設 計 自 動 化 77－13
(1995. 10. 20)

索表による AND-OR-EXOR 論理式最小化の一手法

デブナス デバトシュ，　笹尾 勤
九州工業大学情報工学部電子情報工学科
〒820 飯塚市大字川津 680-4
TEL: 0948-29-7675
debnath@aries20.cse.kyutech.ac.jp
sasao@cse.kyutech.ac.jp

あらまし: 出力段に 2 入力 EXOR 論理素子を一つ用いた AND-OR-EXOR 三段論理回路を考える．二つの論理和形を EXOR で結合した式を EX-SOP と呼ぶ．本問題は, 積項数最小の EX-SOP を求める問題に帰着できる．5 変数までの論理式の最小化アルゴリズムと 6 変数以上に適用できる論理式簡単化アルゴリズムを示す．これらのアルゴリズムでは, 5 変数 NP 代表関数の最小 EX-SOP の表を用いている．また, n 変数関数 ($n \geq 6$) は高々$9 \cdot 2^{n-5}$積項で表現できることを示す．

和文キーワード: 3 段論理回路, AND-EXOR, 論理式簡単化, スペクトラル法, 複雑度, NP 同値類.

An Optimization of AND-OR-EXOR Three-Level Expressions by Table Look-Up

Debatosh DEBNATH and Tsutomu SASAO
Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka 820, Japan
TEL: +81-948-29-7675

Abstract: This paper presents a design method for AND-OR-EXOR three-level networks, where a single two-input EXOR gate is used. The network realizes an Exclusive-OR of two sum-of-products expressions (EX-SOP). The problem is to minimize the total number of product terms. Algorithms for minimization of EX-SOPs with up to five variables are shown. A heuristic algorithm is also presented to simplify EX-SOPs with six or more variables. Up to five variables, all the representative functions of NP-equivalence classes were minimized. For five-variable functions, the upper bound on the number of products in minimum EX-SOPs is found to be 9. For n-variable ($n \geq 6$) functions, minimum EX-SOPs require at most $9 \cdot 2^{n-5}$ products. This upper bound is smaller than 2^{n-1}, the upper bound for the conventional sum-of-products expressions.

Key words: Three-level network, AND-EXOR, logic minimization, spectral method, complexity, NP-equivalence class.

1 Introduction

Logic networks are usually designed by using AND and OR gates. However, it has been observed that the addition of exclusive-OR (EXOR) gates in the design often produce better networks [8]. For example, on the average, five-variable functions require 7.46 products in minimum SOPs (sum-of-products expressions), while 6.16 products in minimum ESOPs (exclusive-or sum-of-products expressions) [9]. On the other hand, to realize an arbitrary function of six variables, minimum SOPs require at most 32 products, while minimum ESOPs require at most 15 products [5]. These reveal the advantages of designing logic networks using EXOR gates. In these designs, EXOR gates with unlimited fun-in are used. However, in most technologies, EXOR gates with many inputs are expensive.

In this paper, we present a design method for AND-OR-EXOR three-level networks. The network realizes an exclusive-OR of two sum-of-products expressions (EX-SOP). Here, only a single two-input EXOR gate is used. Such a network is shown in Fig. 1.1. An EX-SOP of a function f can be written as $F = G \oplus H$, where G and H are SOPs. Our objective is to minimize total number of products in G and H.

2 Definitions and Basic Properties

Definition 2.1 *$\tau(EX\text{-}SOP : F)$ denotes the number of products in F, an EX-SOP for f. $\tau(EX\text{-}SOP : f)$ denotes the total number of products in minimum EX-SOP for f. $\tau(SOP : f)$ denotes the number of products in minimum SOP for f.*

The following theorem is the basis of the minimization of EX-SOPs. We assume that the two SOPs of the EX-SOP do not share products between them.

Theorem 2.1 *Let f be an n-variable function. Let $\mathcal{G}_n$ be the set of all the n-variable functions. Then,*

$$\tau(EX\text{-}SOP : f) = \min_{g \in \mathcal{G}_n} \left\{ \tau(SOP : g) + \tau(SOP : f \oplus g) \right\}. \quad (2.1)$$

(Proof) Suppose that the minimum EX-SOP for f is represented as $f = g \oplus h$. From this, we have $h = f \oplus g$. Since all possible functions g are considered in (2.1), we have the theorem. (Q.E.D.)

Theorem 2.1 shows that for n-variable functions, we have to check 2^{2^n} different g's and take one with the minimum number of product terms. But this search

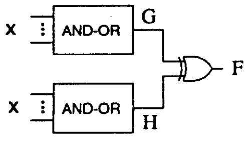

Figure 1.1: AND-OR-EXOR three-level network.

space is very large, even for $n = 5$. The following lemma shows that we can drastically reduce this search space.

Lemma 2.1 *In Theorem 2.1, suppose we need to find an EX-SOP with fewer than p products. If we consider g's so that $\tau(SOP : g)$ are in increasing order, then we have only to consider those g's, such that*

$$\tau(SOP : g) \leq \lceil p/2 - 1 \rceil,$$

where $\lceil k \rceil$ denotes the least integer greater than or equal to k.

(Proof) Suppose we already considered all the g's such that $\tau(SOP : g) \leq \lceil p/2 - 1 \rceil$. Now it is sufficient to prove that a further increase in $\tau(SOP : g)$ by considering other g's cannot produce an EX-SOP F with $\tau(EX\text{-}SOP : F) < p$. We prove this by contradiction. We already considered g's such that $\tau(SOP : g) = 0, 1, \ldots, \lceil p/2 - 1 \rceil$. To obtain $\tau(EX\text{-}SOP : F) < p$, we increase $\tau(SOP : g)$ by 1, i.e., $\tau(SOP : g)$ is now $\lceil p/2 \rceil$. We have $\tau(EX\text{-}SOP : F) = \tau(SOP : g) + \tau(SOP : f \oplus g)$. Therefore, $\tau(SOP : f \oplus g) < \lceil p/2 \rceil$, implies $\tau(SOP : f \oplus g) = \lceil p/2 - 1 \rceil, \ldots, 1$, or 0. But if such an EX-SOP exits, that must be found when we considered $\tau(SOP : g) = 0, 1, \ldots, \lceil p/2 - 1 \rceil$. Thus, $\tau(EX\text{-}SOP : F)$ is not less than p. Similarly, we can show that a further increase in $\tau(SOP : g)$ by considering other g's cannot produce an EX-SOP with fewer products. Hence, we have the lemma. (Q.E.D.)

Example 2.1 *In Theorem 2.1, to find an EX-SOP with fewer than 8 products, we have only to consider those g's such that $\tau(SOP : g) \leq 3$. Similarly to find an EX-SOP with fewer than 9 products, we have only to consider those g's such that $\tau(SOP : g) \leq 4$. (End of Example)*

The Shannon decomposition is stated in the following lemma.

Lemma 2.2 *An n-variable function $f(x_1, x_2, \ldots, x_n)$ can be decomposed into two sub-functions by using the Shannon decomposition, $f = \bar{x}_1 f_0 \vee x_1 f_1$, where $f_0 = f|_{x_1=0}$ and $f_1 = f|_{x_1=1}$.*

Lemma 2.3 *If f and g are disjoint (i.e., $f \cdot g = 0$), then $f(h_{11} \oplus h_{12}) \vee g(h_{21} \oplus h_{22}) = (f h_{11} \vee g h_{21}) \oplus (f h_{12} \vee g h_{22})$.*

(Proof) $f(h_{11} \oplus h_{12}) \vee g(h_{21} \oplus h_{22})$

$$= f(h_{11} \oplus h_{12}) \oplus g(h_{21} \oplus h_{22})$$
$$= (f h_{11} \oplus g h_{21}) \oplus (f h_{12} \oplus g h_{22})$$
$$= (f h_{11} \vee g h_{21}) \oplus (f h_{12} \vee g h_{22}). \quad \text{(Q.E.D.)}$$

Lemma 2.4 *[1] Let $\tau(EX\text{-}SOP : n)$ denote the maximum number of products required to realize an arbitrary n-variable function by a minimum EX-SOP. Then*

$$\tau(EX\text{-}SOP : n) \leq 2\tau(EX\text{-}SOP : n-1).$$

(Proof) An arbitrary n-variable function can be expanded as $f = \bar{x}f_0 \vee xf_1$, where $f_0 = f|_{x=0}$ and $f_1 = f|_{x=1}$. Representing f_0 and f_1 by EX-SOPs, we have an expression F_1 for f.

$$F_1 = \bar{x}(H_{00} \oplus H_{01}) \vee x(H_{10} \oplus H_{11}). \quad (2.2)$$

In (2.2) H_{ij}'s are SOPs. Since f_0 and f_1 are functions of $n-1$ variables, the total number of products in F_1 is at most $2\tau(EX\text{-}SOP : n-1)$.
By applying Lemma 2.3 to (2.2), we have an EX-SOP F of f:

$$F = (\bar{x}H_{00} \vee xH_{10}) \oplus (\bar{x}H_{01} \vee xH_{11}). \quad (2.3)$$

Note that the total number of products in (2.2) and (2.3) are the same. Thus, f can be represented by an EX-SOP with at most $2\tau(EX\text{-}SOP : n-1)$ products. Hence, we have the lemma. (Q.E.D.)

3 Minimization of EX-SOPs with up to Five Variables

In this section, we present algorithms to minimize EX-SOPs with up to five variables. Here, we assume that two SOPs of the EX-SOP do not share products between them. The following is a straightforward algorithm to minimize EX-SOPs with n ($n \leq 5$) variables.

Algorithm 3.1 *(EX-SOP minimization: Straightforward)*

```
1. Let $f$ be the function to be represented as an EX-SOP, and $\mathcal{G}_n$ be the set of all the n-variable functions. $\mathcal{G}_n$ is sorted in ascending order of $\tau(SOP : g)$, where $g \in \mathcal{G}_n$.
2. best shows the minimum number of products in EX-SOPs ever found. sol shows a pair of n-variable functions.
   best ← $\tau(SOP : f)$;  sol ← $(f, 0)$
3. For each $g \in \mathcal{G}_n$ (sequentially from the beginning of $\mathcal{G}_n$) such that $\tau(SOP : g) \leq \lceil$best/2 − 1$\rceil$,
       temp ← $\tau(SOP : g) + \tau(SOP : f \oplus g)$    (3.1)
       If (temp < best) then
           best ← temp    (3.2)
           sol ← $(g, f \oplus g)$
       endif
4. Return best and sol.
```

For up to four-variable functions, Algorithm 3.1 produces solutions very quickly. However, for five-variable functions, it is rather time consuming. In implementing Algorithm 3.1 for five-variable functions, we use the following techniques.

3.1 Strategy for Five-Variable EX-SOP Minimization

The most time consuming part of Algorithm 3.1 is (3.1). In this algorithm, $\mathcal{G}_n$ is sorted in ascending order of $\tau(SOP : g)$ ($g \in \mathcal{G}_n$), and we are considering $\mathcal{G}_n$ sequentially from the beginning. Thus, we can obtain $\tau(SOP : g)$ without much effort. Therefore, the most time consuming part of the algorithm is the computation of $\tau(SOP : f \oplus g)$. To compute $\tau(SOP : f \oplus g)$, we use a cost table instead of doing logic minimization. The total number of five-variable functions is $2^{32} \approx 4.3 \times 10^9$. So, it is impractical to store all the values of $\tau(SOP : f \oplus g)$. Instead, we use the cost table of the NP-representative functions. Fig. 3.1 shows the *cost table* for the NP-representative functions.

The numbers of products in SOPs are invariant under the permutation and/or negation of the input variables. In other words, if $f \overset{NP}{\sim} g$, then $\tau(SOP : f) = \tau(SOP : g)$, where $\overset{NP}{\sim}$ denotes the NP-equivalence relation [6, 4]. The number of NP-equivalence classes of five-variable functions is 1,228,158. We have the cost table of $\tau(SOP : f)$ for all the NP-representative functions f with five variables (Fig. 3.1). However, to get the representative function from a given function is rather time consuming. To speed up the computation, we use $\mu(f)$, the *modified coordinate representation* of f, which will be defined later in this section. $\mu(f)$ is quickly calculated from f, and have the following property.

Property 3.1 *Let f and g be functions. If $f \overset{NP}{\sim} g$, then $\mu(f) = \mu(g)$.*

Among the five-variable functions, there exist functions f and g such that $f \overset{NP}{\not\sim} g$ and $\mu(f) = \mu(g)$. Thus, for some f, $\mu(f)$ corresponds to more than one NP-equivalence class. All the 1,228,158 NP-equivalence classes of five-variable functions have only 149,466 distinct modified coordinate representations. Although $\mu(f)$ cannot uniquely identify the NP-equivalence classes, we can use it to estimate the value of $\tau(SOP : f)$. We use the *modified cost table* storing $p_{low} = \min_{\mu(f)=\mu(g)}\left\{\tau(SOP : g)\right\}$ and $p_{up} = \max_{\mu(f)=\mu(g)}\left\{\tau(SOP : g)\right\}$. Usually, the differences between p_{low} and p_{up} are small, and in many cases we can show that (temp $\geq$ best) in (3.2) in Algorithm 3.1. If there is any possibility that (temp < best), then we use more time consuming routine to obtain the value of $\tau(SOP : f \oplus g)$.

Fig. 3.2 shows the *modified cost table* for the NP-representative functions. In this table, the left part stores the distinct values of $\mu(f)$, and the right parts store the upper and lower bounds on $\tau(SOP : f)$. We can quickly compute the value of $\mu(f)$ for a given f. Because the left part ($\mu(f)$) is the array of 32 integers, we use hash technique to look-up modified cost table.

Definition 3.1 *[2] The coordinate representation of f, $COR(f)$ of a five-variable function consists of 32 integers:*

$$COR(f) = (c_0;\ c_1, c_2, c_3, c_4, c_5;\ c_{12}, c_{13}, \ldots, c_{45};\ c_{123}, c_{124}, \ldots, c_{345};\ c_{1234}, c_{1235}, c_{1245}, c_{1345}, c_{2345};\ c_{12345}),$$

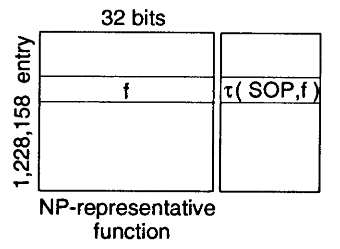

Figure 3.1: Cost table for the NP-representative functions.

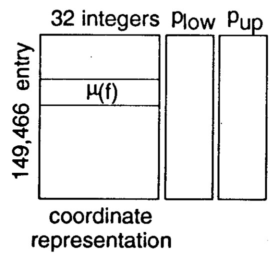

Figure 3.2: Modified cost table for the NP-representative functions.

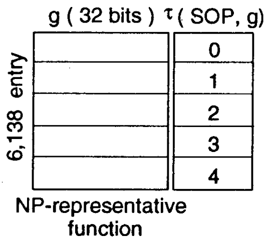

Figure 3.3: Sorted function table.

and is calculated as follows:

$$\begin{aligned} c_0 &= 2^{n-1} - |f|, & \\ c_i &= 2^{n-1} - |f \oplus x_i|, & (i \in L), \\ c_{ij} &= 2^{n-1} - |f \oplus x_i \oplus x_j|, & (i,j \in L), \\ c_{ijk} &= 2^{n-1} - |f \oplus x_i \oplus x_j \oplus x_k|, & (i,j,k \in L), \\ c_{ijkl} &= 2^{n-1} - |f \oplus x_i \oplus x_j \oplus x_k \oplus x_l|, & (i,j,k,l \in L), \\ c_{12345} &= 2^{n-1} - |f \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5|, & \end{aligned}$$

where $|f|$ denotes the number of true minterms of f, $n = 5$, and $L = \{1,2,3,4,5\}$.

Definition 3.2 *The modified coordinate representation of f, $\mu(f)$ of a five-variable function consists of 32 integers, and is calculated from $COR(f)$ as follows:*

$$\mu(f) = (d_0;\ d_1, d_2, d_3, d_4, d_5;\ d_{12}, d_{13}, \ldots, d_{45};\ d_{123}, d_{124}, \ldots, d_{345};\ d_{1234}, d_{1235}, d_{1245}, d_{1345}, d_{2345};\ d_{12345}).$$

$d_0 = c_0$, $d_i (i \in \{1,2,3,4,5\})$ are obtained from c_i by deleting the sign and rearranging in ascending order. d_{ij}, d_{ijk}, and d_{ijkl} are obtained in similar ways. $d_{12345} = |c_{12345}|$.

Theorem 3.1 *$\mu(f)$ is invariant under the permutation and/or complementation of the input variables.*

(Proof) First, we will show that $(d_1, d_2, d_3, d_4, d_5)$ is invariant. By the definition of $COR(f)$, the permutation of the input variable only permutes the values within the group $(c_1, c_2, c_3, c_4, c_5)$. In $\mu(f)$, the values of d_i are rearranged in ascending order. So, $(d_1, d_2, d_3, d_4, d_5)$ is invariant under the permutation of the input variables. Note that,

$$\begin{aligned} c_i = 2^{n-1} - |f \oplus x_i| &= 2^{n-1} - (|\bar{x}_i f| + |x_i \bar{f}|) \\ &= (2^{n-1} - |x_i \bar{f}|) - |\bar{x}_i f| = |x_i f| - |\bar{x}_i f|. \end{aligned}$$

Thus, the complementation of the variable x_i only changes the sign of c_i. But we discard the signs of c_i's. Therefore, $(d_1, d_2, d_3, d_4, d_5)$ is invariant under the complementation of the input variable. For other groups in $\mu(f)$, we can show that the values are invariant under the permutation and complementation of the input variables. (Q.E.D.)

According to Lemma 2.1, we can reduce the total search space by considering g's such that $\tau(SOP : g)$ is in ascending order. For this purpose, we use the *sorted function table* shown in Fig. 3.3. We found that for five-variable functions, maximum value of $\tau(SOP : g)$ is four. Thus, the sorted function table stores only those NP-representative function g's, such that $\tau(SOP : g) \leq 4$. The table is sorted in ascending order of $\tau(SOP : g)$. There are only 6,138 NP-representative functions whose minimum SOPs require four or fewer products.

3.2 Algorithm for Five-Variable EX-SOP Minimization

Algorithm 3.2 *(Minimization of EX-SOPs for five-variable)*

0. *f, g, h, g_{rep}, and h_{rep} represent logic functions, and $p_{g_{rep}}$, p_{exsop}, p_{bound}, p_{low}, p_{up}, and p_{new} represent the number of products.*
1. *Read the cost table, the modified cost table, and the sorted function table for the NP-representative functions.*
2. *Read the function to be minimized. Let it be f.*
3. *p_{exsop} denotes the minimum number of products in EX-SOP ever found. $p_{exsop} \leftarrow 10$. p_{bound} represents the upper bounds on the number of products in the SOP for g. $p_{bound} \leftarrow 4$.*
4. *Take the first representative function g_{rep} from the sorted function table, and $p_{g_{rep}} \leftarrow \tau(SOP : g_{rep})$ from the cost table.*
5. *Generate all the functions of the NP-representative class g_{rep}. Take the first function of this class. Let the function be g.*
6. *$h \leftarrow f \oplus g$.*
7. *Calculate $\mu(h)$. Obtain the upper and the lower bounds on $\tau(SOP : h)$ from the modified cost table. Let them be p_{up} and p_{low}, respectively.*
8. *If $(p_{g_{rep}} + p_{low}) \geq p_{exsop}$, then go to step 13. (Reduction in p_{exsop} is impossible using the current h.)*
9. *If $p_{low} = p_{up}$, then $p_{new} \leftarrow p_{low}$ and go to step 11. (Exact value of $\tau(SOP : h)$ is obtained from the modified cost table.)*
10. *$p_{new} \leftarrow \tau(SOP : h)$. (Using breadth-first search, obtain the NP-representative function h_{rep} of h, then obtain $\tau(SOP : h_{rep})$ from the cost table.)*
11. *If $(p_{g_{rep}} + p_{new}) \geq p_{exsop}$, then go to step 13. (Reduction in p_{exsop} is impossible using the current h.)*

12. *(New solution is found.)* $p_{exsop} \leftarrow p_{g_{rep}} + p_{new}$. *Save* g *and* h *as the latest solution.* $p_{bound} \leftarrow \lceil p_{exsop}/2 \rceil - 1$. *(*$\lceil k \rceil$ *denotes the least integer greater than or equal to* k*.) If* $p_{bound} \leq p_{g_{rep}}$*, then go to step 14.*

13. *Take the next function* g *in the class* g_{rep} *(computed in step 5), and go to step 6. If there is no remaining function in this class, then take the next representative function* g_{rep} *from the sorted function table and* $p_{g_{rep}} \leftarrow \tau(SOP : g_{rep})$ *from the cost table. If* $p_{bound} < p_{g_{rep}}$*, then go to step 14, otherwise go to step 5.*

14. *Print the latest solution saved at step 12, and* p_{exsop} *as the final number of products.*

4 Simplification of EX-SOPs with Six or more Variables

In this section, we present a heuristic algorithm to simplify EX-SOPs with six or more variables. The algorithm is based on the Shannon decomposition and table look-up of minimum EX-SOPs for the representative functions of NP-equivalence classes of five variables.

4.1 A Naive Method

Let f be an n-variable ($n \geq 6$) function. Recursively applying the Shannon decomposition to f, we have

$$f = \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}g_1 \vee \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}x_{n-5}g_2 \vee \bar{x}_1\bar{x}_2\cdots x_{n-6}\bar{x}_{n-5}g_3 \vee \cdots \vee x_1x_2\cdots x_{n-6}x_{n-5}g_N. \quad (4.1)$$

Here g_i's are functions of five variables and $N = 2^{n-5}$ ($n \geq 6$). All the product terms in (4.1) are disjoint. Let $g_i = h_{i1} \oplus h_{i2}$ ($1 \leq i \leq N$). Putting expressions for g_i's into (4.1), we have

$$f = \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}(h_{11} \oplus h_{12}) \vee \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}x_{n-5}(h_{21} \oplus h_{22}) \vee \bar{x}_1\bar{x}_2\cdots x_{n-6}\bar{x}_{n-5}(h_{31} \oplus h_{32}) \vee \cdots \vee x_1x_2\cdots x_{n-6}x_{n-5}(h_{N1} \oplus h_{N2}). \quad (4.2)$$

Now using Lemma 2.3 recursively, we have the following EX-SOP F of f:

$$F = (\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{11} \vee \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}x_{n-5}h_{21} \vee \bar{x}_1\bar{x}_2\cdots x_{n-6}\bar{x}_{n-5}h_{31} \vee \cdots \vee x_1x_2\cdots x_{n-6}x_{n-5}h_{N1}) \oplus (\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{12} \vee \bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}x_{n-5}h_{22} \vee \bar{x}_1\bar{x}_2\cdots x_{n-6}\bar{x}_{n-5}h_{32} \vee \cdots \vee x_1x_2\cdots x_{n-6}x_{n-5}h_{N2}). \quad (4.3)$$

The minimum EX-SOPs for g_i's i.e., h_{ij}'s can be found from the table of minimized EX-SOPs for the representative functions of NP-equivalence classes of five variables.

4.2 An Improved Method

Let us consider a realization of an eight-variable function $f(x_1, \ldots, x_8)$. The decompositions we have used to produce (4.1) are shown in Fig. 4.1 by using a Shannon decomposition tree for f. In this tree, each non-terminal node is labeled by a variable x_i ($1 \leq i \leq 3$), and all the edges are labeled by 0 or 1. The root node represents the function f, and all other nodes represent sub-functions of f. Function represented by a non-terminal node (labeled x_i) is restricted to $x_i = 0(1)$ and is represented by its child node connected to its 0(1) labeled edge.

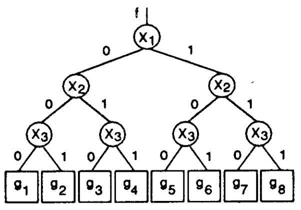

Figure 4.1: Representation of a logic function by using Shannon decompositions.

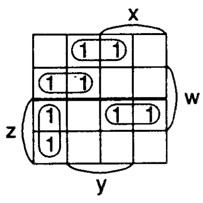

Figure 4.2: Choosing a decomposition variable.

In Fig. 4.1, x_1, x_2 and x_3 are chosen as the decomposition variable at the first, second and third level of the tree, respectively. But, we can choose any variable ($x_1, \ldots, x_8$) at any node of the tree. Suppose we are given a set of cubes, which represent a simplified SOP of a function f. If we choose decomposition variables without considering the shape of the cubes, it often increases the number of cubes. An increase in the number of cubes tends to increase the number of products in the EX-SOPs of the sub-functions. This results an increase in the number of products in the EX-SOP for the function f. Thus, our strategy is to find the decomposition variable at each node of the Shannon decomposition tree for f that increases as few products as possible.

Choosing a Decomposition Variable

Suppose that we have the four-variable function f shown in Fig. 4.2. We want to decompose it into two sub-functions by using the Shannon decomposition. We can do this in four different ways, corresponding to four variables of f. If we decompose f with respect to either x, y or w, then the number of products is increased by one in each case. But, if we choose z as the decomposition variable, then the number of products remains the same. It is shown in Fig. 4.2 by the thick line. The thick line divides the Karnaugh map into two parts without splitting any loop. The resulting expression for f is $\bar{z}(y\bar{w} \vee \bar{x}w) \vee z(\bar{x}\bar{y} \vee xw)$.

It is not always possible to find a decomposition variable which do not increase any product. However, we can choose the decomposition variable that increases as few products as possible.

Effect of REDUCE in Decomposition

A four-variable function f is shown in Fig. 4.3(a). EXPAND [3] operations have already been done on the cubes of this figure. Suppose that we want to decompose f into two sub-functions by using the Shannon decomposition. There are four ways to decompose f. But,

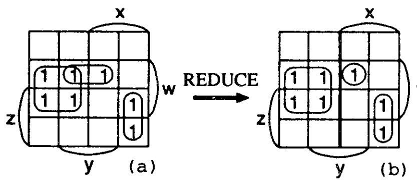

Figure 4.3: Effect of REDUCE in decomposition.

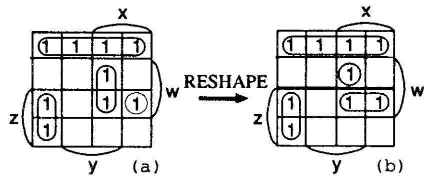

Figure 4.4: Effect of RESHAPE in decomposition.

in all the cases the number of cubes would be increased by one. We perform the REDUCE operation [3] on the cubes of Fig. 4.3(a), and get Fig. 4.3(b). Now we can decompose f into two sub-functions without increasing the number of products. We choose x as the decomposition variable. The two sub-functions of f are separated by the thick line on the Karnaugh map of Fig. 4.3(b).

Effect of RESHAPE in Decomposition

A four-variable function f is shown in Fig. 4.4(a). We want to decompose f into two sub-functions by using Shannon decomposition. But we cannot do this without splitting a cube. By doing RESHAPE [3] operation on the two cubes of Fig. 4.4(a), we have Fig. 4.4(b). Now we can decompose f without splitting any cube. Here, z is the decomposition variable. The thick line on Fig. 4.4(b) shows this decomposition. RESHAPE is very useful to find a good decomposition variable. On a set of cubes we can do RESHAPE operation in different ways until we find a variable which split as few cubes as possible.

Effect of REDUCE in Decomposition of Sub-Functions

A four-variable function f is shown in Fig. 4.5(a). REDUCE and RESHAPE are inapplicable to the cubes of this figure. In Fig. 4.5, 'DECOMP' indicates decomposition. We decompose f into two sub-functions by splitting one cube. The resulting sub-functions are shown in Fig. 4.5(b). Now we want to decompose each sub-function again. RESHAPE is inapplicable to the cubes of the sub-functions. We REDUCE the cubes, and obtain Fig. 4.5(c). Decomposing the sub-functions in Fig. 4.5(c), we have Fig. 4.5(d). From Fig. 4.5(b) to Fig. 4.5(d) the number of cubes remain same. But, if

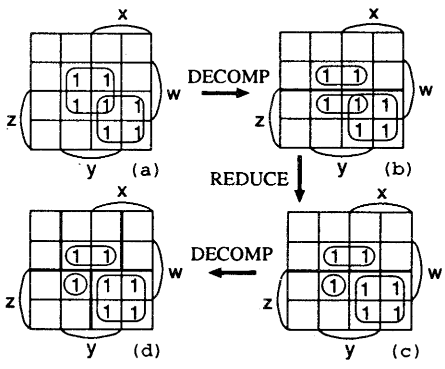

Figure 4.5: Effect of REDUCE in decomposition of sub-functions.

we decompose the sub-functions in Fig. 4.5(b) without performing REDUCE operation, the number of cubes would increase. Thus, when a decomposition increase the number of cubes, a REDUCE operation on the cubes of the sub-functions may help to find a good decomposition variable of the sub-functions.

4.3 Combining the Sub-Solutions

To obtain (4.3) from (4.2), for each pair of products correspond to g_i's, we can choose their positions in two different ways. Such as, for $\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{11}$ and $\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{12}$. We can choose $\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{11}$ in the first SOP and $\bar{x}_1\bar{x}_2\cdots\bar{x}_{n-6}\bar{x}_{n-5}h_{12}$ in the second SOP (as shown in (4.3)), or vice versa. There are N pair of products for an n-variable function, where $N = 2^{n-5}$ ($n \geq 6$). We can form many different EX-SOPs from (4.2). Thus, some heuristic algorithm is necessary to form an EX-SOP of the given function from the EX-SOPs of the sub-functions.

4.4 Further Improvement

Each of the SOP of the EX-SOP can be simplified again. We can simplify two SOPs simultaneously, so that they can share products between them. Further simplification of the EX-SOP may possible by using some heuristic algorithm [10].

4.5 Heuristic Algorithm

Algorithm 4.1 *(Simplification of EX-SOPs with n ($n \geq 6$) variables)*

1. *Accept a simplified or minimized SOP of the given function f (in the cube form).*
2. *Decompose f into 2^{n-5} sub-functions, using the procedure* DecompFunc(f,YES) *shown in Fig. 4.6. Let the sub-functions be g_i ($1 \leq i \leq 2^{n-5}$).*
3. *For all g_i's obtain the minimum EX-SOPs ($g_i = h_{i1} \oplus h_{i2}$) using the procedure* Find5varEX-SOP(g) *shown in Fig. 4.7.*

```
DecompFunc($f$,reduceFlag){
    if reduceFlag = YES
        REDUCE the cubes of $f$;
    do RESHAPE until
        a decomposition variable is found;
    decompose $f$ into sub-functions $f_0$ and $f_1$;
    if # of cubes is increased in above decomposition
        reduceFlag ← YES;
    else
        reduceFlag ← NO;
    if $f_0$ and $f_1$ depend on more than five variables{
        DecompFunc($f_0$,reduceFlag);
        DecompFunc($f_1$,reduceFlag);
    }
}
```

Figure 4.6: Procedure DecompFunc(f,reduceFlag).

Table 5.1: Number of four-variable functions requiring t products.

t	SOP	ESOP	EX-SOP
0	1	1	1
1	81	81	81
2	1804	2268	2316
3	13472	21744	22896
4	28904	37530	37634
5	17032	3888	2608
6	3704	24	
7	512		
8	26		
av	4.13	3.66	3.62

av : average

4. *Form an EX-SOP of f by combining the EX-SOPs of g_i's.*

5. *Simultaneously minimize both SOPs of the EX-SOP, so that they can share products between them.*

6. *Output the simplified EX-SOP.*

5 Experimental Results

Using our EX-SOP minimization program, we minimized all the 1,228,158 representative functions of NP-equivalence classes of five variables. For five-variable functions, when the number of products in the minimum EX-SOP is 9 (worst case), we could minimize it within 38 CPU seconds on a DEC ALPHASTATION 200. When the number of products in the minimum EX-SOP is 6 for the five-variable functions, the computation time was less than 3 CPU seconds on the same machine. The program can run within 25 megabytes of memory space. Although the minimization program is not so time consuming for each function, we spent nearly two months of computation time by using several workstations to minimize over 1.2 million representative functions.

```
Find5varEX-SOP($g$){
  $g_{rep}$ ← NPrep($g$);
    // NPrep($g$) returns NP-representative function of $g$
    // let revNPrep($g$) is the reverse transform of NPrep($g$)
    // i.e., $g$ = revNPrep($g_{rep}$)
  ($h_{rep1}, h_{rep2}$) ← minimum EX-SOP of $g_{rep}$;
    // here $g_{rep} = h_{rep1} \oplus h_{rep2}$
    // use the table of minimized EX-SOPs for
    // the representative functions of the
    // NP-equivalence classes of five variables
  $h_1$ ← revNPrep($h_{rep1}$);
  $h_2$ ← revNPrep($h_{rep2}$);
  return $h_1$ and $h_2$;
    // here $g = h_1 \oplus h_2$
}
```

Figure 4.7: Procedure Find5varEX-SOP(g).

Table 5.2: Number of five-variable functions requiring t products.

t	SOP	ESOP	EX-SOP
0	1	1	1
1	243	243	243
2	20676	24948	25988
3	818080	1351836	1511996
4	16049780	39365190	47838990
5	154729080	545193342	694830748
6	698983656	2398267764	2678055614
7	1397400512	1299295404	870943300
8	1254064246	11460744	1760384
9	571481516	7824	32
10	160200992		
11	34140992		
12	6160176		
13	827120		
14	84800		
15	5312		
16	114		
av	7.46	6.16	6.02

av : average

Table 5.1 and 5.2 show the number of four and five-variable functions requiring t products by different expressions, respectively.[1] In these tables, data for SOPs and ESOPs are taken from [9]. EX-SOPs were minimized by Algorithm 3.1 and 3.2, respectively. For five-variable functions, on the average, EX-SOPs require 6.02 products while SOPs require 7.46 products. For the same five-variable functions, on the average, EX-SOPs require fewer products than ESOPs. We found that for four and five-variable functions, the upper bounds on the number of products in minimum EX-SOPs (when two SOPs do not share products) are 5 and 9, respectively. Thus, minimum EX-SOPs with n variables ($n \geq 6$) require at most $9 \cdot 2^{n-5}$ products (Lemma 2.4).

Table 5.3 compares the average number of products required to realize randomly generated functions. For each value of n, 100 n-variable pseudo-random functions with the specified number of minterms were generated and minimized. In this experiment, SOPs were minimized by the Quine-McCluskey method [9], and ESOPs were simplified by EXMIN2 [8]. Data for the column headed 'Ref.[10]' were generated by the program in [10]. Data for the rightmost column of this table were simplified by Algorithm 4.1.

Table 5.4 compares the number of products required by different expressions to realize some six-variable

[1]In Table 5.1 and 5.2, av = $\left(\sum_t (t \times \text{number of functions requiring } t \text{ products})\right)$ / total number of functions. For n variables, total number of functions is 2^{2^n}.

Table 5.3: Average number of products for randomly generated functions.

% of true minterms	n	SOP	ESOP	EX-SOP	
				Ref.[10]	This
25.00	6	10.13	8.68	8.71	8.96
	7	19.00	15.89	16.33	17.88
	8	36.02	29.02	30.96	33.22
	9	69.78	55.58	62.12	66.60
37.50	6	12.46	9.96	10.43	10.40
	7	22.98	18.15	20.25	20.35
	8	44.17	33.75	40.34	40.13
	9	82.87	64.74	79.28	77.88
50.00	6	13.82	10.92	11.86	11.79
	7	25.34	19.38	22.63	22.56
	8	46.96	35.90	44.56	43.21
	9	88.76	69.27	85.66	82.67
62.50	6	14.17	10.76	12.17	11.94
	7	25.17	19.24	22.84	22.65
	8	46.90	35.88	43.82	43.06
	9	86.63	68.61	82.82	81.00
75.00	6	12.85	9.66	10.88	11.11
	7	23.05	16.64	19.71	20.84
	8	42.03	30.10	37.14	39.29
	9	77.15	57.05	69.82	74.46

% of true minterms : (# of true minterms / 2^n) x 100
n : # of variables

Table 5.4: Number of products for some six-variable functions.

Most complex	Function truth table in hexadecimal	Number of products		
		SOP	ESOP	EX-SOP
ESOP	6bbdbdd6bdd6d66b	27	15	14
	7ee9e997e997977e	30	15	12
SOP	6996966996696996	32	6	8

functions. The first two functions are taken from [5]. Among all the six-variable functions, these functions have the most complex ESOP representations. The last function in this table is a parity function. Among all the six-variable functions, it has the most complex SOP representation.

6 Conclusions

In this paper, we presented minimization algorithms for AND-OR-EXOR three-level networks (EX-SOPs) for up to five-variable functions. A heuristic algorithm is also presented to simplify EX-SOPs with six and more variables. We minimized all the 1,228,158 representative functions of NP-equivalence classes of five variables. We have completed the table of minimum EX-SOPs with up to five variables. We showed that for five-variable functions, the upper bound on the number of products in minimum EX-SOPs is 9, and that for n-variable functions is at most $9 \cdot 2^{n-5}$, when $n \geq 6$. Previously, this upper bound was known to be $5 \cdot 2^{n-4}$ for $n \geq 4$ [1]. Again, this upper bound is smaller than 2^{n-1}, the upper bound for the minimum SOPs. Also, we found that for five-variable functions, on the average, EX-SOPs require 6.02 products while SOPs require 7.46 products. Our heuristic simplification algorithm is based on function decomposition, and table look-up of minimum EX-SOPs for the NP-representative functions of five variables. For some functions, the algorithm produce comparable solutions to that of [10]. In our minimization algorithms, we did not considered the sharing of products between two SOPs of an EX-SOP. If we consider sharing of products, we can realize EX-SOPs with fewer products for many functions. Currently we are developing programs to minimize EX-SOPs considering product sharing.

Acknowledgements

This work was supported in part by a Grant in Aid for the Scientific Research of the Ministry of Education, Science and Culture of Japan. The authors are thankful to Prof. N. Koda, who provided the table of minimized SOPs for the representative functions.

References

[1] 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.

[2] M. A. Harrison, *Introduction to Switching and Automata Theory*, McGraw-Hill Book Company, 1965.

[3] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. & Develop.*, pp. 443-458, Sept. 1974.

[4] S. L. Hurst, D. M. Miller and J. C. Muzio, *Spectral Techniques in Digital Logic*, Academic Press Inc., 1985.

[5] 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.

[6] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, New York, 1979.

[7] 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.

[8] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, No. 5, pp. 621-632, May 1993.

[9] T. Sasao, "AND-EXOR expressions and their optimization," (Sasao ed.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[10] T. Sasao, "A design method for AND-OR-EXOR three-level networks," *Proc. Int. Workshop on Logic Synthesis,* Lake Tahoe, May 1995.

[11] K. Shu, H. Yasuura and S. Yajima, "Optimization of PLDs with output parity gates," (in Japanese) *National Convention, Information Processing Society of Japan*, March 1985.

[12] N. Song and M. A. Perkowski, "EXORCISM-MV-2: Minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions," *Proc. 23rd ISMVL*, pp. 132-137, May 1993.