Takashi Hirayama and Yasuaki Nishitani,
"A Simplification Algorithm of AND-EXOR Expressions Guaranteeing Minimality for Some Class of Logic Functions,"
Systems and Computers in Japan (Scripta Technica, Inc.), Vol. 27, No. 3, pp. 18–25, March 1996.
Translated from *Trans. IEICE*, Vol. J78-D-I, No. 4, pp. 409–415, April 1995.

A Simplification Algorithm of AND-EXOR Expressions Guaranteeing Minimality for Some Class of Logic Functions

Takashi Hirayama, *Nonmember*, **and Yasuaki Nishitani,** *Member*

Faculty of Engineering, Gunma University, Kiryu, Japan 376

SUMMARY

Among all AND-EXOR expressions that represent a logic function f, those with a minimum number of products are called minimum AND-EXOR expressions of f. The number of products in the minimum AND-EXOR expressions of f is denoted by $\tau(f)$. The minimization algorithms of AND-EXOR expressions take a time in $O(2^{2^n})$, where n is the number of variables. So it is necessary to develop a simplification algorithm that does not always find the exact minimum AND-EXOR expressions but runs fast. This paper presents a simplification algorithm of AND-EXOR expressions, which partially guarantees minimality and has the following properties: (1) it computes a minimum AND-EXOR expression for a logic function f with $\tau(f) < 3(k+1)$, where k is a nonnegative integer, (2) if an AND-EXOR expression obtained by this algorithm takes less than or equal to $3(k+1)$ of products, then it is minimum, and (3) its time complexity is $O(\sqrt{3}^{kn^2+(k+C)n})$. From the experimental results for $k = 0, 1, 2$, we can conclude that its computational time is practical for $n = 20, 7, 6$, respectively. Modifying this algorithm, we present an exact minimization algorithm for 5-variable functions, which is faster than previously known ones.

key words: logic design, AND-EXOR expressions, logic minimization algorithms, time complexity

1 Introduction

A design of logic circuits usually uses algorithms to simplify circuits in which AND, OR, and NOT gates are used as basic logic components. However, for arithmetic, telecommunication, and error correcting circuits, their realizations in which EXOR gates are used in addition to conventional AND and OR gates often require fewer gates [1, 2, 7]. In automated design of circuits with EXOR gates, it is necessary to establish the design methods for circuits containing EXOR gates as well as AND and OR gates.

AND-EXOR expressions which are formed by combining product terms with the EXOR operator are called ESOPs (Exclusive-OR Sum-Of-Products expressions). There are many types of AND-EXOR expressions, such as Reed-Muller expansions, general Reed-Muller expansions, et al. ESOPs are most generalized expressions and require the smallest numbers of product terms. Among all ESOPs that represent a function f, those with a minimum number of product terms are called minimum ESOPs of f and the number of product terms of the minimum ESOP of f is denoted by $\tau(f)$. Minimization means to obtain the minimum ESOP of f and simplification means reducing the number of product terms in ESOPs without guaranteeing minimality.

There are no efficient algorithms for minimizing ESOPs. The minimization algorithms previously known use exhaustive searching methods [1, 5, 6, 10]. Since their computing time is $O(2^{2^n})$ or $O(2^{3^n})$, where

n is the number of variables, it is difficult to compute minimum ESOPs for $n \geq 6$ in realistic time. EXMIN [8] and EXMIN2 [9] are famous simplification algorithms which rewrite ESOP by some rewriting rules and reduce the number of product terms step-by-step. Since the rewriting rules are heuristic, however, the minimality of the obtained ESOPs is not guaranteed and there is no theoretical analysis of their computing time.

We present a simplification algorithm based on the minimization algorithm given by Nishitani and Shimizu [5]. This simplification algorithm has the following properties: (1) it always finds a minimum ESOP for any function f with $\tau(f) < 3(k+1)$, where k is a nonnegative integer; (2) if the obtained ESOP has at most $3(k+1)$ product terms, then it is minimum; and (3) the time complexity is $O(\sqrt{3}^{kn^2+(k+C)n})$. No simplification algorithms such that the class of functions which can be minimized is explicitly specified and the time complexity is given have been known previously. When the value of k is large in our simplification algorithm, its computing time is large, while the class of functions which can be minimized is large. It is trade-off between the subclass of functions for which it guarantees minimality and its computing time.

We also present a fast minimization algorithm for 5-variable functions, based on foregoing simplification algorithm. This algorithm runs much faster than previously developed ones. It is useful for simplifying functions with more than 5 variables to obtain minimum ESOPs of 5-variable functions rapidly [4].

We implemented the simplification algorithm presented herein and made experiments on the functions the simplified ESOPs whose minimality is not guaranteed. The experimental results give the average difference between the number of product terms in simplified ESOPs and that in minimum ESOPs and the number of variables of functions which can be simplified in practical time. For the functions with $\tau(f) \geq 3(k+1)$, the the number of product terms of the ESOP simplified by our algorithm becomes larger than that of the minimum ESOP as $\tau(f)$ increases. For $k = 0, 1, 2$, the computational time is practical for $n = 20, 7, 6$, respectively.

2 Definitions and Minimization Theorem

In this section, we give the definitions and the minimization theorem [5] which characterizes the number of product terms in minimum ESOPs.

Definition 1: x and $\bar{x}$ are *literals* of a variable x. A logical product which contains at most one literal for each variable is called a *product term*. Product terms combined with *exclusive-or* operators ($\oplus$) form an *exclusive-or sum-of-products expression* (*ESOP*).

Definition 2: The number of product terms in an ESOP F is denoted by $\tau(F)$. Among all ESOPs of a function f, those with a minimum number of product terms are called *minimum ESOP*s of f. The number of product terms in a minimum ESOP of f is denoted by $\tau(f)$.

Definition 3: For an n-variable function f and a variable x, the subfunctions of f with $x = 0$ and $x = 1$ are denoted by $f_{x=0}$ and $f_{x=1}$, respectively. Furthermore $f_{x=2}$ and $f_{x=3}$ are defined to be $f_{x=0} \oplus f_{x=1}$ and the logical zero function 0, respectively.

A function f can be expanded as the following four forms using the subfunctions $f_{x=0}$, $f_{x=1}$, $f_{x=2}$, and $f_{x=3}$ $(= 0)$: $f = \bar{x}f_{x=0} \oplus xf_{x=1} \oplus f_{x=3} = \bar{x}f_{x=3} \oplus xf_{x=2} \oplus f_{x=0} = \bar{x}f_{x=2} \oplus xf_{x=3} \oplus f_{x=1} = \bar{x}f_{x=1} \oplus xf_{x=0} \oplus f_{x=2}$. Other expansion forms can be constructed by combining an arbitrary $(n-1)$-variable function g with each subfunction of the forgoing expansions. For example, $\bar{x}(f_{x=0} \oplus g) \oplus x(f_{x=1} \oplus g) \oplus (f_{x=3} \oplus g)$ is an expansion of f.

The following is known as the minimization theorem which characterizes $\tau(f)$ in terms of the subfunctions [5].

Theorem 1 (Minimization Theorem): Let f be an n-variable function and $\mathcal{F}^{n-1}$ be the class of all $(n-1)$-variable functions. Then the following equality holds.

$$\tau(f) = \min_{g \in \mathcal{F}^{n-1}} \{\tau(f_{x=0} \oplus g) + \tau(f_{x=1} \oplus g) + \tau(g)\}$$

From this minimization theorem and the fact that a function f is expanded as $\bar{x}(f_{x=0} \oplus g) \oplus x(f_{x=1} \oplus g) \oplus g$, we can obtain the minimum ESOP of f by computing the minimum ESOPs of $f_{x=0} \oplus g$, $f_{x=1} \oplus g$, and g for all $(n-1)$-variable functions g. That is, for g such

that $\tau(f_{x=0} \oplus g) + \tau(f_{x=1} \oplus g) + \tau(g)$ is minimum, letting F_0, F_1, and F_3 be minimum ESOPs of $f_{x=0} \oplus g$, $f_{x=1} \oplus g$, and g, respectively, the minimum ESOP of f is expressed as $\bar{x}F_0 \oplus xF_1 \oplus F_3$.

In another aspect, the above minimization theorem allows us to obtain the minimum ESOP of f by computing the minimum ESOPs of g_0, g_1, and g_2 for all expansions such that $f = \bar{x}g_0 \oplus xg_1 \oplus g_2$. Hence, by computing minimum ESOPs for not only $f_{x=0} \oplus g$, $f_{x=1} \oplus g$, and $f_{x=3} \oplus g = g$ but also $f_{x=2} \oplus g$, we can check four expansions $\bar{x}(f_{x=0} \oplus g) \oplus x(f_{x=1} \oplus g) \oplus (f_{x=3} \oplus g)$, $\bar{x}(f_{x=3} \oplus g) \oplus x(f_{x=2} \oplus g) \oplus (f_{x=0} \oplus g)$, $\bar{x}(f_{x=2} \oplus g) \oplus x(f_{x=3} \oplus g) \oplus (f_{x=1} \oplus g)$, and $\bar{x}(f_{x=1} \oplus g) \oplus x(f_{x=0} \oplus g) \oplus (f_{x=2} \oplus g)$. Then, by finding a function g and an index i $(0 \le i \le 3)$ such that $\sum_{0 \le j \le 3, j \ne i} \tau(f_{x=j} \oplus g)$ is minimum, we can obtain the minimum ESOP of f. From this discussion, the minimization theorem can be modified as follows.

Corollary 1: Let f be an n-variable function and $\mathcal{F}^{n-1}$ be the class of all $(n-1)$-variable functions. Then the following equality holds.

$$\tau(f) = \min_{g \in \mathcal{F}^{n-1}} \{ \min_{0 \le i \le 3} \{ \sum_{0 \le j \le 3, j \ne i} \tau(f_{x=j} \oplus g) \} \}$$

A minimization algorithm based on Corollary 1 is less efficient than one based on Theorem 1 because it minimizes four $(n-1)$-variable functions for each g. However, a simplification algorithm given in the next section is based on an equality similar to Corollary 1.

3 Simplification Algorithm

In this section, we present a simplification algorithm based on the minimization algorithm and give the subclass of functions for which the algorithm guarantees minimality. We also discuss its time complexity.

3.1 General simplification algorithm

Firstly we describe the concept of our simplification algorithms. Algorithms based on Theorem 1 or Corollary 1 must compute minimum ESOPs for all $(n-1)$-variable functions. Their computing time depends on the number of testing functions g's ($|\mathcal{F}^{n-1}| = 2^{2^{n-1}}$). Therefore, in order to develop more efficient algorithms, the number of testing functions must be reduced. We consider an algorithm whose testing function set is smaller, some subset $\mathcal{G}[f]$ of $\mathcal{F}^{n-1}$, instead of $\mathcal{F}^{n-1}$. In general this algorithm is a simplification algorithm, i.e., it does not always guarantee minimality since $\mathcal{G}[f] \subseteq \mathcal{F}^{n-1}$. The set $\mathcal{G}[f]$ is called a *testing set*. The testing set $\mathcal{G}[f]$ takes a parameter f because it may vary by f.

Based on the concept described here, the function $\sigma(f)$ is defined as follows, where n is the number of variables of f, and $\mathcal{G}[f]$ is a testing set of $(n-1)$-variable functions corresponding to f. Minimum ESOPs of all functions with at most m variables (the number of product terms, $\tau(f)$) are assumed to be known.

$$\sigma(f) = \begin{cases} \tau(f) & (n \le m) \\ \min\limits_{g \in \mathcal{G}[f]} \{ \min\limits_{0 \le i \le 3} \{ \sum\limits_{0 \le j \le 3, j \ne i} \sigma(f_{x=j} \oplus g) \} \} & (n > m) \end{cases}$$

From the definition of $\sigma(f)$, we can construct an algorithm in a similar way as with the minimization theorem. It is a simplification algorithm since the testing set $\mathcal{G}[f]$ is not $\mathcal{F}^{n-1}$. Note that the value of $\sigma(f)$ is the number of product terms in the simplified ESOP.

In the following, we show the simplification algorithm based on the equation of $\sigma(f)$, where f is an n-variable function. We have various algorithms by changing $\mathcal{G}[f]$ in this algorithm. To specify the testing set $\mathcal{G}$, our simplification algorithm is denoted by $A[\mathcal{G}]$.

Algorithm 1 ($A[\mathcal{G}]$):

1. For $n \le m$, return the minimum ESOP of f.
2. Find a function $g \in \mathcal{G}[f]$ and an index i $(0 \le i \le 3)$ such that $\sum_{0 \le j \le 3, j \ne i} \sigma(f_{x=j} \oplus g)$ is minimum, where $\sigma(f_{x=0} \oplus g)$, $\sigma(f_{x=2} \oplus g)$, $\sigma(f_{x=2} \oplus g)$, and $\sigma(f_{x=3} \oplus g)$ are obtained by applying this algorithm recursively.
3. For the function g obtained at the above step, let F_0, F_1, F_2, and F_3 be the ESOPs which are simplified for $f_{x=0} \oplus g$, $f_{x=2} \oplus g$, $f_{x=2} \oplus g$, and $f_{x=3} \oplus g$ by this algorithm, respectively. Return the following ESOP F, corresponding to the index i obtained at Step 2.

$$F = \begin{cases} \bar{x}F_2 \oplus xF_3 \oplus F_1 & (i = 0) \\ \bar{x}F_3 \oplus xF_2 \oplus F_0 & (i = 1) \\ \bar{x}F_0 \oplus xF_1 \oplus F_3 & (i = 2) \\ \bar{x}F_1 \oplus xF_0 \oplus F_2 & (i = 3) \end{cases}$$

3.2 Simplification algorithm such that testing set is the class of functions with $\tau(f) \leq k$

Specifying a concrete testing set of the general simplification algorithm of the previous section, we give a simplification algorithm which guarantees minimality for some subclass.

By $\mathcal{T}_k^n$ we denote the class of n-variable functions such that the number of product terms of their minimum ESOP is less than or equal to k. Consider the algorithm $A[\mathcal{T}_k]$ which is given by using $\mathcal{T}_k^{n-1}$ as the testing set $\mathcal{G}[f]$ in Algorithm 1. The number of product terms of an ESOP simplified by $A[\mathcal{T}_k]$, denoted by $\sigma_k(f)$, is defined as follows, where $S_k(f,g,i) = \sum_{0\leq j\leq 3, j\neq i} \sigma_k(f_{x=j} \oplus g)$.

$$\sigma_k(f) = \begin{cases} \tau(f) & (n \leq m) \\ \min\limits_{g\in\mathcal{T}_k^{n-1}} \{ \min\limits_{0\leq i\leq 3} \{S_k(f,g,i)\}\} & (n > m) \end{cases}$$

Then the following theorem and corollary hold.

Theorem 2: Let k be a nonnegative integer. For an arbitrary function f with $\tau(f) < 3(k+1)$, $\sigma_k(f) = \tau(f)$.

Proof: It is proved by induction on n. In the base case of $n \leq m$, it is trivial that the assertion holds from the definition of σ_k. Assume that the assertion holds for $n-1$. Let f be an n-variable function. From Theorem 1, there exists an $(n-1)$-variable function g such that

$$\tau(f) = \tau(f_{x=0} \oplus g) + \tau(f_{x=1} \oplus g) + \tau(g).$$

From the condition $\tau(f) < 3(k+1)$, all of $\tau(f_{x=0} \oplus g)$, $\tau(f_{x=1} \oplus g)$, and $\tau(g)$ are less than $3(k+1)$. Hence, from the induction hypothesis, $\sigma_k(f_{x=0} \oplus g) = \tau(f_{x=0} \oplus g)$, $\sigma_k(f_{x=1} \oplus g) = \tau(f_{x=1} \oplus g)$, and $\sigma_k(g) = \tau(g)$ hold, and then we have the following equality.

$$S_k(f,g,2) = \sigma_k(f_{x=0} \oplus g) + \sigma_k(f_{x=0} \oplus g) + \sigma_k(g) = \tau(f) \quad (1)$$

Furthermore from the condition $\tau(f) < 3(k+1)$, at least one of $\tau(f_{x=0} \oplus g)$, $\tau(f_{x=1} \oplus g)$, and $\tau(g)$ is at most k, i.e., $f_{x=0} \oplus g \in \mathcal{T}_k^{n-1}$ or $f_{x=1} \oplus g \in \mathcal{T}_k^{n-1}$ or $g \in \mathcal{T}_k^{n-1}$.

In the case of $g \in \mathcal{T}_k^{n-1}$, since $\sigma_k(f) \leq S_k(f,g,i)$ for any i $(0 \leq i \leq 3)$, we have $\sigma_k(f) \leq \tau(f)$ from the Eq. (1). In the case of $f_{x=0} \oplus g \in \mathcal{T}_k^{n-1}$, since $\sigma_k(f) \leq S_k(f, (f_{x=0} \oplus g), i)$ for any i $(0 \leq i \leq 3)$ and the equations $f_{x=0} \oplus g = f_{x=3} \oplus (f_{x=0} \oplus g)$, $f_{x=1} \oplus g = f_{x=2} \oplus (f_{x=0} \oplus g)$, and $g = f_{x=0} \oplus (f_{x=0} \oplus g)$ hold, we have $S_k(f,g,2) = S_k(f, (f_{x=0} \oplus g), 1)$. Hence $\sigma_k(f) \leq \tau(f)$. In the case of $f_{x=0} \oplus g \in \mathcal{T}_k^{n-1}$, we have $\sigma_k(f) \leq S_k(f, (f_{x=1} \oplus g), 0) \leq \tau(f)$ in a similar way. From the proceeding discussion, we can conclude that $\sigma_k(f) \leq \tau(f)$ in all cases.

On the other hand, it is trivial that $\tau(f) \leq \sigma_k(f)$. Hence the assertion $\sigma_k(f) = \tau(f)$ is proved. □

Corollary 2: Let k be a nonnegative integer. Then, $\sigma_k(f) = \tau(f)$ if $\sigma_k(f) \leq 3(k+1)$.

Proof: Since $\tau(f) \leq \sigma_k(f)$, we have $\tau(f) \leq 3(k+1)$. When $\tau(f) < 3(k+1)$, $\sigma_k(f) = \tau(f)$ holds from Lemma 2. In the case of $\tau(f) = 3(k+1)$, since $3(k+1) = \tau(f) \leq \sigma_k(f) \leq 3(k+1)$, we have $\sigma_k(f) = \tau(f)$. □

Theorem 2 guarantees that the simplification algorithm $A[\mathcal{T}_k]$ always finds the minimum ESOP for an arbitrary function f with $\tau(f) < 3(k+1)$, and Corollary 2 means that if the number of product terms in an ESOP simplified by $A[\mathcal{T}_k]$ is at most $3(k+1)$, then it is a minimum ESOP.

To implement the algorithm $A[\mathcal{T}_k]$, the testing set $\mathcal{T}_k^{n-1}$ must be generated. While it is easy to generate $\mathcal{T}_0^{n-1}$ and $\mathcal{T}_1^{n-1}$, it is hard to generate exact $\mathcal{T}_k^{n-1}$ for $k \geq 2$. However, we can generate redundant $\mathcal{T}_k^{n-1}$ by combining at most k product terms with EXOR operators, where some functions may be generated multiple times.

3.3 Computing time of $A[\mathcal{T}_k]$ and its improvement

In this section we give the time complexity of the algorithm $A[\mathcal{T}_k]$, and then improve it in terms of average computing time.

Theorem 3: Let n be the number of variables. The time complexity of the algorithm $A[\mathcal{T}_k]$ is $O(\sqrt{3}^{kn^2+(k+C)n})$, where C is a constant.

Proof: In Step 2 in $A[\mathcal{T}_k]$, there are four recursive calls for each $(n-1)$-variable function g in $\mathcal{T}_k^{n-1}$. Denoting by $T_k(n)$ the computing time for an n-variable function, we have the recurrence relation $T_k(n) \leq 4C' \cdot |\mathcal{T}_k^{n-1}| \cdot T_k(n-1)$, where C' is a constant.

The number of all product terms with at most $n-1$ variables is equal to 3^{n-1}. Hence we have $|\mathcal{T}_k^{n-1}| \leq (3^{n-1}+1)^k$. Since $(3^{n-1}+1)^k \leq 3^{kn}$, we have the recurrence relation $T_k(n) \leq 3^{kn+\log_3 4C'} \cdot T_k(n-1)$ from the above relation. Let $C = 2\log_3 4C'$ and $T_k(n) = c\sqrt{3}^{kn^2+(k+C)n}$, where c is a constant, then $T_k(n)$ satisfies the above relation. □

Although the algorithm $A[\mathcal{T}_k]$ computes $S_k(f,g,i)$ $(0 \leq i \leq 3)$ for all g in $\mathcal{T}_k^{n-1}$, the minimum ESOP (g and i such that $S_k(f,g,i)$ is minimum) may be obtained before testing all functions in $\mathcal{T}_k^{n-1}$. If we have a method for verifying whether an intermediate ESOP is minimum, then the average computing time can be reduced by terminating when the minimum ESOP is obtained.

The method for guaranteeing minimality of the intermediate ESOPs is based on Corollary 2. For a non-negative integer k' with $k' \leq k$, if the ESOP simplified by the algorithm $A[\mathcal{T}_{k'}]$ has at most $3(k'+1)$ product terms, then Corollary 2 guarantees that it is minimum. It is unnecessary to apply the algorithm $A[\mathcal{T}_k]$. Specifically, Step 2 of $A[\mathcal{T}_k]$ is divided in $k+1$ substeps by partitioning the testing set $\mathcal{T}_k^{n-1}$ into $\mathcal{T}_0^{n-1}$, $(\mathcal{T}_1^{n-1} - \mathcal{T}_0^{n-1})$, $\cdots$, $(\mathcal{T}_k^{n-1} - \mathcal{T}_{k-1}^{n-1})$, and at the ith substep we check the inequality $\tau(F) \leq 3(i+1)$, where F is an intermediate ESOP. If the inequality holds, we can conclude that F is minimum by Corollary 2 and return it. We present the modified algorithm $B[k]$, based on the above argument, in the following. The algorithm $B[k]$ has the same properties as $A[\mathcal{T}_k]$, i.e., $B[k]$ always obtains the minimum ESOP for any function f with $\tau(f) < 3(k+1)$ and if the number of product terms in the obtained ESOP is at most $3(k+1)$, then it is minimum.

Algorithm 2 ($B[k]$):

1. Let F_{min} be an ESOP which represents f (for example, the minterm expansion of f).
2. For each k' $(k' = 0, 1, \cdots, k)$, perform the following.
   - 2.1 Let F be the ESOP which is obtained by the algorithm $A[\mathcal{T}_{k'} - \mathcal{T}_{k'-1}]$, where $\mathcal{T}_0^{n-1} - \mathcal{T}_{-1}^{n-1} = \mathcal{T}_0^{n-1}$ and recursive simplifications in $A[\mathcal{T}_{k'} - \mathcal{T}_{k'-1}]$ are done by applying $B[k]$. If $\tau(F) < \tau(F_{min})$, assign the ESOP F to F_{min}.
   - 2.2 If $\tau(F_{min}) \leq 3(k'+1)$, then return F_{min} and stop.
3. Return F_{min} and stop.

4 Fast Minimization Algorithm for 5-variable Functions

Slightly modifying algorithm $B[2]$, we give a fast algorithm which finds exact minimum ESOPs for all 5-variable functions. It is known that an arbitrary 5-variable function can be represented by an ESOP with at most nine product terms [2]. Theorem 2 and Corollary 2 guarantee that $B[2]$ exactly minimizes all functions whose minimum number of product terms is at most eight, and that the obtained ESOPs with nine product terms are minimum. There are 7824 functions whose minimum number of product terms is nine [3, 4]. We applied algorithm $B[2]$ to these 7824 functions. For 7776 functions of them, $B[2]$ found ESOPs with 9 product terms, i.e., they were minimized, and there are only 48 functions which cannot be minimized. For these 48 functions, we can make a table of their minimum ESOPs, which are computed by a minimization algorithm based on Theorem 1. This table is called the *auxiliary-table*. Using this table and the algorithm $B[2]$, we can minimize all 5-variable functions: (1) simplify a 5-variable function using $B[2]$; (2) if the number of product terms of the simplified ESOP is less than or equal to nine, return it; (3) otherwise, get the minimum ESOP from the auxiliary-table and return it.

This modified algorithm was implemented in C language and applied to 5-variable functions on Sparc Station 10. In this implementation the terminating condition of the recursive call is $n \leq 4$ $(m = 4)$, because the minimum ESOPs for all 4-variable functions, the number of which is 65536, can be stored in a computer memory. The average computing time is 0.01 seconds, and in the worst case, i.e., when the algorithm tests all functions of $\mathcal{T}_2^4$ $(|\mathcal{T}_2^4| = 2370)$, the computing time is 0.027 seconds. Previously, the method of Koda and Sasao [4] was known as the fastest minimizer for 5-variable functions; its computing time is 0.09 seconds in the average case and seven seconds in the worst case (HP9000/720). Compared to their method, our minimization algorithm can be said to run faster in both the average case and the worst case.

Table 1: Ratio (%) of the number of minimized 5-variable functions

$\tau(f)$	No. of representative functions	$A[\mathcal{T}_0]$	$A[\mathcal{T}_1]$	$A[\mathcal{T}_2]$
0	1	100	100	100
1	1	100	100	100
2	4	100	100	100
3	19	94.7	100	100
4	137	72.3	100	100
5	971	43.5	100	100
6	3572	36.7	93.6	100
7	2143	44.0	95.3	100
8	86	32.6	86.0	100
9	2(7824)	0	99.4	99.4
total	6936	40.7	95.1	99.9

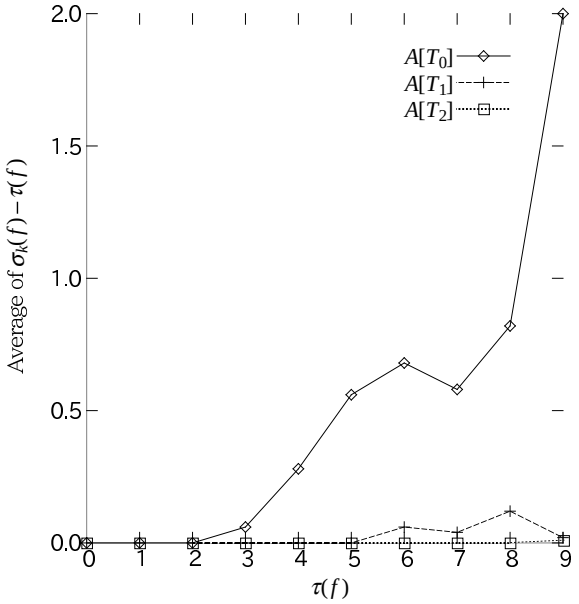

Figure 1: Average difference of the number of products between simplified and minimized ESOPs

5 Experimental Results

The algorithms $A[\mathcal{T}_k]$ and $B[k]$ described in Section 3 were implemented and applied to some functions in order to obtain their computing time and the difference between minimum ESOPs and the results of our algorithms. In our implementation, the terminating condition of the recursive call is $n \leq 4$ ($m = 4$). The programs are coded in CLISP and run on Sparc Station 10.

As described in Section 3, algorithms $A[\mathcal{T}_k]$ and $B[k]$ find the minimum ESOPs for all functions f with $\tau(f) < 3(k+1)$. For functions with $\tau(f) \geq 3(k+1)$, it is interesting to compare the numbers of product terms in ESOPs simplified by $A[\mathcal{T}_k]$ with those in minimum ESOPs. Although the minimum ESOPs of 6-variable functions cannot be computed in practical time, those of 5-variable functions can be computed by a minimization algorithm based on Theorem 1. Thus, for 5-variable functions, we compare their ESOP simplified by $A[\mathcal{T}_k]$ ($k = 0, 1, 2$) with their minimum ESOPs. We applied our algorithms to all representative functions of LP-equivalence classes [1, 2], the number of which is 6936, and all functions with $\tau(f) = 9$, the number of which is 7824. Table 1 shows the ratio of the number of functions which were minimized by our simplification algorithm. Figure 1 shows the error of solutions of our algorithms, where the error is defined to be $\sum_f(\sigma_k(f) - \tau(f))/(\text{the number of } f)$, which is the average of differences between the numbers of product terms of the simplified ESOPs and the minimum ones.

The algorithm $A[\mathcal{T}_k]$ finds minimum ESOPs for more than 90 percent of functions with $\tau(f) = 3(k+1)$. However, for functions whose minimum number of product terms is large, the ratio of the number of functions which can be minimized decreases and the error increases. This is because for the expansions $f = \bar{x}g_0 \oplus xg_1 \oplus g_2$ corresponding to the minimum ESOP of f, all functions g_0, g_1, and g_2 require more than k product terms, when $\tau(f)$ is large.

Table 2 shows the average computing time of algorithm $B[k]$ for n-variable functions ($n \geq 5$). The functions are generated randomly and the number of them varies as n, 10–10,000. From Table 2, we can say that $B[0]$ can simplify the functions with at most 20 variables and $B[1]$ and $B[k]$ can do those with at most 7 and 6 variables, respectively.

To improve the error of solutions of our algorithm for functions with $\tau(f) \geq 3(k+1)$, we increase the number of expansions that are tested in our algorithm. Here, we used the following methods for experiment.

- Extend the testing set $\mathcal{T}_k^{n-1}$ to $\mathcal{S}_k^{n-1}[f] = \{g \oplus h \mid g \in \mathcal{T}_k^{n-1}, h \in \{0, f_{x=0} \cdot f_{x=0}\}\}$, where "$\cdot$" operator denotes logical product.

- Execute $A[\mathcal{G}]$ for each variable $x_1, x_2, \cdots, x_n$.

Table 2: Average time of $B[k]$ (seconds)

n	$B[0]$	$B[1]$	$B[2]$
5	0.0003	0.01	0.05
6	0.0010	6.73	4159.68
7	0.0029	14791.92	
8	0.0090		
9	0.027		
10	0.082		
11	0.36		
12	0.94		
13	2.74		
14	8.10		
15	24.51		
16	72.07		
17	216.37		
18	645.23		
19	1976.67		
20	6014.47		

The former increases the number of testing expansions by extending the testing set and the latter increases the number of testing expansions by permuting variables. The latter derives a different algorithm from the general simplification algorithm in Section 3.1. By $P[\mathcal{G}]$ we denote this new algorithm with testing set $\mathcal{G}[f]$. We made similar experiments on $A[\mathcal{T}_0]$, $A[\mathcal{S}_0]$, $P[\mathcal{T}_0]$, and $P[\mathcal{S}_0]$ as on $A[\mathcal{T}_k]$. The experimental results are shown in Table 3 and Figure 2. While $A[\mathcal{T}_0]$ minimizes 41 percent of 5-variable functions, $A[\mathcal{S}_0]$, $P[\mathcal{T}_0]$, and $P[\mathcal{S}_0]$ minimize 52 percent, 69 percent, and 75 percent of them, respectively (Table 3). The difference between the numbers of product terms in the simplified ESOPs and the minimum ones decreases by the order of $A[\mathcal{T}_0]$, $A[\mathcal{S}_0]$, $P[\mathcal{T}_0]$, and $P[\mathcal{S}_0]$ (Figure 2). By adding $f_{x=0} \cdot f_{x=1}$ to the testing set and simplifying for all variable permutations we can decrease the errors of solutions of our simplification algorithm.

6 Conclusions

We presented the simplification algorithms of ESOPs, $A[\mathcal{T}_k]$ and $B[k]$. $A[\mathcal{T}_k]$ and $B[k]$ guarantee that they always find the minimum ESOPs for all functions f with $\tau(f) < 3(k+1)$ and that the simplified ESOP is minimum if it has at most $3(k+1)$ product terms. Their time complexity is $O(\sqrt{3}^{kn^2+(k+C)n})$. When k is large, the computing time is large, but the class of functions which can be minimized is large. We also presented the fast algorithm to minimize 5-variable functions on an average computing time 0.01 seconds, which uses $B[k]$ and runs much faster than previously known algorithms.

Table 3: Ratio (%) of the number of minimized 5-variable functions (2)

$\tau(f)$	No. of representative functions	$A[\mathcal{T}_0]$	$A[\mathcal{S}_0]$	$P[\mathcal{T}_0]$	$P[\mathcal{S}_0]$
0	1	100	100	100	100
1	1	100	100	100	100
2	4	100	100	100	100
3	19	94.7	100	94.7	100
4	137	72.3	83.9	93.4	95.6
5	971	43.5	56.3	77.2	84.6
6	3572	36.7	49.5	68.5	73.8
7	2143	44.0	53.0	65.3	72.3
8	86	32.6	51.2	50.0	67.4
9	2(7824)	0	0	0	0
total	6936	40.7	52.4	69.1	75.3

We implemented the presented algorithms in LISP and engaged in experimental studies on their computing time and the errors of their solutions. For $k = 0$, $k = 1$, and $k = 2$, we can simplify functions with 20, 7, and 6 variables, in practical time, respectively. The error of their solutions increases as $\tau(f)$ increases. Then, we experimentally show that adding $f_{x=0} \cdot f_{x=1}$ to the testing set and simplifying for all variable permutations improves the solutions.

Our simplification algorithms for n-variable functions simplify $(n-1)$-variable functions recursively. If we store the intermediate ESOPs that are obtained by recursive simplification, the computing time can be reduced since each function is simplified only once.

Since the cost of an EXOR gate and an OR gate is the same in FPGAs, it is useful to use EXOR gates in the design of FPGAs. We can easily construct the algorithms which reduce the number of literals by computing the number of literals adding to the number of product terms in our algorithms. These algorithms may be useful for the design of FPGAs. Extending our simplification algorithm to multiple-output functions remains to be studied.

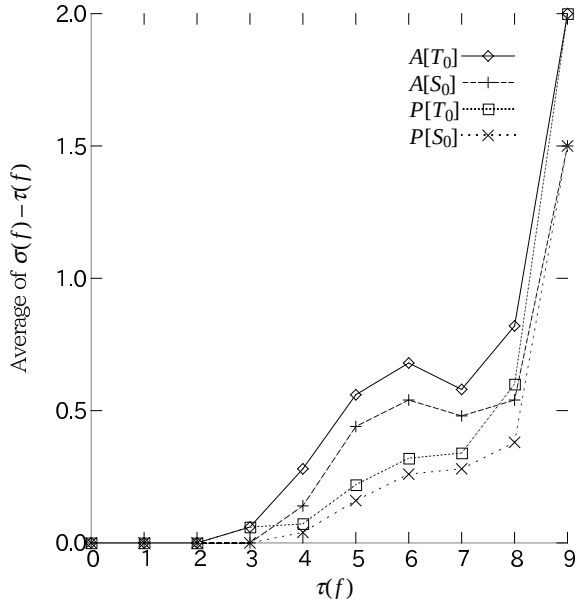

Figure 2: Average difference of the number of products between simplified and minimized ESOPs (2)

References

[1] Koda, N. and Sasao, T., "Four variable AND-EXOR minimum expressions and their properties," *Trans. IEICE*, vol. J74-D-I, no. 11, pp. 765–773, Nov. 1991. (*Computers and Communications in Japan*, vol. 23, no. 10, pp 27–41, May 1993).

[2] Koda, N. and Sasao, T., "An upper bound on the number of product terms in AND-EXOR minimum expressions," *Trans. IEICE*, vol. J75-D-I, no. 3, pp. 135–141, Mar. 1992.

[3] Koda, N. and Sasao, T., "A minimization method for AND-EXOR expressions using lower bound theorem," *Trans. IEICE*, vol. J76-D-I, no. 1, pp. 1–10, Jan. 1993. (*Computers and Communications in Japan*, vol. 24, no. 13, pp 16–27, Apr. 1994).

[4] Koda, N. and Sasao, T., "LP characteristic vector of logic functions and its application," *Trans. IEICE*, vol. J76-D-I, no. 6, pp. 260–268 Jun. 1993.

[5] Nishitani, Y. and Shimizu, K., "Lower bounds on size of periodic functions in Exclusive-OR sum-of-products expressions," *IEICE Trans. Fundamentals*, vol. E77-A, no. 3, pp. 475–482, Mar. 1994.

[6] Perkowski, M. and Chrzanowska-Jeske, M., "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions," *Proc. Int. Symposium on Circuits and Systems '90*, pp. 1652–1655, 1990.

[7] Sasao, T. and Besslich, P., "On the complexity of mod-2 sum PLA's," *IEEE Trans. Comput.*, vol. C-39, no. 2, pp. 262–266, Feb. 1990.

[8] Sasao, T., "EXMIN: A simplification algorithm for exclusive-OR-sum-of-products expressions for multiple-valued input two-valued output functions," *Int. Symposium on Multiple-Valued Logic '90*, pp. 128–135, 1990.

[9] Sasao, T., "EXMIN2: A simplification algorithm for exclusive-or-sum of products expressions for multiple-valued-input two-valued-output functions," *IEEE Trans. on Comput.-Aided Des. Integrated Circuits. and Syst.*, vol. 12, no. 5, pp. 621–632, May 1993.

[10] Sasao, T. and Matsuura, M., "A method to derive exact minimum AND-EXOR expressions using binary decision diagrams," *Technical Report of IEICE*, VLD93-58, pp. 55–60, Oct. 1993.

AUTHORS

Takashi Hirayama received his B.E. degree in Computer Science from Gunma University, Kiryu, Japan, in 1994. He is now working toward his M.E. degree at Gunma University. His current research interests include logic synthesis with EXOR gates and FPGA designs.

Yasuaki Nishitani received his B.E. degree in Electrical Engineering, his M.E. and Dr. of Eng. degrees in Computer Science from Tohoku University, Sendai, Japan, in 1975, 1977, and 1984, respectively. In 1981 he joined the Software Product Engineering Laboratory, NEC Corporation. Since 1987, he has been an associate professor in the Department of Computer Science, Gunma University. His current research interests include software engineering, distributed algorithms and logic circuits.