An Optimization of AND-OR-EXOR Three-Level Networks

Debatosh Debnath and Tsutomu Sasao
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820, Japan

Abstract — **In this paper, we present 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), where the two sum-of-products expressions (SOP) cannot share products. The problem is to minimize the total number of product in the two SOPs. We introduced the μ-equivalence of logic functions to develop minimization algorithms for EX-SOPs with up to five variables. We minimized all the representative functions of NP-equivalence classes for up to five variables and found that five-variable functions require up to 9 products in minimum EX-SOPs. For n-variable functions, minimum EX-SOPs require at most $9 \cdot 2^{n-5}$ $(n \geq 6)$ products. This upper bound is smaller than 2^{n-1}, the upper bound for the conventional sum-of-products expressions.**

Index Terms — **Three-level network, AND-EXOR, logic minimization, spectral method, NP-equivalence, μ-equivalence, coordinate representation, complexity.**

I. 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 [5, 11, 12, 15]. 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) [13]. To realize an arbitrary function of six variables, minimum SOPs require up to 32 products, while minimum ESOPs require up to 15 products [8]. These reveal the advantages of designing logic networks using EXOR gates. In these designs, EXOR gates with unlimited fan-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), where only a single two-input EXOR gate is used. Such a network is shown in Fig. 1. An EX-SOP for a function f can be written as $F = G \oplus H$, where G and H are SOPs. Our objective is to reduce the total number of products in G and H, where G and H *cannot* share products.

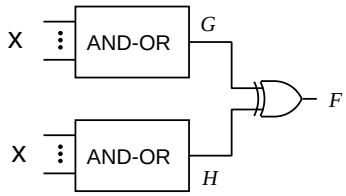

Fig. 1. AND-OR-EXOR three-level network.

AND-OR-EXOR three-level network is suitable for implementing arithmetic functions. For example, the Texas Instruments SN181 arithmetic circuit has EXOR gates in the outputs [18]. Programmable logic arrays (PLAs) with two-input EXOR gates in the outputs efficiently realize adders [17]. AND-OR-EXOR is one of the simplest architecture, since it contains only a single two-input EXOR gate. However, its logic capability is quite high. Because of these, various PLAs with two-input EXOR gates in the outputs were developed. Especially, RICOH, Lattice and AMD (MMI) produced series of such PLAs.

Design methods for AND-OR-EXOR three-level networks were considered in the past [4, 16], but no practical algorithm was reported. A cut-and-try method was reported in [10] and several heuristic algorithms to simplify EX-SOPs were presented in [14]. Upper bounds on the number of products in AND-OR-EXOR expansion was reported in [3].

The paper is organized as follows: Section II introduces first the terminologies used and develops the concept of μ-equivalence of logic functions. Section III presents the key idea for the minimization, a technique to reduce the search space, and an algorithm to minimize EX-SOPs. Section IV shows how the μ-equivalence of logic functions

ASP-DAC '97
0-89791-851-7$5.00

can be used to reduce the computation time for EX-SOPs and presents a minimization algorithm for EX-SOPs with five variables. Section V outlines an idea to simplify EX-SOPs with six or more variables. Section VI shows the experimental results. The last section presents conclusions and comments.

II. Definitions and Basic Properties

In this section, we introduce first several notations and define the NP-equivalence classes of functions. We consider then the modified coordinate representation of functions. By using this representation, we develop the concept of μ-equivalence of logic functions and illustrate its properties, which would be useful in the minimization of EX-SOPs with five variables. In this paper, we distinguish functions and their expressions. We use lower case letters, such as f, g, h, to represent functions, and upper case letters, such as F, G, H, to represent expressions of function.

Definition 2.1: *$\tau(F)$ denotes the number of products in an expression F. An expression F is said to be minimum when $\tau(F)$ is minimum.*

Definition 2.2: *$\tau(EX\text{-}SOP : f)$ denotes the total number of products in a minimum EX-SOP for f. $\tau(SOP : f)$ denotes the number of products in a minimum SOP for f.*

Let a function f be represented as follows:

$$f = g \oplus h. \tag{2.1}$$

Note that g and h correspond to G and H in Fig. 1, respectively. To compute $\tau(EX\text{-}SOP : f)$, we have to choose g and h in different ways so that they satisfy (2.1). Thus, we have $\tau(EX\text{-}SOP : f) = \min\Big\{\tau(SOP : g) + \tau(SOP : h)\Big\}$.

Definition 2.3: *$\tau(EX\text{-}SOP : F)$ denotes the number of products in an EX-SOP F.*

Logic functions can be grouped into classes by using simple transformations.

Definition 2.4: *The set of functions which are identical under (a) the permutation of the variables and/or (b) the complementation (negation) of one or more variables are called NP-equivalent functions [6, 7, 9]. $f \overset{\text{NP}}{\sim} g$ denotes: f and g are NP-equivalent, and $f \overset{\text{NP}}{\not\sim} g$ denotes: f and g are not NP-equivalent.*

For the NP-equivalent functions we have the following:

Property 2.1: *If $f \overset{\text{NP}}{\sim} g$, then $\tau(SOP : f) = \tau(SOP : g)$ and $\tau(EX\text{-}SOP : f) = \tau(EX\text{-}SOP : g)$.*

[1] The proof of the theorems can be found from the authors.

2.1. Modified Coordinate Representation and μ-Equivalence Classes

In the following, we first show a new representation of logic functions, called *modified coordinate representation*, and introduce a novel equivalence relation, called *μ-equivalence* of logic functions. We present then properties of modified coordinate representation and μ-equivalence classes.

Definition 2.5: *$w(f)$ denotes the number of true minterms of the function f.*

Definition 2.6: *[6] The coordinate representation of f, $COR(f)$ of a five-variable function consists of 32 integers:* $COR(f) = (c_0;\ c_1, c_2, c_3, c_4, c_5;\ c_{12}, c_{13}, c_{14}, c_{15}, c_{23}, c_{24}, c_{25}, c_{34}, c_{35}, c_{45};\ c_{123}, c_{124}, c_{125}, c_{134}, c_{135}, c_{145}, c_{234}, c_{235}, c_{245}, c_{345};\ c_{1234}, c_{1235}, c_{1245}, c_{1345}, c_{2345};\ c_{12345})$.

And c's are calculated as follows:

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

where $n = 5$ and $L = \{1, 2, 3, 4, 5\}$. The elements of $COR(f)$ which are separated by ';' (semicolon) form a group.

Definition 2.7: *The modified coordinate representation of f, $\mu(f)$ of a five-variable function consists of 32 integers:* $\mu(f) = (d_0;\ d_1, d_2, d_3, d_4, d_5;\ d_{12}, d_{13}, d_{14}, d_{15}, d_{23}, d_{24}, d_{25}, d_{34}, d_{35}, d_{45};\ d_{123}, d_{124}, d_{125}, d_{134}, d_{135}, d_{145}, d_{234}, d_{235}, d_{245}, d_{345};\ d_{1234}, d_{1235}, d_{1245}, d_{1345}, d_{2345};\ d_{12345})$.

And is calculated from $COR(f)$ as follows: (a) $d_0 = c_0$; (b) d_i ($i \in \{1, 2, 3, 4, 5\}$) are obtained from c_i by deleting the sign and then rearranging the elements of the group in ascending order; d_{ij}, d_{ijk} and $d_{ijk\ell}$ ($i, j, k, \ell \in \{1, 2, 3, 4, 5\}$) are obtained in similar ways; (c) d_{12345} is obtained by deleting the sign of c_{12345}.

Theorem 2.1: *$\mu(f)$ is invariant under (a) the permutation of the variables and/or (b) the complementation of one or more variables.*

Definition 2.8: *Two functions f and g are μ-equivalent, denoted by $f \overset{\mu}{\sim} g$, if and only if, they have the same modified coordinate representation.*

From the definitions of the NP-equivalent and μ-equivalent functions, and by Theorem 2.1, we have the following:

Property 2.2: *If $f \overset{\text{NP}}{\sim} g$, then $f \overset{\mu}{\sim} g$.*

Note that the converse of Property 2.2 is not always true.

Observation 2.1: *Among the five-variable functions, there exist functions f and g such that $f \overset{\mu}{\sim} g$ but $f \overset{\text{NP}}{\not\sim} g$.*

III. Minimization of EX-SOPs

In this section, we develop an algorithm to minimize EX-SOPs, where the two SOPs of the EX-SOP cannot share products.

3.1. Idea for Minimization

The following theorem is the basis of the minimization of EX-SOPs.

Theorem 3.1: *Let f be an n-variable function and $\mathcal{G}_n$ be the set of all the n-variable functions. Then,*

$$\tau(EX\text{-}SOP{:}f)=\min_{g\in\mathcal{G}_n}\Big\{\tau(SOP{:}g)+\tau(SOP{:}f\oplus g)\Big\}. \quad (3.1)$$

3.2. Reduction of Search Space

Theorem 3.1 shows that for the given n-variable function f, we have to check 2^{2^n} different g's, and choose g that produce a minimum value for $\tau(SOP : g) + \tau(SOP : f \oplus g)$. This search space is very large, even for $n = 5$. The following lemma shows that we can drastically reduce this search space.

Theorem 3.2: *In Theorem 3.1, suppose we need to find an EX-SOP with fewer than t 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*

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

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

3.3. Minimization Algorithm: Straightforward

Based on Theorem 3.1, the following is a straightforward algorithm to minimize EX-SOPs:

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

1. *Let f be the n-variable 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 arranged 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` $\leftarrow \tau(SOP : f)$; `sol` $\leftarrow (f, 0)$;
3. *For each $g \in \mathcal{G}_n$ (sequentially from the beginning of $\mathcal{G}_n$) such that $\tau(SOP : g) \leq \lceil$* `best` $/2 - 1\rceil$* **do***
   `temp` $\leftarrow \tau(SOP : g) + \tau(SOP : f \oplus g)$;
   if `temp` $<$ `best` **then**
   `best` $\leftarrow$ `temp`; `sol` $\leftarrow (g, f \oplus g)$;
   endif
   repeat
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. The computation techniques for the five-variable functions are considered in the next section.

IV. Strategy for Five-Variable EX-SOP Minimization

In this section, we first develop several techniques to reduce the computation time for five-variable functions. We then present a new algorithm to minimize EX-SOPs with five variables. The most time consuming part of Algorithm 3.1 is the computation of $\tau(SOP : g)+\tau(SOP : f\oplus g)$ in step 3. The techniques we used to obtain $\tau(SOP : g)$ and $\tau(SOP : f \oplus g)$ are different.

4.1. Obtaining $\tau(SOP : g)$ and Elimination of Redundant Computation

We can quickly obtain $\tau(SOP : g)$ in Algorithm 3.1, by using a table of $g \in \mathcal{G}_n$ and the corresponding $\tau(SOP : g)$. According to Theorem 3.2, we can reduce the number of g's for Theorem 3.1 by considering them (g's) such that $\tau(SOP : g)$ is in ascending order. We found that for five-variable functions, maximum value of $\tau(SOP : g)$ is four. For five-variable, the number of g's such that $\tau(SOP : g) \leq 4$ is 16,888,780, and it is inconvenient to work with a table of this size. To reduce the table size, we use NP-equivalence classes. From Property 2.1, the number of products in minimum SOPs for the NP-equivalent functions are equal. Every NP-equivalence class has an NP-representative function. There are only 6,138 NP-representative functions whose minimum SOPs require up to four products. Thus, we use the *sorted function table* shown in Fig. 2. The left column of the sorted function table stores only those NP-representative function g_{rep}'s, such that $\tau(SOP : g_{rep}) \leq 4$, and the right column stores the corresponding $\tau(SOP : g_{rep})$. The data in this table is arranged in ascending order of $\tau(SOP : g_{rep})$. We access the sorted function table sequentially from the beginning to get an NP-representative function g_{rep} and the corresponding $\tau(SOP : g_{rep})$. We then obtain g's by generating all the function of the class g_{rep}. Since, $\tau(SOP : g_{rep})$ and $\tau(SOP : g)$ are equal, we obtain $\tau(SOP : g)$ without much effort.

4.2. Computation of $\tau(SOP : f \oplus g)$: The Time Consuming Part

We have shown in Section 4.1 that $\tau(SOP : g)$ can be quickly obtained from the sorted function table. Thus, the most time consuming part of Algorithm 3.1 is the

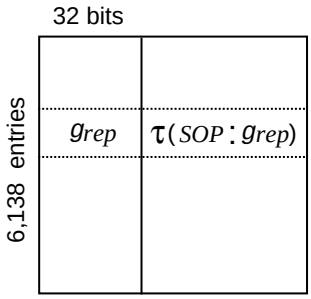

Fig. 2. NP-representative functions with $\tau(SOP : g_{rep}) \leq 4$.

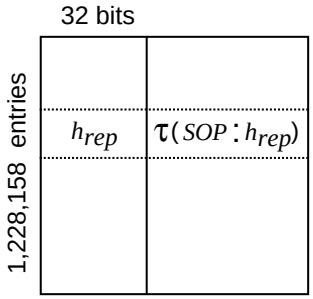

Fig. 3. NP-representative functions.

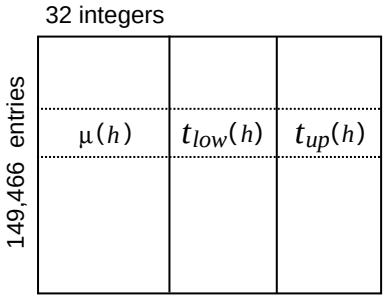

Fig. 4. μ-representative functions.

computation of $\tau(SOP : f \oplus g)$. A straightforward computation of $\tau(SOP : f \oplus g)$ is time consuming. Thus, instead of doing logic minimization, we use a table of all the five-variable functions h and the corresponding $\tau(SOP : h)$. But, the total number of five-variable functions is $2^{32} \approx 4.3 \times 10^9$, and it is impractical to store a table of this size.

4.2.1. Reduction of Table Size: Use of NP-Equivalence Classes

To reduce the table size, we use a cost table for all the NP-representative functions of five variables. The number of NP-equivalence classes of five-variable functions is 1,228,158. Fig. 3 shows the *cost table*. The left column of the cost table stores all the NP-representative functions h_{rep} and the right column stores the corresponding $\tau(SOP : h_{rep})$. We arranged the data in the left column of the cost table, which helps to quickly locate the position of an NP-representative function in the table. Note that the data in the sorted function table is a subset of the data in the cost table. Also the arrangements of data in the two tables are different. For a given function h_{given}, to obtain $\tau(SOP : h_{given})$ from the cost table, first we compute the NP-representative function h_{rep} of h_{given}. Since the number of products is invariant under the NP-equivalence class, we have $\tau(SOP : h_{given}) = \tau(SOP : h_{rep})$. Using h_{rep} for the cost table look-up, we obtain $\tau(SOP : h_{given})$. However, to get the NP-representative function h_{rep} from a given function h_{given} is still rather time consuming.

4.2.2. Quick Estimation of $\tau(SOP : f \oplus g)$: Use of μ-Equivalence Classes

To speed-up the computation, we use μ-equivalence classes, which we have already introduced in Section 2.1. $\mu(h)$, the modified coordinate representation of h, is quickly calculated from h and have Property 2.2. But, Observation 2.1 shows that $\mu(h)$ corresponds to more than one NP-equivalence classes for some h. All the 1,228,158 NP-equivalence classes of five-variable functions can be partitioned into 149,466 μ-equivalence classes. Although μ-equivalence classes cannot uniquely identify the NP-equivalence classes, we can use them to quickly estimate the value of $\tau(SOP : f \oplus g)$ in step 3 of Algorithm 3.1. Thus, we use the *modified cost table*, which is shown in Fig. 4. The left column of the modified cost table stores the distinct values of $\mu(h)$ for all the five-variable function h. The middle and the right columns store the corresponding values of $t_{\ell ow}(h) = \min_{h \overset{\mu}{\sim} h_j} \{\tau(SOP : h_j)\}$ and $t_{up}(h) = \max_{h \overset{\mu}{\sim} h_j} \{\tau(SOP : h_j)\}$, respectively.

Usually the differences between $t_{\ell ow}(h)$ and $t_{up}(h)$ are small. Thus, without computing $\tau(SOP : f \oplus g)$ and using the modified cost table, very often we can show that $\tau(SOP : g) + t_{\ell ow}(f \oplus g) \geq$ `best`, i.e., `temp` $\geq$ `best` in step 3 of Algorithm 3.1. This implies that using the modified cost table, very often we can avoid the computation of $\tau(SOP : f \oplus g)$. If we have $\tau(SOP : g) + t_{\ell ow}(f \oplus g) <$ `best`, i.e., a possibility that `temp` < `best` in step 3 of Algorithm 3.1, only then we compute $\tau(SOP : f \oplus g)$. In many cases $t_{\ell ow}(f \oplus g)$ and $t_{up}(f \oplus g)$ are equal, and we get $\tau(SOP : f \oplus g)$ from the modified cost table. When $t_{\ell ow}(f \oplus g)$ and $t_{up}(f \oplus g)$ are unequal, we obtain $\tau(SOP : f \oplus g)$ from the cost table by using more time consuming routine. Since $\mu(h)$ is an array of 32 integers, we use hash technique to look-up modified cost table.

4.3. Algorithm for Five-Variable EX-SOP Minimization

Based on the above discussions and Theorem 3.1, an algorithm for the minimization of EX-SOPs with five variables is presented in the following:

Algorithm 4.1: *(Minimization of EX-SOPs for five-variable)*

0. f, g, h and g_{rep} represent functions, and t_g, t_{exsop}, t_{bound}, $t_{\ell ow}$, t_{up} and t_h represent the number of products. Read the cost table, the modified cost table and the sorted function table. Read the function to be minimized. Let it be f.

1. *t_{exsop} denotes the minimum number of products in EX-SOP ever found. Let $t_{exsop} \leftarrow 10$. t_{bound} represents the upper bound on the number of products in the SOP for g. Let $t_{bound} \leftarrow 4$.*
2. *From the sorted function table: (a) take the first NP-representative function g_{rep} and (b) obtain $t_g \leftarrow \tau(SOP : g_{rep})$.*
3. *Generate all the functions of the NP-equivalence class g_{rep}. Take the first function of this class. Let the function be g.*
4. *$h \leftarrow f \oplus g$.*
5. *Calculate $\mu(h)$. By using $\mu(h)$, obtain $t_{\ell ow} \leftarrow t_{\ell ow}(h)$ and $t_{up} \leftarrow t_{up}(h)$ from the modified cost table.*
6. *If $t_g + t_{\ell ow} \geq t_{exsop}$ (reduction of t_{exsop} is impossible using the current h), then go to step 11.*
7. *If $t_{\ell ow} = t_{up}$ ($\tau(SOP : h)$ is obtained from the modified cost table), then $t_h \leftarrow t_{\ell ow}$ and go to step 9.*
8. *Obtain $t_h \leftarrow \tau(SOP : h)$ from the cost table.*
9. *If $t_g + t_h \geq t_{exsop}$ (reduction of t_{exsop} is impossible using the current h), then go to step 11.*
10. *(New solution is found.) $t_{exsop} \leftarrow t_g + t_h$. Save g and h as the latest solution. $t_{bound} \leftarrow \lceil t_{exsop}/2 \rceil - 1$. If $t_{bound} \leq t_g$, then go to step 12.*
11. *Take the next function g in the class g_{rep} (computed in step 3), and go to step 4. If there is no remaining function in this class, then from the sorted function table: (a) take the next NP-representative function g_{rep} and (b) obtain $t_g \leftarrow \tau(SOP : g_{rep})$. If $t_{bound} < t_g$, then go to step 12, otherwise go to step 3.*
12. *Print the latest solution saved in step 10, and t_{exsop} as the final number of products.*

The execution time of Algorithm 4.1 mainly depends on t_{bound}, the upper bound on the number of products in the SOP for g of Theorem 3.1. In Algorithm 4.1, t_{bound} is defined in step 1 and it is updated in step 10.

V. Simplification of EX-SOPs with Six or More Variables

A simplified EX-SOP for the n-variable function can be obtained by using a pair of EX-SOPs for the $(n-1)$-variable functions. The idea for this simplification is shown in the following.

Theorem 5.1: *[3] 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*

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

TABLE I
Numbers 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

Based on the idea presented in this section, a heuristic algorithm to simplify EX-SOPs with six or more variables has been developed [1]. The algorithm uses a table of minimum EX-SOPs for the representative functions of NP-equivalence classes of five variables.

VI. Experimental Results

Using our EX-SOP minimization program, we minimized all the 1,228,158 representative functions of NP-equivalence classes of five variables. A five-variable function, on the average, took 9.54 CPU seconds on a DEC AlphaStation 200. This average is obtained by minimizing 10,000 randomly generated functions with 16 true minterms. For five-variable functions, when the number of products in the minimum EX-SOP is 6 and 9 (worst case), we could minimize each function within 3 and 38 CPU seconds, respectively, on the same machine.

Table I shows the numbers of five-variable functions requiring t products by different minimum expressions.[1] In this table, data for SOPs and ESOPs are taken from [13]. EX-SOPs were minimized by Algorithm 4.1. For five-variable functions, on the average, minimum EX-SOPs require 6.02 products while minimum SOPs require 7.46 products. For the five-variable functions, on the average, minimum EX-SOPs require fewer products than minimum ESOPs. We found that for four and five-variable

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

functions, the upper bounds on the number of products in minimum EX-SOPs (when two SOPs cannot share products) are 5 and 9, respectively. Thus, from Theorem 5.1, minimum EX-SOPs with n variables require at most $9 \cdot 2^{n-5}$ $(n \geq 6)$ products. For five-variable functions, there is only one NP-equivalence class whose minimum EX-SOP requires 9 products. The NP-representative function of this class is $\texttt{177e7ee9}_{16}$. In this class, only 32 functions are equivalent.

VII. Conclusions and Comments

In this paper, we presented minimization algorithms for AND-OR-EXOR three-level networks (EX-SOPs) for up to five variables, where two SOPs of the EX-SOP cannot share products. We developed the concept of μ-equivalence of logic functions and used it to reduce the computation time of the minimization program. 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 and showed that minimum EX-SOPs for five-variable functions require up to 9 products. For n-variable functions, the upper bound on the number of products in minimum EX-SOPs is at most $9 \cdot 2^{n-5}$ $(n \geq 6)$. This is tighter than the previously known one: $5 \cdot 2^{n-4} (n \geq 4)$ [3]. This upper bound is smaller than 2^{n-1}, the upper bound for the minimum SOPs. We found that for five-variable functions, on the average, minimum EX-SOPs require 6.02 products while minimum SOPs require 7.46 products. In our minimization algorithms, we did not consider 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. We consider this problem in a separate paper [2], where the tables of minimum EX-SOPs with up to five variables are used. The table of minimum EX-SOPs with five variables is also used in the heuristic simplification program for EX-SOPs with six or more inputs [1].

Acknowledgements

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

References

[1] D. Debnath and T. Sasao, "An optimization of AND-OR-EXOR three-level expressions by table look-up," *IEICE Technical Report,* vol. 95, No. 307, pp. 9–16, Oct. 1995.

[2] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," *Proc. The Sixth Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI '96)*, Fukuoka, Japan, Nov. 1996.

[3] 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.

[4] H. Fleisher, J. Giraldi, D. B. Martin, R. L. Phoenix and M. A. Tavel, "Simulated annealing as a tool for logic optimization in a CAD environment," *Proc. Int. Conf. Computer-Aided Design*, pp. 203–205, Nov. 1985.

[5] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, Cambridge, 1985.

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

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

[8] 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.

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

[10] D. Pellerin and M. Holley, *Practical Design Using Programmable Logic*, Prentice Hall, 1991, p. 180.

[11] 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.

[12] 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.

[13] T. Sasao, "AND-EXOR expressions and their optimization," in T. Sasao, Editor, *Logic Synthesis and Optimization*, Kluwer Academic Publishers, Boston, 1993.

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

[15] T. Sasao, "Representations of logic functions using EXOR operators," in T. Sasao and M. Fujita, Editors, *Representations of Discrete Functions*, Kluwer Academic Publishers, Boston, 1996.

[16] 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.

[17] A. Weinberger, "High-speed programmable logic array adders," *IBM J. Res. & Develop.,* vol. 23, No. 2, pp. 163–178, March 1979.

[18] *The TTL Data Book for Design Engineers*, Texas Instruments Inc., 1973.