESOP's have been developed for two-valued-input functions [2], [6], [10], [11], [14], [23], [28], [29], [37], [42], and for multiple-valued-input functions [24], [35], [38]. In particular, [14] presents an algorithm and minimization results for various AND–EXOR PLA's for multiple-output functions. In [24], the algorithm was extended to treat AND–EXOR PLA's with input decoders. This paper considers the same problem as [24], but uses a different approach. Preliminary versions of this paper have been published as [35] and [38].

This paper is organized as follows. In Section II, multiple-valued-input two-valued-output functions and their ESOP's are introduced. Then, the design problem of multiple-output AND–EXOR PLA's with input decoders is formulated. In Section III a simplification algorithm for ESOP's is presented. In Section IV experimental results are shown and the performance of EXMIN2 is compared with [14], [36], [42], and [6]. In Section V some applications of EXMIN2 are shown. In Section VI conclusion and comments are presented.

II. ESOP's for Multiple-Valued-Input Two-Valued-Output Functions

2.1. Multiple-Valued-Input Two-Valued-Output Functions

An AND–OR PLA with r-bit decoders realizes an SOP of a 2^r-valued-input two-valued-output function [34]. Similarly, an AND–EXOR PLA with r-bit decoder realizes an ESOP of a 2^r-valued-input function [35], [24].

Definition 2.1: A *multiple-valued-input two-valued-output function (function for short)* is a mapping

$$f(X_1, X_2, \cdots, X_n): P_1 \times P_2 \times \cdots \times P_n \rightarrow B,$$

where X_i is a multiple-valued variable, $P_i = \{0, 1, \cdots, p_i - 1\}$ is a set of values that this variable may assume, $B = \{0, 1\}$, and $p_i \geq 1$.

Definition 2.2: Let X be a variable that takes one of the values in $P = \{0, 1, \cdots, p - 1\}$. For any subset $S \subseteq P$, X^S is a *literal* representing the function that

$$X^S = \begin{cases} 1, & \text{if } X \in S \\ 0, & \text{if } X \notin S. \end{cases}$$

If $P = \{0, 1\}$, then the literals are $X^{\{0,1\}}$, $X^{\{0\}}$, $X^{\{1\}}$, and X^{ϕ}. When all the variables are two-valued, these literals may be denoted by 1, $\overline{X}$, X, and 0, respectively.

Definition 2.3: A product of literals $X_1^{S_1} \cdot X_2^{S_2} \cdot \cdots \cdot X_n^{S_n}$ is said to be a *product term* (also called *term* or *product* for short). A sum of product terms

$$\sum_{(S_1, S_2, \cdots, S_n)} \oplus X_1^{S_1} \cdot X_2^{S_2} \cdot \cdots, X_n^{S_n} \qquad (2.1)$$

is an *exclusive-or sum-of-products expression* (ESOP). When $S_i = P_i$, a literal $X_i^{S_i}$ denotes a constant 1, and so the literal is omitted from the products.

Theorem 2.1: An arbitrary multivalued-input two-valued-output function can be represented by an ESOP of the form (2.1).

Proof: Let $(a_1, a_2, \cdots, a_n)$ be an element in B^n such that $f(a_1, a_2, \cdots, a_n) = 1$. Then f can be represented by

$$\sum_{(a_1, a_2, \cdots, a_n)} \oplus X_1^{\{a_1\}} \cdot X_2^{\{a_2\}} \cdots X_n^{\{a_n\}}.$$

This expression is an ESOP. Q.E.D.

For a given function, there exist many ESOP's.

Definition 2.4: For a given function, an ESOP is *minimum* if there exists no ESOP representing the same function with fewer products.

Example 2.1: Table II shows a function

$$f(X_1, X_2, X_3): P_1 \times P_2 \times P_3 \rightarrow B$$

where $P_1 = \{0, 1\}$, $P_2 = \{0, 1, 2, 3\}$, and $P_3 = \{0, 1, 2\}$. Fig. 5 is the map representing a minimum ESOP for the function:

$$f = X_2^{\{0,3\}} \oplus X_1^{\{0\}} \cdot X_3^{\{2\}} \oplus X_1^{\{0\}} \cdot X_2^{\{2\}} \cdot X_3^{\{0\}}.$$

End of Example

Note that in the case of ESOP, every minterm of f must be covered by the loop(s) an odd number of times.

2.2. Multiple-Output Functions and Characteristic Functions

When the circuit has more than two outputs, independent minimization does not always produce total minimization. In this section, we consider a minimization method for multiple-output AND–EXOR PLA's.

Definition 2.5: Let an m-output function be $f_i(X_1, X_2, \cdots, X_n)$, where $i \in M$, and $M = \{0, 1, \cdots, m-1\}$. *A characteristic function* for the multiple-output function is defined as follows [31]:

$$F(X_1, X_2, \cdots, X_n, X_{n+1}) = \bigvee_{i=0}^{m-1} X_{n+1}^{\{i\}} \cdot f_i(X_2, \cdots, X_n)$$

where X_{n+1} denotes the ith output and assumes a value in M.

If the combination of inputs and outputs are allowed in the original multiple-output function, then $F = 1$, else $F = 0$, in the characteristic function.

Example 2.2: Consider the three-input three-output function (f_1, f_2, f_3) shown in Table III. If x_1 is replaced by X_1, (x_2, x_3) is replaced by X_2, and the output part is placed by X_3, then the characteristic function becomes one shown in Table II. A minimum ESOP for the characteristic function is

$$F = X_2^{\{0,3\}} \oplus X_1^{\{0\}} \cdot X_3^{\{2\}} \oplus X_1^{\{0\}} \cdot X_2^{\{2\}} \cdot X_3^{\{0\}}.$$

By restricting the above ESOP to $X_3 = 0$, $X_3 = 1$, and $X_3 = 2$, we have minimum ESOP's for f_1, f_2, and f_3:

$$f_0 = X_2^{\{0,3\}} \oplus X_1^{\{0\}} \cdot X_2^{\{2\}}$$

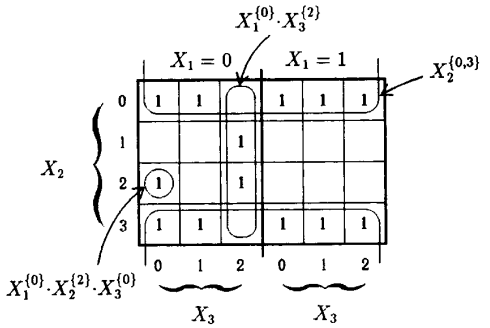

Fig. 5. Map for multiple-valued-input two-valued-output function.

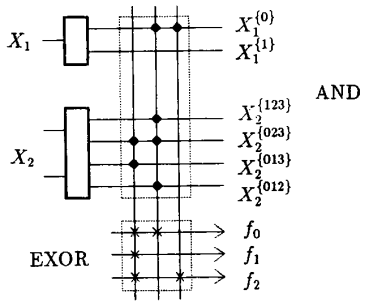

Fig. 6. AND–EXOR PLA for Table III.

TABLE II
FUNCTION f

X_1	X_2	X_3	f
0	0	0	1
0	0	1	1
0	0	2	1
0	1	0	0
0	1	1	0
0	1	2	0
0	2	0	0
0	2	1	0
0	2	2	0
0	3	0	1
0	3	1	1
0	3	2	1
1	0	0	1
1	0	1	1
1	0	2	0
1	1	0	0
1	1	1	0
1	1	2	1
1	2	0	1
1	2	1	0
1	2	2	1
1	3	0	1
1	3	1	1
1	3	2	0

$$f_1 = X_2^{\{0,3\}}$$

$$f_2 = X_3^{\{0,3\}} \oplus X_1^{\{0\}}.$$

Fig. 6 shows the PLA realizing the above ESOP.

The first decoder generates two outputs: $X_1^{\{0\}}$ and $X_1^{\{1\}}$. The second decoder generates four output: $X_2^{\{1,2,3\}}$,

TABLE III
FUNCTION (f_0, f_1, f_2)

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

$X_2^{\{0,2,3\}}$, $X_2^{\{0,1,3\}}$, and $X_2^{\{0,1,2\}}$. If we use ordinary two-valued logic expressions, these outputs are represented as $x_2 \vee x_3$, $x_2 \vee \bar{x}_3$, $\bar{x}_2 \vee x_3$, and $\bar{x}_2 \vee \bar{x}_3$, respectively. The first column of the PLA realizes the product

$$X_2^{\{0,2,3\}} \cdot X_2^{\{0,1,3\}} = X_2^{\{0,3\}} (= x_2 \cdot x_3 \vee \bar{x}_2 \cdot \bar{x}_3)$$

the second column realizes the product

$$X_1^{\{0\}} \cdot X_2^{\{0,2,3\}} \cdot X_2^{\{0,1,3\}} \cdot X_2^{\{0,1,2\}} = X_1^{\{0\}} \cdot X_2^{\{2\}} (= \bar{x}_1 \cdot x_2 \cdot x_3)$$

and the last column realizes the products $X_1^{\{0\}}$ $(= \bar{x}_1)$.

(End of Example)

From the above example, we can see that each product of the ESOP for the characteristic function corresponds to each column of the PLA realizing the multiple-output function. Thus we have the following.

Theorem 2.2: Suppose that a PLA realizes a multiple-output function $(f_0, f_1, \cdots, f_{m-1})$. Then, there exists an ESOP for characteristic function F with the same number of the products. Conversely, given an ESOP for the characteristic function for $(f_0, f_1, \cdots, f_{m-1})$. We can realize a PLA for the multiple-output function with the same number of the products.

Thus in order to minimize the number of the products in PLA for $(f_0, f_1, \cdots, f_{m-1})$, it is sufficient to minimize the number of the products in the ESOP for the characteristic function.

III. Simplification Algorithm

3.1. Outline of the Algorithm

For absolute minimization of ESOP's, no algorithm is known except for the exhaustive method [3], [16] or a method, that is applicable only to small problems [25]. For near-minimum ESOP's, most algorithms use iterative improvement methods [11], [10], [14], [24], [37], [38], [42]. Some of the above algorithms use RME's [1], [28] and others use SOP's of minterms for their initial solutions. However, they require excessive memory space when the number of the inputs are large.

The algorithm EXMIN2 is an improved version of EXMIN [35], [38], and has the following features

1) It simplifies ESOP's for multiple-valued-input two-valued-output functions.
2) As an initial solution, a disjoint SOP is derived from a simplified ESOP. This produces an initial solution of moderate size.
3) Several rules are iteratively used to reduce the number of products in ESOP's.
4) When further reduction of the number of products becomes impossible in the iterative improvement, the number of products is temporarily increased. This process is added in the EXMIN2 version. As a result the quality of the solutions has become considerably better than that of EXMIN.
5) As for multiple-output functions, the characteristic function is simplified again after each function is simplified independently.
6) For problems with many products, the input function is decomposed into sub-functions with reasonable size, and each function is simplified independently. Afterward, the total function is simplified again. This routine is also added in EXMIN2 to reduce the total computation time for the functions with many products.

3.2. Initial Solution

In order to treat large-scale PLA's, the initial solutions must be small enough to be stored in a memory of the computer. In this algorithm, the initial solutions are disjoint SOP's derived from simplified SOP's.

Definition 3.1: A SOP in which each pair of products is disjoint is called a *disjoint sum-of-products expression (DSOP)*.

In a DSOP, the OR operators can be replaced with the EXOR operators without changing the function represented by the expression. We have developed an algorithm that converts SOP's into DSOP's. Some heuristics are used to make the resulting DSOP's have as few products as possible. [32].

3.3. Simplification Rules for ESOP's

Let $A, B, C, D \subseteq P$, where $P = \{0, 1, \cdots, p-1\}$. Let $\cup$, $\cap$, and denote the union, intersection, and complement operations on sets, respectively. We use symbol $\oplus$ to denote the exclusive OR of the two sets: $A \oplus B = (\bar{A} \cap B) \cup (A \cap \bar{B})$. The same symbol $\oplus$ is also used to denote the exclusive-or of two logic functions.

The simplification of ESOP's depends on the following two theorems.

Theorem 3.1: $X^A \oplus X^B = X^{A \oplus B}$.

Theorem 3.2:

$$X^A Y^B \oplus X^C Y^D = X^{(A \oplus C)} Y^B \oplus X^C Y^{(B \oplus D)} = X^A Y^{(B \oplus D)} \oplus X^{(A \oplus C)} Y^D.$$

Proof: As for the first equation, note that

$$\{X^A Y^B \oplus X^C Y^D\} \oplus \{X^{(A \oplus C)} Y^B \oplus X^C Y^{(B \oplus D)}\} = \{X^A \oplus X^{(A \oplus C)}\} Y^B \oplus X^C \{Y^D \oplus Y^{(B \oplus D)}\} = X^C Y^B \oplus X^C Y^B = 0.$$

Hence, we have the first equation. In a similar way, we prove the second one. Q.E.D.

Note that these rules are generalization of those used by [10], [11], and [38]. However, the next theorem shows that the set of these rules is not sufficient to derive minimum ESOP's [6].

Theorem 3.3: Suppose that the laws of a commutative ring (associative, distributive, etc.) hold. A set of rules cannot generate a minimum ESOP from a certain ESOP if it satisfies the following conditions:

1) Each rule changes at most two products at a time.
2) Each rule does not increase the number of the products.

An intuitive proof: The ESOP shown in Fig. 12(a) cannot be minimized by such rules. Q.E.D.

Thus at least one rule that temporarily increases the number of the products is necessary to derive the minimum ESOP from a given ESOP. However, the addition of term-increasing rules tends to increase the computation time. Therefore, a careful selection of such a rule is very important. After conducting experiments on benchmark functions, we found the following rule to be quite effective in reducing the number of products.

Theorem 3.4: $X^P = X^{\bar{A}} \oplus X^A$, where $A \subseteq P$ and $P = \{0, 1, \cdots, p-1\}$.

3.4. Rules in EXMIN2

The rules of Theorem 3.2 are still too general to be applied directly. So, the current version of EXMIN2 uses the following rules:

1) *X*-MERGE

$$X^A \oplus X^B = X^{(A \bullet B)}$$

2) RESHAPE

$$X^A Y^B \oplus X^C Y^D = X^A Y^{(B \cap \bar{D})} \oplus X^{(A \cup C)} Y^D,$$
$$\text{if } (A \cap C = \phi, B \supset D)$$

3) DUAL-COMPLEMENT

$$X^A Y^B \oplus X^C Y^D = X^C Y^{(B \cap \bar{D})} \oplus X^{(\bar{A} \cap C)} Y^B$$
$$\text{if } (A \subset C, B \supset D)$$

4) *X*-EXPAND-1

$$X^A Y^B \oplus X^C Y^D = X^A Y^{(B \cup D)} \oplus X^{(A \cup C)} Y^D$$
$$= X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cup D)},$$
$$\text{if } (A \cap C = \phi, B \cap D = \phi)$$

5) *X*-EXPAND-2

$$X^A Y^B \oplus X^C Y^D = X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cap \bar{D})},$$
$$\text{if } (A \cap C = \phi, B \supset D)$$

6) *X*-EXPAND-3

$$X^A Y^B \oplus X^C Y^D = X^{(A \cup C)} Y^B \oplus X^C Y^{(B \bullet D)}$$
$$= X^A Y^{(B \bullet D)} \oplus X^{(A \cup C)} Y^D,$$
$$\text{if } (A \cap C = \phi, B \cap D \neq \phi)$$

7) *X*-REDUCE-1

$$X^A Y^B \oplus X^C Y^D = X^{(A \cap \bar{C})} Y^B \oplus X^C Y^{(D \cap \bar{B})},$$
$$\text{If } (A \supset C, B \subset D)$$

8) *X*-REDUCE-2

$$X^A Y^B \oplus X^C Y^D = X^{(A \cap \bar{C})} Y^B \oplus X^C Y^{(B \cap \bar{D})}$$
$$= X^A Y^{(B \cap \bar{D})} \oplus X^{(A \cap \bar{C})} Y^D,$$
$$\text{if } (A \supset C, B \supset D)$$

9) *X*-REDUCE-3

$$X^A Y^B \oplus X^C Y^D = X^{(A \cap \bar{C})} Y^B \oplus X^C Y^{(B \bullet D)},$$
$$\text{if } (A \supset C)$$

10) SPLIT

$$X^P = X^{\bar{A}} \oplus X^A.$$

Fig. 7 illustrates the above rules for four-valued-input functions. Among these rules, only *X*-MERGE reduces the number of the products in ESOP's. The other rules do not reduce the number of products, but modify the shape of products to make *X*-MERGE applicable. RESHAPE is also used to modify the shape of the products in SOP's [15]. Other rules are specific to ESOP's. For some classes of functions, such as parity functions, *X*-EXPAND-1 and *X*-EXPAND-2 are effective to reduce the number of products. *X*-EXPAND-1 produces two different results, and DUAL-COMPLEMENT interchanges the two results. Also, note that if RESHAPE (or DUAL-COMPLEMENT) is applied twice to a pair of products, then the rule produces the original pair. *X*-EXPAND-3 is a new rule introduced in EXMIN2. It is useful only for multiple-valued-input functions. Note that for two-valued functions, this rule is identical either to RESHAPE or *X*-EXPAND-2. *X*-REDUCE-1, *X*-REDUCE-2, and *X*-REDUCE-3 are reverse operations of *X*-EXPAND-1, *X*-EXPAND-2, and *X*-EXPAND-3, respectively. *X*-REDUCE-2 produces two different results, and RESHAPE interchanges the two results. It is clear that (2)–(9) are special cases of the rules of Theorem 3.2. SPLIT is also a new rule introduced in EXMIN2. The addition of this rule increases the total computation time, but reduces the number of the products for many functions.

The proposed algorithm relies on *X*-MERGE to reduce the number of the products. The other rules are used when *X*-MERGE is inapplicable. The quality of the solutions is quite sensitive to the order in which the rules are applied. However, the current version of the algorithm uses only a simple heuristic on the order of the rules and products. In the case of a two-valued-input multiple-output function, first we decompose it into single-output functions,

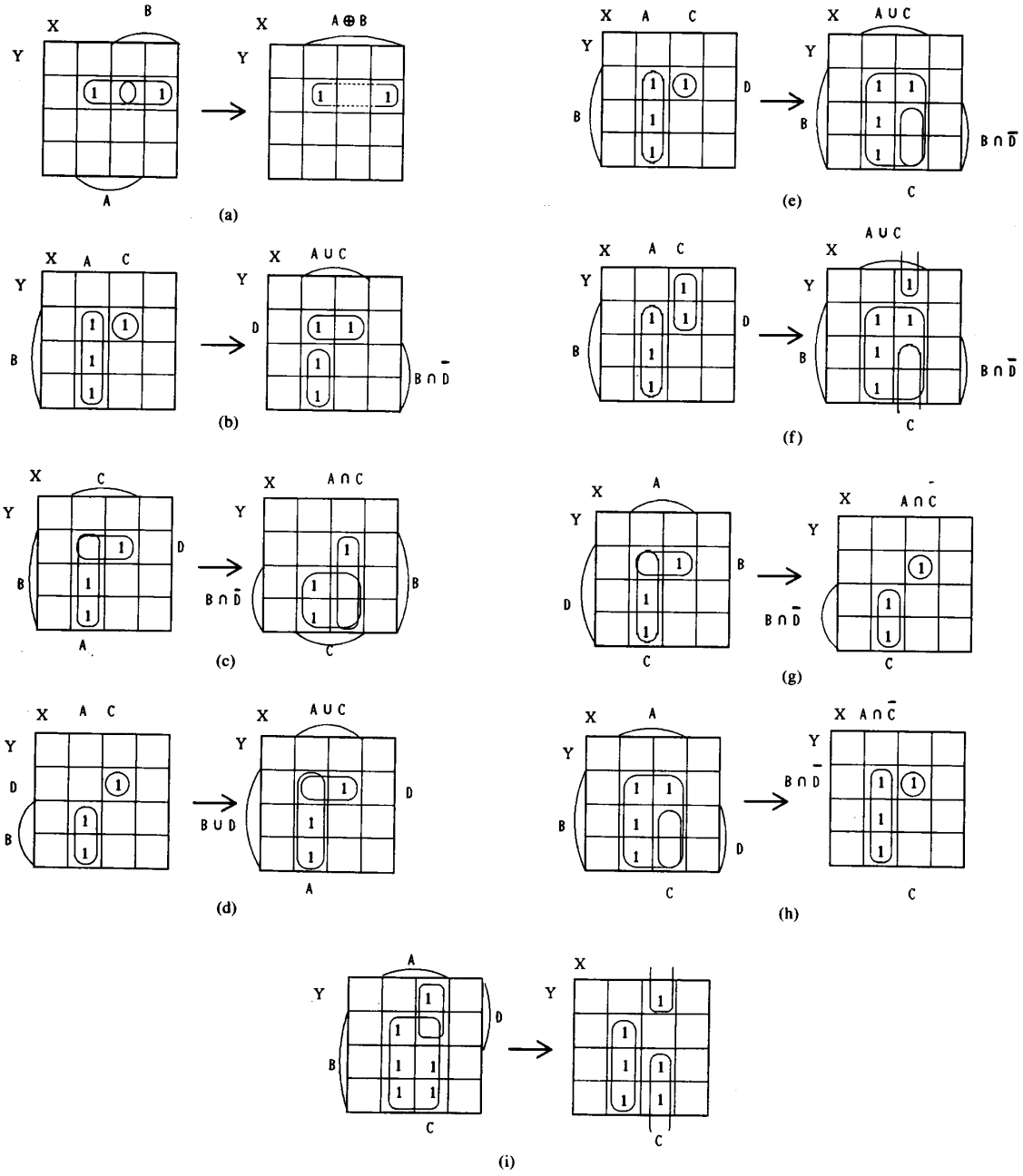

Fig. 7. Examples of simplification rules. (a) *X*-MERGE. (b) RESHAPE. (c) DUAL-COMPLEMENT. (d) *X*-EXPAND-1. (e) *X*-EXPAND-2. (f) *X*-EXPAND-3. (g) *X*-REDUCE-1. (h) *X*-REDUCE-2. (i) *X*-REDUCE-3.

and then simplify each function independently, and finally we simplify the characteristic function again. In the case of a multi-valued-input function, we have to be careful not to fall into a infinite loop of the program. That is, the successive application of *X*-EXPAND's and DUAL-COMPLEMENT can restore the original cover, and so the program may never stop.

Example 3.1: Fig. 8 illustrates that the successive application of *X*-EXPAND-2, DUAL-COMPLEMENT: *X*-EXPAND-2 and DUAL-COMPLEMENT will produce the original cover. End of Example

3.5. *Rule of EXMIN2 for Binary Case*

Although we consider ESOP's with multiple-valued inputs, the binary versions of the rules will help reader to understand the meaning of the rules. The following list

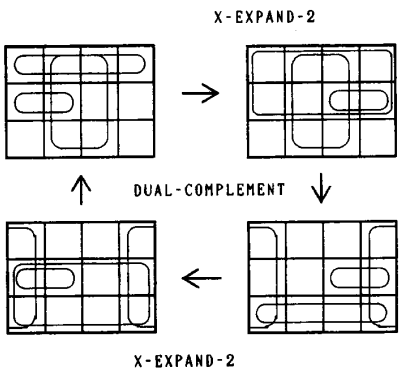

Fig. 8. Example of an infinite loop.

The volume of the original cubes is $V_2 = |A| \cdot |B| + |C| \cdot |D|$. The volume of the cubes after RESHAPE is $V_3 = |A| \cdot (|B| - |D|) + (|A| + |C|) \cdot |D|$. Hence, $V_2 = V_3$. Q.E.D.

Lemma 3.3: DUAL-COMPLEMENT does not change the volume for two-valued-input functions, but may decrease the volume for multi-valued-input functions.

Proof: The volume of the cubes after DUAL-COMPLEMENT is $V_4 = |C| \cdot (|B| - |D|) + (|C| - |A|) \cdot |B|$. So, the difference of the volume is $V_4 - V_2 = 2\{|B| \cdot |C| - |C| \cdot |D| - |A| \cdot |B|\}$. In the case of two-valued-input functions, $|B| = |C| = 2$, and $|A| = |D| = 1$. So, $V_4 = V_2$. In the case of multivalued-input functions, DUAL-COMPLEMENT may decreases the volume of the cubes as shown in Fig. 8. Q.E.D.

shows the rules for binary cases:

1) X-MERGE:	$X \oplus X = 0, X \oplus \bar{X} = 1, X \oplus 1 = \bar{X}, \bar{X} \oplus 1 = X.$
2) RESHAPE:	$X \cdot Y \oplus \bar{Y} = \bar{X} \cdot \bar{Y} \oplus X$
3) DUAL-COMPLEMENT:	$X \oplus Y = \bar{X} \oplus \bar{Y}$
4) X-EXPAND-1:	$X \cdot \bar{Y} \oplus \bar{X} \cdot Y = X \oplus Y$
5) X-EXPAND-2:	$X \cdot Y \oplus \bar{Y} = 1 \oplus \bar{X} \cdot Y$
6) X-REDUCE-1:	$X \oplus Y = X \cdot \bar{Y} \oplus \bar{X} \cdot Y$
7) X-REDUCE-2:	$1 \oplus \bar{X} \cdot Y = X \cdot Y \oplus \bar{Y}$
8) SPLIT:	$1 = \bar{X} \oplus X.$

In [10], four rules $X \oplus \bar{X} = 1$ (Merger), $X \oplus 1 = \bar{X}$ (Exclusion), X-EXPAND-2 (Increase of Order), and X-EXPAND-1 (Bridging) are used. In [11], three rules $X \oplus 1 = \bar{X}$ (Unit Distance Pair merge), X-REDUCE-1 (Two Distance Pair merge), and RESHAPE (Mixed Unit Distance Pair merge) are used. This shows that EXMIN used a much-more-powerful rule set than previously published ones. However, the XLINK operation introduced by [14] is not used in EXMIN2. XLINK operation often increases the number of the products, and is difficult to incorporate into EXMIN2.

3.6. Volume of Cubes

In order to prevent the program falling into an infinite loop, we introduce the notion of a volume of an ESOP.

Definition 3.2: Let $|S|$ denote the number of elements in S. The *volume of a product* $X_1^{S_1} \cdot X_2^{S_2} \cdot \cdots, X_n^{S_n}$ is $|S_1| \cdot |S_2| \cdot \cdots, |S_n|$. The *volume of an ESOP* is sum of volumes of the products in the ESOP.

In the case of two-valued-input single-output functions, the greater the volume, the fewer the literals are in the ESOP's. So, the increase of the volume is desirable for the reduction of the circuit cost. The following lemmas consider the change of the volume of an ESOP in applying the rules.

Lemma 3.1: X-MERGE does not increase the volume.

Proof: Let the original cubes be X^A and X^B. The volume of the original cubes is $V_0 = |A| + |B|$. The volume of the cube after X-MERGE is $V_1 = |A| + |B| - |A \cap B|$. Hence, $V_1 \leqq V_0$. Q.E.D.

Lemma 3.2: RESHAPE does not change the volume.

Proof: Let the original cubes be $X^A Y^B$ and $X^C Y^D$.

Lemma 3.4: X-EXPAND increases the volume.

Proof: 1) Volumes of the cubes after X-EXPAND-1 are

$$V_5 = |A| \cdot (|B| + |D|) + (|A| + |C|) \cdot |D|$$

and

$$V_6 = (|A| + |C|) \cdot |B| + |C| \cdot (|B| + |D|).$$

The differences of the volumes are $V_5 - V_2 = 2 \cdot |A| \cdot |D| > 0$ and $V_6 - V_2 = 2 \cdot |B| \cdot |C| > 0$.

2) The volume of the cubes after X-EXPAND-2 is $V_7 = (|A| + |C|) \cdot |B| + |C| \cdot (|B| - |D|)$. The difference of the volume is $V_7 - V_2 = 2 \cdot |C| \cdot \{|B| - |D|\} > 0$.

3) The volumes of the cubes after X-EXPANDS-3 are

$$V_8 = (|A| + |C|) \cdot |B| + |C| \cdot (|B| + |D| - |B \cap D|)$$

$$V_9 = |A| \cdot (|B| + |D| - |B \cap D|) + (|A| + |C|) \cdot |D|.$$

The differences of the volumes are $V_8 - V_2 = 2 \cdot |B| \cdot |C| - |C| \cdot |B \cap D| > 0$ and $V_9 - V_2 = 2 \cdot |A| \cdot |D| - |A| + |B \cap D| > 0$. Hence, X-EXPAND increases the volume of the cubes. Q.E.D.

Lemma 3.5: X-REDUCE decreases the volume.

Proof: X-REDUCE is the reverse operation of X-EXPAND. So X-REDUCE decreases the volume of the cubes. Q.E.D.

Example 3.2: Consider the function in Fig. 8. X-EXPAND-2 increases the volume, but DUAL-COMPLE-

MENT decrease the volume. This fact implies that the original ESOP is obtained after applying several rules.

End of Example

As shown in the previous example, the application of DUAL-COMPLEMENT after *X*-EXPAND may produce the original ESOP. In the case of two-valued-input functions, RESHAPE and DUAL-COMPLEMENT do not decrease the volume. On the other hand, in the case of multiple-valued-input functions. DUAL-COMPLEMENT may decrease the volume. To avoid the danger of an infinite loop, volume is calculated when the DUAL-COMPLEMENT is applied in the program.

3.7. Simplification Algorithm

Algorithm 3.1: (EXMIN2)

1) Convert a given SOP into a DSOP. Let it be the initial solution.
2) For a multioutput function, decompose it into single-output functions, and simplify each function independently by the method below. Then simplify the characteristic function. If the number of products is greater than some specified value p_1 (say, $p_1 = 40$), then decompose the function, so that each function has at most p_1 products. Then, simplify each function independently by the method below. Finally, simplify the total set of functions again.

   We use the following heuristic to decompose the function F: Find a variable X and a set A that does not increase the number of the products or increase the minimum of the products in $X^A \cdot F \vee X^A \cdot F$.

   2.1) Rearrange the products such that the number of 1's in the positional cube notation [31] is in ascending order.
   2.2) For each pair of products in the ESOP, check if *X*-MERGE is applicable. If so, merge them.
   2.3) For each pair of products in the ESOP, check if RESHAPE, volume non-decreasing DUAL-COMPLEMENT, *X*-EXPAND-2, *X*-EXPAND-1, and *X*-EXPAND-3 are applicable in this order. If so, apply the rule. After this, do the following: For the products modified by the above rules, check if *X*-MERGE is applicable. If so, merge them.
   2.4) For each pair of products in the ESOP, check if *X*-MERGE is applicable. If so, merge them.
   2.5) If *X*-EXPAND-1, *X*-EXPAND-2 or *X*-EXPAND-3 is applied in step (2.3), then go to (2.3).
3) Apply *X*-REDUCE-1, *X*-REDUCE-2, and *X*-EXPAND-3 in this order.
4) Simplify the characteristic function by steps (2.1)–(2.5).
5) If the number of products is reduced in step (4), then go to step (3).
6) In this step, we cannot reduce the number of products by rules 1)–9) of Section 3.4. So, we increase the number of products by SPLIT: Find a variable X and a set A that increases the minimum number of the products in $X^A \cdot F \oplus X^A \cdot F$. Simplify each sub-function independently by steps (2) through (5) of the algorithm. Then, simplify the total function again. Apply this step as long as the reduction of the number of products is possible.
7) Reduction of the number of connections (optional): For each pair of products, check if Theorem 3.2 is applicable and can reduce the connections. If so, apply the rule. This step is also newly introduced in EXMIN2.
8) Verification (optional): Verify that the simplified ESOP is logically equivalent to the original expression.

3.8. Examples of Simplification

In this section, we illustrate the simplification algorithm by using the two-valued-input four-variable function in Fig. 9. Note that the SOP is already a DSOP. Also, note that *X*-MERGE is not applicable. So, the process to start with in EXMIN2 is step 2.2). Example 3.3 will show the simplification by EXMIN2, while Example 3.4 will show the simplification by the same algorithm except that the order in which the rules are applied is interchanged.

Example 3.3: In the ESOP shown in Fig. 10(a), we cannot use *X*-MERGE even if we apply RESHAPE. Because we cannot apply DUAL-COMPLEMENT, nor *X*-EXPAND-1, we try to apply *X*-EXPAND-2. We have three pairs of products to which this rule can be applied: the first pair is (①, ②), the second one is (③, ④) and the third one is (③, ⑤). If we apply *X*-EXPAND-2 to the pair (①, ②), we have the ESOP shown in Fig. 10(b). In this ESOP, we combine ⑦ and ④ by *X*-MERGE, and obtain the ESOP shown in Fig. 10(c). Now, we can combine ③ and ⑧ by *X*-MERGE, again, and have the ESOP shown in Fig. 10(d). So, we have an ESOP with three products. End of Example

Example 3.4: In Fig. 11(a), let us apply *X*-EXPAND-1 instead of *X*-EXPAND-2. We have two options to apply this rule: one is the pair of products (①, ②), and the other is (②, ③). If we apply *X*-EXPAND-1 to the pair (①, ②), we obtain the ESOP shown in Fig. 10(b). In this ESOP, we combine ④ and ⑤ by *X*-MERGE, and have the ESOP shown in Fig. 11(c). So, we have an ESOP with four products. End of Example

As shown in the above examples, the order of the rules in the algorithm affects the quality of the solution. In Example 3.3, if we apply *X*-EXPAND-2 to the pair (③, ⑤), we get the same result as Example 3.4.

Example 3.5: In Fig. 12(a), we cannot apply rules 1) to 9). Now, we try to apply SPLIT. There are four possibilities to split to ESOP into two: $\bar{x}_1 \cdot F \oplus x_1 \cdot F$, $\bar{x}_2 \cdot F \oplus x_2 \cdot F$, $\bar{x}_3 \cdot F \oplus x_3 \cdot F$, and $\bar{x}_4 \cdot \oplus x_4 \cdot F$, and the numbers of the products after SPLIT are 4, 6, 5, 5, respectively. So, the SPLIT's that increase the number of products by the smallest amount are by x_3 and x_4. If we use x_3 to split the function, then we have the ESOP's shown in Fig. 12(b). Independent simplification produces

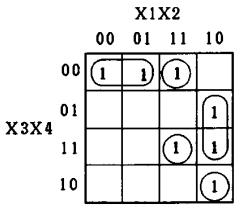

Fig. 9. Map for Examples 3.3 to 3.5.

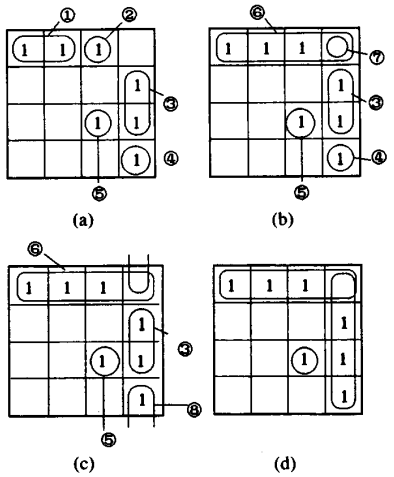

Fig. 10. Simplification by EXMIN2, as in Example 3.3. (a) The original ESOP. (b) *X*-EXPAND-2 is applied. (c) *X*-MERGE is applied. (d) *X*-MERGE is applied again, resulting in an ESOP with three products.

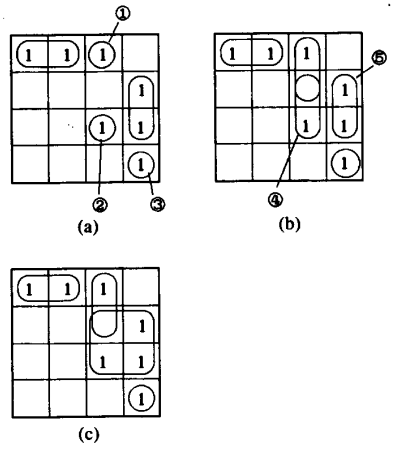

Fig. 11. Simplification by EXMIN2, as in Example 3.4. (a) The original ESOP. (b) *X*-EXPAND-1 is applied. (c) *X*-MERGE is applied, resulting in an ESOP with four products.

two ESOP's in Fig. 12(c). By applying RESHAPE to ① and ②, we have an ESOP in Fig. 12(d). By applying *X*-MERGE to ③ and ④, we have Fig. 12(e) with three products, the same result as obtained in Example 3.3.

End of Example

One advantage of ESOP representation is that the complement of a function is easily obtained.

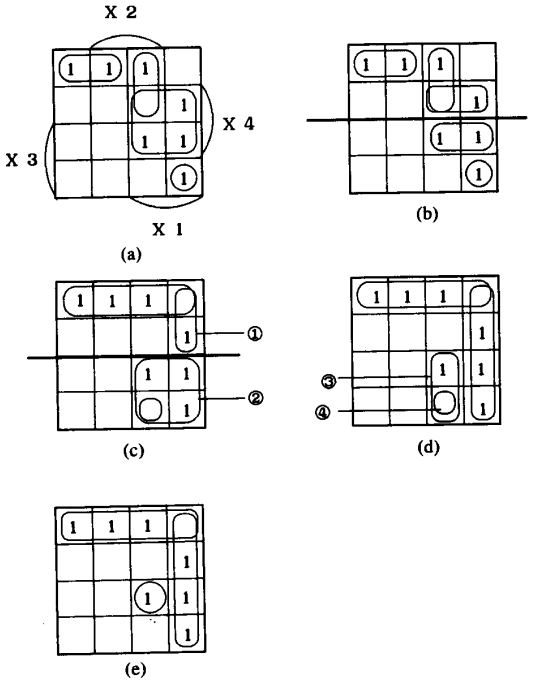

Fig. 12. Simplification by SPLIT. (a) The original ESOP. (b) x_3 used to split the function. (c) Independent simplification produces two ESOP's. (d) RESHAPE is applied. (e) *X*-MERGE is applied, resulting in an ESOP with three products, as in Example 3.3 (Fig. 10).

Example 3.6: Suppose that the SOP $F_1 = x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$ is given. First, EXMIN2 converts F into DSOP with n products:

$$F_2 = x_1 \oplus \bar{x}_1 x_2 \oplus \bar{x}_1\bar{x}_2 x_3 \oplus \bar{x}_1\bar{x}_2\bar{x}_3 x_4 \oplus \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 x_5.$$

Then, by *X*-EXPAND-2,

$$\bar{x}_1\bar{x}_2\bar{x}_3 x_4 \oplus \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 x_5 = \bar{x}_1\bar{x}_2\bar{x}_3(x_4 \oplus \bar{x}_4 x_5)$$

is replaced by $\bar{x}_1\bar{x}_2\bar{x}_3(1 \oplus \bar{x}_4\bar{x}_5)$. By *X*-MERGE, $\bar{x}_1\bar{x}_2\bar{x}_3 \oplus \bar{x}_1\bar{x}_2\bar{x}_3$ is replaced by $\bar{x}_1\bar{x}_2$. Similarly, by *X*-MERGE, $\bar{x}_1 x_2 \oplus \bar{x}_1\bar{x}_2$ is replaced by $\bar{x}_1$, and finally, $x_1 \oplus \bar{x}_1$ is replaced by 1.

Hence, we have $F_3 = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5 \oplus 1$.

End of Example

IV. Experimental Results

We coded EXMIN2 in FORTRAN and implemented it on NEC personal computers as well as on SUN workstations. EXMIN2 simplifies ESOP's with multiple-valued-inputs of arbitrary number of values. We simplified ESOP's for various arithmetic functions. Helliwell and Perkowski developed a simplification algorithm called EXORCISM and reported its performance in detail [14]. They used "primary $x$linking" and "secondary $x$linking" rules to simplify ESOP's. Saul improved the algorithm, and developed a program called HERMES [42]. Brand and Sasao developed a "non-deterministic" simplification algorithm (ND) which produces very good solutions although it is very time-consuming [6].

TABLE IV
COMPARISON WITH OTHER SYSTEMS

Data Name	EXORCISM pt	EXORCISM TIME (1)	HERMES pt	HERMES TIME (1)	EXMIN pt	EXMIN TIME (2)	EXMIN2 pt	EXMIN2 TIME (2)	ND pt
ADR4	34	98	34	24	34	4.6	31	13.5	31
MLP3	29	2	23	2	18	0.6	18	1.9	
MLP4	212	893	92	142	72	16.3	63	42.9	61
SQR6	85	28	39	8	39	2.4	37	15.0	35
9SYM	142	22	95	217	85	92.0	53	27.2	

pt number of the products
(1) cpu seconds by Gould NP1 computer
(2) cpu seconds by SPARCstation 1+

TABLE V
COMPARISON OF AND-OR WITH AND-EXOR FOR ARITHMETIC FUNCTIONS

Data Name	Number of Products: AND-OR 1bit	AND-OR 2bit	AND-EXOR 1bit	AND-EXOR 2bit	Number of Connections: AND-OR 1bit AND	AND-OR 1bit OR	AND-OR 2bit AND	AND-OR 2bit OR	AND-EXOR 1bit AND	AND-EXOR 1bit EXOR	AND-EXOR 2bit AND	AND-EXOR 2bit EXOR
ADR4	75	17	31	11	340	75	82	17	122	40	46	14
LOG8	128	112	99	92	754	257	963	159	526	164	778	166
MLP4	126	91	63	50	726	159	773	95	311	84	343	69
NRM4	120	75	71	52	708	171	665	105	404	132	431	90
RDM8	76	51	32	26	322	76	340	51	112	49	138	43
ROT8	57	42	37	26	298	87	312	62	193	64	193	49
SQR8	180	161	112	108	1057	333	1423	222	546	201	852	226
WGT8	255	54	59	25	1774	296	435	55	263	67	131	28

Table IV compares the number of products and computation time for EXORCISM, HERMES, EXMIN, EXMIN2, and ND. For these arithmetic functions, EXMIN2 produced the best solutions among EXORCISM, HERMES, EXMIN, and EXMIN2. The data for EXORCISM and HERMES were obtained by the Gould NP1 computer [42]. The data for EXMIN and EXMIN2 were obtained by the SPARC Station 1+. EXMIN is slightly modified from [38], so the results are different from [38]. The inputs of the ND are ESOP's simplified by EXMIN, and the computation time is about 50 times that of EXMIN. Thus the data of ND indicate how much room for improvement exists in the solutions if EXMIN2, which is still immature compared with the inclusive OR SOP minimization.

[24] extended the simplification algorithm for ESOP's to treat multiple-valued-input two-valued-output functions. However, the authors did not show experimental results. So, no experimental data have been published for the multiple-valued cases except for [35], [38]. Table V compares the number of products and connections AND-OR PLA's and AND-EXOR PLA's with 1-bit and 2-bit decoders for various arithmetic functions [34].

In general, AND-EXOR's require fewer connections than AND-OR's. However, the circuits with 2-bit decoders sometimes require more connections than those with 1-bit decoders. We used MINI2 [34] to obtain near-minimum solutions for AND-OR's. For the PLA's with 2-bit decoders, we used a heuristic method to find the near-optimum input assignments [31]. For the AND-EXOR PLA's, we used EXMIN2 to obtain near-minimum solutions. As for the number of products in these PLA's, we have the following results: $A > B$, $C > D$, $A > C$, and $B > D$ where

A = # of products in AND-OR PLA's, with 1-bit decoders,
B = # of products in AND-OR PLA's, with 2-bit decoders,
C = # of products in AND-EXOR PLA's, with 1-bit decoders, and
D = # of products in AND-EXOR PLA's, with 2-bit decoders.

Also, it was found again that AND-EXOR PLA's required fewer products than AND-OR PLA's.

Table VI shows the number of products and connections for circuits selected from MCNC and Berkeley benchmarks [8], [43]. In this case, we observed similar tendencies as in Table V except for vg2. vg2 requires more products and connections in the AND-EXOR than the AND-OR. This benchmark function has a pattern similar to the expression $x_1x_2 \vee x_3x_4 \vee \cdots \vee x_{2n-1}x_{2n}$, which requires $2^n - 1$ products in ESOP's [40].

We simplified many benchmark circuits and found that some have similar property as vg2, and others require

TABLE VI
NUMBER OF PRODUCTS AND CONNECTIONS FOR OTHER BENCHMARK CIRCUITS

Data name	Products		Connections				CPU sec
			SOP		ESOP		
	SOP	ESOP	AND	OR	AND	EXOR	
5xp1	67	34	266	76	125	61	13
9sym	85	53	510	85	380	53	25
add6	355	127	2196	355	732	140	430
addm4	192	91	1220	225	521	133	129
b12	41	28	150	53	124	40	4
clip	118	68	616	155	404	113	55
ex7	119	81	754	119	520	81	46
f51m	76	32	322	76	112	49	10
in7	55	35	338	79	274	59	12
intb	629	307	5274	631	2717	319	1353
life	84	54	672	84	361	54	23
m181	41	29	150	53	128	41	5
m4	104	84	668	639	488	295	189
max512	137	89	846	212	560	136	71
mlp6	1892	872	17105	2498	7521	1131	29921
rd53	31	15	140	35	41	19	2
rd73	127	42	765	147	165	56	20
rd84	255	59	1774	296	263	67	45
ryy6	112	40	624	112	328	40	13
sao2	58	29	432	78	246	62	8
seq	350	259	4493	1668	3398	1907	2797
sym10	210	84	1260	210	667	84	154
t3	33	25	218	33	166	43	5
t481	481	13	4752	481	40	13	677
vg2	110	184	804	110	1804	188	163
z4	59	29	252	59	111	34	4

cpu seconds by SPARCstation 1+

almost similar number of products in AND–EXOR's and in AND–OR's. Such circuits are omitted from Table VI. The most impressive circuit ever found was t481, which requires 481 products in the AND–OR but only 13 products in the AND–EXOR realization.

V. APPLICATIONS OF EXMIN2

EXMIN2 converts AND–OR's and AND–EXOR's. Experimental results showed that AND–EXOR's require fewer gates and fewer connections than AND–OR's in many cases. By using similar techniques to the AND–OR cases [5], we can design multilevel AND–EXOR circuits [42]. When the table look-up type field-programmable gate arrays (FPGA's) are used to implement the circuits [22], the cost of OR's and EXOR's are the same. Thus we may have more economical realization than AND–OR based circuits.

However, in many cases, EXOR's are more expensive than OR's. The cost of a two-input EXOR is 3 to 4 times more expensive than an inverter when we use CMOS technology. To solve this problem, we developed an algorithm to convert AND–EXOR's into AND–OR–EXOR's. This algorithm replaces most of the EXOR's into OR's without increasing other gates. For arithmetic circuits, the resulting circuits are much more economical than AND–OR's even if the cost of a two-input EXOR is four times more expensive than that of an inverter [41].

VI. CONCLUSION AND COMMENTS

In this paper, we presented a design method for AND–EXOR PLA's with input decoders, and formulated it as a minimization problem of ESOP's for multiple-valued-input two-valued-output functions. We developed a simplification algorithm called EXMIN2, which uses various rules. We coded it in a Fortran program, simplified various functions, and showed that EXMIN2 produces better solutions than EXORCISM and HERMES. We also simplified various arithmetic functions and showed that in most cases, AND–EXOR PLA's with 2-bit decoders require fewer products than AND–EXOR PLA's with 1-bit decoders. Also, we showed that there is a circuit whose ESOP realization requires more products than SOP realization. The data in Tables V and VI show that AND–EXOR's often require fewer products and connections than AND–OR's, especially in arithmetic circuits. EXMIN2 is currently the most powerful ESOP simplification tool that can treat multiple-valued-input functions. However, the current state of the art is still immature compared with the technique for SOP minimization. We can improve the quality of the solutions by iteratively applying EXMIN2.

ACKNOWLEDGMENT

M. Higashida worked on the first version of EXMIN [35]; Prof. Jon T. Butler helped the author at the Naval Postgraduate School, Monterey, California [38]; Dr. D. Brand developed a "non-deterministic" minimization algorithm [6]; N. Koda developed an exact ESOP minimization algorithm [16]; Prof. P. W. Besslich carefully read the manuscript; and Profs. M. Perkowski, J. Muzio, Prof. M. R. Mukerjee, and M. David sent the author related papers.

REFERENCES

[1] Ph. W. Besslich, "Efficient computer method for EXOR logic design," in *Proc. Inst. Elect. Eng.*, vol. 130, part E, pp. 203–206, 1983.
[2] Ph. W. Besslich and M. W. Riege, "An efficient program for logic synthesis of mod-2 sums expressions," in *EuroASIC'91*, Paris, France, pp. 136–141, May 1991.
[3] G. Bioul, M. Davio, and J. P. Deschamps, "Minimization of ring-sum expansions of Boolean functions," Philips Res. Rep., vol. 28, pp. 17–36, 1973.
[4] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston, MA: Kluwer, 1984.
[5] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multi-level logic optimization systems," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.
[6] D. Brand and T. Sasao, "On the minimization of AND–EXOR expressions," presented at International Workshop on Logic Synthesis, Research Triangle Park, NC, May 1991.
[7] M. Davio, J.-P. Deschamps, and A. Thayse, *Discrete and switching functions*. New York: McGraw-Hill, 1978.
[8] A. J. de Geus, "Logic synthesis and optimization benchmarks for the 1986 Design Automation Conference," in *Proc. 23rd Design Automation Conf.*, pp. 78, 1986.
[9] G. Dueck and D. M. Miller, "A 4-valued PLA using the MOD SUM," in *Proc. 16th Int. Symp. Multiple-valued Logic*, pp. 232–240, May 1986.
[10] S. Even, I. Kohavi, and A. Paz, "On minimal modulo-2 sums of products for switching functions," *IEEE Trans. Electron, Comput.*, vol. EC-16, pp. 671–674, Oct. 1984.

[11] H. Fleisher, M. Tavel, and J. Yeager, "A computer algorithm for minimizing Reed-Muller canonical forms," *IEEE Trans. Comput.*, vol. C-36, pp. 247-250, Feb. 1987.
[12] H. Fujiwara and K. Kinoshita, "A design of programmable logic arrays with universal tests," *IEEE Trans. Comput.*, vol. C-30, pp. 823-828; also *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 1027-1032, Nov. 1981.
[13] D. H. Green and I. S. Taylor, "Multiple-valued switching circuit design by means of generalized Reed-Muller expansions," *Digital Processes*, vol. 2, pp. 63-81, 1976.
[14] M. Helliwell and M. Perkowski, "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," in *Proc. 25th Design Automation Conf.*, pp. 427-432, 1988.
[15] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.*, pp. 443-458, Sept. 1974.
[16] N. Koda and T. Sasao, "Four-variable AND-EXOR minimum expressions and their properties," [in Japanese] *IEICE Trans.*, vol. J74-D-1, no. 11, pp. 765-773, Nov. 1991.
[17] N. Koda and T. Sasao, "An upper bound on the number of product terms in AND-EXOR minimum expressions," [in Japanese] *IEICE Trans.*, vol. J75-D-1, no. 3, pp. 135-142, Mar. 1992.
[18] A. Mukhopadhyay and G. Schmitz, "Minimization of exclusive OR and logical equivalence of switching circuits," *IEEE Trans. Comput.*, C-19, pp. 132-140, 1970.
[19] M. R. Mukerjee, "Minimization of ring-sum expansion of mixed polarity," presented at AMSE Symp. on Modeling and Simulation, Greensboro, NC, Oct. 1990.
[20] D. E. Muller, "Application of Boolean algebra to switching circuit design and to error detection," *IRE Trans. Electron. Comput.*, EC-3, pp. 6-12, 1954.
[21] S. Muroga, *Logic design and Switching Theory*. New York: Wiley, 1979.
[22] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Logic synthesis for programmable gate arrays," in *Proc. 27th Design Automation Conf.*, pp. 620-625, June 1990.
[23] G. Papakonstantinou, "Minimization of modulo-2 sum of products," *IEEE Trans. Comput.*, C-28, pp. 163-167, 1979.
[24] M. Perkowski, M. Helliwell, and P. Wu, "Minimization of multiple-valued input multi-output mixed-radix exclusive sum of products for incompletely specified Boolean functions," in *Proc. 19th Int. Symp. Multiple-valued Logic*, pp. 256-263, May 1989.
[25] M. Perkowski and M. Chrzanowska-Jeske, "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions," in *Proc. ISCAS*, pp. 1652-1655, June 1990.
[26] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IRE Trans. Information Theory*, PGIT-4, pp. 38-49, 1954.
[27] S. M. Reddy, "Easily testable realization for logic functions," *IEEE Trans. Comput.*, C-21, pp. 1083-1088, 1972.
[28] J. P. Robinson and Chia-Lung Yeh, "A method for modulo-2 minimization," *IEEE Trans. Comput.*, C-31, pp. 800-801, 1982.
[29] K. K. Saluja and E. H. Ong, "Minimization of Reed-Muller canonic expansion," *IEEE Trans. Comput.*, C-28, pp. 535-537, 1979.
[30] 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.
[31] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. Comput.*, vol. C-33, pp. 879-894, Oct. 1984.
[32] T. Sasao, "An algorithm to derive the complement of a binary function with multiple-valued inputs," *IEEE Trans. Comput.*, vol. C-34, pp. 131-140, Feb. 1985.
[33] T. Sasao and H. Fujiwara, "A design of AND-EXOR PLA's with universal tests," [in Japanese] IEICE Japan tech. paper, FTS86-25, Feb. 23, 1987.
[34] T. Sasao, "Multiple-valued logic and optimization of programmable logic arrays," *IEEE Computer*, vol. 21, no. 4, pp. 71-80, April 1988.
[35] T. Sasao and M. Higashida, "On a design algorithm for AND-EXOR PLA's with input decoders," [in Japanese] presented at the 20th Workshop on Fault Tolerant Computing, Jan. 1989; also IEICE (Japan) tech. paper VLD89-84, Dec. 15, 1989.
[36] T. Sasao, "On the optimal design of multiple-valued PLA's," *IEEE Trans. Comput.*, vol. 38, pp. 582-592, Apr. 1989.
[37] T. Sasao and P. Besslich, "On the complexity of MOD-2 Sum PLA's," *IEEE Trans. Comput.*, vol. 32, pp. 262-266, Feb. 1990.
[38] T. Sasao, "EXMIN: A simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued input two-valued output functions," in *Proc. Int. Symp. Multiple-Valued Logic*, pp. 128-135, May 1990.
[39] T. Sasao, "A transformation of multiple-valued input two-valued output functions and its application to simplification of exclusive-or sum-of-products expressions," in *Proc. Int. Symp. Multiple-Valued Logic*, pp. 270-279, May 1991.
[40] T. Sasao, "AND-EXOR expressions and their optimization," in *Logic Synthesis and Optimization*, T. Sasao, Ed. Boston, MA: Kluwer Academic Publishers, 1993, pp. 286-312.
[41] T. Sasao, "Logic synthesis with EXOR gates," in *Logic Synthesis and Optimization*, T. Sasao, Ed. Boston, MA: Kluwer Academic Publishers, 1993, pp. 259-285.
[42] J. M. Saul, "An improved algorithm for the minimization of mixed polarity Reed-Muller representation," in *Proc. ICCD-90*, Cambridge, MA., Oct. 1991, pp. 372-375.
[43] S. Yang, "Logic synthesis and optimization benchmark user guide, version 3.0," MCNC, Jan. 1991.

Tsutomu Sasao (S'72-M'77-SM'90) 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 1988, he has been a professor with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka, Japan. From 1977 to 1988, he was on the faculty at Osaka University, Osaka, Japan. Beginning in February 1982, he spent a year at the IBM T. J. Watson Research Center, where he developed an AND-OR PLA minimization system and a multi-level logic synthesis system. In 1990, he spent three months at the Naval Postgraduate School, Monterey, CA, where he improved the AND-EXOR minimization program. His research interests include logic design and switching theory, especially the complexity analysis of logic circuits, design methods for PLA's, and application of multiple-valued logic. He was the Asia Area Program Chairman in 1974, and the Program Chairman in 1992 for the International Symposium on Multiple-Valued Logic (ISMVL). Also, he was the Organizer and the Chairman of the International Symposium on Logic Synthesis and Microprocessor Architecture, Iizuka, Japan, in July 1992. He has published five books on switching theory and logical design, including *Logic Synthesis and Optimization*. Boston, MA: Kluwer Academic Publishers, 1993.

Dr. Sasao is a member of the Institute of Electronics, Information, and Communication Engineers (IEICE), and is an Associate Editor of IEICE Transactions on Information and Systems. He is also a member of Information Processing Society of Japan, and he was Chairman of the Japanese Research Group on Multiple-Valued Logic during 1989-1991. He received the NIWA Memorial Award in 1979, and a Distinctive Contribution Award from the IEEE Computer Society MVL-TC for the paper presented at ISMVL 1986.