IEICE TRANS. FUNDAMENTALS, VOL.E90–A, NO.5 MAY 2007

PAPER Special Section on Discrete Mathematics and Its Applications

A New Equivalence Relation of Logic Functions and Its Application in the Design of AND-OR-EXOR Networks

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

SUMMARY This paper presents a design method for AND-OR-EXOR three-level networks, where a single two-input exclusive-OR (EXOR) gate is used. The network realizes an EXOR of two sum-of-products expressions (EX-SOPs). The problem is to minimize the total number of products in the two sum-of-products expressions (SOPs). We introduce the notion of *μ-equivalence* of logic functions to develop *exact minimization* algorithms for EX-SOPs with up to five variables. We minimized all the NP-representative functions for up to five variables and showed that five-variable functions require 9 or fewer 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}, which is the upper bound for SOPs. We also found that, for five-variable functions, on the average, minimum EX-SOPs require about 40% fewer literals than minimum SOPs.
key words: *three-level networks, AND-EXOR, NP-equivalence, coordinate representation, μ-equivalence, spectral method, logic minimization*

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 produces better networks [18]–[23]. For example, on the average, five-variable functions require 7.46 products in minimum sum-of-products expressions (SOPs), while 6.16 products in minimum EXOR sum-of-products expressions (ESOPs) [19]. To realize an arbitrary function of six variables, minimum SOPs (MSOPs) require 32 or fewer products, while minimum ESOPs require 15 or fewer products [13]. In these designs, EXOR gates with unlimited fan-in are used. However, in most technologies, EXOR gates with many inputs are expensive.

This paper presents an *exact minimization* method for AND-OR-EXOR three-level networks. The network realizes an EXOR of two SOPs (EX-SOPs), where only a single two-input EXOR gate is used (Fig. 1). An EX-SOP for a function f can be written as $F = G \oplus H$, where G and H are SOPs. The objective of the minimization is to reduce the total number of products in G and H.

Manuscript received August 21, 2006.
Manuscript revised November 11, 2006.
Final manuscript received December 28, 2006.
[†]The author is with the Department of Computer Science and Engineering, Oakland University, Rochester, Michigan 48309, U.S.A.
[††]The author is with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820-8502 Japan.
a) E-mail: debnath@oakland.edu
b) E-mail: sasao@cse.kyutech.ac.jp
DOI: 10.1093/ietfec/e90–a.5.932

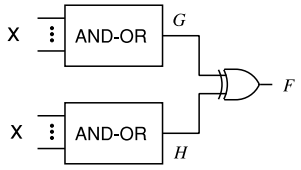

Fig. 1 An AND-OR-EXOR three-level network.

AND-OR-EXOR is one of the simplest three-level architecture, since it contains only a single two-input EXOR gate. However, its logic capability is quite high. Because of this, various programmable logic devices (PLDs) with two-input EXOR gates in the outputs were developed. Especially, RICOH, Lattice and AMD (MMI) produced series of such PLDs [14], [16], [17], and recently millions of complex PLDs (CPLDs) with output EXOR gates have been shipped [1], [2]. An AND-OR-EXOR three-level network is suitable for implementing arithmetic functions. For example, Texas Instruments' SN74LS181 arithmetic logic unit has EXOR gates in the outputs [24]. Programmable logic arrays (PLAs) with two-input EXOR gates in the outputs efficiently realize adders [5], [25].

Design methods for AND-OR-EXOR three-level networks were considered in the past [9], [16], [22]. Upper bounds on the number of products in an AND-OR-EXOR expansion was also reported [7]. During the last several years significant progress in the heuristic minimization of EX-SOPs have been made [4], [8], [12], [20]. However, other than [3], no exact minimization algorithm for EX-SOPs is reported. In this paper, an exact minimization algorithm for EX-SOPs is presented. The algorithm is based on the notion of a new equivalence class, namely *μ-equivalence*, of logic functions.

The rest of the paper is organized as follows: Sect. 2 introduces the terminology and develops the concept of μ-equivalence of logic functions. Section 3 provides the key idea for the minimization. Section 4 describes how the μ-equivalence of logic functions can be used to reduce the computation time and shows a minimization algorithm for EX-SOPs with five variables. Section 5 reports experimental results. Section 6 presents conclusions and comments.

2. Definitions and Basic Properties

This section introduces the notations used in the paper and considers the modified coordinate representation of logic functions. By using the representation, we develop the con-

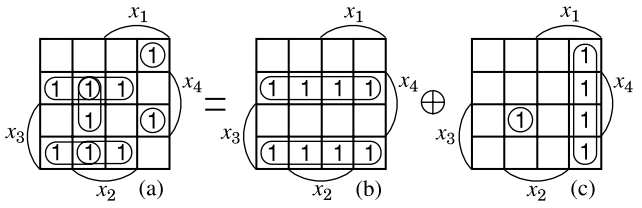

Fig. 2 Karnaugh maps for the SOP and EX-SOP in Example 1.

cept of the μ-equivalence of logic functions and illustrate its properties. In this paper, we distinguish functions and their expressions. We use lower case letters, such as f, g, and h, to represent functions, and upper case letters, such as F, G, and H, to represent expressions of functions.

Definition 1: A sum-of-products expression (***SOP***) is the OR of product terms. An ***EX-SOP*** is the EXOR of two SOPs. An ***ESOP*** is the EXOR of product terms.

Example 1: Figure 2(a) shows an SOP for f: $F_{sop} = \bar{x}_1\bar{x}_3x_4 \vee x_2\bar{x}_3x_4 \vee \bar{x}_1x_2x_4 \vee \bar{x}_1x_3\bar{x}_4 \vee x_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4$. Figures 2(b) and 2(c) show an EX-SOP for f: $F_{exsop} = (\bar{x}_3x_4 \vee x_3\bar{x}_4) \oplus (\bar{x}_1x_2x_3x_4 \vee x_1\bar{x}_2)$. Note that F_{sop} and F_{exsop} represent the same function.

Definition 2: An expression of a function is said to be ***minimum*** if it has the least number of product terms.

Example 2: In Example 1, F_{sop} is a minimum SOP and F_{exsop} is a minimum EX-SOP.

Definition 3: Let τ**(SOP:** f**)** and τ**(EX-SOP:** f**)** be the number of products in a minimum SOP (***MSOP***) for function f and minimum EX-SOP (***MEX-SOP***) for function f, respectively.

Example 3: In Example 1, τ(SOP: f) = 7 and τ(EX-SOP: f) = 4. For this function, MSOP requires 23 literals, while MEX-SOP requires only 10 literals.

Let a function f be represented as follows:

$$f = g \oplus h. \tag{1}$$

Note that g and h correspond to G and H in Fig. 1, respectively. To compute τ(EX-SOP: f), we must choose g and h such that they satisfy Eq. (1). Thus, we have τ(EX-SOP: f) = min$\{\tau$(SOP: g) + τ(SOP: h)$\}$.

Definition 4: Let τ**(EX-SOP:** F**)** be the number of products in an EX-SOP F. We note that τ(EX-SOP: f) which is introduced in Definition 3 and τ(EX-SOP: F) are different.

Example 4: In Example 1, τ(EX-SOP: F_{exsop}) = 4.

Definition 5: Let the minterm expansion of an n-variable function be $f(x_1, x_2, \ldots, x_n) = m_0 \cdot \bar{x}_1\bar{x}_2\cdots\bar{x}_n \vee m_1 \cdot \bar{x}_1\bar{x}_2\cdots x_n \vee \cdots \vee m_{2^n-1} \cdot x_1x_2\cdots x_n$, where $m_0, m_1, \ldots, m_{2^n-1} \in \{0, 1\}$. The 2^n bit binary number $m_0m_1\cdots m_{2^n-1}$ is the ***binary representation*** of f. The hexadecimal number which is obtained from the binary number $m_0m_1\cdots m_{2^n-1}$ is the ***hexadecimal representation*** of f. To denote a binary (hexadecimal) number, a subscripted 2 (16) is used after it.

Example 5: Let the three-variable function $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1$. The binary representation of f is $\mathtt{10001111}_2$. Let the five-variable function $g(x_1, x_2, x_3, x_4, x_5) = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5 \vee x_1$. The hexadecimal representation of g is $\mathtt{8000ffff}_{16}$.

2.1 NP-Equivalence Classes

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

Definition 6: The set of functions which are identical under (a) the permutation of the variables and/or (b) the complementation (i.e., negation) of one or more variables are called ***NP-equivalent functions*** [10], [11], [15]. Let $f \overset{\text{NP}}{\sim} g$ denote that f and g are NP-equivalent, and let $f \overset{\text{NP}}{\not\sim} g$ denote that f and g are not NP-equivalent. NP-equivalent functions form an ***NP-equivalence class*** of functions.

Example 6: Consider the three functions: $f_1(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3$, $f_2(x_1, x_2, x_3) = \bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_3$, and $f_3(x_1, x_2, x_3) = \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3$. Since $f_2(x_3, x_2, x_1) = x_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3 = f_1(x_1, x_2, x_3)$, we have $f_1 \overset{\text{NP}}{\sim} f_2$, and since $f_3(x_1, x_2, \bar{x}_3) = \bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_3 = f_2(x_1, x_2, x_3)$, we have $f_2 \overset{\text{NP}}{\sim} f_3$. Therefore, the functions f_1, f_2 and f_3 belong to the same NP-equivalence class.

Definition 7: The function which has the smallest binary representation among the functions of an NP-equivalence class is the ***NP-representative function*** of the class.

Example 7: Functions x_1x_2, $x_1\bar{x}_2$, $\bar{x}_1x_2$, and $\bar{x}_1\bar{x}_2$ form an NP-equivalence class of two variables. In binary representation: $x_1x_2 = \mathtt{0001}_2$, $x_1\bar{x}_2 = \mathtt{0010}_2$, $\bar{x}_1x_2 = \mathtt{0100}_2$, and $\bar{x}_1\bar{x}_2 = \mathtt{1000}_2$. Since $\mathtt{0001}_2 < \mathtt{0010}_2 < \mathtt{0100}_2 < \mathtt{1000}_2$, the NP-representative function of this class is x_1x_2.

For NP-equivalent functions we have the following:

Property 1: If $f \overset{\text{NP}}{\sim} g$, then τ(SOP: f) = τ(SOP: g) and τ(EX-SOP: f) = τ(EX-SOP: g).

2.2 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 then present properties of modified coordinate representation and μ-equivalence classes.

Definition 8: Let $w(f)$ be the number of true minterms of the function f.

Definition 9 ([10]): The ***coordinate representation*** of a five-variable function f, denoted by $COR(f)$, 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:

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

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

Example 8: Let a five-variable function f be $\mathtt{65c5ab8d}_{16}$. Here $c_4 = 16 - w(\mathtt{65c5ab8d}_{16} \oplus \mathtt{33333333}_{16}) = 16 - w(\mathtt{56f698be}_{16}) = 16 - 19 = -3$. $COR(f) = (-1;\ 1, -1, 3, -3, 1;\ 1, -3, -1, 7, -1, 5, -3, -3, -7, -1;\ 1, -1, 3, -1, -1, 5, 1, 1, 3, 3;\ 3, -1, 1, 1, 3;\ 1)$.

Definition 10: The ***modified coordinate representation*** of a five-variable function f, denoted by $\boldsymbol{\mu(f)}$, consists of 32 integers: $\mu(f) = (d_0;\ d_1, d_2, d_3, d_4, d_5;\ d_{12}, d_{13}, d_{14}, d_{15}, d_{23}, d_{24}, d_{25}, d_{34}, d_{35}, d_{45};\ d_{123}, d_{124}, d_{125}, d_{134}, d_{135}, d_{145}, d_{234}, d_{235}, d_{245}, d_{345};\ d_{1234}, d_{1235}, d_{1245}, d_{1345}, d_{2345};\ d_{12345})$. The $\mu(f)$ 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}.

Example 9: For the function f shown in Example 8, $\mu(f) = (-1;\ 1, 1, 1, 3, 3;\ 1, 1, 1, 1, 3, 3, 3, 5, 7, 7;\ 1, 1, 1, 1, 1, 1, 3, 3, 3, 5;\ 1, 1, 1, 3, 3;\ 1)$.

The definitions of the coordinate representation and the modified coordinate representation for functions with any arbitrary n variables are similar to Definitions 9 and 10, respectively.

Theorem 1: $\mu(f)$ is invariant under (a) the permutation of the variables and/or (b) the complementation of one or more variables of f.

Proof : We will show that $(d_1, d_2, d_3, d_4, d_5)$ is invariant under (a) and (b). The other groups can be shown to be invariant in a similar way. From the definition of $COR(f)$, the permutation of the variables only permutes the values of $(c_1, c_2, c_3, c_4, c_5)$ within the group. The elements of the group $(c_1, c_2, c_3, c_4, c_5)$ are rearranged in ascending order to obtain the group $(d_1, d_2, d_3, d_4, d_5)$ of $\mu(f)$. Thus, $(d_1, d_2, d_3, d_4, d_5)$ is invariant under the permutation of the variables. Note that $c_i = 2^{n-1} - w(f \oplus x_i) = 2^{n-1} - \big(w(\bar{x}_i f) + w(x_i \bar{f})\big) = \big(2^{n-1} - w(x_i \bar{f})\big) - w(\bar{x}_i f) = w(x_i f) - w(\bar{x}_i f)$. This implies that the complementation of the variable x_i only changes the sign of c_i. Note that we discard the signs of c_i's to obtain d_i's. Therefore, $(d_1, d_2, d_3, d_4, d_5)$ is invariant under the complementation of the variables. □

Definition 11: 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. The functions that are μ-equivalent form a ***μ-equivalence class*** of functions.

From the definitions of the NP-equivalent and μ-equivalent functions, and by Theorem 1, we have the following:

Property 2: If $f \overset{\text{NP}}{\sim} g$, then $f \overset{\mu}{\sim} g$.

Example 10: The five-variable functions $\mathtt{fb1e4b3d}_{16}$ and $\mathtt{6bfa79e1}_{16}$ are NP-equivalent, and both have the same modified coordinate representation: $(-4;\ 0, 2, 2, 2, 4;\ 0, 0, 2, 2, 2, 2, 2, 2, 4, 4;\ 0, 0, 0, 0, 2, 2, 2, 2, 4, 8;\ 2, 2, 4, 4, 4;\ 2)$.

For functions with three or fewer variables the converse of Property 2 holds, i.e., if $f \overset{\mu}{\sim} g$ then $f \overset{\text{NP}}{\sim} g$. But the converse of Property 2 is not true for all functions with four or more variables.

Observation 1: Among the five-variable functions, there exist functions f and g such that $f \overset{\mu}{\sim} g$ but $f \overset{\text{NP}}{\not\sim} g$.

Example 11: The five-variable functions $\mathtt{07b4e93e}_{16}$, $\mathtt{166ea5b9}_{16}$, $\mathtt{16979ae5}_{16}$ and $\mathtt{169a9e75}_{16}$ belong to different NP-equivalence classes, but they have the same modified coordinate representation: $(-1;\ 1, 1, 1, 1, 3;\ 1, 1, 1, 1, 1, 1, 3, 3, 3, 3;\ 1, 1, 1, 1, 1, 1, 3, 3, 7, 7;\ 1, 3, 3, 5, 5;\ 3)$.

3. Minimization of EX-SOPs

This section presents several key ideas for the minimization of EX-SOPs.

3.1 Idea for Minimization

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

Theorem 2: Let f be an n-variable function and $\mathcal{G}_n$ be the set of all the n-variable functions. Then,

$$\tau(\text{EX-SOP: } f) = \min_{g \in \mathcal{G}_n} \big\{\tau(\text{SOP: } g) + \tau(\text{SOP: } f \oplus g)\big\}. \quad (2)$$

Proof : Suppose that the MEX-SOP for f is represented as $f = g \oplus h$, which implies $h = f \oplus g$. Since all possible g's are considered in Eq. (2), we have the theorem. □

3.2 Straightforward Minimization Algorithm

Based on Theorem 2, the following is a straightforward algorithm to minimize EX-SOPs.

Let f be the n-variable function to be minimized and $\mathcal{G}_n$ be the set of all the n-variable functions. Let *best* be the minimum number of products among all the EX-SOPs considered so far and *sol* be a pair of n-variable functions.

Algorithm 1: (EX-SOP Minimization: Straightforward)

1. Initialize: $best \leftarrow \tau(\text{SOP: } f)$; $sol \leftarrow (f, 0)$;

2. **for each** $g \in \mathcal{G}_n$ such that $\tau(\text{SOP: } g) < best$ **do**
   $temp \leftarrow \tau(\text{SOP: } g) + \tau(\text{SOP: } f \oplus g)$;
   if $temp < best$ **then**
   $best \leftarrow temp$; $sol \leftarrow (g, f \oplus g)$;
   endif
   repeat
3. Print *best* and *sol*.

3.3 Reduction of Search Space

To obtain a minimum EX-SOP for an n-variable function f by using Algorithm 1, we must check about 2^{2^n} different g's in the worst case, and choose the g that produces a minimum value for $\tau(\text{SOP: } g) + \tau(\text{SOP: } f \oplus g)$. For up to four-variable functions, this search space is relatively small, and Algorithm 1 produces solutions quickly. However, for five-variable functions, the search space is extremely large. The following theorem shows that we can drastically reduce the search space.

Theorem 3: In Algorithm 1, suppose we need to find an EX-SOP with fewer than t products. If we consider g's so that $\tau(\text{SOP: } g)$ is in increasing order, then we have only to consider those g's, such that $\tau(\text{SOP: } g) \leq \lceil t/2 - 1 \rceil$, 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 $\tau(\text{SOP: } g) \leq \lceil t/2 - 1 \rceil$. Now it is sufficient to prove that a further increase in $\tau(\text{SOP: } g)$ by considering other g's cannot produce an EX-SOP F with $\tau(\text{EX-SOP: } F) < t$. We prove this by contradiction. We already considered g's such that $\tau(\text{SOP: } g) = 0, 1, \ldots, \lceil t/2 - 1 \rceil$ and to obtain $\tau(\text{EX-SOP: } F) < t$ we increase $\tau(\text{SOP: } g)$ by 1, i.e., $\tau(\text{SOP: } g)$ is now $\lceil t/2 \rceil$. We have $\tau(\text{EX-SOP: } F) = \tau(\text{SOP: } g) + \tau(\text{SOP: } f \oplus g)$. Therefore, $\tau(\text{SOP: } f \oplus g) < \lceil t/2 \rceil$ which implies $\tau(\text{SOP: } f \oplus g) = \lceil t/2 - 1 \rceil, \ldots, 1$, or 0. But if such an EX-SOP exist, it must have been found when we considered $\tau(\text{SOP: } g) = 0, 1, \ldots, \lceil t/2 - 1 \rceil$. Thus, $\tau(\text{EX-SOP: } F)$ is not less than t. Similarly, we can show that a further increase in $\tau(\text{SOP: } g)$ by considering other g's cannot produce an EX-SOP with fewer products. Hence, we have the theorem. □

To find an EX-SOP with fewer than 8 products, we have only to consider those g's such that $\tau(\text{SOP: } g) \leq 3$. Similarly to find an EX-SOP with fewer than 9 products, we have only to consider those g's such that $\tau(\text{SOP: } g) \leq 4$.

Example 12: The numbers of five-variable functions which require up to three and four products in their minimum SOPs are 839,000 and 16,888,780, respectively. Thus, by using Theorem 3 for five-variable functions, to find an EX-SOP with fewer than 7 or 8 products, we must consider at most 839,000 g's, and to find an EX-SOP with fewer than 9 or 10 products, we must consider at most 16,888,780 g's. These numbers of g's are only 0.0195% and 0.3932% of the total number of five-variable functions, respectively.

4. Minimization of EX-SOPs with Five Variables

In this section, we first develop several techniques to reduce the computation time for EX-SOPs. We then present an algorithm to minimize EX-SOPs with five variables. The most time consuming part of Algorithm 1 is the computation of $\tau(\text{SOP: } g) + \tau(\text{SOP: } f \oplus g)$ in step 2. The techniques we used to obtain $\tau(\text{SOP: } g)$ and $\tau(\text{SOP: } f \oplus g)$ are different which is explained in the following.

4.1 Obtain $\tau(\text{SOP: } g)$: Eliminate Redundant Work

We can quickly obtain $\tau(\text{SOP: } g)$ in Algorithm 1, by using a table of $g \in \mathcal{G}_n$ and the corresponding $\tau(\text{SOP: } g)$. According to Theorem 3, we can reduce the number of g's in step 2 of Algorithm 1 by considering the g's in ascending order of their $\tau(\text{SOP: } g)$. We found that, for five-variable functions, the maximum value of $\tau(\text{SOP: } g)$ is 4. For five variables, the number of g's such that $\tau(\text{SOP: } g) \leq 4$ is 16,888,780, but it is inconvenient to work with a table of such size. To reduce the table size, we use NP-equivalence classes. From Property 1, the number of products in MSOPs for the NP-equivalent functions are equal. There are only 6,138 NP-representative functions whose MSOPs require up to four products. Thus, we use the *sorted function table* (Fig. 3). The left column of the sorted function table stores only those NP-representative function g_{rep}'s such that $\tau(\text{SOP: } g_{rep}) \leq 4$, and the right column stores the corresponding $\tau(\text{SOP: } g_{rep})$'s. The data in the table is arranged in ascending order of $\tau(\text{SOP: } g_{rep})$. We access the sorted function table sequentially from the beginning — i.e., starting with the smallest $\tau(\text{SOP: } g_{rep})$ — to get an NP-representative function g_{rep} and the corresponding $\tau(\text{SOP: } g_{rep})$. We then obtain g's by generating all the functions of the class g_{rep}. Since $\tau(\text{SOP: } g_{rep})$ and $\tau(\text{SOP: } g)$ are equal, we obtain $\tau(\text{SOP: } g)$ quickly.

4.2 Compute $\tau(\text{SOP: } f \oplus g)$: Time Consuming Part

We have shown in Sect. 4.1 that $\tau(\text{SOP: } g)$ can be quickly obtained from the sorted function table. Thus, the most time consuming part of Algorithm 1 is the computation of $\tau(\text{SOP: } f \oplus g)$ in step 2. A straightforward computation of $\tau(\text{SOP: } f \oplus g)$ is time consuming. Thus, instead of doing logic minimization, we can use a table of all the five-variable functions

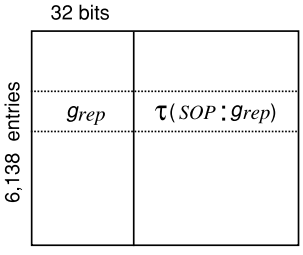

Fig. 3 Sorted function table (NP-representative functions with $\tau(\text{SOP: } g_{rep}) \leq 4$).

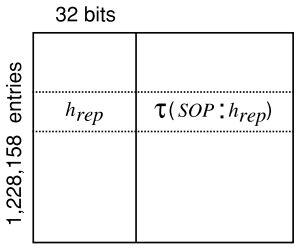

Fig. 4 Cost table (NP-representative functions).

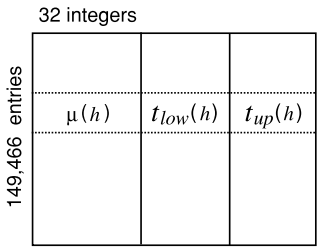

Fig. 5 Modified cost table (μ-representative functions).

h and the corresponding τ(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 Reduce Table Size: Use NP-Equivalence

To reduce the table size, we use a *cost table* (Fig. 4) for all the NP-representative functions of five variables. The number of NP-equivalence classes of five-variable functions is 1,228,158. The left column of the cost table stores all the NP-representative functions h_{rep} and the right column stores the corresponding τ(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 τ(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 τ(SOP: h_{given}) = τ(SOP: h_{rep}). By using h_{rep} to look-up the cost table, we obtain τ(SOP: h_{given}). However, to get the NP-representative function h_{rep} from a given function h_{given} is rather time consuming.

4.2.2 Quickly Estimate τ(SOP: $f \oplus g$): Use μ-Equivalence

Since determination of τ(SOP: $f \oplus g$) by using the cost table requires computation of an NP-representative function which is a time consuming process, we use properties of μ-equivalence classes to quickly estimate it. Often the estimated value is sufficiently accurate to avoid the time consuming computation of NP-representative functions.

Definition 12: Let $t_{\ell ow}(h) = \min_{h \overset{\mu}{\sim} h_j} \tau$(SOP: h_j) and $t_{up}(h) = \max_{h \overset{\mu}{\sim} h_j} \tau$(SOP: h_j).

Example 13: For the four functions shown in Example 11, the number of products in the MSOPs are 10, 9, 10 and 8, respectively. Only the functions of these four NP-equivalence classes have the modified coordinate representation shown in Example 11. Thus, for any function h in these classes, $t_{\ell ow}(h) = 8$ and $t_{up}(h) = 10$.

The modified coordinate representation of h, denoted by $\mu(h)$, is quickly calculated from h and has Property 2 (Sect. 2.2). But Observation 1 shows that, for five variables, $\mu(h)$ 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 the μ-equivalence classes cannot uniquely identify the NP-equivalence classes, we can use them to quickly estimate the value of τ(SOP: $f \oplus g$) in step 2 of Algorithm 1. Thus, we use the *modified cost table* (Fig. 5). The left column of the modified cost table stores the distinct values of $\mu(h)$ for any five-variable function h. The middle and the right columns store the corresponding values of $t_{\ell ow}(h)$ and $t_{up}(h)$, respectively. Since $\mu(h)$ is an array of 32 integers, we use hash technique to look-up the modified cost table.

Observation 2: For any five-variable function h, the differences between τ(SOP: h) and $t_{\ell ow}(h)$ are small, which is shown in the following:

δ	0	1	2	3	4
Ψ	1,664,817,920	2,169,159,536	449,837,280	11,132,720	19,840
%	38.76	50.51	10.47	0.26	0.00

δ: τ(SOP: h) − $t_{\ell ow}(h)$. Ψ: no. of functions. %: % of total functions.

Observation 2 reveals that the differences between τ(SOP: h) and $t_{\ell ow}(h)$ are small. Thus, without computing τ(SOP: $f \oplus g$) and by using the modified cost table, we can often show that τ(SOP: g) + $t_{\ell ow}(f \oplus g) \geq best$, i.e., $temp \geq best$ in step 2 of Algorithm 1. This implies that using the modified cost table, we can often avoid the computation of τ(SOP: $f \oplus g$). If we have τ(SOP: g) + $t_{\ell ow}(f \oplus g) < best$, i.e., a possibility that $temp < best$ in step 2 of Algorithm 1, only then we obtain τ(SOP: $f \oplus g$) from the cost table.

Observation 3: For any five-variable function h, the differences between $t_{\ell ow}(h)$ and $t_{up}(h)$ are small, which is shown in the following:

γ	0	1	2	3	4
Ψ	413,076,664	1,692,187,888	1,913,993,984	273,853,840	1,854,920
%	9.62	39.40	44.56	6.38	0.04

γ: $t_{up}(h) - t_{\ell ow}(h)$. Ψ: no. of functions. %: % of total functions.

For about 9.62% cases $t_{\ell ow}(f \oplus g)$ and $t_{up}(f \oplus g)$ are equal (Observation 3). In these cases we get the actual — not an estimated — value of τ(SOP: $f \oplus g$) from the modified cost table. When we have τ(SOP: g) + $t_{\ell ow}(f \oplus g) < best$ in step 2 of Algorithm 1 and $t_{\ell ow}(f \oplus g) \neq t_{up}(f \oplus g)$, we obtain τ(SOP: $f \oplus g$) from the cost table by using a more time consuming routine.

4.3 Minimization Algorithm

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

Let f be the function to be minimized. Let g, h and g_{rep} represent functions, and t_g, $t_{\ell ow}$, t_{up} and t_h represent the number of products. t_{exsop} denotes the minimum number of products in EX-SOP ever found and t_{bound} represents an upper bound on the number of products in an MSOP for g. From [7], we know that $t_{exsop} < 11$. Section 4.1 shows that $t_{bound} \leq 4$. This algorithm uses three tables: the sorted function table, the cost table, and the modified cost table; we have already introduced them in Sects. 4.1 and 4.2.

Algorithm 2: (EX-SOP Minimization: Five Variables)

1. Initialize: $t_{exsop} \leftarrow 11$; $t_{bound} \leftarrow 4$.
2. $t_g \leftarrow \tau(\text{SOP: } g_{rep})$, where g_{rep} is the first function of the sorted function table.
3. Generate all the functions of the NP-equivalence class g_{rep}. Take the first function g of this class.
4. $h \leftarrow f \oplus g$. Calculate $\mu(h)$. Look-up the modified cost table by using $\mu(h)$, and let $t_{\ell ow} \leftarrow t_{\ell ow}(h)$ and $t_{up} \leftarrow t_{up}(h)$.
5. If $t_g + t_{\ell ow} \geq t_{exsop}$ (i.e., reduction of t_{exsop} is impossible using the current h), then go to step 10.
6. If $t_{\ell ow} = t_{up}$ (i.e., $\tau(\text{SOP: } h)$ is obtained from the modified cost table), then $t_h \leftarrow t_{\ell ow}$ and go to step 8.
7. Look-up the cost table by using h; let $t_h \leftarrow \tau(\text{SOP: } h)$.
8. If $t_g + t_h \geq t_{exsop}$ (i.e., reduction of t_{exsop} is impossible using the current h), then go to step 10.
9. (Found a new solution.) $t_{exsop} \leftarrow t_g + t_h$. 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 11.
10. Take the next function g in the class g_{rep} (g was computed in step 3), and go to step 4.
    If all the functions of the class g_{rep} are already considered, then $t_g \leftarrow \tau(\text{SOP: } g_{rep})$ where g_{rep} is the next function of the sorted function table.
    If $t_{bound} < t_g$, then go to step 11, otherwise go to step 3.
11. Print the latest solution saved in step 9, and t_{exsop} as the final number of products.

The execution time of Algorithm 2 mainly depends on t_{bound}, the upper bound on the number of products in the MSOP for g in Theorem 2. In Algorithm 2, t_{bound} is initialized in step 1 and it is updated in step 9. We have mentioned in Sect. 4.2 that we must compute the NP-representative function h_{rep} of h before look-up the cost table for $\tau(\text{SOP: } h)$. But the computation of h_{rep} is time consuming. This leads to a relatively long execution time for step 7 in Algorithm 2. However, the following observation reveals that the algorithm executes step 7 less frequently.

Observation 4: We have conducted an experiment by using Algorithm 2 for 10,000 pseudo-random functions with 16 true minterms. We found that, for each of the functions, on the average, steps 5, 6 and 7 of Algorithm 2 execute 43,519, 1,972 and 138 times, respectively. It should be noted that although each of the passes through step 7 of Algorithm 2 takes a significant amount of computation time, it is not the most time consuming step of the algorithm. On the average, the step takes about 11% of the execution time of the algorithm. Each execution of the step takes about 28 microseconds on a 3.00 GHz Intel Pentium 4 CPU.

4.4 Technique to Reduce Number of Literals

A given function f may have more than one MEX-SOP, such that $f = g_1 \oplus h_1 = g_2 \oplus h_2 = \cdots = g_k \oplus h_k$, where $\tau(\text{EX-SOP: } f) = \tau(\text{SOP: } g_1) + \tau(\text{SOP: } h_1) = \tau(\text{SOP: } g_2) + \tau(\text{SOP: } h_2) = \cdots = \tau(\text{SOP: } g_k) + \tau(\text{SOP: } h_k)$. But Algorithm 2 finds only one MEX-SOP. We modified steps 5, 8 and 9 of Algorithm 2 to generate a set of MEX-SOPs and took the EX-SOP with the minimum number of literals. To count the literals of an EX-SOP, we must count the literals of two SOPs. Our approach is to count the number of literals of SOPs by table look-up. We use a table, similar to the one shown in Fig. 4. Instead of storing $\tau(\text{SOP: } h_{rep})$, the table stores the number of literals in MSOP for h_{rep} in the right column. Note that we use Quine-McCluskey method [15] to minimize SOPs, where the number of literals may not be the minimum. To reduce the computation time we generate at most 100 MEX-SOPs for a given function and took the EX-SOP with the minimum number of literals.

5. Experimental Results

We implemented the proposed EX-SOP minimization algorithms in C language and carried out experiments on a 3.00 GHz Intel Pentium 4 PC with one gigabytes memory running Red Hat Enterprise Linux WS Release 4. We minimized all the NP-representative functions of four and five variables. A five-variable function, on the average, takes about 34 milliseconds of CPU time; this average is obtained by minimizing 10,000 randomly generated functions. For five-variable functions, when the number of products in the minimum EX-SOP (MEX-SOP) is 6 and 9 (worst case), we could minimize each of the functions within 11 and 140 milliseconds of CPU time, respectively. The program requires about 25 megabytes of memory space.

Table 1 shows the numbers of five-variable functions requiring t products by different minimum expressions. In the table, 'av' indicates average number of products which is equal to $\left(\sum_t (t \times \lambda_t)\right)/2^{32}$, where λ_t represents the number of functions requiring t products and 2^{32} is the total number of five-variable functions. A similar table for four-variable functions is omitted for brevity. In Table 1, data for SOPs and ESOPs are taken from [19], and EX-SOPs were minimized by Algorithm 2. For five-variable functions, on the average, MEX-SOPs require 6.02 products, while minimum SOPs (MSOPs) require 7.46 prod-

ucts. For five-variable functions, on the average, MEX-SOPs require fewer products than minimum ESOPs. We found that, for four- and five-variable functions, the upper bounds on the number of products in MEX-SOPs are 5 and 9, respectively. Thus, from the observations in [7], MEX-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 MEX-SOPs require 9 products. The NP-representative function of the class is $\texttt{177e7ee9}_{16}$; the class has 32 NP-equivalent functions.

Table 2 shows a distribution of five-variable functions with respect to the number of products in MSOPs and MEX-SOPs. It reveals statistical data about the functions. For example, from the table we can state that there are 160 five-variable functions which require 8 products in MSOPs but only 3 products in MEX-SOPs. We found that MEX-SOPs of 84.35% of the five-variable functions require fewer products than MSOPs. The table also shows that, for some functions, the difference between the number of products in an MSOP and an MEX-SOP is up to 10. For example, an MEX-SOP for $\texttt{69969669}_{16}$ (parity function with five variables) requires 6 products, while its MSOP requires 16 products.

To compare the number of literals in MSOPs and MEX-SOPs, we minimized EX-SOPs for all the NP-representative functions of five variables using the technique presented in Sect. 4.4, where minimization of the number of products is the primary objective and minimization of the number of literals is the secondary objective. We note that the number of literals is equal to the total fan-in of the AND gates. Some of the findings of the experiment are as follows: (a) On the average, MEX-SOPs require 16.49 literals and MSOPs require 27.24 literals, where SOPs are minimized using Quine-McCluskey method [15]. (b) For some functions MEX-SOPs require up to 80% fewer literals than MSOPs. For example, an MEX-SOP for $\texttt{69969669}_{16}$ (a parity function) requires 16 literals, while its MSOP requires 80 literals. The EX-SOP can be written as $G \oplus H$, where G and H are the MSOPs for $\texttt{00ffff00}_{16}$ and $\texttt{69696969}_{16}$, respectively. (c) For some (about 0.02%) functions MEX-SOPs require more literals than MSOPs. For example, an MEX-SOP for $\texttt{6ff7fefb}_{16}$ requires 6 products and 25 literals, while its MSOP requires 8 products and 18 literals. The EX-SOP can be written as $G \oplus H$, where G and H are the MSOPs for $\texttt{ffffffff}_{16}$ and $\texttt{90080104}_{16}$, respectively.

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

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

6. Conclusions and Comments

In this paper, we presented minimization algorithms for AND-OR-EXOR three-level networks which implement EX-SOPs with up to five variables. We developed the concept of μ-equivalence of logic functions and used it to reduce the computation time of the minimization program for EX-SOPs with five variables. We minimized all the NP-representative functions with up to five variables, generated the tables of minimum EX-SOPs (MEX-SOPs) for them, and showed that MEX-SOPs for five-variable functions re-

Table 2 Numbers of five-variable functions requiring t_{sop} and t_{exsop} products in minimum SOPs and minimum EX-SOPs, respectively.

	t_{exsop}									
t_{sop}	0	1	2	3	4	5	6	7	8	9
0	1									
1		243								
2			20676							
3			2880	815200						
4			1920	405020	15642840					
5			512	237216	18144480	136346872				
6				51520	10805520	289507904	398618712			
7				2880	2801360	196734672	1077053888	120807712		
8				160	395350	60602400	840982932	351985564	97840	
9					45760	10318900	293832472	266795424	488960	
10					3680	1217280	58113376	100323744	542912	
11						97600	8453632	25172352	417376	32
12						4960	914840	5090088	150288	
13							76080	695040	56000	
14						160	8720	69520	6400	
15							960	3744	608	
16							2	112		

quire 9 or fewer products. We established that, for n-variable functions, the upper bound on the number of products in MEX-SOPs is at most $9 \cdot 2^{n-5}$ ($n \geq 6$). This bound is tighter than the previously known one which is $5 \cdot 2^{n-4}$ ($n \geq 4$) [7]. This upper bound for MEX-SOPs is also smaller than 2^{n-1}, which is the upper bound for minimum SOPs (MSOPs). We showed that, for five-variable functions, on the average, MEX-SOPs require 6.02 products, while MSOPs require 7.46 products. We also found that, for five-variable functions, on the average, MEX-SOPs require about 40% fewer literals than MSOPs. For some five-variable functions MEX-SOPs require up to 80% fewer literals than MSOPs.

In this paper we have not considered the sharing of products between two SOPs of an EX-SOP. We considered this problem in [3] and found that, for five-variable functions, the upper bound on the number of products for MEX-SOPs with product sharing is also 9. The tables of MEX-SOPs are used in another minimization program for EX-SOPs with up to five-variables [3] and in the heuristic simplification program for EX-SOPs with six or more variables [4]. A comparison with other methods is not possible because no other algorithms for the problem with five variables are published. We are presently investigating the usefulness of μ-equivalence classes for Boolean matching in cell-library binding [6].

Acknowledgements

This work was supported in part by the Ministry of Education, Science, Culture, and Sports of Japan. We thank Prof. N. Koda for providing the table NP-representative functions and Prof. J.T. Butler for carefully reviewing the manuscript.

References

[1] Altera Corporation, "MAX EPM7128 celebrates 50 million units," News & Views: Newsletter for Altera Customers, p.28, Fourth Quarter, 2000.
[2] Altera Corporation, MAX 9000 Programmable Logic Device Family Data Sheet, June 2003.
[3] 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.
[4] D. Debnath and T. Sasao, "A heuristic algorithm to design AND-OR-EXOR three-level networks," Proc. Asia and South Pacific Design Automation Conf., pp.69–74, Feb. 1998.
[5] D. Debnath and T. Sasao, "Output phase optimization for AND-OR-EXOR PLAs with decoders and its application to design of adders," IEICE Trans. Inf. & Syst., vol.E88-D, no.7, pp.1492–1500, July 2005.
[6] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.
[7] 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," Electron. Lett., vol.31, no.7, pp.541–542, March 1995.
[8] E.V. Dubrova, D.M. Miller, and J.C. Muzio, "AOXMIN-MV: A heuristic algorithm for AND-OR-XOR minimization," 4th Int. Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.37–53, Aug. 1999.
[9] 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. IEEE/ACM Int. Conf. Computer-Aided Design, pp.203–205, Nov. 1985.
[10] M.A. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, 1965.
[11] S.L. Hurst, D.M. Miller, and J.C. Muzio, Spectral Techniques in Digital Logic, Academic Press, 1985.
[12] A. Jabir and J. Saul, "Minimization algorithm for three-level mixed AND-OR-EXOR/AND-OR-EXNOR representation of Boolean functions," IEE Proc. Computers and Digital Techniques, vol.149, no.3, pp.82–96, May 2002.
[13] N. Koda and T. Sasao, "An upper bound on the number of products in minimum ESOPs," IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp.94–101, Aug. 1995.
[14] Monolithic Memories Inc., PAL/PLE DEVICE: Programmable Logic Array Handbook, Fifth Edition, 1986.
[15] S. Muroga, Logic Design and Switching Theory, John Wiley & Sons, 1979.
[16] D. Pellerin and M. Holley, Practical Design Using Programmable Logic, Prentice Hall, 1991.
[17] RICOH, CMOS Electrically Programmable Logic, Series 20, no.85-02, 1985.
[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, "AND-EXOR expressions and their optimization," in Logic Synthesis and Optimization, ed. T. Sasao, Kluwer Academic Publishers, 1993.
[20] T. Sasao, "A design method for AND-OR-EXOR three-level networks," IEEE/ACM Int. Workshop on Logic Synthesis, pp.8:11–8:20, May 1995.
[21] T. Sasao, "Representations of logic functions using EXOR operators," in Representations of Discrete Functions, eds. T. Sasao and M. Fujita, Kluwer Academic Publishers, 1996.
[22] K. Shu, H. Yasuura, and S. Yajima, "Optimization of PLDs with output parity gates," National Convention, Information Processing Society of Japan, March 1985.
[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. Integr. Circuits Syst., vol.15, no.4, pp.385–395, April 1996.
[24] Texas Instruments Inc., The TTL Data Book for Design Engineers, 1973.
[25] A. Weinberger, "High-speed programmable logic array adders," IBM J. Res. Dev., vol.23, no.2, pp.163–178, March 1979.

Debatosh Debnath received the B.Sc.Eng. and M.Sc.Eng. degrees from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 1991 and 1993, respectively, and the Ph.D. degree from the Kyushu Institute of Technology, Iizuka, Japan, in 1998. He held research positions at the Kyushu Institute of Technology from 1998 to 1999 and at the University of Toronto, Ontario, Canada, from 1999 to 2002. In 2002, he joined the Department of Computer Science and Engineering at the Oakland University, Rochester, Michigan, as an Assistant Professor. His research interests include logic synthesis, design for testability, multiple-valued logic, and CAD for field-programmable devices. He was a recipient of the Japan Society for the Promotion of Science Postdoctoral Fellowship.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in electronics engineering from Osaka University, Osaka, Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, the IBM T.J. Watson Research Center, Yorktown Heights, New York, and the Naval Postgraduate School, Monterey, California. He is now a Professor of the Department of Computer Science and Electronics at the Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than nine books on logic design, including *Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis*, and *Logic Synthesis and Verification*, Kluwer Academic Publishers, 1993, 1996, 1999, and 2001, respectively. He has served as Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan, in 1998. He received the NIWA Memorial Award in 1979, Distinctive Contribution Awards from the IEEE Computer Society MVL-TC for papers presented at ISMVLs in 1986, 1996, 2003 and 2004, and Takeda Techno-Entrepreneurship Award in 2001. He has served as an Associate Editor of the *IEEE Transactions on Computers*. He is a fellow of the IEEE.