

A Faster Algorithm of Minimizing AND-EXOR Expressions

Takashi HIRAYAMA^{†a)}, Yasuaki NISHITANI[†], *Regular Members*,
and Toru SATO[†], *Nonmember*

SUMMARY It has been considered difficult to obtain the minimum AND-EXOR expression of a given function with six variables in a practical computing time. In this paper, a faster algorithm of minimizing AND-EXOR expressions is proposed. We believe that our algorithm can compute the minimum AND-EXOR expressions of any six-variable and some seven-variable functions practically. In this paper, we first present a naive algorithm that searches the space of expansions of a given n -variable function f for a minimum expression of f . The space of expansions are generated by using all combinations of $(n-1)$ -variable product terms. Then, how to prune the branches in the search process and how to restrict the search space to obtain the minimum solutions are discussed as the key point of reduction of the computing time. Finally a faster algorithm is constructed by using the methods discussed. Experimental results to demonstrate the effectiveness of these methods are also presented.

key words: AND-EXOR two-level circuit, AND-EXOR expression, exclusive-or sum-of-products expression, logic minimization algorithm

1. Introduction

Logic circuits including exclusive-or (EXOR) gates have some advantages over traditional circuits with only AND and OR gates. EXOR-based realization can improve the testability [8], [16], [21] and often reduces the circuit area [2], [4], [13]. For arithmetic functions, error-correcting functions, and tele-communication functions, AND-EXOR circuits are smaller than AND-OR ones [17].

AND-EXOR logical expressions, which correspond to AND-EXOR circuits, have been studied as the fundamentals of the EXOR-based realization. There are several classes of AND-EXOR expressions [9], [20] such as PPRMs (positive-polarity Reed-Muller expressions), FPRMs (fixed-polarity Reed-Muller expressions), DFPRMs (double-fixed-polarity Reed-Muller expressions), and ESOPs (exclusive-or sum-of-products expressions). ESOPs are expressions such that arbitrary product terms are combined by EXORs. Among the classes of AND-EXOR expressions, ESOPs are the most general AND-EXOR expressions and require the fewest product terms to represent logic functions. The number

of product terms of an ESOP F is called the size of F . Among all ESOPs that represent a logic function f , those with the minimum size are called minimum ESOPs of f . The size of a minimum ESOP of f is denoted by $\tau(f)$. Minimization means to obtain the minimum ESOP of f and simplification means to reduce the number of product terms in ESOPs without guaranteeing minimality.

A lot of algorithms of simplifying ESOPs by applying heuristic rewriting rules have been proposed [1], [5], [6], [24]; Sasao's algorithm [18] and Song-Perkowski's algorithm [23] are known to be especially efficient. However, these algorithms do not guarantee the minimality of the resulting ESOPs. On the other hand, studies about algorithms of minimizing ESOPs are fewer [14], [15], [19]; so far, no efficient minimization algorithms for ESOPs are known. Although a simple algorithm based on the minimization theorem [7] computes a minimum ESOP of a given function f with n variables by testing all $(n-1)$ -variable functions, the algorithm requires a huge computing time if $n \geq 6$. It is due to the number of $(n-1)$ -variable functions, $2^{2^{n-1}}$, which is double exponential. In practice, minimization algorithms proposed previously can be applied only to five-variable functions and a small fraction of the functions with more variables. It has been considered difficult to obtain the minimum ESOPs for functions with six or more variables.

In general, minimization is much more time consuming than simplification. The minimization results, however, can be used to evaluate the performance of the simplification algorithms. Even for a larger function that cannot be minimized, it is important to obtain the minimum expressions of their subfunctions since an arbitrary function can be represented by a combination of its subfunctions. These expressions can be used to efficient initial forms in the simplification algorithms.

In this paper, a faster minimization algorithm is proposed. We believe that the algorithm can compute the minimum ESOPs of any six-variable and some seven-variable functions in a practical computing time. To describe the basic concept of minimization, we refer to the original minimization algorithm based on the minimization theorem and give an simplification algorithm named *naive-tau* $[k](f)$ in Sect.2. While

Manuscript received March 15, 2002.

Manuscript revised June 14, 2002.

Final manuscript received August 1, 2002.

[†]The authors are with the Faculty of Engineering, Iwate University, Morioka-shi, 020-8551 Japan.

a) E-mail: hirayama@cis.iwate-u.ac.jp

the original minimization algorithm tests all $(n-1)$ -variable functions g , $naive\text{-}tau[k](f)$ tests g such that $\tau(g) \leq k$ only, where k is a nonnegative integer. For a given function f , the size of the ESOP obtained by $naive\text{-}tau[k](f)$ is denoted by $\tau[k](f)$. $Naive\text{-}tau[k](f)$ searches ESOPs of f with generating candidate functions g dynamically by combining at most k products. So, $naive\text{-}tau[k](f)$ can be viewed as a program searching for a solution. Section 3 describes a faster version of $naive\text{-}tau[k](f)$, called $fast\text{-}tau[k](f)$, which can prune the branches in the search process. This pruning method does not affect the size of the resulting ESOPs, that is, the size of the ESOP obtained by $fast\text{-}tau[k](f)$ is also $\tau[k](f)$. Since the search space of $fast\text{-}tau[k](f)$ expands with k , a minimum ESOP of f is obtained if k is large enough. To avoid unnecessary search, the upper bound on k such that $\tau[k](f) = \tau(f)$ are discussed in Sect. 4. By using the upper bound, a minimization algorithm $min\text{-}tau(f)$ is obtained from $fast\text{-}tau[k](f)$. Experimental results about the effectiveness of the pruning method and the upper bound on k are given in Sect. 5.

2. Preliminaries

In this section, we describe the basic concept of minimization.

Definition 1: The size of an ESOP F is the number of product terms of F , denoted by $\tau(F)$. Among all ESOPs that represent a logic function f , those with the minimum size are called minimum ESOPs of f . The size of a minimum ESOP of f is denoted by $\tau(f)$. \square

Definition 2: 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 $f_{x:\{0,1\}}$ and $f_{x:\{\}}$ are defined as $f_{x:\{0\}} \oplus f_{x:\{1\}}$ and the logical zero function 0, respectively. In general, $f_{x:(I \oplus J)}$ ($I, J \subseteq \{0, 1\}$, $I \oplus J = (I \cup J) - (I \cap J)$) is defined as $f_{x:I} \oplus f_{x:J}$. \square

With the above subfunctions, an arbitrary function f can be expanded as follows [20], where $I \subseteq \{0, 1\}$.

$$f = x f_{x:(I \oplus \{0\})} \oplus \bar{x} f_{x:(I \oplus \{1\})} \oplus f_{x:(I \oplus \{0,1\})} \quad (1)$$

The expansion with $I = \{0, 1\}$ is known as the Shannon expansion, and the expansions with $I = \{1\}$ and $I = \{0\}$ are known as the positive Davio and the negative Davio expansions, respectively. With an arbitrary $(n-1)$ -variable function g , a more general expansion holds:

$$f = x(f_{x:(I \oplus \{0\})} \oplus g) \oplus \bar{x}(f_{x:(I \oplus \{1\})} \oplus g) \oplus (f_{x:(I \oplus \{0,1\})} \oplus g), \quad (2)$$

since $(x \oplus \bar{x} \oplus 1)g = 0$. Equation (1) is a special case of Eq. (2) with $g = 0$.

Definition 3: Let f and g be functions with n variables and $(n-1)$ variables, respectively. $T_{x:I}(f, g)$ ($I \subseteq \{0, 1\}$) and $T(f, g)$ are defined as follows.

$$\begin{aligned} T_{x:I}(f, g) &= \tau(f_{x:(I \oplus \{0\})} \oplus g) + \tau(f_{x:(I \oplus \{1\})} \oplus g) \\ &\quad + \tau(f_{x:(I \oplus \{0,1\})} \oplus g) \\ T(f, g) &= \min\{T_{x:\{0\}}(f, g), T_{x:\{1\}}(f, g), T_{x:\{0,1\}}(f, g)\} \end{aligned}$$

\square

An ESOP F can be written in the form $F = \bar{x}F_a \oplus xF_b \oplus F_c$, where F_a , F_b , and F_c are ESOPs without literals of the variable x . $F = xF_a \oplus \bar{x}F_b \oplus F_c$ is called an ESOP corresponding to $T_{x:I}(f, g)$ if F_a , F_b , and F_c are minimum ESOPs of $f_{x:(I \oplus \{0\})} \oplus g$, $f_{x:(I \oplus \{1\})} \oplus g$, and $f_{x:(I \oplus \{0,1\})} \oplus g$, respectively. As a property of an ESOP F corresponding to $T_{x:I}(f, g)$, F represents f from Eq. (2), and $\tau(F) = T_{x:I}(f, g)$ holds from $\tau(F_a) = \tau(f_{x:(I \oplus \{0\})} \oplus g)$, $\tau(F_b) = \tau(f_{x:(I \oplus \{1\})} \oplus g)$, and $\tau(F_c) = \tau(f_{x:(I \oplus \{0,1\})} \oplus g)$. An ESOP F corresponding to $T_{x:I}(f, g)$ is called an ESOP corresponding to $T(f, g)$ if $T_{x:I}(f, g) = T(f, g)$. If F is an ESOP corresponding to $T(f, g)$, F represents f and $\tau(F) = T(f, g)$ holds.

The following theorem is the basic concept of minimization [7], [14].

Theorem 1 (Minimization Theorem): Let \mathcal{F}^{n-1} be the set of all $(n-1)$ -variable functions. For an arbitrary n -variable function f , the following equation holds.

$$\tau(f) = \min\{T(f, g) \mid g \in \mathcal{F}^{n-1}\}$$

\square

From Theorem 1, we can construct a simple minimization algorithm, which is shown as $min\text{-}esop$ in Fig. 1. In the algorithm, a minimum ESOP of an arbitrary n -variable function f for $n \leq m$ and the number of products $\tau(f)$ are assumed to be known as the terminal condition of the recursive algorithm. For example, m

```
function min-esop(f) : (integer, ESOP);
{ f is an n-variable function }
var t0, t1, t2, t3, s : integer;
var F0, F1, F2, F3, F : ESOP;
begin
  if n ≤ m then return (τ(f), a minimum ESOP of f);
  s := a large integer;
  for g ∈ Fn-1 do begin
    (t0, F0) := min-esop(fx:{0} ⊕ g);
    (t1, F1) := min-esop(fx:{1} ⊕ g);
    (t2, F2) := min-esop(fx:{0,1} ⊕ g);
    (t3, F3) := min-esop(fx:{} ⊕ g);
    if T(f, g) ≤ s then (s, F) := (T(f, g), an ESOP corresponding to T(f, g));
  end;
  return (s, F)
end;
```

Fig. 1 $min\text{-}esop$: A simple minimization algorithm.

can be 4 since a list of minimum ESOPs for all functions with four or less variables has been presented in [10]. In *min-esop*, $T(f, g)$ can be computed from t_0, t_1, \dots, t_3 , and an ESOP corresponding to $T(f, g)$ can be obtained from F_0, F_1, \dots, F_3 .

In *min-esop*, a minimum ESOP of f can be obtained together with $\tau(f)$, that is, a minimum ESOP and $\tau(f)$ can be obtained by the same algorithm. Although, in the rest of this paper, we focus on obtaining $\tau(f)$ to simplify the discussion, the discussion includes obtaining a minimum ESOP on the analogy of Theorem 1 and *min-esop*.

The simple minimization algorithm *min-esop* requires very large computing time since the time complexity mainly depends on the number of functions of \mathcal{F}^{n-1} ($|\mathcal{F}^{n-1}| = 2^{2^{n-1}}$). Then we attempt to reduce the set \mathcal{F}^{n-1} to some subset specified by an optional parameter k .

Definition 4: Let f be an n -variable function and k be a non-negative integer. $\tau[k](f)$ is defined as follows.

$$\tau[k](f) = \min\{T(f, g) \mid g \in \mathcal{F}^{n-1} \text{ and } \tau(g) \leq k\} \quad \square$$

An algorithm obtaining $\tau[k](f)$ is shown in Fig. 2, in which $(n-1)$ -variable functions g such that $\tau(g) \leq k$ are generated by EXOR-combinations of at most k products. \mathcal{P}^{n-1} used in *naive-tau* is the set of all $(n-1)$ -variable functions that can be represented by exactly one product. Although the values of $\tau(g)$ and $T(f, g)$ should be passed to the procedure $S(g, \mathcal{P})$ as its additional arguments to avoid computing the same values for practical programming, those arguments are omitted in Fig. 2 for simplicity of the description.

Theorem 2: *naive-tau* $[k](f) = \tau[k](f)$

Proof: It is obvious that the algorithm *naive-tau* computes $T(f, h)$ for all functions $h = g \oplus p$ such that $\tau(g \oplus p) \leq k$, and does not compute $T(f, h)$ for functions

```

function naive-tau $[k](f)$  : integer;
{  $f$  is an  $n$ -variable function and  $k$  is a nonnegative integer }
var  $s$  : integer;
procedure  $S(g, \mathcal{P})$ ;
{  $g$  is an  $(n-1)$ -variable function and  $\mathcal{P}$  is a set of products }
begin
  if  $\tau(g) \geq k$  or  $\mathcal{P} = \emptyset$  then return;
   $p \in \mathcal{P}$ ;
  if  $\tau(g \oplus p) = \tau(g) + 1$  then begin
     $s := \min\{s, T(f, g \oplus p)\}$ ;
     $S(g \oplus p, \mathcal{P} - \{p\})$ 
  end;
   $S(g, \mathcal{P} - \{p\})$ 
end;
begin
  if  $n \leq m$  then return  $\tau(f)$ ;
   $s := T(f, 0)$ ;
   $S(0, \mathcal{P}^{n-1})$ ;
  return  $s$ 
end;

```

Fig. 2 *naive-tau*: An algorithm for $\tau[k](f)$.

h such that $\tau(h) > k$. Hence we have the theorem. \square

The following lemmas are used in the later sections.

Lemma 1: Let f and g be functions with n variables and $(n-1)$ variables, respectively. The following equation holds for arbitrary subscripts I and J ($I, J \subseteq \{0, 1\}$).

$$T_{x:I}(f, g) = T_{x:J}(f, f_{x:(I \oplus J)} \oplus g)$$

Proof:

$$\begin{aligned}
 T_{x:J}(f, f_{x:(I \oplus J)} \oplus g) &= \tau(f_{x:(J \oplus \{0\})} \oplus f_{x:(I \oplus J)} \oplus g) \\
 &\quad + \tau(f_{x:(J \oplus \{1\})} \oplus f_{x:(I \oplus J)} \oplus g) \\
 &\quad + \tau(f_{x:(J \oplus \{0,1\})} \oplus f_{x:(I \oplus J)} \oplus g) \\
 &= \tau(f_{x:(I \oplus \{0\})} \oplus g) + \tau(f_{x:(I \oplus \{1\})} \oplus g) \\
 &\quad + \tau(f_{x:(I \oplus \{0,1\})} \oplus g) \\
 &= T_{x:I}(f, g)
 \end{aligned}$$

\square

Lemma 2: The following inequality holds for arbitrary functions f and h .

$$\tau(f \oplus h) \geq \tau(f) - \tau(h)$$

Proof: Since $f \oplus h \oplus h = f$, $\tau(f \oplus h) + \tau(h) \geq \tau(f)$. \square

3. Pruning for Fast Computation

For a product set \mathcal{P} , let $\mathcal{H}(\mathcal{P})$ be the set of all functions that can be represented by one product or EXOR-combinations of products in \mathcal{P} . From Fig. 2, it is observed that the procedure $S(g, \mathcal{P})$ searches $T(f, g \oplus h)$ on $h \in \mathcal{H}(\mathcal{P})$ such that $\tau(g \oplus h) = \tau(g) + \tau(h) \leq k$, and s is updated as $s := T(f, g \oplus h)$ if $T(f, g \oplus h) < s$. From the behavior, $S(g, \mathcal{P})$ can be considered as a program searching for the minimum solution. In this section, we give a lemma to reduce the search space and improve the algorithm.

Lemma 3: Let f and g be functions with n variables and $(n-1)$ variables, respectively. For any $(n-1)$ -variable function h such that $\tau(g \oplus h) = \tau(g) + \tau(h)$, the following inequality holds.

$$T(f, g \oplus h) \geq T(f, g) - \tau(h)$$

Proof: From the definition of $T(f, g)$, $T(f, g \oplus h) = \min\{T_{x:\{0\}}(f, g \oplus h), T_{x:\{1\}}(f, g \oplus h), T_{x:\{0,1\}}(f, g \oplus h)\}$. Without loss of generality, we assume that $T(f, g \oplus h) = T_{x:\{0,1\}}(f, g \oplus h) = \tau(f_{x:\{1\}} \oplus g \oplus h) + \tau(f_{x:\{0\}} \oplus g \oplus h) + \tau(h \oplus g)$, where h is a function specified by the above lemma. From Lemma 2 and the assumption of the above lemma, the following relations hold for $\tau(f_{x:\{1\}} \oplus g \oplus h)$, $\tau(f_{x:\{0\}} \oplus g \oplus h)$, and $\tau(g \oplus h)$.

$$\tau(f_{x:\{1\}} \oplus g \oplus h) \geq \tau(f_{x:\{1\}} \oplus g) - \tau(h)$$

```

function fast-tau[k](f) : integer;
var s : integer;
procedure S(g, P);
begin
  if  $T(f, g) - (k - \tau(g)) \geq s$  or  $P = \emptyset$  then return;
   $p \in P$ ;
  if  $\tau(g \oplus p) = \tau(g) + 1$  then begin
     $s := \min\{s, T(f, g \oplus p)\}$ ;
     $S(g \oplus p, P - \{p\})$ 
  end;
   $S(g, P - \{p\})$ 
end;
begin
  if  $n \leq m$  then return  $\tau(f)$ ;
   $s := T(f, 0)$ ;
   $S(0, P^{n-1})$ ;
  return s
end;

```

Fig. 3 fast-tau: An algorithm for $\tau[k](f)$.

$$\begin{aligned} \tau(f_{x:\{0\}} \oplus g \oplus h) &\geq \tau(f_{x:\{0\}} \oplus g) - \tau(h) \\ \tau(g \oplus h) &= \tau(g) + \tau(h) \end{aligned}$$

From the above relations, we have the following.

$$\begin{aligned} T(f, g \oplus h) &= T_{x:\{0,1\}}(f, g \oplus h) \\ &\geq T_{x:\{0,1\}}(f, g) - \tau(h) \geq T(f, g) - \tau(h) \end{aligned}$$

□

From Lemma 3, for any h such that $\tau(g \oplus h) = \tau(g) + \tau(h) \leq k$, $T(f, g \oplus h) \geq s$ is guaranteed if $T(f, g) - (k - \tau(g)) \geq s$. In other words, there exists no smaller $T(f, g \oplus h)$ than s in the search space of $S(g, P)$ when $T(f, g) - (k - \tau(g)) \geq s$. Recall that $S(g, P)$ searches $T(f, g \oplus h)$ on h such that $\tau(g \oplus h) = \tau(g) + \tau(h) \leq k$. We call the condition $T(f, g) - (k - \tau(g)) \geq s$ the pruning condition.

From the above discussion, we have an improved algorithm, *fast-tau*, shown in Fig. 3. The difference between *fast-tau* and *naive-tau* is indicated by the underline in Fig. 3, in which the pruning condition is used as the terminal condition of $S(g, P)$.

Theorem 3: $\text{fast-tau}[k](f) = \tau[k](f)$

Proof: The difference between *naive-tau* and *fast-tau* is the terminal condition of $S(g, P)$; the pruning condition $T(f, g) - (k - \tau(g)) \geq s$ is used in *fast-tau* while the condition $\tau(g) \geq k$ is used in *naive-tau*. The pruning condition can be rewritten as $\tau(g) + (T(f, g) - s) \geq k$. Moreover $T(f, g) \geq s$ holds when $S(g, P)$ starts the execution. So, if the condition $\tau(g) \geq k$ in *naive-tau* is true, the pruning condition in *fast-tau* is also true. Hence we have $\text{naive-tau}[k](f) \leq \text{fast-tau}[k](f)$.

On the other hand, there is a case where the condition $\tau(g) \geq k$ is false but the pruning condition $T(f, g) - (k - \tau(g)) \geq s$ is true. In this case, the behavior of *fast-tau* is different from that of *naive-tau*; $S(g, P)$ of *fast-tau* computes none of $T(f, g \oplus p)$, $S(g \oplus p, P - \{p\})$, and $S(g, P - \{p\})$. However, in this case, Lemma 3 guarantees $T(f, g \oplus h) \geq T(f, g) -$

$\tau(h) \geq T(f, g) - (k - \tau(g)) \geq s$ for any h such that $\tau(g) + \tau(h) = \tau(g \oplus h) \leq k$. Thus the difference of the behavior does not affect the resulting s , i.e., $\text{naive-tau}[k](f) = \text{fast-tau}[k](f)$. Since $\text{naive-tau}[k](f) = \tau[k](f)$ from Theorem 2, we have $\text{fast-tau}[k](f) = \tau[k](f)$. □

4. Upper Bound on k for Minimization

We gave an algorithm to compute $\tau[k](f)$ in the previous section. Our goal is to develop a faster algorithm to compute $\tau(f)$. Since $\tau[0](f) \geq \tau[1](f) \geq \dots \geq \tau(f)$, $\tau(f)$ can be obtained by computing $\tau[k](f)$ with a large enough k . In this section, we consider the upper bound on k such that $\tau(f) = \tau[k](f)$.

Definition 5: The minimum k such that $\tau[k](f) = \tau(f)$ is denoted by $\kappa(f)$, namely, $\kappa(f) = \min\{k \mid \tau[k](f) = \tau(f)\}$. □

From the definitions of $\kappa(f)$ and $\tau[k](f)$, we have the following properties.

Property 1: $\tau(f) = \tau[k](f)$ if $k \geq \kappa(f)$. □

Property 2: $\kappa(f) \leq \tau(g)$ if $\tau(f) = T_{x:I}(f, g)$. □

We present two lemmas about the upper bound on $\kappa(f)$.

Lemma 4: $\kappa(f) \leq \lfloor \tau(f)/3 \rfloor$

Proof: From Theorem 1, there exist $g \in \mathcal{F}^{n-1}$ and $I \in \{\{0\}, \{1\}, \{0, 1\}\}$ such that

$$\begin{aligned} \tau(f) &= T_{x:I}(f, g) \\ &= \tau(f_{x:(I \oplus \{0\})} \oplus g) + \tau(f_{x:(I \oplus \{1\})} \oplus g) \\ &\quad + \tau(f_{x:(I \oplus \{0,1\})} \oplus g). \end{aligned}$$

Let $\tau(f_{x:(I \oplus J)} \oplus g)$ ($J \in \{\{0\}, \{1\}, \{0, 1\}\}$) be the minimum one among $\tau(f_{x:(I \oplus \{0\})} \oplus g)$, $\tau(f_{x:(I \oplus \{1\})} \oplus g)$, and $\tau(f_{x:(I \oplus \{0,1\})} \oplus g)$. Then, $\tau(f_{x:(I \oplus J)} \oplus g) \leq \lfloor \tau(f)/3 \rfloor$ holds. Since $T_{x:I}(f, g) = T_{x:J}(f, f_{x:(I \oplus J)} \oplus g)$ from Lemma 1, $\tau(f) = T_{x:J}(f, f_{x:(I \oplus J)} \oplus g)$ holds. From this result and Property 2, we have $\kappa(f) \leq \tau(f_{x:(I \oplus J)} \oplus g) \leq \lfloor \tau(f)/3 \rfloor$. □

Definition 6: $\gamma(f) = \max\{\tau(f_{x:J}) \mid J \in \{\{0\}, \{1\}, \{0, 1\}\}\}$ □

Lemma 5: $\kappa(f) \leq \tau(f) - \gamma(f)$

Proof: $\tau(f)$ and $\gamma(f)$ can be written as $\tau(f) = T_{x:I}(f, g)$ and $\gamma(f) = \tau(f_{x:J})$, respectively, where $I, J \in \{\{0\}, \{1\}, \{0, 1\}\}$.

If the relation

$$\tau(f_{x:(I \oplus J)} \oplus g) \leq \tau(f) - \tau(f_{x:J}) \quad (3)$$

is proved, the lemma is proved; since $\tau(f) = T_{x:I}(f, g) = T_{x:J}(f, f_{x:(I \oplus J)} \oplus g)$ from Lemma 1, we have $\kappa(f) \leq \tau(f_{x:(I \oplus J)} \oplus g) \leq \tau(f) - \tau(f_{x:J}) = \tau(f) - \gamma(f)$ from Property 2 and the relation (3). In

the following, therefore, we prove the relation (3).

Assume that $J = \{0\}$ (the cases of $J = \{1\}$ and $J = \{0, 1\}$ can be discussed similarly). Since $\tau(f) = T_{x:I}(f, g) = \tau(f_{x:(I \oplus \{0\})} \oplus g) + \tau(f_{x:(I \oplus \{1\})} \oplus g) + \tau(f_{x:(I \oplus \{0, 1\})} \oplus g)$, $\tau(f_{x:(I \oplus J)} \oplus g)$ is written as

$$\begin{aligned} & \tau(f_{x:(I \oplus J)} \oplus g) \\ &= \tau(f_{x:(I \oplus \{0\})} \oplus g) \\ &= \tau(f) - (\tau(f_{x:(I \oplus \{1\})} \oplus g) + \tau(f_{x:(I \oplus \{0, 1\})} \oplus g)). \end{aligned}$$

Moreover, since $(f_{x:(I \oplus \{1\})} \oplus g) \oplus (f_{x:(I \oplus \{0, 1\})} \oplus g) = f_{x:\{0\}} = f_{x:J}$, $\tau(f_{x:(I \oplus \{1\})} \oplus g) + \tau(f_{x:(I \oplus \{0, 1\})} \oplus g) \geq \tau(f_{x:\{0\}}) = \tau(f_{x:J})$ holds. Thus, we have

$$\begin{aligned} & \tau(f_{x:(I \oplus J)} \oplus g) \\ &= \tau(f_{x:(I \oplus \{0\})} \oplus g) \\ &\leq \tau(f) - \tau(f_{x:\{0\}}) = \tau(f) - \tau(f_{x:J}). \end{aligned}$$

□

From Lemma 4 and 5, we have $\kappa(f) \leq \min\{\lfloor \tau(f)/3 \rfloor, \tau(f) - \gamma(f)\}$. Since $\tau(f) \leq T(f, g)$ holds for any $g \in \mathcal{F}^{n-1}$, we have the following corollary.

Corollary 1: Let f be an n -variable function and g be an arbitrary $(n-1)$ -variable function.

$$\kappa(f) \leq \min\{\lfloor T(f, g)/3 \rfloor, T(f, g) - \gamma(f)\}$$

□

From the above corollary, $\text{fast-tau}[k](f)$ returns $\tau(f)$ if the value of the parameter k is set as $k = \min\{\lfloor T(f, g)/3 \rfloor, T(f, g) - \gamma(f)\}$. Based on this idea, we have an algorithm obtaining $\tau(f)$ by modifying fast-tau . The algorithm is called min-tau0 and shown in Fig. 4. In min-tau0 , the value of k is set at the underlined statements. The initial k is $\min\{\lfloor T(f, 0)/3 \rfloor, T(f, 0) - \gamma(f)\}$, in which $T(f, 0)$ and $\gamma(f)$ are computed by applying min-tau0 recursively. Similarly, $\tau(g \oplus p)$ and $T(f, g \oplus p)$ in $S(g, \mathcal{P})$ are computed by applying min-tau0 recursively. During the execution of $S(g, \mathcal{P})$, the value of k may be updated

```
function min-tau0(f) : integer;
var s, k : integer;
procedure S(g, P);
begin
  if T(f, g) - (k - τ(g)) ≥ s or P = ∅ then return;
  p ∈ P;
  if τ(g ⊕ p) = τ(g) + 1 then begin
    s := min{s, T(f, g ⊕ p)};
    k := min{⌊s/3⌋, s - γ(f)};
    S(g ⊕ p, P - {p});
  end;
  S(g, P - {p});
end;
begin
  if n ≤ m then return τ(f);
  s := T(f, 0);
  k := min{⌊s/3⌋, s - γ(f)};
  S(0, Pn-1);
  return s;
end;
```

Fig. 4 min-tau0 : An algorithm for $\tau(f)$.

to be smaller if a smaller $s (= T(f, g))$ is found. The search space of min-tau0 is reduced dynamically as the value of k decreased.

Theorem 4: $\text{min-tau0}(f) = \tau(f)$

Proof: It is obvious that $\text{min-tau0}(f) \geq \tau(f)$. We prove $\text{min-tau0}(f) \leq \tau(f)$. Let s_{\min} be the return value of the algorithm $\text{min-tau0}(f)$. Then let us consider the return value of the algorithm $\text{fast-tau}[k_{\min}](f)$, where $k_{\min} = \min\{\lfloor s_{\min}/3 \rfloor, s_{\min} - \gamma(f)\}$. While $\text{fast-tau}[k_{\min}](f)$ computes $T(f, g)$ for $(n-1)$ -variable functions g such that $\tau(g) \leq k_{\min}$, $\text{min-tau0}(f)$ computes $T(f, g)$ for all functions g used in $\text{fast-tau}[k_{\min}](f)$ and some more functions g . Hence we have $\text{fast-tau}[k_{\min}](f) \geq \text{min-tau0}(f)$. From Theorem 3, $\tau[k_{\min}](f) = \text{fast-tau}[k_{\min}](f) \geq \text{min-tau0}(f)$ holds. And $\tau(f) = \tau[k_{\min}](f)$ holds because $\kappa(f) \leq \min\{\lfloor s_{\min}/3 \rfloor, s_{\min} - \gamma(f)\} = k_{\min}$ holds from Corollary 1. Thus, we have $\tau(f) = \tau[k_{\min}](f) \geq \text{min-tau0}(f)$. □

It is important to decrease the value k for the reduction of computing time. Under some conditions, k can be slightly smaller than that in Corollary 1.

Lemma 6: If $\tau(f) < s$, $\kappa(f) \leq \min\{\lfloor (s-1)/3 \rfloor, s-1 - \gamma(f)\}$ holds.

Proof: From the assumption of the lemma, $\tau(f) \leq s-1$. From Lemma 4 and 5, $\kappa(f) \leq \min\{\lfloor \tau(f)/3 \rfloor, \tau(f) - \gamma(f)\}$. By replacing $\tau(f)$ with $s-1$, we have the lemma. □

From Lemma 6, we have an improved algorithm, min-tau , shown in Fig. 5. The difference between min-tau0 and min-tau is indicated by the underlines in Fig. 5, in which $\min\{\lfloor (s-1)/3 \rfloor, s-1 - \gamma(f)\}$ is used instead of $\min\{\lfloor s/3 \rfloor, s - \gamma(f)\}$. From the modification, min-tau assumes $\tau(f) < s$ and tries obtaining a smaller solution than s . Since the assumption is usually true during the execution of min-tau , Lemma 6 is useful to reduce the computing time. If the assumption is false

```
function min-tau(f) : integer;
var s, k : integer;
procedure S(g, P);
begin
  if T(f, g) - (k - τ(g)) ≥ s or P = ∅ then return;
  p ∈ P;
  if τ(g ⊕ p) = τ(g) + 1 then begin
    s := min{s, T(f, g ⊕ p)};
    k := min{⌊(s-1)/3⌋, s-1 - γ(f)};
    S(g ⊕ p, P - {p});
  end;
  S(g, P - {p});
end;
begin
  if n ≤ m then return τ(f);
  s := T(f, 0);
  k := min{⌊(s-1)/3⌋, s-1 - γ(f)};
  S(0, Pn-1);
  return s;
end;
```

Fig. 5 min-tau : An algorithm for $\tau(f)$.

in the execution, this means $s = \tau(f)$ is guaranteed.

Theorem 5: $\min\text{-tau}(f) = \tau(f)$

Proof: Let s_{\min} be the return value of the algorithm $\min\text{-tau}(f)$ and k_{\min} be $\min\{\lfloor (s_{\min} - 1)/3 \rfloor, s_{\min} - 1 - \gamma(f)\}$. From the similar discussion of the proof of Theorem 4, we have $\tau[k_{\min}](f) = \text{fast}\text{-tau}[k_{\min}](f) \geq \min\text{-tau}(f) = s_{\min}$. Suppose $\tau(f) < s_{\min}$ in contradiction. Since $\kappa(f) \leq k_{\min}$ from Lemma 6, $\tau(f) = \tau[k_{\min}](f) \geq \min\text{-tau}(f) = s_{\min}$. This is a contradiction. \square

5. Experimental Results

In the final version of the minimization algorithm $\min\text{-tau}$, we used the following two methods to reduce the search space.

1. Lower bound on $T(f, g \oplus h)$: *naive-tau* and *fast-tau*.
2. Upper bound on $\kappa(f)$: $\lfloor s/3 \rfloor$, $s - \gamma(f)$, $\lfloor (s - 1)/3 \rfloor$, and $s - 1 - \gamma(f)$.

To test the effectiveness of the above methods, we implemented twelve algorithms corresponding to all combinations of these methods and counted the number of calls for $T(f, g \oplus p)$ in each algorithm. In our implementation, the terminating condition of the recursive call is $n \leq 4$ ($m = 4$). We applied these algorithms to all representative functions of LP-equivalence classes [10], [11] of five-variable functions, the number of which is 6,936. Table 1 shows an average number of calls for $T(f, g)$ per representative function. In Table 1, $\min\text{-tau0}$ and $\min\text{-tau}$ represent $\kappa(f) \leq \min\{\lfloor s/3 \rfloor, s - \gamma(f)\}$ and $\kappa(f) \leq \min\{\lfloor (s - 1)/3 \rfloor, s - 1 - \gamma(f)\}$, respectively, and *naive* and *fast* represent algorithms with and without using the lower bound on $T(f, g \oplus h)$, respectively. In other words, *naive* corresponds to using $\tau(g) \geq k$ as the terminating condition of $S(g, \mathcal{P})$, and *fast* corresponds to using $T(f, g) - (k - \tau(g)) \geq s$ instead. From the table, the number of calls is reduced significantly by changing *naive* to *fast*. For the best reduction, both of $\lfloor (s - 1)/3 \rfloor$ and $s - 1 - \gamma(f)$ are required as the upper bound on $\kappa(f)$.

The real computing time of $\min\text{-tau}$ is also measured. We implemented the algorithm in C language, and the program was executed on a computer with AMD Athlon XP 1900+ 1.61 GHz, whose operating system is Free BSD 4.3-Release. The above 6,936

representative functions were minimized in 0.204 seconds; the average computing time was 2.94×10^{-5} seconds. All the six-variable symmetric functions, the number of which is 128, were minimized in 1764.7 seconds; the average time was 13.8 seconds and the worst time was 100.4 seconds. Among these 128 functions, the maximum $\tau(f)$ is 15. The most complex six-variable function 6bbd-bdd6-bdd6-d66b [11], which is non-symmetric and its $\tau(f)$ is 15, was minimized in 3.7 seconds. No other minimization algorithms that can minimize such large functions in a practical computing time are known. The computing time of our algorithm depends on $\tau(f)$ and the number of variables. We believe that our program can minimize any six-variable functions in a practical computing time since it is known that $\tau(f) \leq 15$ holds for any six-variable functions [11]. We also made experiments on seven-variable functions and found that seven-variable functions with $\tau(f) \leq 10$ can be minimized within ten minutes.

By extending our program to the minimization of the characteristic functions [18], [22], we minimized some multiple-output functions. Among the results, we found $\tau(f) = 7$ for ROT4, which is smaller than the simplification result consisting of eight products [12] computed by EXMIN2 [18]. ROT4 is the 4-input 3-output square-root function defined as $f = \lfloor \sqrt{x} + 0.5 \rfloor$ [22].

6. Conclusions

We presented a faster algorithm of minimizing ESOPs called $\min\text{-tau}$. For a given n -variable function f , the algorithm searches $(n - 1)$ -variable functions g for the minimum $T(f, g)$, in which the functions g are generated by combining at most k products with EXORs. To reduce the computing time, two methods were introduced: the lower bound on $T(f, g)$ to prune the branches in the search process and the upper bound on k to restrict the search space. From the experimental results, it was confirmed that using both of the above two methods reduces the computing time effectively. The experimental results on six- or seven-variable functions suggest that $\min\text{-tau}$ is fast enough to minimize any six-variable and some seven-variable functions. Our algorithm is faster than any other minimization algorithms that were ever proposed.

References

- [1] D. Brand and T. Sasao, "Minimization of AND-EXOR expressions using rewrite rules," IEEE Trans. Comput., vol.42, no.5, pp.568–576, May 1993.
- [2] S. Chattopadhyay, S. Roy, and P.P. Chaudhuri, "KGP-MIN: An efficient multilevel multioutput AND-OR-XOR minimizer," IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst., vol.16, no.3, pp.257–265, March 1997.
- [3] M. Chatterjee, D.K. Pradhan, and W. Kunz, "LOT: Logic optimization with testability—New transformations

Table 1 Average number of calls for $T(f, g)$.

$\kappa(f)$	<i>naive</i>	<i>fast</i>
$\kappa(f) \leq \lfloor s/3 \rfloor$	2537.2	254.2
$\kappa(f) \leq s - \gamma(f)$	11187.0	676.7
$\min\text{-tau0}$	2211.5	240.6
$\kappa(f) \leq \lfloor (s - 1)/3 \rfloor$	1030.0	174.2
$\kappa(f) \leq s - 1 - \gamma(f)$	579.0	128.5
$\min\text{-tau}$	526.5	120.1

- for logic synthesis," *IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst.*, vol.17, no.5, pp.386-399, May 1998.
- [4] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," *IEICE Trans. Inf. & Syst.*, vol.E80-D, no.10, pp.1001-1008, Oct. 1997.
 - [5] H. Fleisher, M. Tavel, and J. Yeager, "Computer algorithm for minimizing Reed-Muller canonical forms," *IEEE Trans. Comput.*, vol.C-36, no.2, pp.247-250, Feb. 1987.
 - [6] M. Helliwell and M.A. Perkowski, "A fast algorithm to minimize multi-output mixed-polarity generalized Reed-Muller forms," *Proc. 25th ACM/IEEE Design Automation Conference*, pp.427-432, June 1988.
 - [7] T. Hirayama and Y. Nishitani, "A simplification algorithm of AND-EXOR expressions guaranteeing minimality for some subclass of logic functions," *IEICE Trans. Inf. & Syst.* (Japanese Edition), vol.J78-D-I, no.4, pp.409-415, April 1995.
 - [8] T. Hirayama, G. Koda, Y. Nishitani, and K. Shimizu, "Easily testable realization based on single-rail-input OR-AND-EXOR expressions," *IEICE Trans. Inf. & Syst.*, vol.E82-D, no.9, pp.1278-1286, Sept. 1999.
 - [9] T. Hirayama, K. Nagasawa, Y. Nishitani, and K. Shimizu, "Double fixed-polarity Reed-Muller expressions: A new class of AND-EXOR expressions for compact and testable realization," *Trans. IPS Japan*, vol.42, no.4, pp.983-991, April 2001.
 - [10] N. Koda and T. Sasao, "Four variable AND-EXOR minimization expressions and their properties," *IEICE Trans. Inf. & Syst.* (Japanese Edition), vol.J74-D-I, no.11, pp.765-773, Nov. 1991.
 - [11] N. Koda and T. Sasao, "An upper bound on the number of products in minimum ESOPs," *Proc. IFIP WG10.5 Reed-Muller'95*, Japan, pp.94-101, Aug. 1995.
 - [12] N. Koda and T. Sasao, "A simplification method for AND-EXOR expressions for multiple-output functions," *IEICE Trans. Inf. & Syst.* (Japanese Edition), vol.J79-D-I, no.2, pp.43-52, Feb. 1996.
 - [13] F. Luccio and L. Pagli, "On a new boolean function with applications," *IEEE Trans. Comput.*, vol.48, no.3, pp.296-310, March 1999.
 - [14] Y. Nishitani and K. Shimizu, "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, March 1994.
 - [15] M.A. Perkowski and M. Chrzanowska-Jeske, "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions," *Proc. International Symposium Circuits & Syst.*, USA, pp.1652-1655, May 1990.
 - [16] S.M. Reddy, "Easily testable realization for logic functions," *IEEE Trans. Comput.*, vol.C-21, no.11, pp.1183-1188, Nov. 1972.
 - [17] T. Sasao and P. Besslich, "On the complexity of mod-2 sum PLA's," *IEEE Trans. Comput.*, vol.39, no.2, pp.262-266, Feb. 1990.
 - [18] 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.12, no.5, pp.621-632, May 1993.
 - [19] T. Sasao, "An exact minimization of AND-EXOR expressions using BDD's," *Proc. IFIP WG10.5 Reed-Muller'93*, Germany, pp.91-98, 1993.
 - [20] T. Sasao, "Representations of logic functions using EXOR operators," in *Representations of Discrete Functions*, ed. T.

Sasao and M. Fujita, pp.29-54, Kluwer Academic Publishers, 1996.

- [21] T. Sasao, "Easily testable realizations for generalized Reed-Muller expansions," *IEEE Trans. Comput.*, vol.46, no.6, pp.709-716, June 1997.
- [22] T. Sasao, *Switching Theory For Logic Synthesis*, Kluwer Academic Publishers, 1999.
- [23] 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.15, no.4, pp.385-395, April 1996.
- [24] Y. Ye and K. Roy, "An XOR-based decomposition diagram and its application in synthesis of AND/XOR networks," *IEICE Trans. Fundamentals*, vol.E80-A, no.10, pp.1742-1748, Oct. 1997.



research interests include high level and logic synthesis and design for testability.

Takashi Hirayama received his B.E., M.E., and Ph.D. degrees in computer science from Gunma University, Kiryu, Japan, in 1994, 1996, and 1999, respectively. From 1999 to 2001 he was a research assistant in the Department of Electrical and Electronics Engineering, Ashikaga Institute of Technology. He is currently a lecturer in the Department of Computer and Information Sciences, Faculty of Engineering, Iwate University. His



department of Computer and Information Sciences, Faculty of Engineering, Iwate University. His current research interests include switching theory, software engineering, and distributed algorithms.

Yasuaki Nishitani received his B.E. degree in electrical engineering, his M.E. and Ph.D. degrees in computer science from Tohoku University, Sendai, Japan, in 1975, 1977, and 1984, respectively. In 1981 he joined the Software Product Engineering Laboratory at the NEC corporation. From 1987 to 2000 he was an associate professor in the Department of Computer Science, Gunma University. Since 2000 he has been a professor in the



Toru Sato received his B.E. degree from Iwate University, Morioka, Japan, in 2001. He is currently working toward his M.E. degree at Iwate University. His research interests include logic synthesis and optimization algorithms.