PAPER Special Issue on Synthesis and Verification of Hardware Design

Minimization of AND-OR-EXOR Three-Level Networks with AND Gate Sharing

Debatosh DEBNATH[†], *Nonmember and* **Tsutomu SASAO**[†], *Member*

SUMMARY This paper presents an exact minimization algorithm for AND-OR-EXOR three-level networks, where a single two-input exclusive-OR (EXOR) gate is used. The network realizes an EXOR of two sum-of-products expressions (EX-SOP), where the two sum-of-products expressions (SOP) can share products. The objective is to minimize the total number of different products in the two SOPs. An algorithm for the exact minimization of EX-SOPs with up to five variables are shown. Up to five variables, EX-SOPs for all the representative functions of NP-equivalence classes were minimized. For five-variable functions, we confirmed that minimum EX-SOPs require up to 9 products. For n-variable functions, minimum EX-SOPs require at most $9 \cdot 2^{n-5}$ $(n \geq 6)$ products.

key words: *three-level networks, AND-EXOR, logic minimization, complexity of logic networks, NP-equivalence*

1. Introduction

Logic networks are usually designed by using AND and OR gates. However, two-level AND-EXOR networks are often more compact and easily testable than two-level AND-OR networks [6], [13], [14], [17], [19]. For example, to realize five-variable functions, on the average, minimum SOPs (sum-of-products expressions) require 7.46 products, while minimum ESOPs (exclusive-OR sum-of-products expressions) require 6.16 products [15]. To realize an arbitrary function of six variables, minimum SOPs require up to 32 products, while minimum ESOPs require up to 15 products [9]. Thus, these are advantages of using EXOR gates. In these designs, EXOR gates with large fan-in are used. However, in most technologies, EXOR gates with many inputs are expensive.

The network shown in Fig. 1 realizes an exclusive-OR of two sum-of-products expressions (EX-SOP), where only a single two-input EXOR gate is used. An EX-SOP for a function f can be written as $F = F_a \oplus F_b$, where F_a and F_b are SOPs. Recently, we developed a minimization algorithm for EX-SOPs [3]. The objective was to minimize the total number of products in F_a and F_b, where no product sharing was permitted between F_a and F_b. Figure 1 shows the realization of an EX-SOP for f with no product sharing.

We can often reduce the total number of products in EX-SOPs if we permit product sharing between two SOPs [16]. The network shown in Fig. 2 realizes an EX-SOP for f with product sharing. An EX-SOP for f with product sharing can be written as $F = (F_a \vee F_s) \oplus (F_b \vee F_s)$, where F_a, F_b, and F_s are SOPs, and F_s represents shared products. In this paper, our objective is to minimize the total number of products in F_a, F_b, and F_s.

AND-OR-EXOR three-level network is suitable for implementing arithmetic functions. For example, the Texas Instruments SN181 arithmetic circuit has EXOR gates in the outputs [20]. Programmable logic arrays (PLAs) with two-input EXOR gates in the outputs efficiently realize adders [21]. AND-OR-EXOR is one of the simplest three-level architecture, since it contains only a single two-input EXOR gate. However, its logic capability is quite high. Because of these, various PLAs with two-input EXOR gates in the outputs were developed. Especially, RICOH, Lattice and AMD (MMI) produced series of such PLAs.

Design methods for AND-OR-EXOR three-level networks were considered in the past [5], [18], but no practical algorithm was reported. A cut-and-try method was reported in [12] and several heuristic algorithms to simplify EX-SOPs were presented in [16], but they cannot guarantee the minimality of the solutions. Upper bounds on the number of products in AND-OR-EXOR expansion were reported in [3], [4].

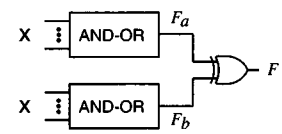


Fig. 1 AND-OR-EXOR three-level network with no AND gate sharing.

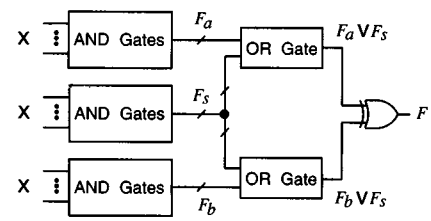


Fig. 2 AND-OR-EXOR three-level network with AND gate sharing.

Manuscript received February 3, 1997.
Manuscript revised May 8, 1997.

[†]The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820 Japan.

This paper is organized as follows: Sect. 2 introduces the notations. Section 3 shows the properties of EX-SOPs. Section 4 illustrates the idea for the minimization of EX-SOPs and a technique to reduce the search space. Section 5 presents an exact minimization algorithm for EX-SOPs with five variables. Section 6 outlines an idea to simplify EX-SOPs with six or more variables. Section 7 presents the experimental results.

2. Preliminaries

In this section, we introduce notations. We show the differences between EX-SOPs with and with no product sharing. In this paper, we distinguish functions and their expressions. We use lower case letters, such as f, g, h, to represent functions, and upper case letters, such as F, H, S, to represent expressions of function.

Definition 1: Let $\tau(F)$ be the number of products in an expression F. A logic expression F for f is said to be *minimum* when $\tau(F)$ is minimum.

Definition 2: Let $\tau(SOP : f)$ be the number of products in a minimum SOP for f. Let $\tau(SOP : F)$ be the number of products in an SOP F.

Definition 3: Let $\tau(EX\text{-}SOP_{NS} : f)$ be the number of products in a minimum EX-SOP for f with no product sharing (NS), and let $\tau(EX\text{-}SOP_{PS} : f)$ be the number of different products in a minimum EX-SOP for f with product sharing (PS).

Let a logic function f be represented as follows:

$$f = (f_a \vee g) \oplus (f_b \vee g). \tag{1}$$

Note that f_a, f_b and g correspond to F_a, F_b and F_s in Fig. 2, respectively. To compute $\tau(EX\text{-}SOP_{PS} : f)$, we choose f_a, f_b and g such that Eq. (1) is satisfied. Thus, we have $\tau(EX\text{-}SOP_{PS} : f) = \min\{\tau(SOP : g) + \tau(SOP : f_a) + \tau(SOP : f_b)\}$.

When $F_s = 0$, the network in Fig. 2 reduces to the network in Fig. 1. To compute $\tau(EX\text{-}SOP_{NS} : f)$, we have to consider $g = 0$, and choose f_a and f_b such that they satisfy Eq. (1). Thus, we have $\tau(EX\text{-}SOP_{NS} : f) = \min\{\tau(SOP : f_a) + \tau(SOP : f_b)\}$.

Definition 4: $\tau(EX\text{-}SOP_{PS} : F)$ denotes the number of different products in an EX-SOP F with product sharing.

3. Properties of EX-SOPs

In this section, we show properties of EX-SOPs, which are useful in minimizing EX-SOPs with product sharing.

Definition 5: On the Karnaugh map of a function, a cell that contain 1 is called a 1-cell and a cell that contain 0 is called a 0-cell.

Property 1: Consider the Karnaugh map for an EX-SOP. Any 1-cell must be covered by the loop(s) for exactly one SOP. If a 0-cell is covered by a loop, then it must be covered by at least one loop from both SOPs.

Definition 6: Let $g(\mathbf{x})$ and $h(\mathbf{x})$ be n-variable logic functions, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Let $B = \{0, 1\}$. If every $\boldsymbol{a} \in B^n$ satisfying $g(\boldsymbol{a}) = 1$ also satisfies $h(\boldsymbol{a}) = 1$, then $g(\mathbf{x}) \subseteqq h(\mathbf{x})$. For simplicity, we write $g \subseteqq h$. If every $\boldsymbol{a} \in B^n$ satisfying $g(\boldsymbol{a}) = 1$ also satisfies $h(\boldsymbol{a}) = 1$, and there exist $\boldsymbol{a} \in B^n$ such that $g(\boldsymbol{a}) = 0$ and $h(\boldsymbol{a}) = 1$, then $g(\mathbf{x}) \subset h(\mathbf{x})$. For simplicity, we write $g \subset h$.

Definition 7: If p is a product and $p \subseteqq f$, then p is an implicant of f.

Lemma 1: If p is a shared product of an EX-SOP for a function f, then p is an implicant of $\bar{f}$.

Proof: An EX-SOP for f with a shared product p can be written as $f = (f_a \vee p) \oplus (f_b \vee p)$. This implies $f = (f_a \oplus f_b) \cdot \bar{p}$. Thus, $\bar{f} = \overline{(f_a \oplus f_b)} \vee p$, i.e., $p \subseteqq \bar{f}$. Therefore, from the definition of the implicant, we have the lemma. □

Corollary 1: If g represents shared products of an EX-SOP for a function f, then $g \subseteqq \bar{f}$.

4. Minimization of EX-SOPs

In this section, we develop an exact minimization technique for EX-SOPs with product sharing.

4.1 Preparatory

The following two lemmas are used for the minimization of EX-SOPs in Sects. 4.2 and 4.3.

Definition 8: Let g represents the shared products of an EX-SOP of function f. The number of different products in a minimum EX-SOP for f with product sharing is denoted by $\tau(EX\text{-}SOP_{PS} : f : g)$.

To compute $\tau(EX\text{-}SOP_{PS} : f : g)$ using the expansion in Eq. (1), g is fixed and we choose f_a and f_b such that Eq. (1) is satisfied. Thus, we have $\tau(EX\text{-}SOP_{PS} : f : g) = \tau(SOP : g) + \min\{\tau(SOP : f_a) + \tau(SOP : f_b)\}$. Note that $\tau(EX\text{-}SOP_{PS} : f : g) \geqq \tau(EX\text{-}SOP_{PS} : f)$ and $\tau(EX\text{-}SOP_{PS} : f : 0) = \tau(EX\text{-}SOP_{NS} : f)$.

To obtain a minimum EX-SOP for f with g representing the shared products, we must minimize EX-SOP for f with no product sharing where g represents don't cares of f.

Lemma 2: $\tau(EX\text{-}SOP_{PS} : f : g) = \tau(SOP : g) + \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h)$.

Proof: Consider f and $g \subseteqq \bar{f}$. From the definition, we have

$$\begin{aligned}&\tau(EX\text{-}SOP_{PS} : f : g)\\ &\quad = \tau(SOP : g) + \min\{\tau(SOP : f_a) + \tau(SOP : f_b)\},\end{aligned} \tag{2}$$

where,

$$\begin{aligned}f &= (f_a \vee g) \oplus (f_b \vee g)\\ &= (f_a \oplus f_b) \cdot \bar{g}.\end{aligned} \tag{3}$$

In Eq. (3), when $g = 1$, $f = 0$. This implies that in the computation of f_a and f_b in Eq. (3), g acts as a don't care set for f. Thus, there exists a function h_s, such that $f \vee h_s = f_a \oplus f_b$ and $h_s \subseteqq g$. If f_a and f_b are represented by SOPs, then $f_a \oplus f_b$ corresponds to an EX-SOP for $f \vee h_s$ with no product sharing.

To obtain a minimum value of $\tau(SOP : f_a) + \tau(SOP : f_b)$, compute the minimum EX-SOP for each $f \vee h$ with no product sharing, where $h \subseteqq g$, and then take the minimum of all $\tau(EX\text{-}SOP_{NS} : f \vee h)$. Thus,

$$\min\{\tau(SOP : f_a) + \tau(SOP : f_b)\} = \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h). \quad (4)$$

Therefore, from Eqs. (2) and (4), we have the lemma. □

Example 1: Let us apply Lemma 2 to the logic function f shown in Fig. 3 (a), where g represents the shared products of an EX-SOP for f. To obtain $\tau(EX\text{-}SOP_{PS} : f : g)$, we have to compute $\tau(EX\text{-}SOP_{NS} : f \vee h)$ for all $h \subseteqq g$ and choose h that makes $\tau(EX\text{-}SOP_{NS} : f \vee h)$ minimum. The candidates for h are 0, $\bar{x}y\bar{z}\bar{w}$, $\bar{x}y\bar{z}w$, and $\bar{x}y\bar{z}$.

Figure 3 (b) shows the Karnaugh map of $f \vee h$, where we choose $h = \bar{x}y\bar{z}w$. Figure 3 (c) corresponds to an EX-SOP for $f \vee h$ with no product sharing. q_a and q_b denote the functions represented by the two loops. Therefore, $f \vee h = q_a \oplus q_b$ and $\tau(EX\text{-}SOP_{NS} : f \vee h) = 2$. For any other $h \subseteqq g$, $\tau(EX\text{-}SOP_{NS} : f \vee h) > 2$. Thus, $\min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h) = 2$.

Thus, from Lemma 2, $\tau(EX\text{-}SOP_{PS} : f : g) = 3$, since $\tau(SOP : g) = 1$. When the shared products represents g in Fig. 3 (d), we have the minimum EX-SOP: $f = (s_a \vee g) \oplus (s_b \vee g)$. □

Lemma 3: To obtain a minimum EX-SOP for a function f, it is sufficient to consider only the prime implicants of $\bar{f}$ as the candidates for the shared products.

Proof: Let q be a shared product of an EX-SOP for f that is not a prime implicant of $\bar{f}$. From Lemma 1, q is an implicant of $\bar{f}$. There exists a prime implicant p of $\bar{f}$, such that $q \subset p$. Note that a replacement of q by p does not change the function represented by the EX-SOP. Therefore, it is sufficient to consider only the prime implicants of $\bar{f}$ as the candidates for the shared products, instead of all the implicants of $\bar{f}$. □

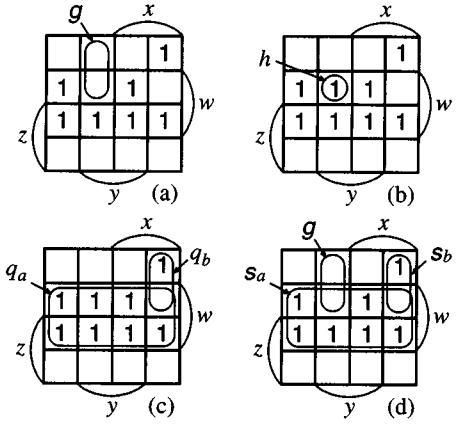


Fig. 3 Karnaugh maps for Example 1.

4.2 Candidates for the Shared Products

To obtain a minimum EX-SOP with product sharing, we use the candidate functions for the shared products. The computation technique of the candidate functions is shown in the following:

Definition 9: Let $PI(\bar{f})$ be the set of prime implicants of $\bar{f}$. Let G_i $(i \geqq 1)$ be the set of minimum SOPs with i products, where each product is an element of $PI(\bar{f})$. If there are two or more candidate SOPs for G_i each representing the same function, eliminate all but one. Let $G_0 = \{0\}$. G_i $(i \geqq 0)$ is called the i-th *shared set* for f.

Example 2: Figure 4 shows the Karnaugh map of a logic function f. The prime implicants of $\bar{f}$ are shown by p_1, p_2, p_3, and p_4, and a false minterm of f is represented by m in the figure. A false minterm means a 0-cell. From the definition of the shared set, the following are true:

1. $p_1 \vee p_2 \in G_2$, since $\tau(SOP : p_1 \vee p_2) = 2$, and $p_1, p_2 \in PI(\bar{f})$.
2. Although $p_2, p_3, p_4 \in PI(\bar{f})$, $p_2 \vee p_3 \vee p_4 \notin G_3$, since $p_2 \vee p_3 \vee p_4$ is not a minimum SOP.
3. Although $\tau(SOP : m \vee p_4) = 2$, $m \vee p_4 \notin G_2$, since $m \notin PI(\bar{f})$. $m \vee p_4$ is not an element of any shared set for f.
4. $G_1 = \{p_1, p_2, p_3, p_4\}$, $G_2 = \{p_1 \vee p_2, p_1 \vee p_3, p_1 \vee p_4, p_2 \vee p_3, p_2 \vee p_4, p_3 \vee p_4\}$ and $G_3 = \{p_1 \vee p_2 \vee p_4, p_1 \vee p_3 \vee p_4\}$.
5. Since $\tau(SOP : \bar{f}) = 3$, for $i \geqq 4$, $G_i = \phi$ (null). □

Example 3: Figure 5 shows the Karnaugh map of a logic function f. The prime implicants of $\bar{f}$ are shown by p_1, p_2, p_3, p_4, p_5, and p_6. From the definition of the shared set, either $p_1 \vee p_3 \vee p_5 \in G_3$ or $p_2 \vee p_4 \vee p_6 \in G_3$, but $\{p_1 \vee p_3 \vee p_5, p_2 \vee p_4 \vee p_6\} \notin G_3$, since $p_1 \vee p_3 \vee p_5$ and $p_2 \vee p_4 \vee p_6$ represent the same function. □

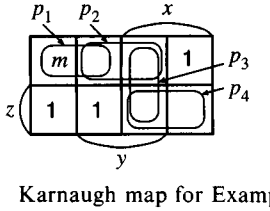


Fig. 4 Karnaugh map for Example 2.

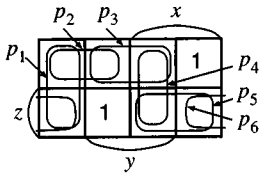


Fig. 5 Karnaugh map for Example 3.

Lemma 4: The i-th shared set (G_i) for f provides sufficient candidates for the i shared products of EX-SOPs for f.

Proof: From Lemma 3 and by the definition of the shared set, we have the lemma. □

4.3 Idea for Minimization

The key idea for the minimization of EX-SOPs with product sharing is shown in the following theorem:

Theorem 1: For a given function f, let $s = \tau(SOP : \bar{f})$ and G_i $(i = 0, 1, \ldots, s)$ be the i-th shared set for f. Then,

$$\tau(EX\text{-}SOP_{PS} : f) = \min_{i=0,1,\ldots,s} \left\{ i + \min_{g \in G_i} \left\{ \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h) \right\} \right\}. \quad (5)$$

Proof: Note that in Eq. (5), $g \in G_i$. From Lemma 4, g is a candidate function for the shared products. If g represents the shared products, then from Lemma 2, the number of different products in a minimum EX-SOP for f is $\tau(SOP : g) + \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h)$.

Let t_i be the number of different products in a minimum EX-SOP for f with exactly i shared products. From Lemma 4, G_i represents all the candidate functions for the i shared products. Therefore, to obtain t_i, we must consider all the elements of G_i as the candidates for the shared products, and choose the EX-SOP with the fewest products. Thus,

$$t_i = \min_{g \in G_i} \left\{ \tau(SOP : g) + \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h) \right\}. \quad (6)$$

In Eq. (6), for any $g \in G_i$, $\tau(SOP : g) = i$. Thus, we have

$$t_i = i + \min_{g \in G_i} \left\{ \min_{h \subseteqq g} \tau(EX\text{-}SOP_{NS} : f \vee h) \right\}. \quad (7)$$

To obtain the number of different products in a minimum EX-SOP for f with product sharing, we must choose the minimum of all the t_i's, where $i = 0, 1, \ldots, s$. Thus,

$$\tau(EX\text{-}SOP_{PS} : f) = \min_{i=0,1,\ldots,s} \{ t_i \}. \quad (8)$$

Hence, from Eqs. (7) and (8), we have the theorem. □

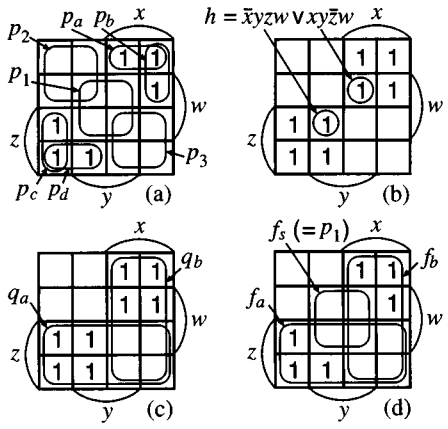


Fig. 6 Karnaugh maps for Examples 4 and 6.

Example 4: Let us apply Theorem 1 to the logic function f shown in Fig. 6 (a). The 0-th shared set is $G_0 = \{0\}$. This implies $g = 0$, where $g \in G_0$. Since $g = 0$, we have $\tau(EX\text{-}SOP_{PS} : f : g) = \tau(EX\text{-}SOP_{NS} : f)$. Thus, we consider an EX-SOP with no product sharing as the initial solution for the EX-SOP with product sharing. The prime implicants of f are shown in Fig. 6 (a) by p_a, p_b, p_c, and p_d. $\tau(SOP : f) = 4$ and $\tau(EX\text{-}SOP_{NS} : f) = 4$. Therefore, we consider $p_a \vee p_b \vee p_c \vee p_d$ as an initial solution for EX-SOP of f with product sharing. Figure 6 (a) also shows the prime implicants of $\bar{f}$: $p_1 = yw$, $p_2 = \bar{x}\bar{z}$, and $p_3 = xz$. From the definition of the shared set, we have $G_1 = \{p_1, p_2, p_3\}$, $G_2 = \{p_1 \vee p_2, p_1 \vee p_3, p_2 \vee p_3\}$, and $G_3 = \{p_1 \vee p_2 \vee p_3\}$.

We search for an improved result by considering the elements of G_1 as the candidates for the shared products. Let $g = p_1$, where $g \in G_1$. Here, p_1 represents the shared product. h has 16 candidate functions, where $h \subseteqq g$. Let us choose $h = \bar{x}yzw \vee xy\bar{z}w$. Figure 6 (b) shows the Karnaugh map of $f \vee h$. Figure 6 (c) corresponds to an EX-SOP for $f \vee h$ with no product sharing. The functions represented by the two loops are denoted by q_a and q_b. Thus, $f \vee h = q_a \oplus q_b$ and $\tau(EX\text{-}SOP_{NS} : f \vee h) = 2$.

For any other $h \subseteqq g$ and $g \in G_i$ $(i = 1, 2, 3)$, $\tau(EX\text{-}SOP_{NS} : f \vee h) > 2$. Thus, $\tau(EX\text{-}SOP_{PS} : f) = 3$, and $p_1 (= yw)$ represents the shared product. Figure 6 (d) corresponds to a minimum EX-SOP for f with product sharing: $f = (f_a \vee f_s) \oplus (f_b \vee f_s)$. □

4.4 Reduction of Search Space

Theorem 1 shows that for a given function f, we must compute $\tau(EX\text{-}SOP_{NS} : f \vee h)$ for all $h \subseteqq g$, $g \in G_i$ and $i = 0, 1, 2, \ldots, \tau(SOP : \bar{f})$. This search space is very large, even for functions with as few as five variables. The following observation reveals this:

Observation 1: Five-variable functions have up to 32 prime implicants. Some shared set for many five-variable functions have more than 100,000 elements.

Example 5: Consider the logic function $f(x, y, z, v, w) = \sum(7, 11, 13, 14, 17, 18, 20, 24, 31)$. This representation is defined in [7], [10], [11]. $\tau(SOP : \bar{f}) = 12$ and we have the following:

i	1	2	3	4	5	6
$\|G_i\|$	26	325	2440	11845	38014	79763
i	7	8	9	10	11	12
$\|G_i\|$	106094	86139	40364	9967	1184	82

where $|G_i|$ denotes the number of elements in the shared set G_i $(1 \leqq i \leqq 12)$. □

A technique to drastically reduce the search space for Theorem 1 is shown in the following:

Theorem 2: In Theorem 1, suppose we need to find an EX-SOP with fewer than t different products, then we must only consider those i's, such that $i \leqq t - 2$.

Proof: Suppose we already considered all the i's such that $i \leqq t - 2$. Now it is sufficient to prove that a further increase in i cannot produce an EX-SOP F with $\tau(EX\text{-}SOP_{PS} : F) < t$. We prove this by contradiction.

To obtain $\tau(EX\text{-}SOP_{PS} : F) < t$, we increase i by 1, i.e., i is now $t - 1$. In Theorem 1, i represents the number of shared products. An EX-SOP with product sharing must have at least one non-shared product. Thus, $\tau(EX\text{-}SOP_{PS} : F) \geqq t$. Similarly, we can show that a further increase in i cannot produce an EX-SOP with fewer products. Hence, we have the theorem. □

Example 6: Let us consider the logic function f shown in Fig. 6 (a). We have $\tau(SOP : \bar{f}) = 3$. This implies that without Theorem 2, we must consider those i's in Theorem 1, such that $i \leqq 3$. We have $\tau(EX\text{-}SOP_{NS} : f) = 4$. We need to find an EX-SOP for f with fewer than four different products. Thus, from Theorem 2, we have only to consider those i's in Theorem 1, such that $i \leqq 2$. Example 4 shows that when $i = 1$, we found an EX-SOP for f with three different products. Thus, from Theorem 2, we have only to consider those i's in Theorem 1, such that $i \leqq 1$. □

Example 7: Let us consider the logic function f shown in Example 5. Without Theorem 2, we have to consider those i's in Theorem 1, such that $i \leqq 12$. We have $\tau(EX\text{-}SOP_{NS} : f) = 7$. Thus, from Theorem 2, we have only to consider those i's in Theorem 1, such that $i \leqq 5$. The values of $|G_i|$ shows that the number of elements in the shared sets to consider in Theorem 1 is reduced by 86%. □

5. An Algorithm Using Table Look-Up

The minimization algorithm for EX-SOPs with product sharing is based on Theorem 1. To compute $\tau(EX\text{-}SOP_{PS} : f)$ using Theorem 1, we have to compute $\tau(EX\text{-}SOP_{NS} : f \vee h)$ many times. A straightforward computation of $\tau(EX\text{-}SOP_{NS} : f \vee h)$ is time consuming. Thus, instead of doing logic minimization, we use a table of all the values for $\tau(EX\text{-}SOP_{NS} : f \vee h)$.

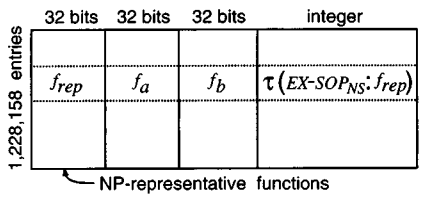


Fig. 7 EX-SOP solution and cost table with no product sharing for the NP-representative functions of five variables.

Up to four-variable functions, we can use a table of all the values for $\tau(EX\text{-}SOP_{NS} : f \vee h)$. In the case of five-variable, the total number of functions is $2^{32} \approx 4.3 \times 10^9$, and it is impractical to store all the values for $\tau(EX\text{-}SOP_{NS} : f \vee h)$. Thus, we use a cost table for the NP-representative functions.

The number of products in a minimum EX-SOP for a function is invariant under the permutation and/or negation of the input variables. In other words, if $f \overset{NP}{\sim} g$, then $\tau(EX\text{-}SOP_{NS} : f) = \tau(EX\text{-}SOP_{NS} : g)$, where $\overset{NP}{\sim}$ denotes the NP-equivalence relation [7], [8], [11]. The number of NP-equivalence classes of five-variable functions is 1,228,158. Figure 7 shows the table we use. We have computed this table in [3] and 16 megabytes memory space is necessary to keep it in the memory. The left most column of the table in Fig. 7 contains all the NP-representative functions f_{rep}. The two columns at the middle store the values of f_a and f_b, where $f_{rep} = f_a \oplus f_b$, such that $\tau(SOP : f_a) + \tau(SOP : f_b)$ is minimum. The right most column stores the values of $\tau(EX\text{-}SOP_{NS} : f_{rep})$. For a given function f_{given}, to obtain $\tau(EX\text{-}SOP_{NS} : f_{given})$ from Fig. 7, first we compute the NP-representative function f_{rep} of f_{given}. Since the number of products is invariant under the NP-equivalence class, we have $\tau(EX\text{-}SOP_{NS} : f_{given}) = \tau(EX\text{-}SOP_{NS} : f_{rep})$. Using f_{rep} for the table look-up in Fig. 7, we obtain $\tau(EX\text{-}SOP_{NS} : f_{given})$.

The minimization algorithm for EX-SOP with product sharing is implemented as procedure SHARE_EX-SOP, the pseudocode of which is shown in Fig. 8. A minimum EX-SOP for the given function with no product sharing is taken as an initial solution. Then, a minimum EX-SOP with the fewer shared products is searched first and a new solution is saved if it reduces the number of different products. Thus, we have the following:

Remark 1: The procedure SHARE_EX-SOP produces a minimum EX-SOP with the least number of shared products.

6. EX-SOPs with More Than Five Variables

When no product sharing is permitted, an EX-SOP for an n-variable function can be derived from a pair of EX-SOPs for $(n - 1)$-variable functions [3], [4]. In this section, we prove that this is also true when the sharing

```
/*  obtain f_a, f_b, f_s, and product, where f = (f_a ∨      */
/*  f_s) ⊕ (f_b ∨ f_s), such that product = min{τ(SOP :      */
/*  f_s) + τ(SOP : f_a) + τ(SOP : f_b)}                      */
1 procedure SHARE_EX-SOP(function f) {
2   var /* define variables */
3       product, bound, i, u : integer;
4       P, Q, S, G_i : set of logic functions;
5       f_a, f_b, f_s, f_rep, g, h, fh_rep : logic function;
6   make S and G_i empty;
7   P ← {all the prime implicants of f̄};
8   f_rep ← NP-representative function of f;
9   product ← τ(EX-SOP_NS : f_rep); /* from Fig. 7 */
10  obtain f_a and f_b by using f_rep;   /* from Fig. 7 */
11  f_s ← 0;
12  bound ← product −2;
13  for i ← 1 to bound do {
14      for each Q ⊆ P such that |Q| = i do {
15          g ← logical OR of all the elements in Q;
16          if g ∉ S and g ∉ G_i then
17              store g in G_i;
18      }
19      for each g ∈ G_i do {
20          for each h ⊆ g such that h ∉ S do {
21              store h in S;
22              fh_rep ← NP-representative function of f∨h;
23              u ← τ(EX-SOP_NS : fh_rep);
24              if u + i < product then {
25                  product ← u + i;
26                  obtain f_a and f_b by using fh_rep;
27                  f_s ← g;
28                  bound ← product −2;
29              }
30          }
31      }
32  }
33  print f_a, f_b, f_s, and product as the final solution;
34 }
```

Fig. 8 Pseudocode of the procedure SHARE_EX-SOP.

of products is permitted in EX-SOPs.

Lemma 5: If f and g are disjoint (i.e., $f \cdot g = 0$), then $f \cdot (h_{11} \oplus h_{12}) \vee g \cdot (h_{21} \oplus h_{22}) = (fh_{11} \vee gh_{21}) \oplus (fh_{12} \vee gh_{22})$.

Proof:

$$\begin{aligned} & f \cdot (h_{11} \oplus h_{12}) \vee g \cdot (h_{21} \oplus h_{22}) \\ &= f \cdot (h_{11} \oplus h_{12}) \oplus g \cdot (h_{21} \oplus h_{22}) \\ &= (fh_{11} \oplus gh_{21}) \oplus (fh_{12} \oplus gh_{22}) \\ &= (fh_{11} \vee gh_{21}) \oplus (fh_{12} \vee gh_{22}). \end{aligned}$$ □

Theorem 3: Let $\tau(EX\text{-}SOP_{PS} : n)$ be the maximum number of different products required to realize an n-variable function by a minimum EX-SOP with product sharing. Then

$$\tau(EX\text{-}SOP_{PS} : n) \leqq 2\tau(EX\text{-}SOP_{PS} : n-1).$$

Proof: An arbitrary n-variable function $f(x_1, x_2, \ldots, x_n)$ can be decomposed into two sub-functions by using the Shannon decomposition [7], [11], [17], $f = \bar{x}_1 f_0 \vee x_1 f_1$, where $f_0 = f|_{x_1=0}$ and $f_1 = f|_{x_1=1}$. By representing f_0 and f_1 by minimum EX-SOPs with product sharing, we have an expression F_1 for f:

$$\begin{aligned} F_1 = \bar{x}_1\{(H_{00} \vee S_0) \oplus (H_{01} \vee S_0)\} \\ \vee x_1\{(H_{10} \vee S_1) \oplus (H_{11} \vee S_1)\}, \end{aligned} \quad (9)$$

where, H_{ij}'s are minimum SOPs. S_0 and S_1 are minimum SOPs representing shared products for the minimum EX-SOPs of f_0 and f_1, respectively. Then,

$$\begin{aligned} \tau(F_1) = \tau(H_{00}) + \tau(H_{01}) + \tau(S_0) \\ + \tau(H_{10}) + \tau(H_{11}) + \tau(S_1). \end{aligned} \quad (10)$$

f_0 and f_1 are functions of $n-1$ variables, thus from Eq. (9) we have

$$\tau(F_1) \leqq 2\tau(EX\text{-}SOP_{PS} : n-1). \quad (11)$$

By applying Lemma 5 to Eq. (9), we have an EX-SOP F_2 for f:

$$\begin{aligned} F_2 &= \{(\bar{x}_1(H_{00} \vee S_0) \vee x_1(H_{10} \vee S_1)\} \\ &\quad \oplus \{(\bar{x}_1(H_{01} \vee S_0) \vee x_1(H_{11} \vee S_1)\}. \\ \Rightarrow F_3 &= \{(\bar{x}_1 H_{00} \vee x_1 H_{10}) \vee (\bar{x}_1 S_0 \vee x_1 S_1)\} \\ &\quad \oplus \{(\bar{x}_1 H_{01} \vee x_1 H_{11}) \vee (\bar{x}_1 S_0 \vee x_1 S_1)\}. \\ \Rightarrow F_4 &= (H_a \vee S) \oplus (H_b \vee S). \end{aligned} \quad (12)$$

In Eq. (12), H_a, H_b, and S represent minimum SOPs of $\bar{x}_1 H_{00} \vee x_1 H_{10}$, $\bar{x}_1 H_{01} \vee x_1 H_{11}$, and $\bar{x}_1 S_0 \vee x_1 S_1$, respectively. S represents shared products of the EX-SOP F_4 for f. Note that the number of products in Eq. (12) does not increase from Eq. (9). Thus, we have

$$\begin{aligned} \tau(EX\text{-}SOP_{PS} : n) \leqq \tau(H_{00}) + \tau(H_{10}) + \tau(H_{01}) \\ + \tau(H_{11}) + \tau(S_0) + \tau(S_1). \end{aligned} \quad (13)$$

Hence, from Eqs. (10), (11), and (13), we have the theorem. □

By using the minimization algorithm presented in Sect. 5, we can obtain a table of minimum EX-SOPs for the NP-representative functions of five variables. From the idea presented in Theorem 3, this table can be used to develop a heuristic simplification program of EX-SOPs with product sharing for six or more inputs [1].

7. Experimental Results

We minimized expressions for all the 1,228,158 NP-representative functions of five variables. Table 1 compares numbers of five-variable functions requiring t products for different classes of minimum expressions†. In this table, the data for SOPs and ESOPs are taken from [15]. EX-SOPs with no product sharing were minimized by the algorithm in [3]. EX-SOPs with product sharing were minimized by the procedure SHARE_EX-SOP presented in Sect. 5.

†In Table 1, av = $\left(\sum_t (t \times \text{number of functions requiring } t \text{ products})\right)$ / total number of functions. Total number of five-variable functions is 2^{32}.

Table 1 Numbers of five-variable functions requiring t products in minimum expressions.

t	SOP	ESOP	EX-SOP	
			NS	PS
0	1	1	1	1
1	243	243	243	243
2	20676	24948	25988	25988
3	818080	1351836	1511996	1532236
4	16049780	39365190	47838990	49658670
5	154729080	545193342	694830748	727980932
6	698983656	2398267764	2678055614	2697565894
7	1397400512	1299295404	870943300	816721636
8	1254064246	11460744	1760384	1481664
9	571481516	7824	32	32
10	160200992			
11	34140992			
12	6160176			
13	827120			
14	84800			
15	5312			
16	114			
av	7.4635	6.1616	6.0185	5.9971

NS : with no product sharing.
PS : with product sharing.
av : average.

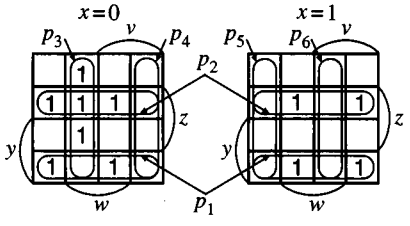


Fig. 9 Karnaugh map for a five-variable function.

For five-variable functions, on the average, minimum EX-SOPs with and with no product sharing require 5.9971 and 6.0185 products, respectively, while minimum SOPs require 7.4635 products. For five-variable functions, on the average, minimum EX-SOPs require fewer products than minimum ESOPs. On the average, minimum EX-SOPs with and with no product sharing require nearly the same number of products. For, about 92 million five-variable functions, we could reduce the number of products up to two by using product sharing. For example, minimum EX-SOPs for $f(x,y,z,v,w) = \sum(7,8,11,12,13,14,16,19,20,21,22,31)$ with and with no product sharing require five and seven products, respectively. The number of shared products in a minimum EX-SOP for five-variable functions is up to three. For example, a minimum EX-SOP for $f(x,y,z,v,w) = \sum(1,2,4,5,7,13,14,21,22,25,26)$ requires three shared products, which is shown in Fig. 9. This EX-SOP can be represented as $(p_1 \vee F_s) \oplus (p_2 \vee p_3 \vee F_s)$, where $F_s = p_4 \vee p_5 \vee p_6$ represents the shared products, and p's are shown in Fig. 9.

In [3], we showed that five-variable functions require up to 9 products in minimum EX-SOPs with no product sharing. Table 1 shows that this is also true for EX-SOPs with product sharing. Thus, from Theorem 3, minimum EX-SOPs for n variables with product sharing require at most $9 \cdot 2^{n-5}$ $(n \geq 6)$ products.

The execution time of the procedure SHARE_EX-SOP for function f depends on many factors. They include $\tau(EX\text{-}SOP_{NS} : f)$, the number of prime implicants of $\bar{f}$, the number of minterms in each of these prime implicants, and the distribution of these prime implicants on Karnaugh map. On the average, a five-variable function took 41 CPU seconds on a DEC ALPHASTATION 200. This average is obtained by minimizing 10,000 randomly generated functions with 16 true minterms. Although the minimization program is not so time consuming for each function, we spent nearly two and half months of computation time by using several workstations to minimize over 1.2 million NP-representative functions.

8. Conclusions and Comments

In this paper, we presented an exact minimization algorithm for AND-OR-EXOR three-level networks (EX-SOPs) for up to five-variable functions. We minimized EX-SOPs with product sharing for all the 1,228,158 NP-representative functions of five variables and completed the table of minimum EX-SOPs with product sharing. We confirmed that the minimum EX-SOPs for five-variable functions require up to 9 products and that for n-variable functions require at most $9 \cdot 2^{n-5}$ $(n \geq 6)$ products. This bound is tighter than the previously known one: $5 \cdot 2^{n-4}$ $(n \geq 4)$ [4]. This upper bound is smaller than 2^{n-1}, the tight upper bound for the minimum SOPs. Also, we found that for five-variable functions, on the average, minimum EX-SOPs with and with no product sharing require 5.9971 and 6.0185 products, respectively, while minimum SOPs require 7.4635 products. The table of minimum EX-SOPs is useful in a heuristic simplification program for EX-SOPs with six or more inputs [1]. An exact logic minimizer is useful to evaluate the minimality of heuristic logic minimizers. In order to develop a good heuristic logic minimizer, an exact logic minimizer is essential.

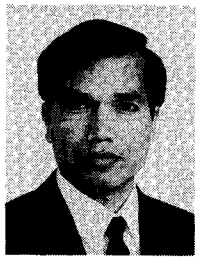
Acknowledgements

This work was supported in part by a Grant in Aid for the Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan. The authors would like to thank Prof. N. Koda for providing the table of NP-representative functions of five variables, Prof. Jon. T. Butler for carefully reviewing the manuscript, and anonymous referees for giving constructive suggestions. This paper is based on [2].

References

[1] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level expressions by table look-up," IEICE

Technical Report, VLD95-90, FTS95-52, Oct. 1995.
[2] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," Proc. The Sixth Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI'96), Fukuoka, Japan, pp.67–73, Nov. 1996.
[3] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level networks," Proc. Asia and South Pacific Design Automation Conference, pp.545–550, Jan. 1997.
[4] E.V. Dubrova, D.M. Miller, and J.C. Muzio, "Upper bounds on the number of products in AND-OR-XOR expansion of logic functions," Electronics Letters, vol.31, no.7, pp.541–542, March 1995.
[5] H. Fleisher, J. Giraldi, D.B. Martin, R.L. Phoenix, and M.A. Tavel, "Simulated annealing as a tool for logic optimization in a CAD environment," Proc. Int. Conf. Computer-Aided Design, pp.203–205, Nov. 1985.
[6] H. Fujiwara, "Logic Testing and Design for Testability," The MIT Press, Cambridge, MA, 1985.
[7] M.A. Harrison, "Introduction to Switching and Automata Theory," McGraw-Hill Book Company, New York, 1965.
[8] S.L. Hurst, D.M. Miller, and J.C. Muzio, "Spectral Techniques in Digital Logic," Academic Press Inc., 1985.
[9] N. Koda and T. Sasao, "An upper bound on the number of products in minimum ESOPs," Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, Makuhari, Japan, pp.94–101, Aug. 1995.
[10] E.J. McCluskey, "Logic Design Principles," Prentice-Hall International, London, 1986.
[11] S. Muroga, "Logic Design and Switching Theory," John Wiley & Sons, New York, 1979.
[12] D. Pellerin and M. Holley, "Practical Design Using Programmable Logic," Prentice Hall, 1991.
[13] T. Sasao and P. Besslich, "On the complexity of MOD-2 sum PLA's," IEEE Trans. Comput., vol.C-39, no.2, pp.262–266, Feb. 1990.
[14] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued input two-valued output functions," IEEE Trans. Comput. Aided Des. Integrated Circuits & Syst., vol.CAD-12, no.5, pp.621–632, May 1993.
[15] T. Sasao, "AND-EXOR expressions and their optimization," in Logic Synthesis and Optimization, ed. T. Sasao, Kluwer Academic Publishers, Boston, 1993.
[16] T. Sasao, "A design method for AND-OR-EXOR three-level networks," Proc. Int. Workshop on Logic Synthesis, Lake Tahoe, California, pp.8:11–8:20, May 1995.
[17] T. Sasao, "Representations of logic functions using EXOR operators," in Representations of Discrete Functions, eds. T. Sasao and M. Fujita, Kluwer Academic Publishers, Boston, 1996.
[18] K. Shu, H. Yasuura, and S. Yajima, "Optimization of PLDs with output parity gates," National Convention, Information Processing Society of Japan, March 1985.
[19] N. Song and M.A. Perkowski, "Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions," IEEE Trans. Comput. Aided Des. Integrated Circuits & Syst., vol.CAD-15, no.4, pp.385–395, April 1996.
[20] Texas Instruments Inc., "The TTL Data Book for Design Engineers," 1973.
[21] A. Weinberger, "High-speed programmable logic array adders," IBM J. Res. & Develop., vol.23, no.2, pp.163–178, March 1979.



Debatosh Debnath received the B.S. degree in Electrical and Electronic Engineering in 1991, and the M.S. degree in Computer Science and Engineering in 1993, both from the Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. He is currently a Ph.D. candidate in Computer Science and Electronics at the Kyushu Institute of Technology, Iizuka, Japan. He has been a recipient of the Japanese Government Scholarship since October 1993. His research interests include logic synthesis and design for testability. He is a Student Member of IEEE.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronic Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He was with Osaka University Japan, IBM T.J. Watson Research Center and Naval Postgraduate School in Monterey, California. Now, he is a Professor of Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published seven books including, "Logic Synthesis and Optimization," and "Representation of Discrete Functions," Kluwer Academic Publishers 1993, and 1996, respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic many times. He organized the International Symposium on Logic Synthesis and Microprocessor Architecture, Iizuka, Japan in 1992, and the IFIP Workshop of the Reed-Muller Expansion in Circuit Design in Makuhari, Japan in 1995. He received the NIWA Memorial Award in 1979, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC in 1987 and 1996. He is a Fellow of IEEE.